Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «МАГІСТР»

РОЗРОБКА ТА ДОСЛІДЖЕННЯ ANDROID-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ З ІНТЕГРАЦІЄЮ МЕТОДІВ ОПТИМІЗАЦІЇ ДЛЯ БЮДЖЕТНОГО ПЛАНУВАННЯ

DEVELOPMENT AND RESEARCH OF AN ANDROID APPLICATION FOR MANAGING PERSONAL FINANCES WITH THE INTEGRATION OF OPTIMIZATION METHODS FOR BUDGET PLANNING

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗм-21
Вакулюк І. В.
Керівник:
д.т.н., професор
Андрущак І. Є.

Кваліфікаційну роботу
допущено до захисту
«___» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент Суринович О. М.
_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *магістр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 202__ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ДРУГОГО (МАГІСТЕРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ

Вакулюк Іванні Володимирівні

1. Тема кваліфікаційної роботи: Розробка та дослідження Android-додатку для управління особистими фінансами з інтеграцією методів оптимізації для бюджетного планування.
Керівник роботи: Андрущак Ігор Євгенович, д.т.н., професор.
_____
затверджені наказом закладу вищої освіти від «29» березня 2025 року № 190/01-02
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи: 04 грудня 2025 р.
3. Вихідні дані до роботи *середовище розробки Android Studio, мова програмування Kotlin, мікрофреймворк Flask для серверної частини, мова Python.*
4. Зміст розрахунково-пояснювальної записки: *аналіз сучасного стану проблеми, огляд існуючих програмних рішень та досліджень, аналіз і вибір засобів математичної оптимізації, побудова та опис математичної моделі об'єкта дослідження, обґрунтування технічних інструментів та засобів розробки, опис функціональних можливостей об'єкта проектування, розробка та пояснення структури системи та її компонентів, експериментальне дослідження результативності та ефективності розроблюваного підходу.*
5. Перелік графічного матеріалу 14 рисунків, 12 таблиць, 4 лістинги коду.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Андрущак І. Є.*		
Теоретичне дослідження та практична реалізація	*Андрущак І. Є.*		
Експериментальне дослідження системи	*Андрущак І. Є.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Андрущак І. Є.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Андрущак І. Є.*		

7. Дата видачі завдання *«02 квітня 2025 р».*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1	Провести огляд літературних джерел по темі кваліфікаційної роботи	02.05.2025	
2	Провести аналіз загальної проблеми і вибір напрямків дослідження	24.09.2025	
3	Розробити функціональну модель та архітектуру системи	01.11.2025	
4	Описати засоби розробки об’єкта проектування	19.11.2025	
5	Практична реалізація об’єкта проектування	26.11.2025	
6	Розробити методику для проведення експерименту	05.11.2025	
7	Провести аналіз результатів експерименту	15.11.2025	
8	Здача чистового варіанту кваліфікаційної роботи на кафедру	04.12.2025	

Здобувач вищої освіти _____ Вакулюк І. В.

Керівник кваліфікаційної роботи _____ Андрущак І. Є.

АНОТАЦІЯ

Вакулюк І. В. Розробка та дослідження Android-додатку для управління особистими фінансами з інтеграцією методів оптимізації для бюджетного планування. Рукопис.

Кваліфікаційна робота магістра ОП «Інженерія програмного забезпечення». Спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025. 66 с.

У роботі досліджено можливість інтеграції методів математичної оптимізації у мобільні застосунки для управляння особистими фінансами. Розроблено Android-додаток, що забезпечує планування персонального бюджету користувача на основі методів оптимізації, зокрема із застосуванням підходу квадратичного програмування. Проведено дослідження роботи алгоритму бюджетного планування, здійснено порівняння квадратичної та лінійної моделей оптимізації, оцінено продуктивність алгоритму.

Кваліфікаційна робота магістра складається з вступу, 3 розділів, висновків, списку використаних джерел, додатків.

Ключові слова: мобільний застосунок, Android-додаток, математична оптимізація, лінійне програмування, квадратичне програмування, Kotlin, Python, Android Studio, Firebase, шифрування даних.

ABSTRACT

Vakuliuk I. V. Development and Research of an Android Application for Managing Personal Finances With the Integration of Optimization Methods for Budget Planning. Manuscript.

Master's Thesis in the educational program «Software Engineering». Specialty 121 «Software Engineering». Lutsk National Technical University. Lutsk, 2025. 66 p.

The thesis investigates the possibilities of integrating mathematical optimization methods into mobile applications for personal finance management. An Android application has been developed to provide personalized budget planning based on optimization techniques, in particular through the use of a quadratic programming approach. The study includes an evaluation of the budget planning algorithm, a comparative analysis of quadratic and linear optimization models, as well as an assessment of the performance of the optimization algorithm.

The master's thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

Keywords: mobile application, Android application, personal finance management, budget planning, mathematical optimization, linear programming, quadratic programming, Kotlin, Python, Android Studio, Firebase, data encryption.

ЗМІСТ

ВСТУП

Сьогодні смартфони є невід'ємною частиною життя, а ринок мобільних застосунків активно розвивається, охоплюючи побутові, навчальні, бізнесові та фінансові сфери. Мобільні технології спрощують виконання повсякденних завдань і створюють нові можливості для користувачів. Цифрові рішення суттєво вплинули й на фінансову сферу: більшість банків пропонують додатки для платежів, переказів, контролю балансу чи інвестування. Паралельно зростає кількість додатків для особистого фінансового менеджменту – контролю, плануванню та аналізу бюджету [1]. Однак їхній функціонал здебільшого обмежується обліком доходів і витрат, формуванням звітів та аналітикою. Більшість застосунків не пропонують інструменти для побудови оптимального бюджету, оптимізації витрат, не враховують індивідуальні фінансові цілі користувача. Це формує прогалину на ринку та зумовлює актуальність розробки нових рішень – систем, які здатні не лише аналізувати витрати, а й будувати оптимальний бюджет із використанням моделей математичної оптимізації.

Метою дослідження є повноцінний мобільний застосунок для особистого фінансового планування з функціоналом формування оптимізованих бюджетних планів, що підтверджує можливість та доцільність інтеграції методів математичної оптимізації у мобільні фінансові рішення.

Об'єкт дослідження – процес управління особистими фінансами користувача у мобільному програмному забезпеченні.

Предмет дослідження – методи та алгоритми математичної оптимізації, що застосовуються для автоматизованого розподілу персонального бюджету за категоріями витрат у мобільному застосунку.

Завдання дослідження:

– здійснити аналіз предметної області, результатів існуючих досліджень;

– провести огляд популярних мобільних застосунків для персонального бюджетування та їх функціональних можливостей;

– визначити основні вимоги до функціоналу, інтерфейсу та архітектури

розроблюваного додатку для управління особистими фінансами;

– обрати метод оптимізації, який найкраще підходить для вирішення задачі оптимального розподілу бюджету, обґрунтувати його використання та побудувати математичну модель;

– здійснити вибір та обґрунтувати використання технологій, інструментів і засобів для розробки застосунку;

– розробити мобільний додаток для операційної системи Android, який реалізує функціонал планування, обліку фінансів та статистику;

– реалізувати модуль оптимізаційного розподілу бюджету та інтегрувати його до Android-застосунок;

– створити інтуїтивний та зручний інтерфейс;

– провести тестове експериментальне дослідження розробленого підходу, протестувати роботу додатку в різних сценаріях.

Наукова новизна отриманих результатів роботи полягає у формуванні нового підходу до функціоналу фінансових додатків, які не лише забезпечують контроль особистих фінансів, а й допомагають користувачу формувати бюджетний план, оптимально розподіляти витрати та досягати фінансових цілей.

Практична цінність роботи полягає у тому, що запропонований алгоритм допомагає ефективніше планувати особисті та сімейні фінанси. Подібний підхід можна застосовувати для корпоративного бюджетування, планування витрат бізнесу та й у будь-яких сферах, де потрібно оптимально розподіляти ресурси.

Щодо апробації результатів роботи, окремі аспекти дослідження та узагальнені висновки були використані при підготовці наукової статті у співавторстві, опублікованій у фаховому журналі «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк: Луцький НТУ, 2025. Випуск № 59 [2]. Публікація наведена у додатку А.

Тема застосування розробленого в роботі підходу до шифрування чутливих даних у фінансових застосунках була представлена у вигляді тез доповіді на науковій конференції «XIV International Scientific and Practical Conference. Sofia, Bulgaria» [3]. Тези доповіді та сертифікат наведені в додатку Б.

РОЗДІЛ 1
АНАЛІЗ ПРОБЛЕМАТИКИ ЗА ТЕМОЮ РОБОТИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд і аналіз предметної області проблеми, результатів існуючих теоретичних та експериментальних досліджень

Розвиток цифрових фінансових сервісів відкриває широкі можливості для контролю та аналізу власних витрат. Проте доступ до інструментів не гарантує вміння правильно ними користуватися. Фінансова грамотність – це сукупність знань, навичок та компетенцій, які дозволяють людині ефективно управляти своїми фінансами, приймати обґрунтовані фінансові рішення, планувати бюджет, контролювати витрати, накопичувати заощадження та інвестувати кошти. В контексті роботи доцільно проаналізувати чи люди у нашій країні мають достатню фінансову грамотність. Звіт Національного Банку України, підготовлений за підтримки USAID, надає детальний аналіз рівня фінансової грамотності серед українців, охоплюючи фінансові знання, поведінку та ставлення до грошей. Дослідження показало, що середній індекс фінансової грамотності в Україні зріс до 12,3 бала, з 11,6 у 2018. Проте все ще залишається нижчим за мінімальний рекомендований рівень у 14 балів за стандартами ОЕСР [4]. Це свідчить про те, що навіть за наявності доступу до сучасних фінансових інструментів, ресурсів і навчальної літератури багато людей не володіють достатніми знаннями для раціонального управління власними фінансами. Низький рівень усвідомлення власних фінансових можливостей часто призводить до імпульсивних витрат, відсутності довгострокових цілей та фінансової нестабільності.

У відповідь на цю проблему Національний банк України розробив Стратегію розвитку фінансової грамотності до 2030 року, яка, серед іншого, передбачає підвищення цифрової фінансової компетентності громадян, зокрема через доступні фінансові послуги, інтерактивні інструменти навчання. Це підтверджує актуальність розробки та вдосконалення мобільних застосунків для

підвищення фінансової грамотності [5].

Як зазначено вище, однією з ключових складових фінансової грамотності є вміння планувати та контролювати власний бюджет. Для цього існують різні стратегії бюджетування, що визначають принципи розподілу доходів, підходи до накопичень і пріоритети у витратах. Деякі стратегії орієнтовані на простоту та швидку організацію фінансів, інші – на точний контроль або досягнення конкретних фінансових цілей. Проведемо аналіз існуючих підходів, це дозволить зрозуміти, які методи використовують фінансово грамотні люди, та оцінити, наскільки сучасні мобільні додатки для особистих фінансів інтегрують подібні стратегії. Це допоможе обрати принципи організації алгоритму планування бюджету, щоб він відповідав реальним сценаріям і був зручним для користувача.

Метод zero-based-budgeting заснований на принципі «дай кожній гривні роботу» (англ. Give every dollar a job). За цією ідеєю всі гроші, які ви отримуєте, розподіляються по конкретних категоріях витрат або заощаджень, а новий період (місяць або тиждень) починається з нульового балансу. Але нульовий бюджет не означає, що ваша мета – витратити все, що ви заробляєте. Навпаки, будь-які залишки необхідно відразу спрямувати на заощадження або інвестування, а не переносити на наступний період. Таким чином, в бюджеті не залишається незапланованих коштів або спонтанних витрат [6].

Метод конвертів (англ. Envelope method) полягає у розподілі коштів за категоріями по конвертах (н-д: харчування, транспорт). В деяких джерелах є навіть підхід розділення місячного доходу на 4 конверти – по одному на тиждень. Цей підхід допомагає побудувати фінансову дисципліну. Наприклад, якщо в «конверті» на розваги залишилося 0 гривень, людина не купує нову гру чи квитки на концерт, а відкладає ці покупки на наступний місяць [6].

Існують також інші, більш відомі серед широкого загалу, правила розподілу доходів, які можна віднести до пропорційного бюджетування. Мабуть кожен чув про правило 50/30/20, яке передбачає що 50 % доходу йде на життєві потреби (житло, їжа, транспорт), 30 % – на бажані витрати (розваги, покупки, хобі), а 20 % – на заощадження чи погашення боргів. Також популярними є схеми

розподілу витрат та заощаджень 80/20 та 60/40, які обирають залежно від фінансових цілей і рівня доходу.

Дуже цікавою є стратегія «Плати собі спочатку» (англ. Pay yourself first). Ключова ідея полягає в зміні стандартного сценарію: спочатку відкласти певну частину доходу на заощадження або інвестиції, а решту витрачати на життя. Це дозволяє зробити накопичення автоматичними, а не другорядними. Наприклад, можна налаштувати автоматичний переказ певного відсотка зарплати на окремий рахунок, щоб не витратити ці кошти на спонтанні покупки [6].

Останнім розглянемо пріоритетне бюджетування, яке полягає у розподілі коштів за важливістю. Спершу людина визначає, на що слід витратити гроші першочергово – тобто які категорії витрат необхідно покрити в першу чергу. Наприклад, це можуть бути рахунки за квартиру, транспорт або навчання. Лише після цього залишок доходу можна розподілити на менш пріоритетні потреби, такі як розваги чи покупки. Даний підхід допомагає раціонально розподілити ресурси – зрозуміти, що важливо, а що можна відкласти або скоротити.

Фінансові додатки успішно інтегрують зазначені стратегії: наприклад, популярний застосунок YNAB базується на принципах zero-based budgeting, а GoodBudget реалізує класичний метод «конвертів». Тому правильний вибір стратегії є важливим для розробки застосунку для планування бюджету, що й розглядається в межах кваліфікаційної роботи.

Та чи справді застосунки для особистого бюджетування, які реалізують наведені стратегії можуть покращити фінансову грамотність? Чи є цьому наукові підтвердження? Відповідь на це запитання дають результати декількох досліджень. Учасниками першого дослідження [7] стали 390 молодих студентів. Вони отримали доступ до додатку FinDiary, який дозволяв записувати та відстежувати транзакції (витрати та доходи). Моніторинг проводився протягом шести місяців, під час якого учасники отримували сповіщення про необхідність заощаджень та нагадування про введення даних. Основні результати та висновки дослідження:

а) середній бал фінансової грамотності учасників експериментальної групи

зріс на 0,08 стандартних відхилень, що свідчить про статистично значуще покращення;

б) покращення обізнаності щодо ринкових цін склало 0,34 стандартних відхилень, що вказує на значне підвищення рівня знань серед учасників;

в) не було виявлено статистично значущих змін у звичках щодо складання бюджету або ймовірності заощаджень серед учасників експериментальної групи;

г) понад 90 % користувачів відзначили – додаток допоміг краще розуміти свої фінансові ресурси та важливість заощаджень, а також визначати, на що витрачати менше коштів;

ґ) 88 % оцінили його як більш корисний порівняно з іншими додатками на своїх смартфонах;

д) після закінчення дослідження 60 % учасників і надалі використовували застосунок;

е) стандартні відхилення показують невелику варіативність у відповідях учасників, тобто результати стабільні серед більшості користувачів.

Таким чином, дослідження однозначно підтверджує, що відстеження бюджету за допомогою мобільного додатку сприяє покращенню фінансової грамотності. Зважаючи на результат «в», можна відзначити, що якби додаток не лише відстежував витрати, а й надавав можливості для планування бюджету, це могло б ще більше підвищити фінансові навички користувачів.

У межах другого експерименту [8] чотири мобільні додатки – «Money Matters» – були надані членам кредитного союзу віком від 16 до 65 років. До складу пакету входили: додаток для порівняння відсоткових ставок кредитів, додаток для порівняння витрат, календар готівки та додаток для управління боргом. Моніторинг проводився у форматі рандомізованого контрольованого експерименту (RCT). Основні результати та висновки дослідження:

– учасники експериментальної групи значно покращили свої знання, базові навички та мотивацію щодо фінансового управління;

– це призвело до поліпшення фінансової поведінки: частіше відстежували доходи та витрати, а також демонстрували більшу стійкість перед фінансовими

шоками;

– оцінка змін у фінансовому благополуччі показала позитивний вплив на загальну фінансову спроможність;

– автори відзначають, що ефект посилюється при персоналізації контенту та легкості його сприйняття.

Висновки обох експериментів доводять позитивний вплив застосунків для особистого бюджетування на фінансову грамотність. Зв'язок між такими додатками з правильною стратегією планування та підвищенням фінансової обізнаності користувачів очевидний.

Перейдемо до аналізу джерел, у яких досліджується функціональність фінансових мобільних додатків, зокрема тих, що спрямовані на управління особистими коштами. У науковій статті [9] автори проаналізували 1335 мобільних додатків на платформах Google Play та App Store, із яких після фільтрації за рейтингом і кількістю відгуків було відібрано 45 найпопулярніших застосунків для детального аналізу. Метою дослідження було оцінити, наскільки сучасні програми підтримують не лише фіксацію транзакцій, а й справжнє бюджетування відповідно до теорії ментального обліку. Оглянемо основні результати дослідження:

– усі 45 досліджених додатків підтримують відстеження транзакцій, проте приблизно третина з них не мають функцій повноцінного бюджетування, таких як розподілення коштів за категоріями чи конвертами;

– лише 26 із 45 застосунків реалізують механізм мультибюджетування – можливість створювати кілька окремих бюджетів для різних типів витрат (продукти, оренда, подорожі тощо);

– у програмах, в яких наявне бюджетування, користувач має самостійно розподіляти кошти між категоріями, автоматичного розподілу бюджету системою не передбачено;

– спостерігається плутанина у термінах: поняття «рахунок» і «транзакція» використовуються неоднозначно, що може ускладнювати сприйняття для користувачів;

– лише поодинокі застосунки пропонують додаткові функції, такі як автоматичне імпортування банківських транзакцій або порівняння витрат із доходами.

Дане дослідження підкреслює актуальність впровадження методів оптимізації у процес бюджетного планування. Більшість наявних мобільних додатків орієнтовані переважно на відстеження витрат, тоді як ті, що підтримують функції бюджетування, здебільшого передбачають ручний розподіл коштів користувачем між рахунками або категоріями. Водночас автоматичне бюджетування – тобто формування та пропонування індивідуальних планів відповідно до цілей, рівня доходу та середніх витрат за категоріями – у таких системах практично не реалізоване. Отже можна сміливо створювати нові рішення.

Дослідники попередньо оглянутої праці проаналізували функціональні можливості застосунків для персонального бюджетування з оглядової точки зору, узагальнивши їх ключові переваги, недоліки та статистичні дані. Для глибшого аналізу аналогів доречно детальніше розглянути окремі популярні рішення. Це є корисним для створенні власного застосунку, оскільки дозволяє визначити, які функції варто реалізувати, які підходи можна успішно адаптувати, а які – є малоефективними.

Одним із найвідоміших застосунків цього типу є You Need A Budget (YNAB), що вирізняється ретельно продуманим процесом адаптації користувача. Після встановлення програма проводить коротке опитування, формуючи стартовий набір категорій витрат на основі фінансових звичок. Основою YNAB є стратегія zero-based budgeting (описана вище), яка передбачає повний розподіл усього доступного доходу між категоріями витрат і заощаджень, не залишаючи невикористаних коштів. Розподіл може виконуватись як вручну, так і автоматично відповідно до заданих пріоритетів.

Серед корисних функцій YNAB варто виділити встановлення фінансових цілей, трекінг прогресу, а також великий обсяг навчальних матеріалів – статей, підкастів та інтерактивних порад. Недоліком є те, що YNAB є повністю платним

сервісом. Важливо зазначити, що застосунок має як мобільну, так і десктопну версію. На рисунку 1.1 наведено приклад інтерфейсу сторінки бюджету. Видно, що витрати структуровані не лише за категоріями (продукти, оренда тощо), але й за тематичними групами – обов'язкові платежі, погашення кредиту, розваги та інші. Це робить інтерфейс більш логічним і зрозумілим, дозволяючи користувачу гнучко адаптувати структуру під власні потреби. Також доступний місячний трекінг витрат, статистика та дані за попередні періоди. Інтерфейс YNAB можна вважати одним з найбільш продуманих серед існуючих аналогів.

Рисунок 1.1 – Сторінка «Бюджет» у YNAB [10]

Другим розглянемо GoodBudget, який реалізує класичний метод «конвертів» – розподіл доходу по категоріях, які виступають у ролі окремих віртуальних гаманців. Після внесення доходу кошти автоматично розподіляються між конвертами, однак не за алгоритмом розподілу, а просто відповідно суми яку, користувач виділив а певну категорію. Такий підхід стимулює користувачів планувати витрати наперед. Проте додаток не має інтеграції з банками, що ускладнює процес обліку, адже всі транзакції вводяться вручну. З цікавих функцій наявне спільне використання бюджету (в парі, родині).

Рисунок 1.2 – Основні сторінки GoodBudget [11]

MoneyFy – найпростіший серед розглянутих застосунків, орієнтований лише на ведення базового обліку фінансів. Усі основні функції зібрані на одному екрані. На головній сторінці користувач одразу бачить витрати, прибуток за місяць і секторну діаграму розподілу. Однак додаток не має функцій планування, аналітики чи лімітів по категоріях, що обмежує його використання як повноцінного інструменту бюджетування.

Spendee вирізняється сучасним дизайном і розвиненою системою аналітики. Застосунок підтримує синхронізацію з банками, зокрема українськими (Monobank, Ощадбанк), а також із криптогаманцями. У ньому передбачено велику кількість категорій, тегів та зручних графіків для аналізу витрат. Однак функціонал планування обмежується лише створенням кількох бюджетів із заданими категоріями, наприклад: бюджет 5000 грн на місяць для категорій «одяг та взуття» та «краса і здоров'я». Такий підхід дозволяє користувачу самостійно розподіляти кошти й контролювати перевищення встановлених меж.

Останній – додаток Wallet (BudgetBacker) є найбільш функціонально наповненим серед проаналізованих. Він підтримує синхронізацію з великою кількістю банків, зокрема українських, та має розширену систему категорій.

Застосунок пропонує широкий набір графічних звітів і може використовуватися не лише для особистого обліку, а й для малого бізнесу чи інвестицій. Важливо, що більшість функцій доступна безкоштовно. Функціонал планування включає внесення запланованих платежів і ручного створення бюджетів на певний період (місяць, рік) з прив'язкою потрібних категорій та рахунків. Додаток також надсилає сповіщення про перевищення бюджету або досягнення 75 % від встановленої межі.

Проведений аналіз інтерфейсів, функціоналу та особливостей аналогів допоміг сформувати чітке бачення структури та ключових функцій майбутнього застосунку. Він також підтвердив висновки наведеного раніше дослідження: сучасні рішення здебільшого зосереджені на обліку доходів і витрат, а реалізовані інструменти бюджетування зазвичай не включають автоматичного розподілу коштів та формування оптимізованих планів для користувача. Отже, основний акцент більшості фінансових застосунків на ринку спрямований на контроль бюджету, а не на його повноцінне планування.

Для реалізації функціоналу планування бюджету, зокрема створення оптимальних планів його розподілу, доцільним є застосування методів математичної оптимізації. Постає питання наукового підґрунтя такого підходу: чи розглядалося використання оптимізаційних методів у бюджетному плануванні в наукових дослідженнях?

Аналіз джерел показав, що такі підходи використовуються у різних сферах, де потрібно розподіляти обмежені ресурси. Хоч роботи стосуються інших галузей, вони демонструють, що методи оптимізації універсальні і можуть ефективно застосовуватися й для планування особистих фінансів.

У статті [12] представлено модель цільового програмування для розподілу бюджету в IT-організації. Дослідники працювали з корпоративним бюджетом, де є багато конкуруючих цілей та обмежень. Модель дозволяє знаходити оптимальний варіант розподілу коштів між підрозділами так, щоб одночасно задовольнити кілька критеріїв ефективності. Автори показали, що використання оптимізаційних методів може значно підвищити обґрунтованість фінансових

рішень і дозволяє приймати їх більш системно, а не інтуїтивно. Хоча робота присвячена IT-сфері, сам підхід легко адаптується для бюджетів будь-якого типу.

У дослідженні О. Артюшенка [13] вже розглядається більш прикладна сфера – підвищення ефективності бюджетних видатків у військовому фінансуванні. Автор застосовує методи лінійного програмування та оптимізації для того, щоб знайти найкращі варіанти розподілу коштів між різними напрямами забезпечення військ. Робота демонструє, що оптимізаційні моделі добре працюють навіть у складних системах з великою кількістю обмежень, де важливо забезпечити максимальний ефект від кожної витраченої одиниці ресурсу. Це ще раз підтверджує, що математична оптимізація є дієвим інструментом там, де потрібно правильно розставити фінансові пріоритети.

Третє джерело – магістерська робота [14] де розроблено модель оптимального розподілу рекламного бюджету кафедри. На відміну від попередніх досліджень, ця робота ближча до тематики персонального планування: йдеться про невеликий бюджет, який потрібно розподілити так, щоб отримати максимальний результат. Автор показує на практиці, як оптимізаційні алгоритми дозволяють обрати найкращі варіанти витрат з урахуванням обмежень та встановлених цілей. Такий підхід дуже схожий на логіку планування особистого бюджету, коли користувач має певні фінансові цілі та обмежену суму коштів.

Таким чином, аналіз наукових джерел показує, що методи лінійного та цільового програмування вже успішно застосовуються у сфері бюджетування – як на рівні великих організацій, так і у моделях, близьких за масштабом до персональних фінансів. Це доводить, що використання математичної оптимізації у застосунку для автоматичного формування планів та розподілу бюджету має цілком обґрунтовану наукову основу.

1.2 Огляд і аналіз методів та засобів розробки Android-додатку для управління особистими фінансами для вирішення проблеми дослідження

Для розробки алгоритму планування та розподілу фінансових ресурсів (бюджету) важливо обрати метод математичної оптимізації. Це дозволить побудувати точну модель та знайти її оптимальне рішення. Чому ж розглядаються саме методи оптимізації? Однією з ключових задач оптимізації є задача математичного програмування, яка полягає у пошуку екстремуму (мінімуму або максимуму) цільової функції за умови дотримання певних обмежень [15]. Задачу розподілу бюджету можна також формалізувати як задачу максимізації або мінімізації цільової функції, де обмеження визначають межі витрат у категоріях та наявний обсяг фінансових ресурсів. Тобто вона за своєю суттю і є класичною задачею математичного програмування. Обмеженнями задачі є лінійні нерівності (наприклад, сума витрат не має перевищувати дохід користувача), а от структура цільова функція буде залежати від обраного методу оптимізації. Розглянемо три поширені методи, які можна використати, їх переваги та недоліки та чим вони можуть бути корисними в контексті задачі оптимізації бюджету.

Лінійне програмування (англ. Linear Programming, LP) використовується для оптимізації лінійної цільової функції при обмеженнях, що мають лінійну форму. Воно дозволяє знаходити максимум або мінімум функції, коли всі взаємозв'язки між змінними пропорційні. Цей метод ефективний у задачах, де важлива точність та швидкість розрахунків при чітко заданих лінійних обмеженнях [15].

Перевагами LP є:

– гарантоване знаходження глобального оптимуму у межах моделі;

– висока обчислювальна ефективність;

– просте моделювання та формулювання задач;

– широка підтримка у вигляді готових алгоритмів та бібліотек.

До недоліків можна віднести:

– неможливість повністю врахувати нелінійні взаємозв'язки між змінними (наприклад, збільшення витрат у одній категорії призводить до нелінійного впливу на іншу категорію);

– не підходить для задач із «м'якими» обмеженнями (коли обмеження можна порушувати) або адаптивними сценаріями.

У задачі персонального бюджету QP ідеальне для швидкого, жорсткого планування витрат, де є фіксовані категорії та суворі ліміти (наприклад, витрати на оренду не можуть перевищувати 10000).

Квадратичне програмування (англ. Quadratic Programming, QP) розширює підхід лінійного програмування, дозволяючи оптимізувати цільову функцію, яка містить квадратичні члени, при збереженні лінійних обмежень. Це дає змогу враховувати взаємозв'язки між змінними та мінімізувати коливання або дисбаланс у системі. QP застосовується у задачах, де важлива не лише величина результату, а й його стабільність або оптимальне співвідношення між змінними. Важливим аспектом є те, що гарантія знаходження найкращого рішення (глобального оптимуму) залежить від опуклості цільової функції. Якщо функція опукла (має форму чаші), то задача QP є задачею опуклого програмування. У цьому випадку гарантується знаходження єдиного і найкращого рішення. Якщо ж функція не опукла, QP може мати кілька локальних мінімумів («ямок»), у які алгоритм може провалитися, не знайшовши при цьому справжнього глобального оптимуму [16].

Переваги QP:

– можливість враховувати взаємозв'язки між змінними;

– оптимізація дисбалансів і коливань у системі;

– дозволяє моделювати дисперсію (визначає, на скільки значення змінної відрізняється від середнього або очікуваного значення).

Недоліки:

– вища обчислювальна складність порівняно з LP;

– складніше налаштовувати модель і потрібно більше даних для точного моделювання;

– гарантія глобального оптимуму залежить від опуклості цільової функції.

У контексті фінансів, метод є корисним для портфельного підходу до бюджету (наприклад, інвестицій) або коли потрібно мінімізувати відхилення фактичних витрат від певних рекомендованих чи ідеальних значень у категоріях.

Цільове програмування (Goal Programming, GP) дозволяє одночасно враховувати декілька (множину) цільових функцій і пріоритетів. На відміну від класичного лінійного програмування, воно вводить поняття відхилень від бажаних цілей та можливість встановлювати їхню важливість. Об'єктом оптимізації в GP є не прямий екстремум однієї функції, а мінімізація суми небажаних відхилень від кожної цілі, зважена відповідно до їхніх пріоритетів. Це дозволяє знайти найкращий компроміс, який максимально задовольняє цілі найвищого рангу. GP застосовується у багатокритеріальних задачах, де необхідно знайти компромісне рішення між кількома суперечливими цілями [17]. Наведемо переваги GP:

– дозволяє працювати з кількома цілями одночасно;

– враховує пріоритети та важливість кожної цілі;

– дає змогу знайти компромісне рішення у складних задачах.

Недоліки:

– складність формулювання моделі та налаштування пріоритетів;

– більша обчислювана складність ніж LP, QP;

– результатом є компроміс (задовільне рішення), а не математичний оптимум.

Такий підхід незамінний, коли потрібно збалансувати конфліктуючі цілі, наприклад: не перевищити загальний ліміт; мінімізувати витрати на розваги, досягти цілі заощаджень (мінімум 20 % доходу).

Вибір оптимального методу (LP, QP або GP) – це перший етап. Наступний крок – практичне розв'язання сформованої математичної моделі. Для цього необхідні спеціалізовані бібліотеки, розроблені спеціально для вирішення задач оптимізації. Хоч для розробки основного функціоналу Android-додатку буде використовуватися мова Kotlin, вона не має підтримки подібних засобів. У сфері

математичного програмування мова Python має безперечну перевагу, вона пропонує найпотужніші бібліотеки для чисельних обчислень. Тож, в будь якому випадку розв'язання задачі необхідно реалізовувати мовою Python та інтегрувати у застосунок.

Вибір конкретної бібліотеки для реалізації буде залежати від класу обраної математичної моделі. Для вирішення задачі LP використовується класичний та найпоширеніший алгоритм – симплекс-метод. Це ітеративний алгоритм, розроблений для знаходження оптимального розв'язку в LP-задачах. Його принцип полягає у систематичному переміщенні по кутових точках (вершинах) області допустимих рішень. Ця область, визначена лінійними обмеженнями, завжди є опуклим багатогранником. На кожному кроці алгоритм перевіряє суміжну кутову точку. Якщо значення цільової функції у новій точці покращується (наприклад, зростає при максимізації), алгоритм переходить до неї. Цей процес триває, доки не буде знайдена точка, в якій подальше переміщення по кутових точках не дає покращення. Основні Python-бібліотеки, які реалізують симплекс-метод це Scipy та Pulp.

Scipy – це наукова бібліотека Python, яка містить модулі для математики, статистики, та головне для оптимізації. Функція linprog, яка входить до пакету scipy.optimize є інструментом для розв'язання задач LP та містить симплекс-метод як один із своїх внутрішніх алгоритмів. Перевага використання SciPy полягає в її інтеграції з іншими науковими інструментами Python (наприклад NumPy) та надійності її реалізації.

Pulp – це бібліотека для моделювання LP-задач. Вона дозволяє описувати задачу оптимізації у більш інтуїтивно зрозумілій формі (наближеній до математичної нотації), а не лише через матриці та вектори. PuLP не містить власного симплекс-методу, але вона виступає інтерфейсом, який автоматично викликає і керує зовнішніми солверами (вирішувачами), як-от GLPK (GNU Linear Programming Kit).

Задачі QP зазвичай вирішуються методами внутрішньої точки. На відміну від симплекс-методу (який рухається по межах області), метод внутрішньої

точки починає рух із точки всередині області допустимих рішень, рухається до оптимуму, залишаючись всередині цієї області. Метод характеризується швидкою збіжністю (збільшення кількості змінних не призводить до експоненційного зростання часу вирішення). Для QP-задач використовуються бібліотеки, які спеціалізуються на опуклому програмуванні: CVXPY, CVXOPT.

CVXPY – це основний, високорівневий інтерфейс для моделювання QP. Він дозволяє формулювати задачу у зручній формі та автоматично перевіряє модель на опуклість. CVXPY не містить власних алгоритмів, але виступає інтерфейсом, який автоматично обирає і викликає спеціалізовані солвери (OSQP, ECOS або CVXOPT), які безпосередньо реалізують методи внутрішньої точки.

CVXOPT – це бібліотека, яка містить власні реалізації солверів для загального опуклого та квадратичного програмування. Вона може використовуватися як безпосередньо, так і як один із внутрішніх солверів, що підключаються через CVXPY.

Задачі GP не вирішуються одним алгоритмом безпосередньо. Вони вимагають трансформації у послідовність стандартних LP-задач. Для цього використовується ієрархічний симплекс-метод. Цей алгоритм послідовно застосовує симплекс-метод: він мінімізує відхилення цілі найвищого пріоритету, а потім переходить до другого, і т.д., щоразу обов'язково не погіршуючи результати, досягнуті для попередніх, більш важливих цілей. Таку ієрархічну логіку можна реалізувати за допомогою бібліотек PuLP або Pyomo.

Оскільки модуль оптимізації можливо реалізувати лише мовою Python, постає питання його інтеграції у мобільний додаток. Існує два можливих підходи: пряма інтеграція та архітектура мікросервісу.

Для безпосереднього вбудовування Python-коду у мобільний додаток на Android існує бібліотека Chaquopy. Це плагін для Android Gradle, який дозволяє запускати Python-код та сторонні Python-бібліотеки безпосередньо в середовищі Java/Kotlin. Вона виконує компіляцію Python-пакетів та їхнє пакування в APK-файл, дозволяючи викликати функції з Python із коду Kotlin або Java. Chaquopy інтегрує багато стандартних наукових бібліотек, включаючи NumPy та SciPy. Це

означає, що, наприклад, моделі Лінійного Програмування (LP), розв'язані за допомогою scipy.optimize.linprog, можуть бути безпосередньо вбудовані в додаток. Однак, пряма інтеграція складних обчислювальних пакетів, як-от CVXPY, неможлива або дуже складна. Це пов'язано з тим, що CVXPY залежить від зовнішніх, нативних розв'язувачів, які написані на мовах C/C++/Fortran, а Chaquopy не підтримує їхню інтеграцію. Тому для методів, що вимагають QP, необхідно застосовувати архітектурно інший підхід.

Якщо пряма інтеграція неможлива, рішенням є винесення розрахункового блоку на віддалений сервер. Це передбачає створення архітектури «клієнт-сервер». Серверним компонентом (backend) виступає сервер на пайтон (н-д Flask, FastAPI). На ньому встановлюються всі необхідні бібліотеки (CVXPY, NumPy, Solvers), і тут виконується функція оптимізації. Мобільний додаток виступає клієнтом та надсилає запити з вхідними даними (дохід, обмеження, пріоритети) на сервер. Це можна реалізувати з допомогою бібліотеки для мережевих запитів (API), наприклад Retrofit – вона спрощує виконання мережевих запитів та автоматично перетворює відповіді (наприклад, у форматі JSON) на об'єкти Kotlin. Для розміщення сервера потрібна надійна платформа (хостинг), наприклад: PythonAnywhere, Heroku, AWS або Google Cloud Platform.

1.3 Постановка завдання на кваліфікаційну роботу магістра

На основі проведеного аналізу сформовано цілісне розуміння проблеми, можливих шляхів її вирішення та відповідних технічних засобів. Отримані результати дали змогу визначити основні цілі, яких необхідно досягти в межах виконання кваліфікаційної роботи магістра. Завдання включає такі пункти:

– здійснити аналіз предметної області, результатів існуючих досліджень;

– провести огляд популярних мобільних застосунків для персонального бюджетування та їх функціональних можливостей;

– визначити основні вимоги до функціоналу, інтерфейсу та архітектури розроблюваного додатку для управління особистими фінансами;

– обрати метод оптимізації, який найкраще підходить для вирішення задачі оптимального розподілу бюджету, обґрунтувати його використання та побудувати математичну модель;

– здійснити вибір та обґрунтувати використання технологій, інструментів і засобів для розробки застосунку;

– розробити мобільний додаток для операційної системи Android, який реалізує функціонал планування, обліку фінансів та статистику;

– реалізувати модуль оптимізаційного розподілу бюджету та інтегрувати його до Android-застосунок;

– створити інтуїтивний та зручний інтерфейс;

– провести тестове експериментальне дослідження розробленого підходу, протестувати роботу додатку в різних сценаріях.

Отже, проблема управління особистими фінансами є актуальною, як на рівні користувачів, так і на рівні цифрових сервісів. Аналіз існуючих продуктів допоміг сформулювати картину функціоналу майбутнього застосунку, та виявити чого не вистачає – підтримки функціоналу планування бюджету.

Для інтеграції методів оптимізації є достатнє наукове підґрунтя. Вони вже досліджувались для використання у фінансовій сфері та демонструють успішні результати у схожих задачах. А реалізація такого застосунку є технічно можливою: доступні інструменти, API-сервіси та фреймворки забезпечують умови для створення функціоналу як обліку, так і автоматизованого планування бюджету.

РОЗДІЛ 2
ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ ТА ПРАКТИЧНА РЕАЛІЗАЦІЯ ANDROID-ДОДАТКУ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

2.1 Обґрунтування вибору шляхів, технологій, алгоритмів і засобів вирішення поставленого завдання

Розподіл особистого бюджету між різними категоріями витрат є складним завданням через необхідність врахування одночасно кількох факторів. Щоб забезпечити оптимальне використання коштів, доцільно застосовувати методи математичної оптимізації. Вони дозволяють формалізувати правила та обмеження, які природно виникають при плануванні витрат, і знайти такий розподіл, який максимально відповідає встановленим пріоритетам користувача.

Спираючись на теоретичну базу методів оптимізації, розглянутих раніше (у підрозділі 1.2), наступним кроком є формалізація задачі, вибір оптимального методу та побудова математичної моделі для алгоритму розподілу бюджету.

Задача формулюється як необхідність знайти оптимальний та збалансований розподіл коштів між категоріями витрат, одночасно враховуючи загальний доступний бюджет, встановлені верхні та нижні межі витрат для кожної категорії, а також їхню важливість (пріоритети). Вхідними даними для моделі слугують доступна сума бюджету, зазначені межі витрат у категоріях та пріоритети категорій, тоді як вихідними даними є безпосередньо розподіл бюджету по кожній категорії.

Спочатку для моделювання розглядався метод лінійного програмування, але найкращим вибором для цієї задачі є метод квадратичного програмування. Чому квадратичне програмування виявилося кращим, стає зрозуміло після аналізу роботи моделі лінійного програмування.

Модель лінійного програмування формується наступним чином (формула 2.1), де визначається цільова функція, пріоритети та межі для кожної категорії витрат:

$$min_x \sum_{i=1}^{n} (-w_i) x_i,$$

$$\text{за умов } \sum_i x \leq B, L_i \leq x_i \leq U_i \quad (2.1)$$

де x_i – обсяг витрат у категорії і;

w_i – ваговий коефіцієнт (пріоритет категорії);

B – доступний бюджет;

L_i, U_i – нижні та верхні межі категорії відповідно.

Розв'язання моделі лінійного програмування, наприклад, із використанням функції optimize.linprog бібліотеки SciPy, дає розподіл витрат x_i, де категорії з найвищим пріоритетом w_i отримують максимально можливе фінансування, за умови дотримання загального бюджету та встановлених для них меж.

Важливий момент тут – перетворення задачі на мінімізацію. Оскільки більшість стандартних алгоритмів оптимізації, як-от у SciPy, за замовчуванням шукають мінімум цільової функції, потрібно використовувати від'ємні коефіцієнти w_i для моделювання максимізації пріоритетів. Чим вищий пріоритет, тим більшим за абсолютною величиною, але від'ємним, є коефіцієнт w_i. Таким чином, мінімізація цієї від'ємної суми фактично призводить до максимізації «корисності» витрат, спрямованих у пріоритетні категорії. У цьому контексті цільова функція виконує роль показника «якості життя» користувача, відображаючи ступінь його задоволення розподілом коштів. Цей підхід направляє основну частину бюджету туди, де він найпотрібніший, одночасно гарантуючи, що всі інші категорії отримають хоча б базове фінансування, визначене їхніми нижніми межами L_i.

Недолік цього підходу полягає в тому, що розподіл є надто агресивним: майже весь бюджет виділяється на 1-2 найважливіші категорії, інші ж ігноруються, навіть якщо їх верхні межі дозволяють вкласти більше. Це виглядає досидь нелогічно, адже не враховується те, що будь-який користувач потребує хоч якихось коштів для всіх основних категорій.

Для вирішення цієї проблеми було обрано підхід квадратичного

програмування з використанням бібліотеки CVXPY. Цей метод дозволяє формулювати цільову функцію у вигляді комбінації лінійної та квадратичної частин. Цільова функція – гнучка, вона враховує як пріоритети, так і бажаний «плавний» розподіл бюджету між категоріями. Це та ж сама функція «якості життя» однак більш збалансована (формула 2.2) :

$$min_x(\beta \sum_{i=1}^{n}(-w_i)x_i + \alpha \sum_{i=1}^{n}(x_i - t_i)^2),$$
$$\text{за умов } \sum_i x \leq B,\ L_i \leq x_i \leq U_i \qquad (2.2)$$

де x_i – обсяг витрат у категорії I;

w_i – ваговий коефіцієнт (пріоритет категорії);

t_i – бажані витрати (н-д: середнє між мінімумом та максимумом);

α – коефіцієнт, що регулює ступінь «м'якості» обмежень;

β – коефіцієнт ваги для пріоритетів.

Лінійна частина тягне витрати в бік найважливіших категорій – як і у прикладі наведеному вище.

Квадратична частина штрафує за занадто великі відхилення від бажаних витрат – тобто, не дозволяє «занулити» інші категорії. Якщо витрати x_i сильно відрізняються від бажаного рівня t_i, то значення $(x_i - t_i)^2$ зростає квадратично. Це збільшує значення цільової функції, і оптимізатор уникає таких рішень, де одна категорія отримує надто багато або надто мало. Таким чином, квадратична частина діє як згладжувач: вона стримує надто радикальні розподіли, де деякі категорії отримують усе, а інші – нічого. Це забезпечує більш справедливий та стабільний розподіл бюджету між усіма категоріями.

Уявімо, що користувач має обмежений бюджет і 10 категорій витрат. При використанні лише пріоритетів (LP), система може витратити все на 1-2 категорії й решту залишити з нульовим фінансуванням. Натомість у моделі з QP, враховується ще й бажаний баланс (наприклад, середні межі кожної категорії). У результаті, витрати розподіляються з урахуванням пріоритетів, але м'яко, без

різких перекосів.

Коефіцієнти α (вага штрафу за відхилення від бажаного розподілу) та β (вага пріоритетів) дозволяють тонко налаштовувати поведінку оптимізатора:

– якщо α≫β, алгоритм надає перевагу вирівнюванню витрат до бажаних значень t_i, навіть якщо деякі категорії мають низький пріоритет. Такий підхід зменшує вагу пріоритетів і забезпечує більш рівномірний розподіл;

– якщо β≫α, система фокусується на пріоритетах, подібно до лінійного програмування, однак квадратичний штраф не дає повністю «обнулити» інші категорії, що запобігає надмірній концентрації бюджету.

У ході тестування найкращий баланс було досягнуто при α = 0.003 та β = 13. Параметри забезпечили розумний компроміс: бюджет було спрямовано згідно з пріоритетами, але водночас жодна категорія не була повністю проігнорована – кожна отримала принаймні мінімум поверх нижньої межі.

Отже, задача та модель сформована, а для розв'язання задачі буде використовуватися Python-бібліотека CVXPY, яка описана у підрозділі 1.2. У подальшому буде продемонстровано, яким чином ця задача будується та розв'язується мовою Python за допомогою засобів CVXPY.

Так як обраний метод оптимізації – QP, можливості прямої інтеграції Python-коду з пакетами CVXPY через бібліотеку Chaquopy або інші засоби неможлива. Отже, єдине рішення для впровадження розрахункового блоку в застосунок – винесення та сервер.

Для серверної частини застосунку було обрано хостинг PythonAnywhere – це інтегроване онлайн-середовище розробки та сервіс веб-хостингу, засноване на програмуванні Python. Вибір засобу зумовлений багатьма перевагами. У платформі є безкоштовний тариф, який має обмежену, але достатню кількість щоденних запитів для навчального проекту. Очевидно, надається повна підтримка Python-середовища, що є особливо важливим оскільки оптимізаційний модуль реалізований з використанням складних пакетів CVXPY. Середовище підтримує WSGI-додатки, що дозволяє без проблем запускати Flask-сервер. І нарешті, середовище просте для використання, доступні важливі

інструментів, серед яких: консоль, перегляд логів доступу та помилок.

Для написання самого серверу, обрано мікрофреймворк на Python – Flask. Його роль – обробити запити, виконати розрахунки та повернути результат у вигляді відповіді API. Вибір засобу обґрунтований простотою реалізації, він не нав'язує структури або складних конфігурацій, що дозволяє зосередитись саме на логіці обробки запитів.

З боку Android-застосунку як клієнта надано перевагу бібліотеці Retrofit. Вона є стандартом де-факто для мережевої взаємодії в Android. Бібліотека дозволяє описувати запити через інтерфейси та автоматично перетворює відповіді сервера у Kotlin-моделі. Завдяки вбудованій підтримці корутин (асинхронних операцій) і роботі поверх OkHttp Retrofit забезпечує зручну обробку помилок та перехоплення запитів. Тому, взаємодія з Flask-сервером виконується надійно та з мінімальними зусиллями.

Перейдемо до інструментів, обраних для розробки Android-застосунку. Основним середовищем є Android Studio – офіційна IDE від Google, що забезпечує всі необхідні засоби для створення, налагодження та тестування мобільних додатків. Вона підтримує Kotlin та Java, інтегрується з Gradle, надає емулятори пристроїв і полегшує роботу з бібліотеками та хмарними сервісами. Основною мовою програмування обрано Kotlin, яка є сучаснішою та більш зручною для розробки порівняно з Java. Код, написаний нею, менш громіздкий, та й нові бібліотеки та інструменти Android в більшості підтримують Kotlin [18].

Розглянемо засоби, які вирішено використовувати в самому середовищі. Почнемо з Jetpack Compose – це сучасний декларативний фреймворк для створення UI на Android, який дозволяє описувати інтерфейс у вигляді функцій Kotlin і автоматично оновлювати його при зміні стану. Якщо казати простими словами: раніше кожен екран потрібно було малювати у окремому XML-файлі, а тепер з допомогою Compose інтерфейс створюється прямо в коді й одразу реагує на зміни даних. Основні Compose-бібліотеки, які використовуються:

– Navigation Compose – бібліотека навігацій, дозволяє переходити між Composable-екранами, управляти стеком навігацій, передавати аргументи;

– Material Compose – набір готових UI-компонентів: теми, кнопки, текстові поля та інше;

– Compose Runtime – бібліотека, що містить механізми управління станом (mutableStateOf, State, remember) та відповідає за рекомпозицію інтерфейсу;

– Animation APIs – засоби для створення анімацій і плавних переходів між станами.

Розглянемо й інші бібліотеки, зокрема Hilt – офіційна бібліотека від Android для ін'єкцій залежностей (англ. Dependency Injection, DI). DI – автоматичне надання потрібних залежностей в різні частини програми. Без нього довелося б самостійно створювати й передавати, наприклад, клієнт мережевих запитів у різні класи. Hilt бере це на себе: один раз визначається, як створюються об'єкти, а все інше він робить автоматично. У результаті застосунок простіше розширювати, легше тестувати й немає «ручного» перенесення залежностей між класами.

Фонова та асинхронна робота реалізована через Kotlin Coroutines. Вони дозволяють виконувати довгі операції – такі як мережеві запити чи взаємодія з базою даних без зависання інтерфейсу (UI-компонентів).

Для реалізації основного функціоналу використовуються не лише бібліотеки, а також і зовнішні сервіси. Firebase – хмарний сервіс, який надає низку інструментів для розробки веб- та мобільних застосунків. Наприклад, для авторизації використовується Firebase Authentication з допомогою Google акаунту. Вхід через Google дозволяє користувачу заходити у застосунок з різних пристроїв, забезпечуючи доступ до свого профілю. Сервіс автоматично керує токенами доступу, оновленням сесій та базовими механізмами захисту. Для зберігання даних використовується Firebase Realtime Database – хмарна нереляційна база даних, що зберігає дані у форматі єдиного дерева JSON. Це дозволяє відмовитися від локального зберігання даних (наприклад Room) та забезпечити їх доступність на будь-якому пристрої користувача. Оскільки відповіді API (наприклад, з банку або Flask-сервера) надходять у форматі JSON, структуру даних легко безпосередньо перетворювати у моделі Kotlin та зберігати

у базі. База забезпечує миттєву синхронізацію змін, а це важливо, адже оперуться динамічні фінансові дані, наприклад транзакції з банку.

Важливо зазначити що, у застосунку виникла потреба в інтеграції банківського сервісу. Вибір зупинився на Monobank, оскільки це єдиний банк, який надає простий персональний токен для доступу до свого API. Інтеграція з Monobank через Monobank Open API дозволяє отримувати дані про баланс та виписки транзакцій, якщо користувач надасть свій персональний токен, який він може згенерувати в особистому кабінеті Monobank. Це важливо для фінансового менеджменту, тому що дає змогу автоматично завантажувати історію витрат, не вимагаючи ручного введення. API надає лише доступ читання, що робить підхід безпечним. Monobank Open API зручний для навчальних чи персональних проектів, оскільки не вимагає складного погодження корпоративних прав доступу, на відміну від API інших банків для бізнес-провайдерів. Для корпоративних застосунків у цьому банку, наприклад, існує окреме API для провайдерів послуг [19].

Оскільки застосунок працює з конфіденційними фінансовими даними, постало завдання забезпечити їх захищене зберігання. Для цього використана власна кастомна реалізацію конвертного шифрування – криптографічного підходу, який працює на багаторівневій архітектурі, що поєднує декілька ключів, які обгортають один одного, створюючи «конверт» безпеки. Така схема дозволяє не лише безпечно зберігати ключі, а й гарантує, що навіть у випадку перехоплення даних доступ до них буде неможливим без всіх ключів. Реалізація виконана на основі стандартних інструментів Java Cryptography Extension (JCE), які забезпечують шифрування та захист чутливих даних, включно з генерацією та обробкою криптографічних ключів, а також безпечним зберіганням і передачею інформації.

Зрештою, проаналізувавши всі засоби та інструменти для розробки, сформуємо технологічний стек Android-додатку для управління особистими фінансами у таблиці 2.1. Визначимо також основні типи компонентів і їх призначення.

Таблиця 2.1 – Технологічний стек проекту

Назва	Тип	Призначення
Android Studio	IDE	Середовище розробки
Kotlin	Мова програмування	Основна мова програмування, на якій написано додаток
Python	Мова програмування	Для написання алгоритму оптимізації
PythonAnywhwere	Сервіс веб-хостигну	Для хостингу сервера
Flask	Python-фреймворк	Для реалізації сервера
CVXPY	Python бібліотека	Для розв'язання задачі квадратичного програмування у модулі оптимізації бюджету
Firebase Realtime Database	Нереляційна хмарна БД	Зберігання даних користувача в хмарі та синхронізація їх у реальному часі між пристроями
Firebase Authentication	Хмарний сервіс автентифікації	Забезпечує безпечну автентифікацію користувачів через Google
Jetpack Compose	UI-фреймворк	Для створення користувацького інтерфейсу на основі декларативного підходу
Hilt	Android-бібліотека	Для ін'єкцій залежностей
Retrofit	Android-бібліотека	Використовується для обміну даними з мережевими ресурсами (API)
Kotlin Coroutines	Kotlin-бібліотека	Для виконання асинхронних операцій
Monobank API	Банківське API	Надає доступ фінансових даних користувача
Java Cryptography Extension (JCE)	Вбудований крипто-фреймворк Java / Android SDK	Забезпечує шифрування та захист чутливих даних

Узагальнюючи викладене, можна сказати, що вибір конкретних засобів і технологій для розробки Android-застосунку базувався на потребах проекту щодо зручності розробки, інтеграції з серверною частиною, обробки фінансових даних та забезпечення безпеки.

2.2 Практична реалізація об'єкта проектування

Після аналізу предметної області, шляхів та методів вирішення проблеми, вибору інструментів та засобів настає найважливіший крок – реалізація. Першим етапом практичної реалізації, звичайно ж є визначення вимог до функціоналу. Це дозволяє структурувати роботу, визначити пріоритетні та другорядні функції.

Перш за все, варто подивитись на застосунок з боку користувача. В цьому

допоможе Use Case діаграма (рис. 2.1). Вона формується на етапі проектування і потрібна для того, щоб показати всі дії, які може виконувати користувач у застосунку (use case), як ці дії пов'язані між собою та як на них реагує система. Це допомагає сформувати вимоги до системи, продумати логіку роботи, перевірити чи система відповідає очікуванням користувача ще до початку розробки.

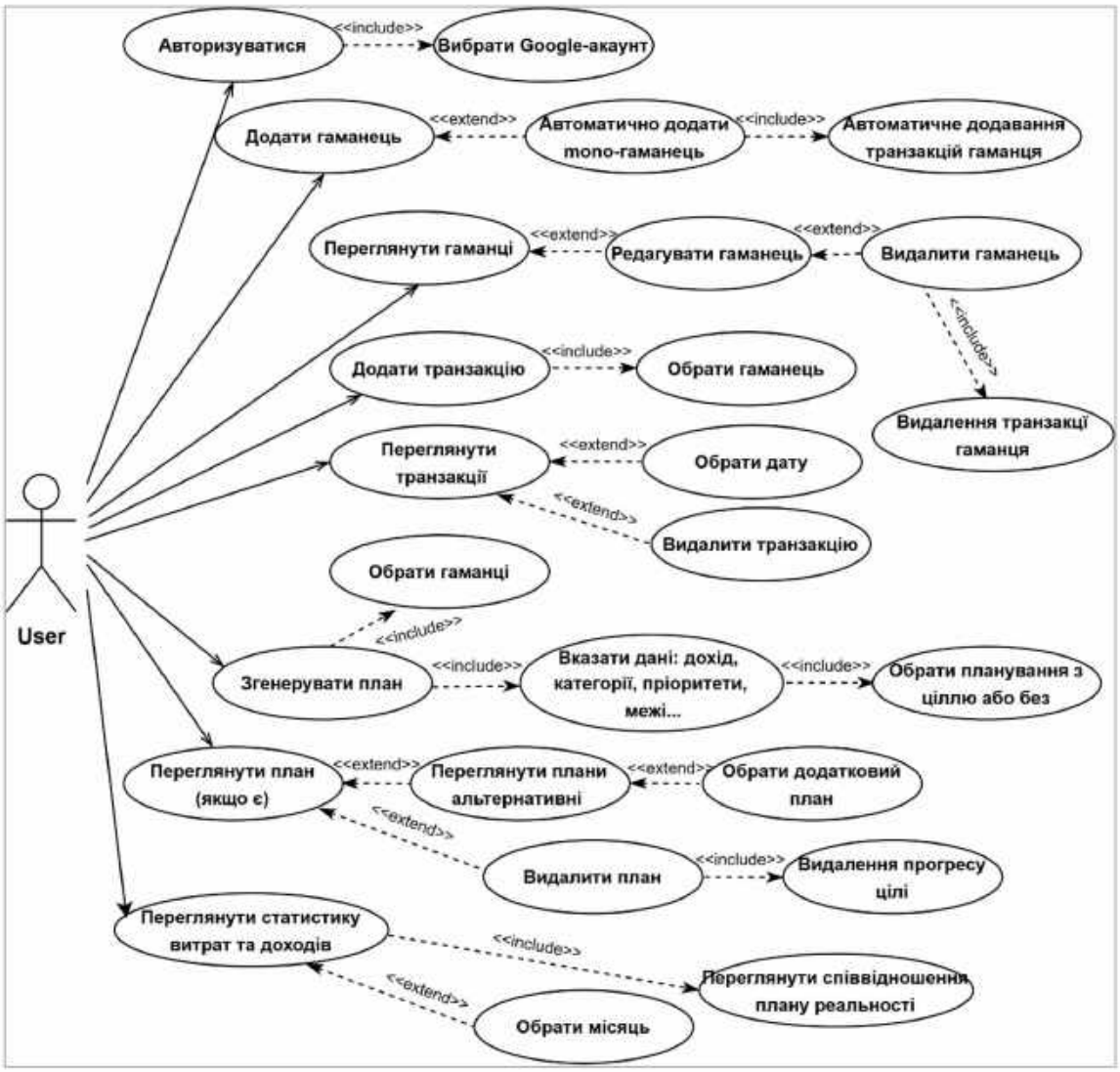

Рисунок 2.1 – Діаграма варіантів використання застосунку

Джерело: розроблено автором

Трохи про зв'язок між use case:

– «include» – обов'язкове включення – один use case завжди виконує інший;

– «extend» – додаткова опція, яка виконується не завжди, тобто один use case може розширювати інший у певних умовах.

На основі діаграми варіантів використання, можна деталізувати вимоги до функціоналу застосунку:

– вхід та авторизація через Google;

– інтеграція з банком для синхронізації транзакцій та гаманців;

– синхронізація транзакцій має відбуватися так, щоб дані оновлювалися автоматично та завжди були актуальними;

– можливість введення транзакцій (дохід, витрата, переказ) вручну;

– можливість ручного створення гаманця (рахунку: н-д: готівка, карта);

– перегляд та фільтрація транзакцій за датою;

– можливість редагувати, видалити гаманець з автоматичним видаленням прив'язаних транзакцій;

– можливість перегляду або видалення транзакції лише у випадку, якщо вона не синхронізована з банком;

– автоматичне планування місячного бюджету (варіанти: з ціллю та без) з пропозиціями альтернативних планів на різний термін;

– введення необхідних вхідних даних користувачем для створення плану;

– якщо планування з ціллю: на основі введених даних, знайти мінімальний можливий термін для накопичення для забезпечення перевірки;

– можливість видалити план, замінити на інший доступний або створити новий;

– перегляд відповідності плану до поточних витрат по категоріях;

– наведення статистики та графіків;

– можливість обирати термін при виведення статистики;

– забезпечення перевірок введених даних користувачем;

– якщо ціль досягнуто, термін плану ще не закінчився – користувач може, як змінити план, так і залишитись на поточному;

– якщо термін плану минув – план видаляється автоматично;

– якщо користувач обрав планування без цілі – план терміну не має.

Окрім основного функціоналу, були сформовані вимоги до безпеки, оскільки застосунок працює з чутливими даними користувача. Зокрема:

– токен для прив'язки до банку має бути надійно збереженим;

– необхідна можливість скидання токену в налаштуваннях із одночасним видаленням усіх даних (гаманців, транзакцій);

– всі чутливі дані (сума транзакції, баланс гаманця) повинні шифруватися перед збереженням у БД.

Основною функцією застосунку, тієї яка вирішує задачу роботи є планування бюджету – створення автоматизованого розподілу бюджету користувача з допомогою методу оптимізації. Для цього був розроблений алгоритм – блок-схема на рисунку 2.2. Опишемо деякі ключові деталі реалізації алгоритму.

На початковому етапі система отримує всі необхідні дані від користувача, які він вводить на спеціальних сторінках у застосунку:

– фінансова ціль та сума;

– щомісячний дохід та дохід поточного місяці;

– категорії витрат (обираються потрібні з всіх доступних);

– межі витрат категорій (нижня – мінімальна та верхня – бажана);

– фіксовані витрати (щомісячні та поточного місяця);

– пріоритети категорій;

– гаманці, які необхідно включити до планування.

З обраних гаманців зчитується та обраховується поточний баланс. Всі ці дані є вхідними для алгоритму. Після отримання всіх початкових параметрів система проводить перевірку їх коректності: чи заповнені всі необхідні поля, чи не суперечать значення одне одному, чи не перевищують межі витрат загальний доступний бюджет. Це дозволяє уникнути некоректних розрахунків. Після успішної перевірки дані передаються безпосередньо в алгоритм планування бюджету для обчислення оптимального розподілу.

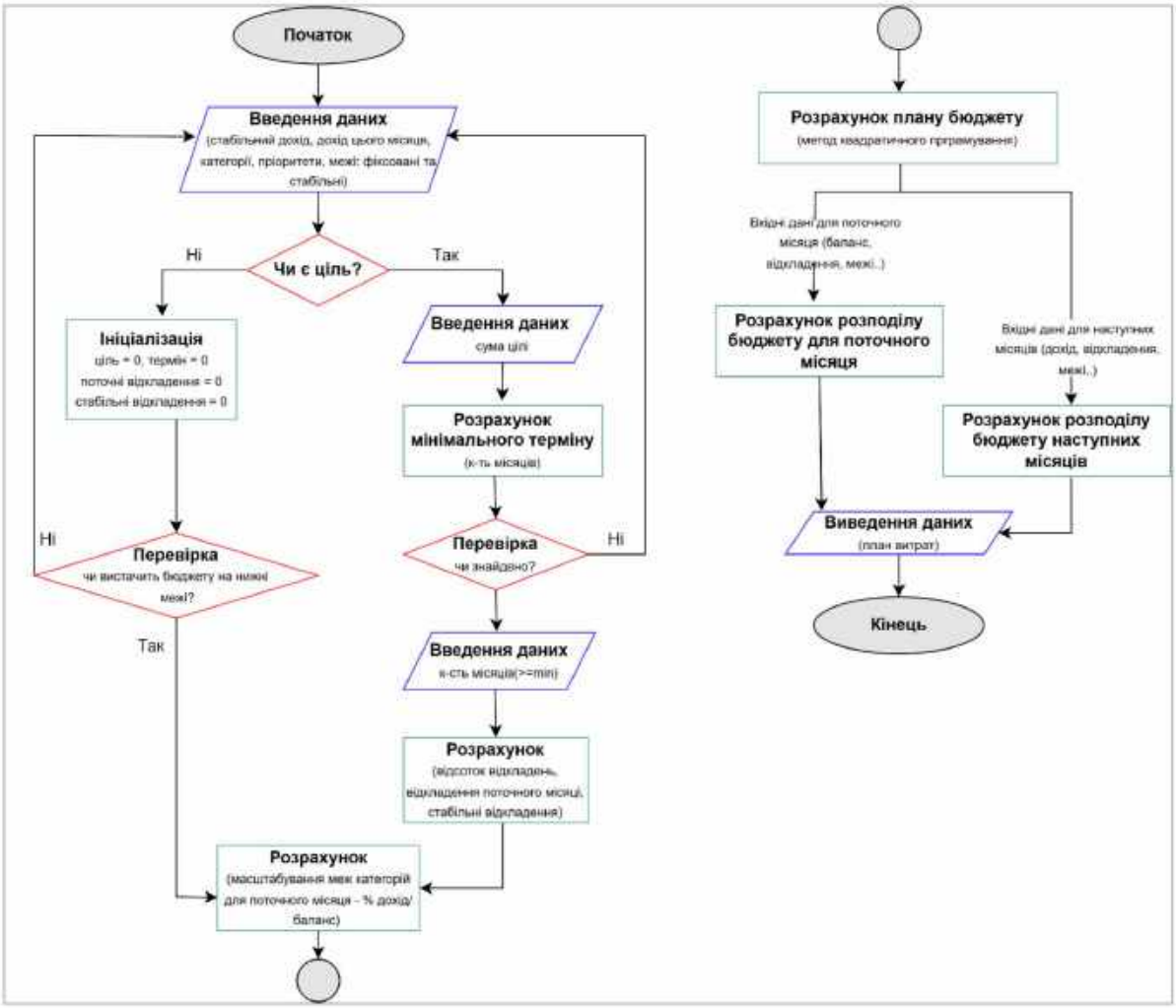

Рисунок 2.2 – Блок-схема алгоритму

Джерело: розроблено автором

Як бачимо з блок-схеми алгоритм містить декілька важливих нюансів, які необхідно роз’яснити.

По перше, користувачу надається можливість вибору одного з двох варіантів плану:

– планування з ціллю – бажанням користувача є спланувати бюджет, який допоможе накопичити на певну ціль за визначений період. Наприклад, накопичити на подорож 50000 гривень за 4 місяці;

– планування без цілі (без терміну) – бажанням користувача є допомога у складанні бюджету, раціональному розподіленню витрат без накопичень.

Якщо користувач задав ціль і суму, система додатково обчислює

мінімальний можливий термін її досягнення. Для цього аналізуються доступні ресурси (баланс + дохід), обов'язкові витрати та межі категорій. Якщо ціль може бути досягнута – користувачу наводиться мінімальний термін, менше нього він обрати не може. Якщо ж ціль не може бути досягнута, можна просто підправити дані: суму цілі, нижні межі категорій.

Наступним важливим аспектом є поділ плану на поточний період та майбутні місяці. Під час проектування алгоритму було вирішено працювати саме з помісячними інтервалами, оскільки такий підхід є звичним і зрозумілим для користувача: більшість регулярних доходів та витрат прив'язані саме до місяця (заробітна плата, комунальні платежі). Проте виникла практична проблема: користувач може розпочати планування в будь-який момент, наприклад 15 числа. Починати новий бюджетний період із середини місяця було б нелогічно та ускладнювало б розрахунки. Отже, було обрано таку стратегію: залишок місяця враховується як додаткові дні до плану, і розподіл на поточний місяць треба рахувати окремо. Таким чином користувач отримує коректний план вже для поточного місяця, а всі наступні місяці обчислюються як повні.

Якщо план формується із заощадженням на фінансову ціль, то спершу визначається універсальний відсоток накопичень, який має бути застосований однаково до балансу та майбутніх доходів. На основі цього відсотка обчислюються суми відкладень для поточного місяця та усіх наступних, після чого вони округлюються та коригуються таким чином, щоб загальний обсяг накопичень точно відповідав поставленій цілі.

Кінцевий етап – на основі встановлених пріоритетів та меж витрат по категоріях виконується оптимізація бюджету. Алгоритм спочатку покриває обов'язкові фіксовані витрати, після чого забезпечує мінімально необхідні значення для змінних категорій. Лише тоді залишок доступних коштів розподіляється відповідно до заданих пріоритетів та меж (методом квадратичного програмування). Якщо користувач визначив фінансову ціль, система додатково резервує необхідну суму накопичень, щоб гарантувати можливість її досягнення у визначений термін.

Вхідні дані для розрахунку поточного та сталого (щомісячного) планів відрізняються. Для постійного плану бюджетом виступає повний щомісячний дохід. Для поточного місяця бюджет складається з доступного балансу на гаманцях у поєднанні з доходом, який користувач ще отримає до кінця місяця. Через це межі витрат категорій масштабуються: вони пропорційно коригуються відповідно до того, наскільки бюджет поточного місяця відрізняється від стандартного місячного доходу. Це дозволяє адаптувати рекомендації під реальну фінансову ситуацію користувача на момент планування.

Наведемо окремі важливі аспекти реалізації алгоритму. CVXPY використовується як інструмент для побудови та розв'язання математичної моделі. Лістинг 2.1 – демонстрація функції, яка здійснює оптимізацію. Формування задачі починається зі створення вектора змінних x, який представляє шукані суми витрат у кожній категорії. Далі на основі вхідних даних генеруються параметри: масштабовані межі категорій, вектор пріоритетів та цільові значення, до яких має наближатися розподіл.

На наступному етапі за допомогою функцій CVXPY формується цільова функція, яка передається у cp.Minimize(). Після цього задається система обмежень. Вони визначають допустимі значення змінних: кожен елемент x має належати до свого інтервалу мінімум-максимум, а сума всіх витрат не повинна перевищувати доступний бюджет. Всі обмеження описуються у вигляді математичних нерівностей, які CVXPY збирає у внутрішню структуру задачі.

Після формування цільової функції та обмежень вони об'єднуються у об'єкт Problem. Виклик методу problem.solve() запускає процес оптимізації: CVXPY автоматично аналізує тип моделі, обирає відповідний солвер і передає йому задачу у стандартизованому форматі.

У результаті розв'язання бібліотека повертає:

– problem.status – статус задачі (чи знайдено оптимальне рішення);

– problem.value – значення цільової функції в оптимумі;

– x.value – оптимальний вектор витрат (бюджетний план).

Також solvers надають технічну інформацію: точність рішення, час роботи.

Лістинг 2.1 – Демонстрація використання CVXPY для розподілу бюджету

```
def calculate_plan(balance, income, fixed_expenses, savings, bounds,
priorities, alpha=0.01, beta=13):
    """
    Розрахунок оптимального плану витрат по категоріях за допомогою
квадратичного програмування.
    """
    # Масштабування меж під поточний баланс (для поточного місяця)
    scale = balance / income
    scaled_bounds = [(low * scale, high * scale) for (low, high) in bounds]

   # Формування target-значень (верхніх меж)
    n = len(scaled_bounds)
    targets = np.array([high for _, high in scaled_bounds])

    # Змінні оптимізації: x_i – витрати по категоріях
    x = cp.Variable(n)
    priorities = np.array(priorities)

    # Цільова функція: пріоритети + штраф за відхилення від target
    objective = cp.Minimize(
        beta * (priorities @ x) +
        alpha * cp.sum_squares(x - targets))

    # Обмеження: залишатися в межах категорій і бюджету
    constraints = [
        x >= [low for low, _ in scaled_bounds],
        x <= [high for _, high in scaled_bounds],
        cp.sum(x) <= balance - fixed_expenses - savings
    ]
    # Створення та розв'язання задачі
    problem = cp.Problem(objective, constraints)
    problem.solve()
    # Формування результату
    class Result:
        def __init__(self, status, values):
            self.success = (status == cp.OPTIMAL)
            self.x = values
    return Result(problem.status, x.value)
```

Кінець лістингу 2.1

Для інтеграції модуля оптимізації в застосунок використовується архітектура клієнт-сервер. Клієнтська частина на Android через Retrofit надсилає запити до серверної частини, реалізованої на Flask, яка обробляє дані та повертає результат у форматі JSON. На стороні Android-клієнта Retrofit конфігурується через Hilt для забезпечення ін'єкцій залежностей – демонстрація на лістингу 2.2.

Лістинг 2.2 – Демонстрація Retrofit-instance

```
@Provides
@Singleton
@Named("BackendRetrofit")
fun provideBackendRetrofit(): Retrofit {
    return Retrofit.Builder()
        .baseUrl("http://ivannavakuliukkkk.pythonanywhere.com/")
        .addConverterFactory(MoshiConverterFactory.create())
        .client(okHttpClient)
        .build()
}
@Provides
@Singleton
fun provideBackendService(
    @Named("BackendRetrofit") retrofit: Retrofit
): BackendService {
    return retrofit.create(BackendService::class.java)
}
```

Кінець лістингу 2.2

Сервісний інтерфейс визначає методи для відправки запитів на сервер – лістинг 2.3.

Лістинг 2.3 – Демонстрація Сервісного інтерфейсу

```
interface BackendService {
    @POST("min_months")
    suspend fun getMinMonths(@Body request: MinMonthsRequest):
MinMonthsResponse
    @POST("optimize_goal")
    suspend fun getPlanWithGoal(@Body request: BudgetPlanRequest):
BudgetPlanResponse
    @POST("optimize_simple")
    suspend fun getPlanSimple(@Body request: BudgetPlanRequest):
BudgetPlanResponseSimple
}
```

Кінець лістингу 2.3

На стороні сервера Flask обробка запитів виглядає так, як наведено на лістингу 2.4, на прикладі одного з запитів – оптимізація з ціллю.

Лістинг 2.4 – Демонстрація обробки запитів Flask-сервера

```
# Шлях для запиту - оптимізація з ціллю
@app.route("/optimize_goal", methods=["POST"])
def optimize_goal():
    data = request.json
    if not data:
        return jsonify({"error": "No data provided"}), 400
    try:
        response = optimize_goal_logic(
            balance=data.get("balance", 0),
            income=data.get("income", 0),
            fixed_expenses_current=data["fixed_expenses"]["current"],
            fixed_expenses_stable=data["fixed_expenses"]["stable"],
            bounds=[tuple(b) for b in data["bounds"]],
            priorities=data["priorities"],
            goal=data.get("goal"),
            months=data.get("months", 1),
            categories=data.get("categories")
        )
        return jsonify(response)
    except ValueError as ve:
        return jsonify({"error": str(ve)}), 400
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Кінець лістингу 2.4

Повний код серверної частини, що реалізує алгоритм оптимізації та обробку запитів, наведено в додатку В. Разом із сервером представлені репозиторій та use case Android-додатку, які забезпечують взаємодію з сервером, обробку отриманих даних та подальше використання їх у логіці додатку.

Окрім модуля планування та оптимізації, у системі реалізовано ще один важливий механізм – власний симетричний алгоритм шифрування даних, який забезпечує конфіденційність фінансової інформації під час збереження в хмарній базі даних. Його створення було необхідним для того, щоб чутливі записи – транзакцій, баланси гаманців та інші фінансові дані – залишались захищеними.

У традиційних схемах конвертного шифрування використовується багаторівнева структура: головний ключ, асиметричний ключ для шифрування ключа даних (KEK) та симетричний ключ для шифрування самих даних (DEK). У розробленому рішенні підхід істотно змінено, щоб пристосувати його до

мобільного застосунку та забезпечити мобільність ключів. Опишемо основні відмінності реалізованої моделі.

PIN-код виступає аналогом головного ключа. Він створюється користувачем під час першого входу та зберігається у Encrypted Shared Preferences. Саме він є основою для подальшого формування ключового матеріалу й дозволяє відновити доступ до даних на будь-якому пристрої без необхідності синхронізації ключів у хмарі.

На основі PIN-коду генерується Master Key, який виконує роль KEK. Він створюється за допомогою алгоритму PBKDF2 з унікальною сіллю. Такий підхід забезпечує стійкість до атак перебором і дозволяє створити надійний 256-бітний AES-ключ.

DEK – випадково згенерований симетричний ключ, яким шифруються всі фінансові записи. Він зберігається виключно у зашифрованому вигляді в базі даних, а у пам'яті додатка перебуває лише в активній сесії.

Таким чином, було сформовано власний, більш мобільний варіант конвертного шифрування, адаптований для багатопристроєвого використання та збереження даних у Firebase .

Процес роботи алгоритму (перший вхід користувача):

– після авторизації користувач задає PIN, який зберігається у захищеному сховищі;

– на основі солі та PIN генерується Master Key (симетричний KEK);

– створюється DEK, який одразу шифрується Master Key та надсилається у Firebase;

– генерується короткий тестовий рядок (canary), який шифрується DEK і зберігається у базі даних для майбутньої перевірки PIN;

– усі чутливі записи (транзакції, баланси, тощо) шифруються DEK перед надсиланням у Firebase і розшифровуються лише локально на пристрої;

– після завершення роботи додатку DEK та Master Key видаляються з оперативної пам'яті.

Оскільки ключі не прив'язані до апаратного сховища, доступ із іншого

пристрою виглядає так:

– користувач вводить PIN-код;

– на його основі знову відтворюється Master Key;

– з Firebase завантажується DEK і розшифровується Master Key;

– перевіряється тестовий рядок (canary);

– якщо аутентифікація успішна, DEK використовується для шифрування та розшифрування даних так само, як і на попередньому пристрої.

Завдяки цій схемі не потрібно синхронізувати ключі або передавати їх між пристроями – достатньо лише PIN та збережених у базі службових даних.

Перейдемо до організації коду. Було обрано один з найпопулярніших архітектурних шаблон мобільної розробки MVVM (Model-View-ViewModel), додатково доповнений шаром UseCase, відповідно до принципів Чистої архітекрути (англ. Clean Architecture). Даний підхід дуже зручний – він дозволяє відокремити логіку роботи з даними, бізнес-логіку та відображення інтерфейсу. На рисунку 2.4 структура проекту у Androd Studio.

Шар Model відповідає за роботу з даними. Він включає, моделі, репозиторії, сервісні інтерфейси для роботи з API (Monobank та Backend). Всі моделі даних, які описані та використовуються:

– основні сутності застосунку: транзакція, гаманець, категорія;

– модель транзакції-відповіді, що надходить із банку;

– моделі для модулів планування та бюджетних налаштувань;

– моделі запитів і відповідей для оптимізаційного модуля на сервері.

Репозиторії виконують роль посередників між джерелами даних і бізнес-логікою. Вони інкапсулюють доступ до сервісів та забезпечують єдиний інтерфейс для всіх операцій. У застосунку реалізовано чотири репозиторії:

– Firebase Repository – робота з хмарною базою даних: збереження, оновлення та отримання зашифрованих фінансових записів;

– MonobankRepository – виконання запитів до банківського API, обробка історії транзакцій, категоризація тощо;

– BackendRepository – взаємодія з серверним модулем оптимізації, відправлення вхідних даних та отримання результатів;

– CryptoRepository – генерація і відновлення ключів, шифрування й дешифрування даних, перевірка коректності ключового матеріалу.

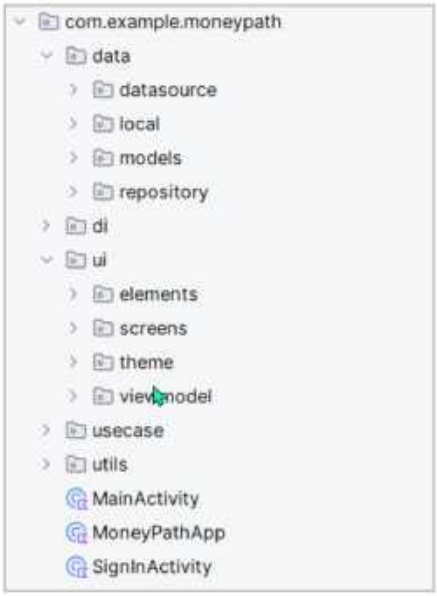

Рисунок 2.3 – Структура проекту

Джерело: розроблено автором

Шар Use Case (Interactor) реалізує бізнес-логіку системи. У ньому зосереджені сценарії, які оперують кількома джерелами даних (використовують декілька репозиторіїв) або потребують складних операцій. Завдяки цьому ViewModel не містить зайвого коду та відповідає лише за стан інтерфейсу. Use case поділені на 3 пакети:

– бізнес логіка: додавання, видалення транзакцій з корекцією балансу гаманця, виклик оптимізації та збереження результату у БД, менеджер синхронізації з банком (при кожному вході та кожних 2 хвилини онлайн) та інші;

– ініціалізація: сценарії формування або відновлення ключів під час входу в застосунок, перевірка доступу та підготовка криптографічного контексту;

– криптографія: операції шифрування та дешифрування конкретних полів, наприклад, отримання зашифрованої транзакції за ID із БД.

ViewModel виконує роль зв'язувальної ланки між Use Case та інтерфейсом. Вона отримує дані з Use Case, трансформує їх за потреби та передає у View через MutableState. Крім цього, ViewModel зберігає стан екрану, обробляє події користувача та забезпечує реактивну зміну UI. Кожному екрану відповідає окрема ViewModel, що дозволяє уникнути перетину логіки.

Шар View – це Composable-екрани, які відповідають виключно за відображення даних і взаємодію користувача з інтерфейсом. Уся логіка винесена за межі View, у ViewModel та Use Case, що робить код чистим.

Завдяки такій структурі код залишається модульним, легко розширюваним і зручним для повторного використання компонентів у майбутніх версіях.

Базою для збереження даних виступає Firebase Realtime Database. Опишемо структуру даних, які там зберігаються. Вони організовані за ієрархічною схемою. У корені бази міститься колекція users, де кожен користувач представлений унікальним ідентифікатором (UID), отриманим під час авторизації у Firebase Authentication. Структура БД наведена на рисунку 2.4.

Рисунок 2.4 – Структура БД

Джерело: розроблено автором

До запису користувача прив'язані такі основні вузли:

а) wallets – зберігає інформацію про всі гаманці користувача: назва, тип, поточний баланс (зашифрований) та службові метадані;

б) transactions – вузол з історією транзакцій, що містить зашифровані суми, категорії, дату, прив'язку до гаманця та додаткові атрибути;

в) planning – вузол планування бюджету, містить:

1) set_up – початкові налаштування бюджету;

2) total_plan – сформований загальний план на місяць;

3) additional_plans – збережені альтернативні сценарії планування;

г) security – службова інформація, пов'язана зі шифруванням:

1) збережений зашифрований DEK (wrapped key);

2) salt для генерації Master Key;

3) test-рядок (canary).

Перейдемо до опису користувацького інтерфейсу мобільного застосунку. Почнемо з карти сторінок (англ. Site map) на рисунку 2.5, яка створювалась на етапі проектування інтерфейсу. Вона відображає ієрархію та логіку переходів між екранами. Карта дозволяє одразу бачити, які сторінки існують та як користувач буде переміщуватися між ними.

Рисунок 2.5 – Карта сторінок застосунку

Джерело: розроблено автором

Першим кроком людина потрапляє на екран авторизації, де пропонується вибрати свій Google-акаунт, а також створити або ввести раніше встановлений

PIN-код. Відповідні інтерфейси показано на рисунку 2.6.

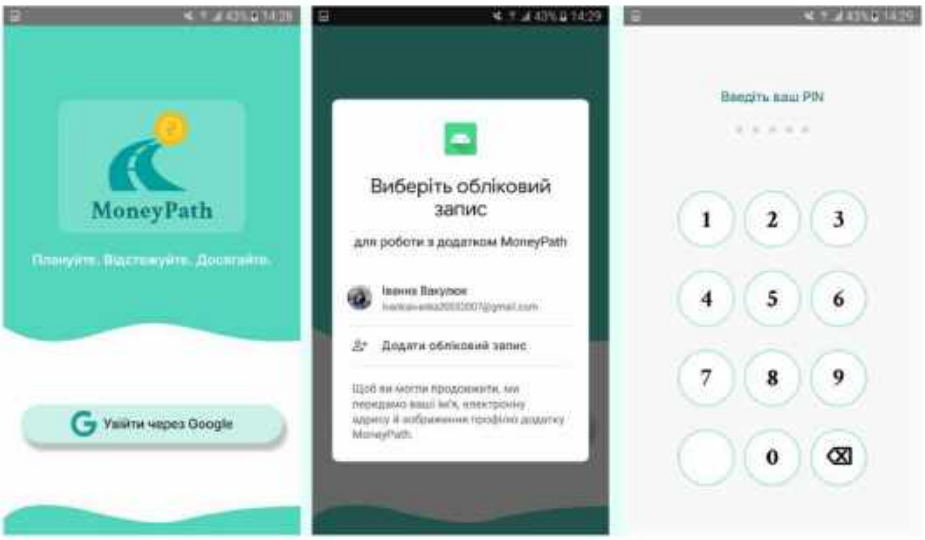

Рисунок 2.6 – Сторінки авторизації

Джерело: розроблено автором

Головна сторінка дає змогу переглядати транзакції за вибрану дату та бачити всі гаманці з їх балансом (рис. 2.7). Звідси можна додати транзакцію чи гаманець, переглянути або змінити деталі транзакції. Додавання гаманця Monobank доступне на сторінці «Інше».

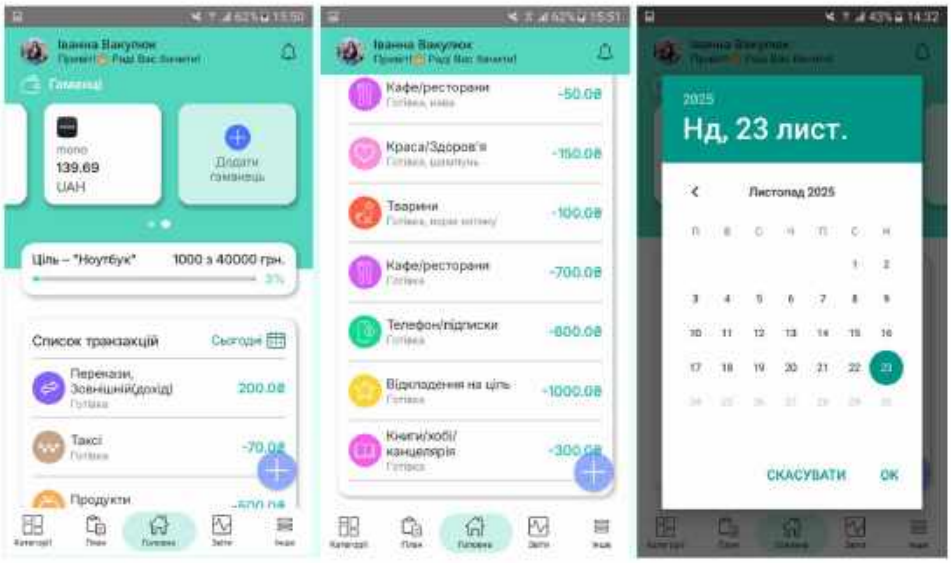

Рисунок 2.7 – Головна сторінка «MoneyPath»

Джерело: розроблено автором

На сторінці додавання гаманця, клієнт сервісу повинен вказати назву, тип та баланс (рис 2.8). Він також може змінити ці дані на сторінці редагування. Сторінку додавання транзакції наведено на рисунку 2.9. Тут можна обрати тип операції – дохід, витрата або переказ (внутрішній між гаманцями чи зовнішній). Також доступний вибір категорії, дати, гаманця та можливість додати нотатку.

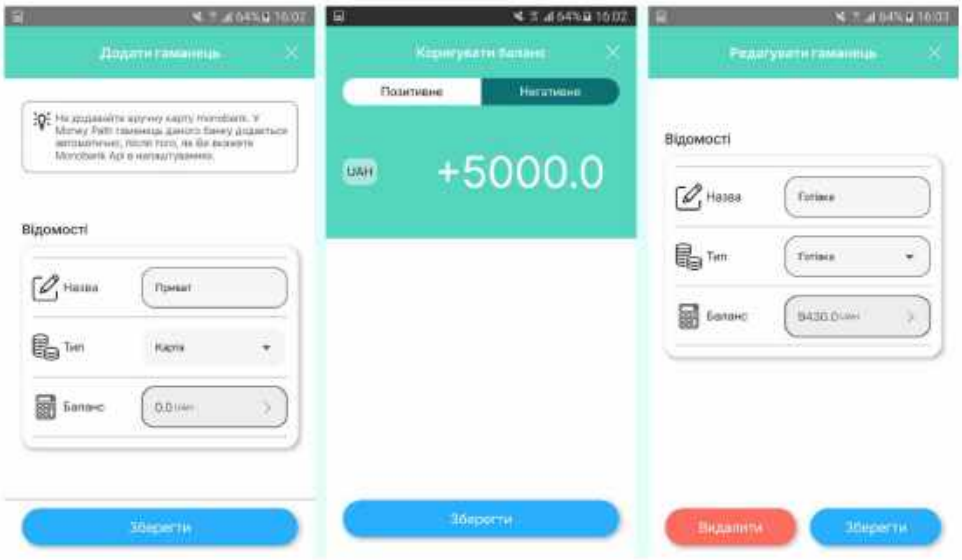

Рисунок 2.8 – Сторінки додавання гаманця

Джерело: розроблено автором

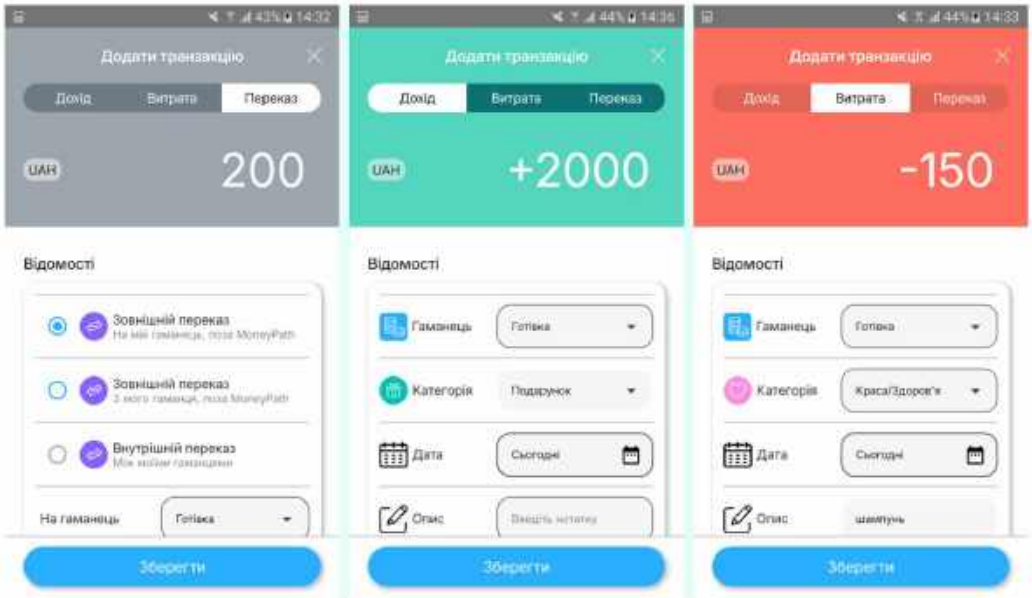

Рисунок 2.9 – Сторінки додавання транзакції

Джерело: розроблено автором

Форма налаштування плану складається з 10 екранів, якщо обрано планування з фінансовою ціллю, і з 8 – якщо без неї. У режимі без цілі пропускаються кроки введення суми, назви та терміну досягнення. На цих екранах користувач послідовно вводить усі вхідні дані для алгоритму у зручному та зрозумілому форматі. Основні екрани наведені на рисунку 2.10.

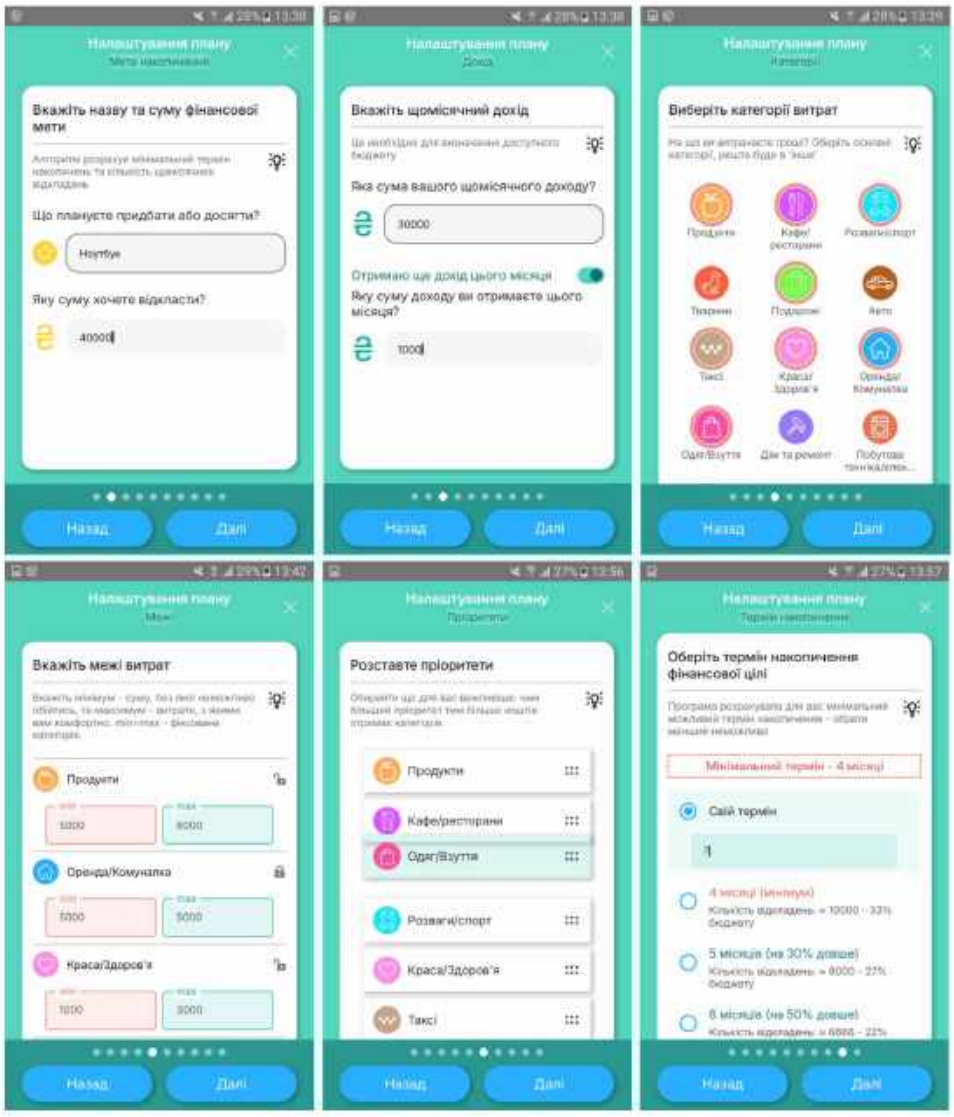

Рисунок 2.10 – Основні сторінки форми налаштування плану

Джерело: розроблено автором

Останніми розглянемо сторінки плану (рис. 2.11) та категорій (рис. 2.12). На першій подано результат оптимізації: стовпчасту діаграму та текстовий список категорій із сумами. Також доступний перегляд альтернативних планів та

вибір одного з них. Сторінка категорій дає змогу оцінити, наскільки фактичні витрати відповідають запланованим. Тут наведено загальні суми доходів і витрат, реальні витрати за вибраний період, прогрес-бари використання бюджету по категоріях, а також кругові діаграми фактичного та запланованого розподілу.

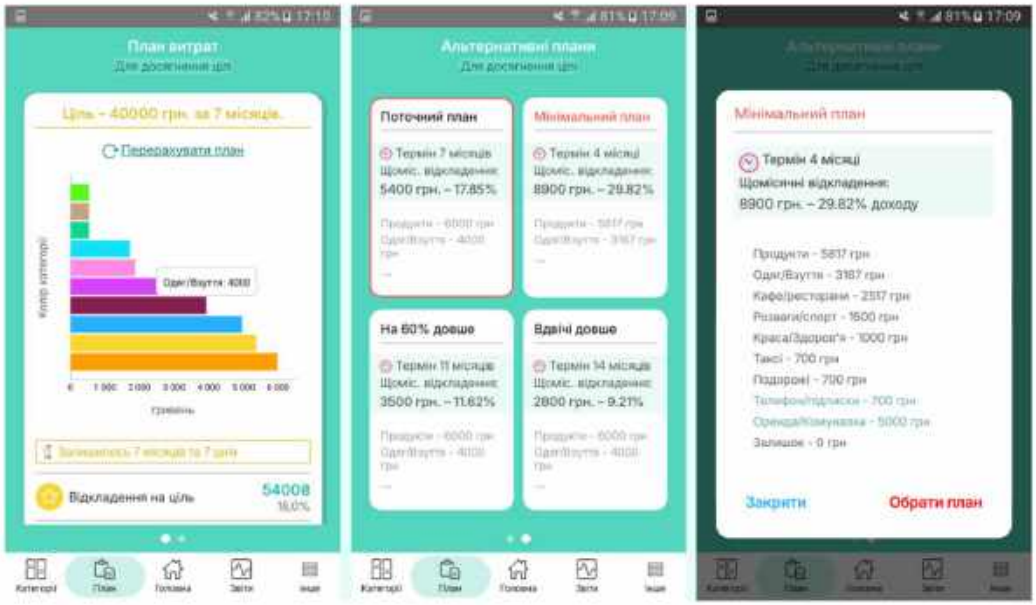

Рисунок 2.11 – Сторінка плану

Джерело: розроблено автором

Рисунок 2.12 – Сторінка категорій

Джерело: розроблено автором

В загальному інтерфейс налічує 21 сторінки та 10 спливаючих вікон: для підтвердження операцій видалення даних (транзакції, плану, токена, акаунту та інше) та сповіщення про досягнення цілі або закінчення плану. Для кожної сторінки з введенням даних відбувається виведення помилок, а сторінки на яких відбувається завантаження даних з ресурсів містять прогрес бари, щоб уникнути неправильної побудови UI поки асинхронна операція не завершилась.

Матеріали, викладені у розділі 2, дають змогу сформулювати кілька роботи. Цей метод дозволяє враховувати пріоритети, працювати з гнучкими обмеженнями та збалансовувати витрати – тобто реалізовувати ті механізми, які неможливо повністю забезпечити класичними лінійними підходами.

По-друге, аналіз практичної реалізації та обраного технологічного стеку засвідчив, що використані засоби розробки повністю підтримують інтеграцію математичної моделі у мобільний застосунок. Вивчення архітектури, обробки даних, взаємодії з API та інструментів побудови інтерфейсу показало, що теоретичні алгоритми можуть працювати разом зі стандартним функціоналом без ускладнення структури програми.

У результаті було розроблено повнофункціональний застосунок із зручним інтерфейсом, який, окрім звичного обліку фінансів та аналізу статистики доходів і витрат, надає користувачу можливість планувати бюджет, відстежувати виконання плану та ефективніше досягати фінансових цілей.

РОЗДІЛ 3
ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ РЕЗУЛЬТАТИВНОСТІ АЛГОРИТМУ МАТЕМАТИЧНОЇ ОПТИМІЗАЦІЇ ДЛЯ УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

3.1 Методика проведення дослідження

Застосунок уже реалізований, має багато функцій та гарний інтерфейс, проте важливо дослідити його результативність. Дослідження проводиться з метою оцінки ефективності розробленої моделі особистого бюджетування на основі методів оптимізації витрат та підтвердження її результативності. Методика включає три основні напрями: порівняльний аналіз алгоритмів, імітаційне моделювання різних сценаріїв та оцінку продуктивності алгоритму.

Метод імітаційного моделювання полягає у тестуванні моделі на різних сценаріях, що дозволяє оцінити стійкість та універсальність алгоритму при змінних вхідних даних та умовах. Таким чином можна проводити дослідження без необхідності залучення реальних користувачів, моделюючи різні рівні доходів, витрат та фінансових цілей.

Гіпотеза, яка висувається в межах методу імітаційного моделювання це ефективність алгоритму для різних сценаріїв, стійкість до збільшення кількості категорій, мінімального бюджету та різниці в пріоритетах.

Для кількісної оцінки ефективності розподілу бюджету по категоріях, застосовується метрика нормалізованої частки розподілу у межах категорії. Вона визначає, наскільки фактичне значення витрат x наближається до бажаного бюджету (верхньої межі) відносно мінімально необхідного рівня (нижньої межі) і визначається формулою 3.1:

$$a_i = \frac{x_i - L_i}{U_i - L_i} \tag{3.1}$$

де x_i – результат знайдений оптимізаційною моделлю (для категорії i);

L_i – нижня межа витрат (мінімально необхідний бюджет категорії);

U_i – верхня межа витрат (бажаний або комфортний бюджет).

Ця метрика базується на класичному підході лінійної нормалізації, що часто застосовується в багатокритеріальному прийнятті рішень (MCDM), машинному навчанні та економіко-математичному моделюванні. Вона дозволяє трансформувати змінні до стандартного діапазону (зазвичай 0-1) та полегшує порівняння результатів [20].

Інтерпретація результатів в контексті задачі:

– 0 – виділено рівно мінімум, потреба задоволена лише на базовому рівні;

– 1 – виділено максимально бажану суму;

– 0,5 – виділено половину між мінімумом і максимумом;

– більше 1 або менше 0 – можливі лише при помилках даних.

Застосування цієї метрики є доцільним для задачі оптимізації, оскільки вона враховує нижню та верхню межу категорії, має чіткий інтервал та демонструє ступінь задоволення потреб користувача. Крім того, нормалізована частка дозволяє легко та наочно аналізувати розподіл коштів, що спрощує порівняння результатів різних сценаріїв та моделей.

Наведемо основні сценарії, які будуть використовуватись для імітаційного моделювання та вхідні дані.

Перший тестовий сценарій моделює користувача із стабільним, але низьким доходом (наприклад, студента), для якого головним пріоритетом є покриття базових витрат та мінімальні заощадження. Сценарій також дозволяє оцінити роботу моделі в умовах обмеженого бюджету, зокрема:

– при наявності великих фіксованих витрат, що займають значну частину доходу (наприклад, іпотека або оренда);

– при спробі накопичення значної суми за короткий термін.

Метою проведення тесту є також перевірка того, чи алгоритм повідомляє користувача про неможливість досягнення фінансової цілі, а також чи забезпечується фінансування малопріоритетних категорій.

Вхідні дані до першого сценарію наведені на таблиці 3.1.

Таблиця 3.1 – Вхідні дані до першого тестового сценарію

Дохід (грн)	Ціль (грн)	Термін (міс)	Назва категорії	Нижня межа (грн)	Верхня межа (грн)	Пріоритет
10000	10000	3	1. Розваги та спорт	1000	2000	1
			2. Продукти	2000	3000	2
			3. Транспорт	500	800	3
			4. Інше	0	1000	4
			5. Телефон та підписки	300	800	5
			6. Навчання	3000	3000	-

Другий тестовий сценарій моделює користувача із високим доходом, для якого цільові накопичення не встановлені або не є актуальними (табл. 3.2). Особливістю цього сценарію є однакові межі для кожної категорії, що дозволяє наочно продемонструвати плавність розподілу ресурсів. Результат повинен показати, яку роль у моделі відіграють саме пріоритети, перевіряючи, чи витрати розподіляються пропорційно до них. У таких умовах легше оцінити роботу алгоритму, ніж у реальних сценаріях, де межі категорій можуть значно відрізнятися, що є тяжчим для сприйняття.

Таблиця 3.2 – Вхідні дані до другого тестового сценарію

Дохід (грн)	Ціль (грн)	Термін (міс)	Назва категорії	Нижня межа (грн)	Верхня межа (грн)	Пріоритет
40000	-	-	1. Продукти	2000	5000	1
			2. Одяг та взуття	2000	5000	2
			3. Кафе та ресторани	2000	5000	3
			4. Краса та здоров'я	2000	5000	4
			5. Розваги та спорт	2000	5000	5
			6. Транспорт	2000	5000	6
			7. Подорожі	2000	5000	7
			8. Книги та хобі	2000	5000	8
			9. Побутова техніка	2000	5000	9
			10. Інше	2000	5000	10

Третій тестовий сценарій є класичним прикладом сприятливих умов для роботи моделі (табл. 3.3). Користувач має стабільний дохід та помірну цільову суму для накопичення за помірний термін, визначено декілька фіксованих категорій. Цей сценарій дозволяє оцінити ефективність алгоритму в оптимальних умовах.

Таблиця 3.3 – Вхідні дані до третього тестового сценарію

Дохід (грн)	Ціль (грн)	Термін (міс)	Назва категорії	Нижня межа (грн)	Верхня межа (грн)	Пріоритет
30000	40000	8	1. Продукти	5000	6000	1
			2. Одяг та взуття	2000	4000	2
			3. Кафе та ресторани	2000	4000	3
			4. Краса та здоров'я	1000	3000	4
			5. Розваги та спорт	1500	3500	5
			6. Таксі	700	3000	6
			7. Подорожі	700	1500	7
			8. Оренда та комуналка	5000	5000	-
			9. Телефон та підписки	700	700	-

Метою четвертою сценарію є перевірка роботи алгоритму для довгострокових цілей. Користувач має середній бюджет та мету – накопичити велику ціль за 3 роки. Вхідні дані на таблиці 3.4.

Таблиця 3.4 – Вхідні дані до четвертого тестового сценарію

Дохід (грн)	Ціль (грн)	Термін (міс)	Назва категорії	Нижня межа (грн)	Верхня межа (грн)	Пріоритет
25000	300000	36	1. Продукти	1500	4000	1
			2. Транспорт	1000	3000	2
			3. Кафе та ресторани	2000	4000	3
			4. Краса та здоров'я	1000	2000	4
			5. Одяг та взуття	500	1000	5
			6. Благодійність	500	2000	6
			7. Інше	0	2000	7
			8. Авто	2000	4000	8
			9. Оренда та комуналка	6000	6000	-

Останній тестовий сценарій моделює ситуацію з максимально деталізованим обліком, коли користувач обирає понад 20 категорій для планування та не має жодних фіксованих витрат. Одночасно ставиться мета накопичити певну суму протягом трьох місяців. Такий сценарій дає змогу оцінити, наскільки ефективно алгоритм працює за великої кількості категорій, більшість із яких має нульові нижні межі. Особливий інтерес становить перевірка того, чи будуть виділені кошти на категорії з дуже низьким

пріоритетом та як алгоритм поводитиметься за умов високої конкуренції між витратами. Вхідні дані наведено в таблиці 3.5.

Таблиця 3.5 – Вхідні дані до п'ятого сценарію

Дохід (грн)	Ціль (грн)	Термін (міс)	Назва категорії	Нижня межа (грн)	Верхня межа (грн)	Пріоритет
35000	25000	3	1. Продукти	1200	5000	1
			2. Оренда та комуналка	4000	7000	2
			3. Транспорт	300	800	3
			4. Таксі	400	1500	4
			5. Пошта	100	500	5
			6. Освіта	0	2000	6
			7. Краса та здоров'я	0	1000	7
			8. Одяг та взуття	0	1000	8
			9. Подарунки	0	500	9
			10. Професійні послуги	0	800	10
			11. Кафе та ресторани	0	1000	11
			12. Телефон та підписки	0	500	12
			13. Дім та ремонт	0	1500	13
			14. Книги, хобі	0	2000	14
			15. Розваги та спорт	0	2000	15
			16. Побутова техніка	0	1500	16
			17. Державні платежі	0	800	17
			18. Благодійність	0	300	18
			19. Резервний фонд	0	2000	19
			20. Інше	0	500	20

Для кожного зі сценаріїв навмисно обрано різну кількість категорій (5-20), оскільки це є необхідним для другого методу дослідження – оцінки продуктивності алгоритму. Оцінювання обчислювальної складності базуватиметься на часі пошуку оптимального розв'язку та кількості ітерацій, що виконуються під час оптимізації. Бібліотека CVXPY надає можливість отримати ці показники безпосередньо з результатів оптимізації. Гіпотеза полягає в тому, що обчислювальна складність зростає із збільшенням кількості категорій витрат.

Останнім етапом дослідження є порівняльний аналіз методів оптимізації. У цьому підході результати роботи моделі квадратичного програмування зіставляються з результатами початкової версії алгоритму, побудованої на основі лінійного програмування. Для об'єктивності порівняння використовуються ті

самі вхідні дані, що й у попередніх сценаріях та метрика нормалізованої частки розподілу. Метою цього аналізу є перевірка доцільності застосування квадратичної моделі в контексті задачі бюджетування. Висунута гіпотеза полягає в тому, що квадратичне програмування забезпечує більш плавний та збалансований розподіл бюджету порівняно з лінійним підходом.

3.2 Обробка та аналіз отриманих результатів

Після опису методології дослідження перейдемо до аналізу отриманих результатів. Вихідні дані першого сценарію для алгоритмів квадратичного програмування (QP) та лінійного програмування (LP) наведено у таблиці 3.6. У таблиці фіксовані категорії позначено буквою «ф», а відкладення на ціль – символом «*». x_i – результат розподілу по категорії i (виділена сума). a_i – нормалізована частка розподілу.

Мінімальний термін накопичення, обчислений системою, склав 4 місяці, хоча користувач зазначав бажання досягти цілі за 3 місяці. Система повідомляє про неможливість досягнення цілі через обмежений бюджет, тому наведено щомісячний розподіл витрат за мінімально можливий термін – 4 місяці.

Результати демонструють класичний сценарій для критично обмеженого бюджету: наявних коштів достатньо лише для покриття базових потреб, а мінімальний залишок розподіляється серед категорій з найвищим пріоритетом. QP навіть у критичних умовах демонструє більш рівномірний розподіл – залишок розподіляється між двома верхніми категоріями, тоді як LP виділяє все лише на перший пріоритет. Отже, QP має перевагу навіть у випадках обмеженого бюджету.

Таблиця 3.6 – Результати виконання сценарію 1

Номер категорії	1	2	3	4	5	6 (ф)	7*
x_i QP(грн)	1458	2242	500	0	300	3000	2500
a_i (QP)	0,46	0,24	0	0	0	1	-
x_i LP(грн)	1700	2000	500	0	300	3000	2500
a_i (LP)	0,70	0	0	0	0	1	-

Результати алгоритмів з вхідними даними другого сценарію явно демонструють значну відмінність між QP та LP (табл. 3.7). Змодельовані вхідні дані з однаковими межами дозволили явно побачити цю різницю.

Метод QP забезпечує плавний, пропорційний розподіл коштів між категоріями залежно від їхніх пріоритетів: найбільше виділяється на високопріоритетні категорії, поступово зменшуючи суму до категорій з нижчими пріоритетами. Нормалізовані показники a_i, які відображають ступінь задоволення потреб категорій, демонструють чітку пропорційність – від 0,83 у першій категорії до 0,18 у останній.

Натомість метод LP розподіляє ресурси більш жорстко: верхні межі виділяються для п'яти категорій із найвищим пріоритетом, тоді як категорії з нижчими пріоритетами отримують лише мінімальні суми. Це наочно демонструє, що LP схильний концентрувати кошти на обмеженому числі пріоритетних категорій, ігноруючи плавність розподілу.

Хоч сума нормалізованих часток розподілу однакова – 5, бачимо абсолютно різну поведінку алгоритмів.

Таблиця 3.7 – Результати виконання сценарію 2

Номер категорії	1	2	3	4	5	6	7	8	9	10
x_i QP(грн)	4475	4258	4042	3825	3608	3392	3175	2958	2742	2552
a_i (QP)	0,83	0,75	0,68	0,61	0,54	0,46	0,39	0,32	0,25	0,18
x_i LP(грн)	5000	5000	5000	5000	5000	2000	2000	2000	2000	2000
a_i(LP)	1	1	1	1	1	0	0	0	0	0

Вхідні дані третього сценарію є найбільш реалістичними та сприятливими для роботи алгоритму – результат алгоритму у таблиці 3.8. Це є протилежний випадок до першого сценарію з обмеженим бюджетом. Алгоритм показує свою роботу у всій «красі» і легко оцінити його ефективність.

Результат, який отримав користувач можна сформулювати так: досягнення цілі за 8 місяців є досидь комфортним. На категорії з високим пріоритетом виділяється достатньо хороший бюджет (продукти, одяг та інше), однак для досягнення цілі потрібно зекономити на таксі та подорожах. LP знову показує

тенденцію до концентрації на верхніх пріоритетах, однак це вже не так помітно, оскільки бюджет більший. QP знову демонструє свою перевагу покриваючи середні пріоритети.

Таблиця 3.8 – Результат виконання сценарію 3

Номер категорії	1	2	3	4	5	6	7	8 (ф)	9(ф)	10*
x_i QP(грн)	5725	3508	3292	2075	2358	1642	700	5000	700	5000
a_i(QP)	0,73	0,75	0,65	0,54	0,43	0,41	0	100	100	-
x_i LP(грн)	6000	4000	4000	2400	1500	700	700	5000	700	5000
a_i(LP)	1	1	1	0,7	0	0	0	100	100	-

Результати виконання четвертого сценарію підтверджують ефективність алгоритму для довгострокових накопичень в 3 роки (табл. 3.9). Через велику ціль бюджет виявився досить обмеженим – обидва алгоритми змогли повністю покрити лише перші три категорії. При цьому QP демонструє нову поведінку: навіть при обмеженому бюджеті кошти можуть виділитися на категорії з низьким пріоритетом, враховуючи не тільки сам пріоритет, а й розрив між фактичним виділенням та бажаною верхньою межею. Чим більший розрив, тим більший «штраф» за недоотримання, що забезпечує більш збалансований розподіл. Наприклад, категорії з 7 та 8 пріоритетом отримали певні кошти завдяки цьому механізму.

Таблиця 3.9 – Результат виконання сценарію 4

Номер категорії	1	2	3	4	5	6	7	8	9(ф)	10*
x_i QP(грн)	3704	2463	3222	1000	500	500	259	2019	6000	8333
a_i (QP)	0,88	0,73	0,61	0	0	0	0,129	0,009	1	-
x_i LP(грн)	4000	3000	2667	1000	500	500	0	2000	6000	8333
a_i (LP)	1	1	0,33	0	0	0	0	0	1	-

Дану особливість можна розглядати як додаткову перевагу алгоритму QP: він не концентрує весь бюджет на верхніх категоріях, а розподіляє його з урахуванням пріоритетів і величини розриву між межами. LP, у свою чергу, знову ж показує свою класичну поведінку.

Результат п'ятого сценарію, який включає 20 категорій підтверджує результативність алгоритму й для деталізованого планування (табл. 3.10). За

наявності доступного бюджету, QP покрив 18 з 20 категорій, зважаючи на те, що майже всі мали нульові нижні межі. LP ж виділив ресурс на 15 з 20 категорій. Особливість врахування розриву між межами знову простежується в розподілі QP – на 19 категорію виділилась частка 0,59.

Таблиця 3.10 – Результат виконання сценарію 5

Номер категорії	1	2	3	4	5	6	7	8	9	10
x_i QP(грн)	5000	7000	800	1500	500	2000	1000	954	454	537
a_i (QP)	1	1	1	1	1	1	1	0,954	0,9	0,67
x_i LP(грн)	5000	7000	800	1500	500	2000	1000	1000	500	800
a_i (LP)	1	1	1	1	1	1	1	1	1	1

Продовження таблиці 3.10

Номер категорії	11	12	13	14	15	16	17	18	19	20
x_i QP(грн)	737	237	1020	1520	1304	804	104	0	1195	0
a_i (QP)	0,73	0,47	0,68	0,76	0,65	0,53	0,13	0	0,59	0
x_i LP(грн)	1000	500	1500	2000	1567	0	0	0	0	0
a_i (LP)	1	1	1	1	0,78	0	0	0	0	0

Отже, алгоритм на основі квадратичного програмування показав кращі результати у всіх п'яти сценаріях. Розподіл коштів виходить більш рівномірним і забезпечує виділення грошей не тільки на найважливіші категорії. Спочатку алгоритм враховує пріоритет категорій, а потім звертає увагу на розрив між верхньою межею та мінімальною сумою. Це дозволяє навіть при обмеженому бюджеті частково фінансувати менш пріоритетні категорії, що робить розподіл більш реалістичним. Саме тому метод квадратичного програмування є більш придатним для вирішення задачі планування особистого бюджету.

Перейдемо до обчислюваної складності – результат оцінки продуктивності алгоритму у таблиці 3.11.

Таблиця 3.11 – Результат оцінки продуктивності алгоритму.

Номер сценарію	К-ть категорій	К-сть ітерацій	Час знаходження оптимуму (мкс)
1	5	125	79
2	10	125	100
3	7	175	106
4	8	125	129
5	20	150	180

Аналіз часу виконання та кількості ітерацій показав, що час знаходження оптимального розв'язку зростає разом із збільшенням кількості категорій витрат. Наприклад, сценарій із 5 категоріями виконується за 79 мікросекунд, а сценарій із 20 категоріями – за 180 мікросекунд. Водночас кількість ітерацій не залежить від числа категорій і залишається приблизно стабільною, що пояснюється особливостями роботи розв'язувача CVXPY: алгоритм використовує внутрішні методи оптимізації, які не масштабуються лінійно з кількістю змінних.

Отже, обчислювальну складність по часі можна оцінити як залежну від кількості категорій, приблизно $O(n)$, а по кількості ітерацій – практично константна $O(1)$.

Аналіз сценаріїв довів, що розроблений алгоритм формує реалістичні та зрозумілі плани витрат. Метод квадратичного програмування забезпечує збалансований розподіл бюджету в різних умовах – від критичних до сприятливих. Порівняння з лінійним методом показало очевидну перевагу QP у якості планування. Час виконання алгоритму дуже малий – вимірюється у мікросекундах. У сукупності це підтверджує, що обраний підхід є найбільш результативний для задачі роботи та є доцільним для використання у схожих задачах розподілу ресурсів.

ВИСНОВКИ

У кваліфікаційній роботі магістра виконано комплексне дослідження та розроблення Android-застосунку для управління особистими фінансами. Основою розробки стали методи оптимізації бюджетного планування. Усі завдання, визначені у вступі та структуровані в розділі 1.3, виконані повністю, зокрема:

– здійснено аналіз предметної області, включно з рівнем фінансової грамотності населення, впливом мобільних додатків для персонального бюджетування та результатами існуючих досліджень щодо використання методів математичної оптимізації у подібних сферах;

– проведено огляд найпопулярніших мобільних застосунків для персонального бюджетування та їх функцій: YNAB, Monefy, GoodBudget, Spendee, Wallet;

– визначено основні вимоги до функціоналу, інтерфейсу та архітектури розроблюваного застосунку для управління особистими фінансами;

– обрано метод оптимізації – квадратичне програмування, обґрунтовано його вибір порівнянням з іншими методами та побудовано математичну модель для подальшого формування алгоритму;

– обрано та обґрунтовано вибір технологій та інструментів розробки: Android-застосунок – Kotlin, Android Studio, Retrofit, Jetpack Compose, JCE, Firebase, банківське API; модуль оптимізації – Python, CVXPY, Flask, PythonAnywhere.

– розроблено мобільний застосунок для операційної системи Android, який реалізує функціонал планування бюджету, обліку витрат та доходів, наведення статистики, інтеграції з банком і шифрування даних;

– реалізовано модуль оптимізаційного розподілу бюджету та інтегровано до Android-застосунку з допомогою архітектури «клієнт-сервер»;

– створено інтуїтивний та зручний інтерфейс;

– проведене тестове експериментальне дослідження розробленого підходу,

з використанням імітаційного моделювання сценаріїв, порівняльного аналізу з результатами моделі лінійного програмування та оцінки обчислювальної складності.

Результатом є повнофункціональний Android-застосунок, який окрім стандартних функцій обліку витрат та доходів, пропонує користувачу можливість спланувати бюджет методом математичної оптимізації, досягти фінансових цілей та відстежувати прогрес. Планування працює ефективно та швидко для різних сценаріїв: з обмеженим бюджетом та достатнім, для досягнення довгострокових та короткострокових цілей.

Матеріали кваліфікаційної роботи магістра, зокрема розроблений підхід до планування бюджету з використанням методу квадратичного програмування, мають потенціал практичного застосування. Запропонована модель може використовуватися у сфері персонального фінансового менеджменту, у фінтех-сервісах та на освітніх платформах, що сприяють підвищенню фінансової грамотності населення. Розроблений алгоритм придатний як для індивідуального бюджетування, так і як складова частина комплексних фінансових систем або основа для створення автоматизованих рекомендаційних сервісів у банківських застосунках. Завдяки універсальності математичної моделі, її можна адаптувати й до інших задач оптимального розподілу ресурсів – від планування бюджетів проектів до вирішення аналогічних задач у інших фінансових сферах.

Можливими подальшими шляхами вдосконалення розробленого рішення можуть бути розширення за рахунок стохастичних та адаптивних методів, які дозволяють врахувати нестабільність доходу та коливання витрат. Також перспективним є впровадження алгоритмів для прогнозування майбутніх витрат і доходів та механізмів для надання персоналізованих фінансових рекомендацій. Можливим є створення веб-версії застосунку, версії для платформи IOS та інтеграції API інших банків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mobile application market size, share & trends analysis report. Grand View Research. URL: https://surl.li/lmrdhk (дата звернення: 05.08.2025).

2. Андрущак І. Є., Вакулюк І. В. Персональне бюджетування: аналіз, програмних рішень, та оптимізаційних можливостей. Науковий журнал «Комп'ютерно-інтегровані технології: освіта, наука, виробництво». Луцьк: Луцький НТУ, 2025. Вип. № 59. С. 29-36.

3. Вакулюк І. В., Андрущак І. Є. Криптографічний захист даних у мобільних додатках: огляд інструментів і приклад реалізації конвертного шифрування. Proceedings of the XIV International Scientific and Practical Conference. Sofia, Bulgaria. 2025. С. 21-27. URL: https://surl.li/akxaqi (дата звернення: 02.12.2025).

4. Національний банк України. Фінансова грамотність, фінансова інклюзія та фінансовий добробут в Україні у 2021 році. Нац. банк України; підтримка USAID. Київ, 2021. 64 с. URL: https://surl.li/hdistb (дата звернення: 15.08.2025).

5. Національний банк України. Національна стратегія розвитку фінансової грамотності до 2030 року. URL: https://surli.cc/swxorm (дата звернення: 15.08.2025).

6. 10 Budgeting Strategies and Methods: Find Your Best Fit. MoneyLion. URL: https://surl.li/xhghso (дата звернення: 09.09.2025).

7. Frisancho V., Herrera A., Prina S. Can a mobile-app-based behavioral intervention teach financial skills to youth? Experimental evidence from a financial diaries study. Journal of Economic Behavior & Organization. 2023. Vol. 214. P. 595-614.

8. French D., McKillop D., Stewart E. The effectiveness of smartphone apps in improving financial capability. Financial Literacy and Responsible Finance in the FinTech Era. 2021. P. 6-22.

9. Alenazi M., Sas C. Evaluating Budgeting Apps: Limited Support for Budgeting Compared to Tracking. 36th International BCS Human-Computer

Interaction Conference. 2023.

10. Wealth of Advice: 25 Tips to Learn the Super Budgeting App YNAB. PC MAG. URL: https://surl.li/cqkylp (дата звернення: 15.09.2025).

11. How to get started on Android. GoodBudget. URL: https://surli.cc/uahxlv (дата звернення: 15.09.2025).

12. Prathyusha G., Kumara K. N. U., Kumara K. M. P. A Goal Programming Model for Budgetary Allocation of an IT Organization. Communications in Mathematics and Applications. 2023. Vol. 14, № 1. P. 429-437.

13. Артюшенко О. Застосування методів лінійного програмування та оптимізації для підвищення ефективності бюджетних видатків у сфері фінансового забезпечення військ. Вісник Київського національного університету імені Тараса Шевченка. Військово-спеціальні науки. 2023. № 3(55). С. 41-47.

14. Ханін Г. С. Модель оптимального розподілу рекламного бюджету кафедри : пояснювальна записка до кваліфікаційної роботи здобувача вищої освіти на другому (магістерському) рівні, спеціальність 113 Прикладна математика. М-во освіти і науки України, Харків. нац. ун-т радіоелектроніки. Харків, 2025. 62 с.

15. Григорків В. С., Григорків М. В., Ярошенко О. І. Оптимізаційні методи та моделі : підручник. Чернівці : Чернівецький нац. ун-т, 2022. 440 с.

16. Formulation of quadratic programming problems. Fiveable. URL: https://surl.li/ydmtlr (дата звернення: 10.10.2025).

17. Linear Programming: Soft Constraints and Goal Programming. Medium. URL: https://surl.li/mrjvwl (дата звернення: 15.10.2025).

18. Android Studio. Офіційна документація. URL: https://surl.li/vcflsk (дата звернення: 01.11.2025).

19. Monobank Open Api (v250818). Mono Api. URL: https://surl.lt/ywrzjj (дата звернення: 02.11.2025).

20. Noor Fatima. A Comprehensive Guide to Normalization in Machine Learning. Medium. URL: https://surl.li/inghxf (дата звернення: 10.11.2025).