ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ

Факультет міжнародних економічних відносин та інформаційних технологій

Кафедра комп'ютерних наук, прикладної та вищої математики

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

Прототипування мобільного застосунку для відстеження особистих фінансів

Виконав: здобувач вищої освіти
першого (бакалаврського) рівня
ОПП "Комп'ютерні науки"
спеціальності 122 "Комп'ютерні науки"
_____ **Вацюк Микола Іванович**
(підпис)

Науковий керівник
_____ старший викладач Крутяк М. Б.
(підпис)

Львів – 2026

ЛЬВІВСЬКИЙ ТОРГОВЕЛЬНО-ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
Факультет міжнародних економічних відносин та інформаційних технологій
Кафедра комп'ютерних наук, прикладної та вищої математики
Перший (бакалаврський) рівень вищої освіти
Спеціальність 122 "Комп'ютерні науки"
Освітньо-професійна програма "Комп'ютерні науки"

"ЗАТВЕРДЖУЮ"

Завідувач кафедри комп'ютерних наук,
прикладної та вищої математики

_____ З. Г. Новосад

13.10.2025 р.

Завдання
до кваліфікаційної роботи
Вацюку Миколі Івановичу

(прізвище, ім'я, по батькові)

1.Тема роботи **Прототипування мобільного застосунку для відстеження особистих фінансів**

керівник роботи **Крутяк Марія Богданівна, *старший викладач***

затверджена наказом по Університету ***№ 992/03 від 06 жовтня 2025 р.***

2.Термін здачі здобувачем вищої освіти роботи 12 червня 2026 р.
3. Вхідні дані до роботи: література, документація, матеріали та сайти Internet, що пов'язані з прототипування мобільного застосунку.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці):
   1. Аналіз предметної області мобільних фінансових застосунків
   2. Проєктування та прототипування мобільного застосунку
   3. Розробка та реалізація мобільного застосунку
5. Перелік графічного матеріалу до 12 слайдів у форматі PowerPoint
6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 13.10.2025 р.

Календарний план

№	Найменування етапів виконання кваліфікаційної роботи	Терміни виконання етапів	Примітка
	Отримання завдання і аналіз інформаційних джерел за темою роботи	до 11.12.2025	
	Аналіз предметної області та постановка задачі кваліфікаційної роботи	до 13.01.2026	
	Функціональні вимои до мобільного застосунку для відстеження особистих фінансів	до 07.02.2026	
	Проектування архітектури мобільного застосунку	до 01.04.2026	
	Тестування та аналіз результатів прототипування	до 22.04.2026	
	Оформлення кваліфікаційної роботи. Подання роботи на перевірку керівнику	до 16.05.2026	
	Доопрацювання кваліфікаційної роботи з врахуванням зауважень керівника. Подання роботи для перевірки на рівень запозичень	до 20.05.2026	
	Подання кваліфікаційної роботи керівнику для отримання відгуку	до 27.05.2026	
	Представлення кваліфікаційної роботи на зовнішнє рецензування. Подання кваліфікаційної роботи на кафедру комп'ютерних наук	до 5.06.2026	
	Попередній захист кваліфікаційної роботи	09.06.2026-10.06.2026	
	Захист кваліфікаційної роботи на засіданні екзаменаційної комісії	15.06.2026 – 19.06.2026 згідно графіку роботи ЕК	

Здобувач вищої освіти _____ Вацюк М.І.

Керівник роботи _____ Крутяк М.Б.

РЕФЕРАТ

Об'єктом дослідження є процес управління особистими фінансами за допомогою мобільних інформаційних систем.

Предметом дослідження є методи, технології та програмні засоби проєктування і розробки мобільних застосунків для обліку особистих фінансів.

Метою роботи є проєктування та розробка прототипу мобільного застосунку для відстеження особистих фінансів із використанням технологій Flutter та SQLite.

Для досягнення поставленої мети було проведено аналіз предметної області персонального фінансового менеджменту, досліджено сучасні мобільні застосунки для обліку фінансів, визначено функціональні та нефункціональні вимоги до програмного продукту, виконано проєктування структури даних, бази даних та користувацького інтерфейсу, а також реалізовано програмний прототип мобільного застосунку.

У процесі розробки використано фреймворк Flutter, мову програмування Dart, локальну базу даних SQLite, пакет Provider для керування станом застосунку та бібліотеку fl_chart для побудови статистичних графіків і діаграм.

Результатом роботи є мобільний застосунок, який забезпечує ведення обліку доходів і витрат, роботу з категоріями фінансових операцій, формування статистичних звітів та контроль особистого бюджету користувача.

Практична цінність роботи полягає у можливості використання розробленого програмного продукту як основи для створення повноцінної системи управління особистими фінансами з подальшим розширенням функціональних можливостей.

Ключові слова: мобільний застосунок, особисті фінанси, Flutter, Dart, SQLite, фінансовий менеджмент, бюджет, доходи, витрати, статистика.

ABSTRACT

The object of research is the process of personal finance management using mobile information systems.

The subject of research is the methods, technologies, and software tools for designing and developing mobile applications for personal finance accounting.

The aim of the thesis is to design and develop a prototype of a mobile application for personal finance tracking using Flutter and SQLite technologies.

To achieve this goal, an analysis of the personal financial management domain was conducted, modern mobile finance applications were studied, functional and non-functional requirements for the software product were defined, the data structure, database, and user interface were designed, and a software prototype of the mobile application was implemented.

During the development process, the Flutter framework, Dart programming language, SQLite local database, Provider package for state management, and the fl_chart library for creating statistical charts and diagrams were used.

The result of the work is a mobile application that provides income and expense tracking, management of financial operation categories, generation of statistical reports, and personal budget control.

The practical significance of the work lies in the possibility of using the developed software product as a foundation for creating a full-featured personal finance management system with further functional expansion.

Keywords: mobile application, personal finance, Flutter, Dart, SQLite, financial management, budget, income, expenses, statistics.

ЗМІСТ

ВСТУП 7

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОБІЛЬНИХ ФІНАНСОВИХ ЗАСТОСУНКІВ ..10

1.1 Поняття та роль персонального фінансового менеджменту 11

1.2 Аналіз сучасних мобільних застосунків для обліку фінансів 13

1.3 Функціональні вимоги до системи обліку особистих фінансів 16

1.4 Огляд технологій розробки мобільних застосунків 18

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА ПРОТОТИПУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ 23

2.1 Формування вимог до застосунку 24

2.2 Проєктування структури даних 27

2.3 UML-моделювання системи 29

2.4 Проєктування бази даних SQLite 32

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ 36

3.1 Архітектура Flutter-застосунку 37

3.2 Реалізація бази даних SQLite 41

3.3 Реалізація екрана доходів та витрат 47

3.4 Реалізація категорій фінансових операцій 53

3.5 Реалізація статистики та графіків 58

ВИСНОВКИ 65

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 67

ВСТУП

У сучасних умовах цифровізації суспільства мобільні технології стали невід'ємною частиною повсякденного життя людини. Смартфони використовуються не лише для комунікації, а й для вирішення широкого спектра практичних завдань, серед яких особливе місце займає управління особистими фінансами. Зростання кількості безготівкових платежів, онлайн-покупок та електронних фінансових сервісів сприяє збільшенню обсягів фінансових операцій, які користувачеві необхідно контролювати на регулярній основі.

Раціональне управління особистими коштами є важливою складовою фінансової грамотності. Ведення обліку доходів і витрат дозволяє аналізувати структуру витрат, планувати бюджет, контролювати фінансові цілі та приймати обґрунтовані фінансові рішення. Проте значна кількість користувачів не веде систематичного обліку власних коштів через складність існуючих програмних продуктів або відсутність зручних інструментів для щоденного використання.

Сучасний ринок мобільних застосунків пропонує значну кількість рішень для фінансового обліку, однак багато з них мають надлишкову функціональність, складний інтерфейс або потребують платної підписки. Це обумовлює актуальність розробки мобільних застосунків, які поєднують простоту використання, сучасний дизайн та необхідний набір функцій для контролю особистих фінансів.

Актуальність теми полягає у необхідності створення зручного мобільного застосунку для обліку особистих фінансів, який дозволяє користувачам ефективно контролювати доходи та витрати, аналізувати фінансову активність і планувати бюджет за допомогою сучасних мобільних технологій.

Об'єктом дослідження є процес управління особистими фінансами за допомогою мобільних інформаційних систем.

Предметом дослідження є методи, технології та програмні засоби проєктування і розробки мобільних застосунків для обліку особистих фінансів.

Метою роботи є проєктування та розробка прототипу мобільного застосунку для відстеження особистих фінансів із використанням технологій Flutter та SQLite.

Для досягнення поставленої мети необхідно вирішити такі **завдання**:

– дослідити сучасні підходи до управління особистими фінансами;

– проаналізувати існуючі мобільні застосунки для фінансового обліку;

– визначити функціональні та нефункціональні вимоги до програмного продукту;

– обґрунтувати вибір технологій розробки;

– спроєктувати структуру бази даних мобільного застосунку;

– розробити прототип інтерфейсу користувача в середовищі Figma;

– реалізувати мобільний застосунок засобами Flutter;

– реалізувати механізми збереження та обробки фінансових операцій за допомогою SQLite;

– виконати тестування програмного продукту та оцінити результати його роботи.

Під час виконання роботи використано методи аналізу наукової літератури, порівняльного аналізу програмних продуктів, проєктування інформаційних систем, моделювання баз даних, програмування та тестування програмного забезпечення.

Практичне значення роботи полягає у створенні мобільного застосунку, який дозволяє здійснювати облік доходів і витрат, формувати статистичні звіти та контролювати особистий бюджет користувача.

Робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної області та сучасних мобільних фінансових застосунків.

У другому розділі розглянуто процес проєктування та створення прототипу мобільного застосунку. У третьому – описано реалізацію програмного продукту, структуру бази даних і результати тестування.

Загальний обсяг роботи 74 сторінки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОБІЛЬНИХ ФІНАНСОВИХ ЗАСТОСУНКІВ

Стрімкий розвиток інформаційних технологій та поширення мобільних пристроїв суттєво змінили підходи до управління особистими фінансами. Сучасні користувачі все частіше використовують мобільні застосунки для контролю власних доходів і витрат, планування бюджету та аналізу фінансової активності. Завдяки доступності смартфонів та розвитку мобільних платформ фінансові інструменти стали невід'ємною частиною повсякденного життя багатьох людей.

Одним із важливих напрямів розвитку мобільного програмного забезпечення є створення застосунків для обліку особистих фінансів. Такі системи дозволяють автоматизувати процес ведення фінансового обліку, зменшити ймовірність помилок під час розрахунків та забезпечити швидкий доступ до актуальної інформації про фінансовий стан користувача. Використання мобільних застосунків сприяє підвищенню фінансової грамотності населення та формуванню навичок ефективного управління коштами.

На сьогодні існує значна кількість програмних рішень для контролю особистих фінансів. Вони відрізняються набором функціональних можливостей, інтерфейсом користувача, методами зберігання даних та способами візуалізації статистичної інформації. Найпоширенішими функціями таких застосунків є ведення обліку доходів і витрат, формування фінансових звітів, планування бюджету, аналіз структури витрат та побудова графіків.

Особливу роль у розвитку мобільних фінансових систем відіграють сучасні засоби розробки програмного забезпечення. Використання кросплатформних технологій дозволяє створювати застосунки, які можуть працювати на різних мобільних операційних системах із мінімальними витратами часу та ресурсів. Однією з найбільш популярних платформ для розробки мобільних застосунків є Flutter, що забезпечує високу продуктивність,

сучасний інтерфейс користувача та можливість швидкої розробки програмних продуктів.

Під час створення мобільного застосунку для обліку особистих фінансів важливо враховувати не лише технічні аспекти реалізації, а й потреби кінцевих користувачів. Інтерфейс повинен бути інтуїтивно зрозумілим, а основні функції — доступними та зручними для щоденного використання. Крім того, значну увагу необхідно приділити питанням безпеки даних, продуктивності роботи застосунку та можливості подальшого розширення функціоналу.

У межах даного розділу буде проведено аналіз основних принципів персонального фінансового менеджменту, розглянуто сучасні мобільні застосунки для обліку фінансів, визначено функціональні вимоги до майбутньої системи та виконано огляд технологій, які можуть бути використані під час розробки мобільного застосунку для відстеження особистих фінансів.

1.1 Поняття та роль персонального фінансового менеджменту

У сучасному суспільстві ефективне управління особистими фінансами є важливою складовою фінансової грамотності людини. Постійне зростання кількості фінансових операцій, використання банківських карток, електронних платіжних систем та онлайн-сервісів створює необхідність систематичного контролю над власними доходами та витратами. Саме тому персональний фінансовий менеджмент набуває все більшого значення як інструмент планування та контролю фінансової діяльності.

Персональний фінансовий менеджмент являє собою сукупність методів, інструментів та підходів, спрямованих на ефективне управління особистими фінансовими ресурсами. Основними завданнями такого управління є облік доходів і витрат, формування бюджету, планування заощаджень, контроль фінансових цілей та аналіз фінансового стану користувача.

Застосування принципів персонального фінансового менеджменту дозволяє підвищити рівень контролю над власними коштами та забезпечує

можливість прийняття обґрунтованих фінансових рішень. Регулярний аналіз структури витрат дає змогу виявляти нераціональні витрати, оптимізувати бюджет та підвищувати ефективність використання фінансових ресурсів.

Сучасні інформаційні технології значно спростили процес ведення особистого фінансового обліку. Якщо раніше для цього використовувалися паперові записи або електронні таблиці, то сьогодні більшість користувачів надає перевагу спеціалізованим мобільним застосункам. Такі рішення забезпечують швидке внесення фінансових операцій, автоматичний розрахунок показників та візуалізацію статистичних даних.

Основними перевагами використання мобільних застосунків для фінансового менеджменту є мобільність, зручність використання, швидкий доступ до даних та можливість автоматизованого аналізу фінансової інформації. Завдяки цьому користувач може в будь-який момент отримати актуальну інформацію про стан свого бюджету, структуру витрат та рівень накопичень.

Персональний фінансовий менеджмент також сприяє формуванню відповідального ставлення до власних фінансів. Використання сучасних цифрових інструментів дозволяє не лише контролювати поточні витрати, але й прогнозувати майбутні фінансові потреби, встановлювати фінансові цілі та відстежувати їх досягнення.

В умовах розвитку мобільних технологій особливого значення набувають мобільні застосунки для управління фінансами, які поєднують функції обліку, аналізу та планування в єдиному програмному продукті. Саме такі рішення є найбільш перспективним напрямом розвитку систем персонального фінансового менеджменту.



Рис. 1.1 – Основні складові персонального фінансового менеджменту

До основних складових персонального фінансового менеджменту належать облік доходів, облік витрат, планування бюджету, фінансовий аналіз та контроль досягнення фінансових цілей. Взаємозв'язок цих елементів забезпечує комплексний підхід до управління особистими фінансами та підвищує ефективність прийняття фінансових рішень.

Таким чином, персональний фінансовий менеджмент виступає важливим інструментом управління особистими фінансами, а використання сучасних мобільних застосунків дозволяє значно спростити процес обліку та аналізу фінансової інформації.

1.2 Аналіз сучасних мобільних застосунків для обліку фінансів

Сучасний ринок мобільних застосунків пропонує значну кількість програмних продуктів для ведення особистого фінансового обліку. Їх основною метою є допомога користувачам у контролі доходів і витрат, плануванні

бюджету, аналізі фінансової активності та формуванні заощаджень. Використання таких застосунків дозволяє автоматизувати процес ведення фінансової статистики та забезпечує швидкий доступ до актуальної інформації про стан особистих фінансів.

Одними з найбільш популярних застосунків для обліку фінансів є Wallet, Monefy, CoinKeeper та Money Manager. Кожен із них має власні особливості, переваги та недоліки, що впливають на зручність використання та функціональні можливості системи.

Застосунок Wallet є одним із найвідоміших рішень для управління особистими фінансами. Він надає користувачам можливість вести облік доходів та витрат, планувати бюджет, створювати фінансові цілі та формувати детальні звіти. Перевагою системи є підтримка синхронізації між різними пристроями та можливість інтеграції з банківськими сервісами. Разом із тим частина функціоналу доступна лише у платній версії застосунку.

Monefy орієнтований на швидке внесення фінансових операцій та простоту використання. Інтерфейс застосунку побудований таким чином, щоб користувач міг додати нову транзакцію за декілька секунд. Система підтримує категоризацію доходів і витрат, а також візуалізацію статистики у вигляді діаграм. Недоліком є обмежений набір аналітичних інструментів порівняно з більш функціональними рішеннями.

CoinKeeper відрізняється оригінальним підходом до організації інтерфейсу. Основні фінансові категорії представлені у вигляді окремих елементів, між якими користувач може переміщувати кошти. Такий підхід робить процес ведення фінансового обліку більш наочним, однак може потребувати додаткового часу для освоєння інтерфейсу новими користувачами.

Застосунок Money Manager забезпечує широкий набір функцій для ведення фінансового обліку. Система підтримує формування звітів, аналіз фінансової активності, побудову графіків та експорт даних. Перевагою рішення є детальна статистика та можливість роботи з великим обсягом фінансових записів.

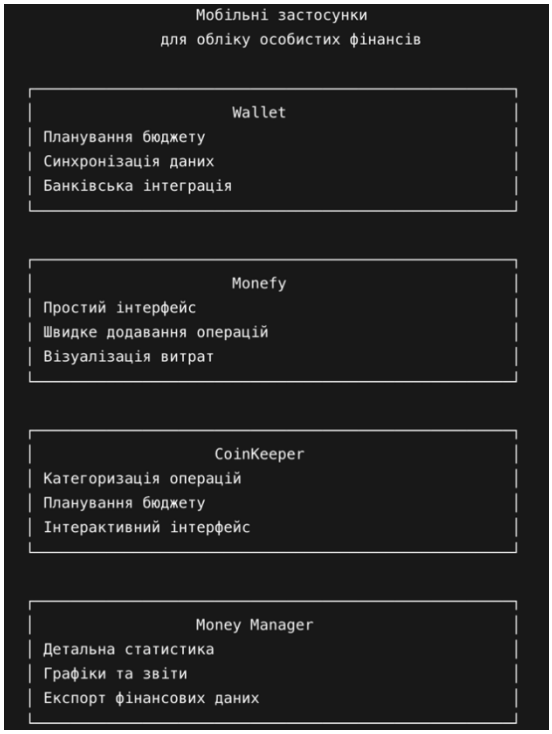

Рис 1.2 – Популярні мобільні застосунки для обліку фінансів

Як показано на рисунку 1.2, сучасні мобільні застосунки для управління особистими фінансами мають схожий базовий функціонал, який включає ведення обліку доходів і витрат, формування статистичних звітів та контроль бюджету. Водночас кожен із розглянутих програмних продуктів має власні особливості та додаткові можливості, орієнтовані на різні категорії користувачів. Результати аналізу існуючих рішень були враховані під час формування функціональних вимог до розроблюваного мобільного застосунку.

Порівняльний аналіз функціональних можливостей популярних застосунків наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння сучасних фінансових застосунків

Функція	**Wallet**	**Monefy**	**CoinKeeper**	**Money Manager**
Облік доходів та витрат	+	+	+	+
Планування бюджету	+	-	+	+
Статистика та графіки	+	+	+	+
Хмарна синхронізація	+	+	+	-
Банківська інтеграція	+	-	-	-
Експорт даних	+	-	+	+

Проведений аналіз показав, що існуючі програмні продукти мають широкий набір функціональних можливостей, проте часто характеризуються складним інтерфейсом, наявністю платних функцій або надлишковою кількістю налаштувань. Це може ускладнювати використання застосунків користувачами, які потребують простого та зрозумілого інструменту для щоденного ведення особистого фінансового обліку.

На основі проведеного дослідження можна зробити висновок, що майбутній мобільний застосунок повинен поєднувати простоту використання, сучасний дизайн та достатній набір функцій для контролю особистих фінансів. Особливу увагу доцільно приділити швидкому внесенню операцій, візуалізації статистики та зручній навігації між основними розділами системи.

1.3 Функціональні вимоги до системи обліку особистих фінансів

Одним із найважливіших етапів розробки будь-якого програмного продукту є визначення функціональних вимог. Саме вони формують перелік можливостей системи та визначають її поведінку під час взаємодії з користувачем. Для мобільного застосунку обліку особистих фінансів функціональні вимоги повинні забезпечувати зручне ведення фінансового обліку, аналіз фінансової активності та підтримку процесу планування бюджету.

На основі аналізу існуючих програмних продуктів та потреб потенційних користувачів було визначено перелік основних функцій, які повинні бути реалізовані в розроблюваному мобільному застосунку.

Першочерговою вимогою є можливість реєстрації та авторизації користувача. Дана функція забезпечує персоналізацію даних та дозволяє кожному користувачеві працювати зі своєю фінансовою інформацією. Авторизація також створює передумови для подальшого розширення функціоналу та інтеграції із зовнішніми сервісами.

Наступною важливою вимогою є ведення обліку доходів та витрат. Користувач повинен мати можливість швидко додавати нові фінансові операції

із зазначенням суми, категорії, дати та короткого опису. Усі записи повинні зберігатися в базі даних та бути доступними для подальшого аналізу.

Система повинна підтримувати роботу з категоріями витрат і доходів. Це дозволяє структурувати фінансову інформацію та отримувати детальну статистику щодо використання коштів у різних сферах життя.

Важливою функцією є формування статистичних звітів та візуалізація фінансових даних. Користувач повинен мати можливість переглядати інформацію про структуру витрат, співвідношення доходів і витрат, а також аналізувати фінансову активність за певний період часу.

Окрему увагу необхідно приділити функції планування бюджету. Система повинна дозволяти встановлювати фінансові обмеження на певний період та контролювати їх виконання. Такий підхід сприяє більш ефективному управлінню особистими фінансами та формуванню навичок фінансового планування.

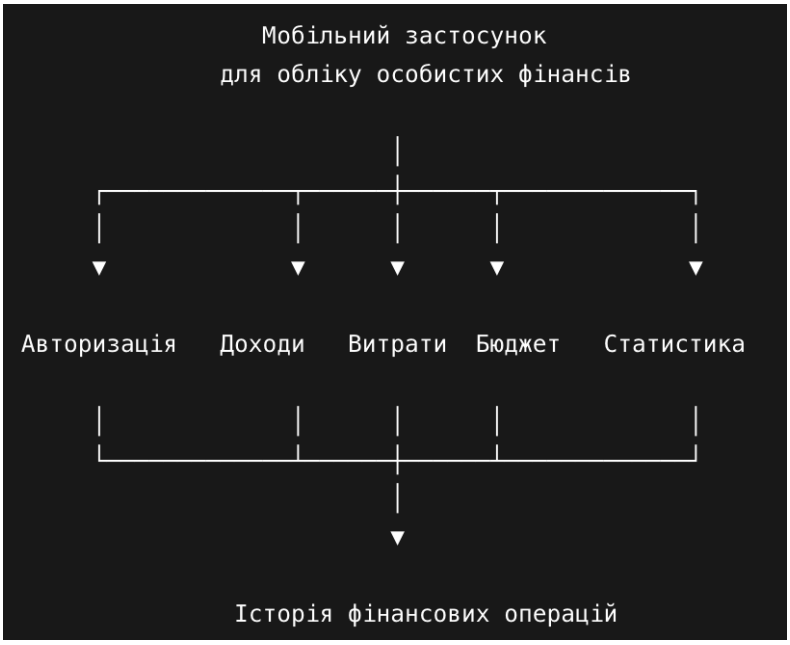

Рис. 1.3 – Основні функціональні можливості мобільного застосунку

Основні функціональні вимоги до системи наведені в таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги до мобільного застосунку

№	Функціональна вимога	Опис
1	Реєстрація користувача	Створення облікового запису
2	Авторизація	Вхід до системи
3	Додавання доходів	Внесення фінансових надходжень
4	Додавання витрат	Внесення фінансових витрат
5	Керування категоріями	Створення та редагування категорій
6	Перегляд історії операцій	Аналіз попередніх транзакцій
7	Статистика	Побудова графіків та діаграм
8	Планування бюджету	Контроль фінансових лімітів
9	Пошук операцій	Фільтрація фінансових записів
10	Зберігання даних	Робота з локальною базою SQLite

Крім функціональних вимог, важливу роль відіграють нефункціональні вимоги. До них належать швидкодія застосунку, зручність інтерфейсу, безпека збереження даних, стабільність роботи та можливість подальшого масштабування системи. Застосунок повинен забезпечувати швидке завантаження екранів, коректну роботу з базою даних та інтуїтивно зрозумілий інтерфейс користувача.

Таким чином, сформований перелік функціональних та нефункціональних вимог є основою для подальшого проєктування архітектури мобільного застосунку та реалізації його програмної частини.

1.4 Огляд технологій розробки мобільних застосунків

Вибір технологій розробки є одним із ключових етапів створення сучасного програмного забезпечення. Від обраних інструментів залежить продуктивність застосунку, швидкість розробки, зручність підтримки та можливість подальшого масштабування системи. У сфері мобільної розробки сьогодні існує значна кількість технологій, які дозволяють створювати програмні продукти для платформ Android та iOS.

Умовно всі підходи до розробки мобільних застосунків можна поділити на нативні та кросплатформні. Нативна розробка передбачає створення окремих

програм для кожної операційної системи із використанням спеціалізованих мов програмування та інструментів. Для Android найчастіше використовуються Java або Kotlin, а для iOS — Swift.

Перевагою нативної розробки є максимальна продуктивність застосунків та повний доступ до можливостей мобільної платформи. Проте такий підхід потребує створення окремих програмних продуктів для різних операційних систем, що збільшує витрати часу та ресурсів на розробку.

Альтернативою є кросплатформні технології, які дозволяють створювати один програмний код для декількох платформ одночасно. Найбільш популярними рішеннями є Flutter, React Native та Xamarin. Використання таких інструментів значно скорочує тривалість розробки та спрощує подальшу підтримку програмного продукту.

Особливу популярність останніми роками отримала платформа Flutter, розроблена компанією Google. Flutter використовує мову програмування Dart та надає широкий набір компонентів для створення сучасних мобільних інтерфейсів. Однією з головних переваг Flutter є можливість створення високопродуктивних застосунків із єдиною кодовою базою для Android та iOS.

Для зберігання даних у мобільних застосунках широко використовуються локальні бази даних. Одним із найбільш поширених рішень є SQLite — компактна вбудована система керування базами даних, яка не потребує встановлення окремого серверного програмного забезпечення. SQLite забезпечує швидке збереження та отримання інформації, що робить її оптимальним вибором для застосунків персонального фінансового обліку.

Для керування станом застосунку в Flutter використовуються різні підходи, серед яких найбільш популярними є Provider, Riverpod та Bloc. У межах даної роботи було обрано Provider завдяки простоті реалізації, хорошій продуктивності та широкому поширенню серед розробників Flutter-застосунків.

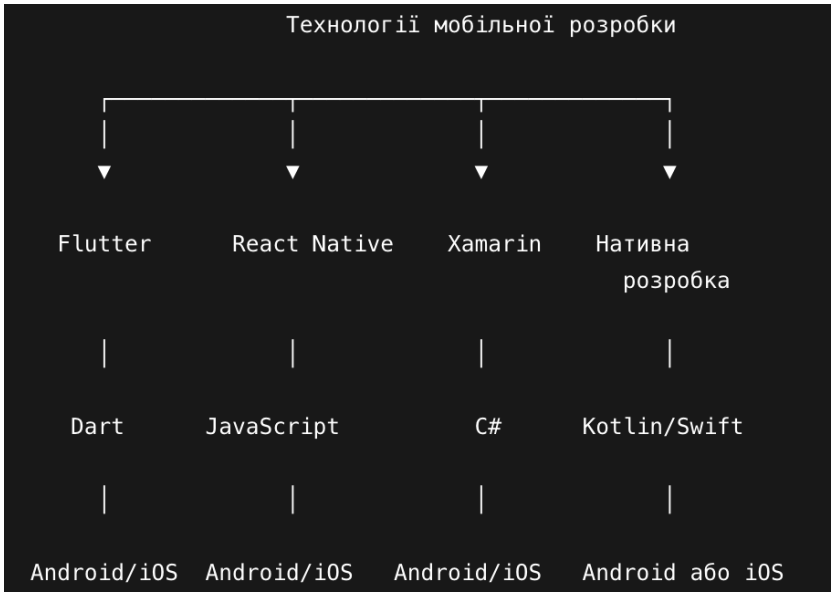

Рис. 1.4 – Порівняння технологій розробки мобільних застосунків

Для порівняння основних технологій мобільної розробки складено таблицю 1.3.

Таблиця 1.3 – Порівняння технологій мобільної розробки

Технологія	**Мова програмування**	**Платформи**	**Переваги**
Flutter	Dart	Android, iOS	Висока продуктивність, єдина кодова база
React Native	JavaScript	Android, iOS	Швидка розробка, велика спільнота
Xamarin	C#	Android, iOS	Інтеграція з .NET
Kotlin	Kotlin	Android	Нативна продуктивність
Swift	Swift	iOS	Повна підтримка можливостей iOS

Проведений аналіз показав, що для реалізації мобільного застосунку відстеження особистих фінансів найбільш доцільним є використання Flutter у поєднанні з SQLite. Такий підхід дозволяє створити сучасний кросплатформний програмний продукт із високою продуктивністю, зручним інтерфейсом користувача та можливістю локального зберігання фінансових даних.

Таким чином, результати аналізу сучасних технологій мобільної розробки стали основою для вибору програмних засобів, які будуть використані під час проєктування та реалізації мобільного застосунку для обліку особистих фінансів.

У першому розділі було проведено аналіз предметної області мобільних застосунків для управління особистими фінансами. Встановлено, що в умовах

активного розвитку цифрових технологій та поширення мобільних пристроїв зростає потреба у використанні програмних засобів для контролю доходів, витрат та планування особистого бюджету.

Розглянуто сутність персонального фінансового менеджменту та визначено його основні завдання, серед яких облік фінансових операцій, аналіз структури витрат, формування бюджету та контроль фінансових цілей. Встановлено, що використання спеціалізованих мобільних застосунків значно спрощує процес управління особистими фінансами та підвищує рівень фінансової грамотності користувачів.

У процесі дослідження було виконано аналіз сучасних мобільних застосунків для обліку фінансів, зокрема Wallet, Monefy, CoinKeeper та Money Manager. Проведене порівняння дозволило визначити їхні переваги, недоліки та найбільш затребувані функціональні можливості. Встановлено, що основними функціями сучасних фінансових систем є ведення обліку доходів і витрат, планування бюджету, побудова статистичних звітів та візуалізація фінансових показників.

На основі аналізу існуючих програмних продуктів сформовано функціональні та нефункціональні вимоги до майбутнього мобільного застосунку. Визначено необхідність реалізації механізмів реєстрації та авторизації користувачів, обліку фінансових операцій, роботи з категоріями доходів і витрат, формування статистики та контролю бюджету.

Також було проведено огляд сучасних технологій мобільної розробки та виконано їх порівняльний аналіз. За результатами дослідження встановлено, що найбільш доцільним рішенням для реалізації проєкту є використання фреймворку Flutter, мови програмування Dart та локальної бази даних SQLite. Обраний технологічний стек забезпечує високу продуктивність застосунку, можливість кросплатформної розробки та зручність подальшої підтримки програмного забезпечення.

Отримані результати аналізу предметної області та сформовані вимоги стали основою для проєктування структури системи, бази даних та

користувацького інтерфейсу мобільного застосунку, що буде розглянуто у наступному розділі.

2. ПРОЄКТУВАННЯ ТА ПРОТОТИПУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

Проєктування є одним із найважливіших етапів розробки програмного забезпечення, оскільки саме на цьому етапі визначаються структура майбутньої системи, її функціональні можливості, архітектура та особливості взаємодії користувача із програмним продуктом. Якісно виконане проєктування дозволяє мінімізувати кількість помилок під час реалізації програмного забезпечення, забезпечити логічну організацію даних та створити зручний інтерфейс користувача.

Для розробки мобільного застосунку відстеження особистих фінансів необхідно виконати комплексне проєктування системи, яке включає визначення функціональних вимог, моделювання структури даних, проєктування бази даних, створення UML-діаграм та розробку прототипу користувацького інтерфейсу. Отримані результати дозволяють сформувати цілісне уявлення про майбутній програмний продукт ще до початку програмної реалізації.

Особливу увагу під час проєктування мобільного застосунку приділено зручності використання та простоті взаємодії користувача із системою. Оскільки фінансовий облік передбачає регулярне внесення інформації про доходи та витрати, інтерфейс повинен забезпечувати швидкий доступ до основних функцій і мінімізувати кількість дій, необхідних для виконання типових операцій.

Одним із ключових завдань проєктування є визначення структури даних, яка буде використовуватися для зберігання інформації про користувачів, фінансові операції, категорії доходів та витрат, а також бюджетні обмеження. Раціонально спроєктована база даних забезпечує ефективне збереження інформації та швидке виконання операцій пошуку, сортування й формування статистичних звітів.

Важливою складовою етапу проєктування є створення UML-моделей, які дозволяють візуалізувати функціональну структуру системи, взаємодію користувачів із програмним продуктом та зв'язки між основними об'єктами

предметної області. Використання UML-діаграм спрощує процес аналізу архітектури системи та дозволяє виявити можливі недоліки ще до початку програмування.

Для розробки інтерфейсу користувача використовується середовище Figma, яке дозволяє створювати інтерактивні прототипи мобільних застосунків. Прототипування дає можливість оцінити зручність навігації, логіку розташування елементів керування та загальний зовнішній вигляд програмного продукту до початку його програмної реалізації.

У межах даного розділу буде виконано формування функціональних вимог до системи, розроблено UML-моделі, спроєктовано структуру бази даних, створено прототип мобільного застосунку та визначено архітектуру програмного забезпечення, яка буде використана під час реалізації програмного продукту засобами Flutter.

2.1 Формування вимог до застосунку

Формування вимог є одним із найважливіших етапів проєктування програмного забезпечення, оскільки саме на його основі визначаються функціональні можливості майбутньої системи та особливості її реалізації. Від правильності визначення вимог залежить ефективність роботи застосунку, рівень задоволення потреб користувачів та якість кінцевого програмного продукту.

Мобільний застосунок для відстеження особистих фінансів призначений для автоматизації процесів обліку доходів і витрат, аналізу фінансової активності користувача та контролю виконання бюджетних обмежень. Основною цільовою аудиторією системи є користувачі мобільних пристроїв, які бажають вести облік власних фінансів за допомогою сучасних цифрових інструментів.

Під час визначення вимог було враховано результати аналізу існуючих фінансових застосунків, проведеного у першому розділі роботи. Це дозволило

виділити функції, які є найбільш затребуваними серед користувачів та повинні бути реалізовані в розроблюваному програмному продукті.

До основних функціональних вимог належить можливість реєстрації та авторизації користувачів. Наявність облікового запису забезпечує персоналізацію даних та створює умови для захисту фінансової інформації користувача.

Важливою функцією застосунку є ведення обліку доходів та витрат. Користувач повинен мати можливість додавати нові фінансові операції, зазначаючи суму, категорію, дату та опис транзакції. Вся інформація повинна автоматично зберігатися в локальній базі даних та бути доступною для подальшого перегляду.

Система повинна підтримувати роботу з категоріями доходів і витрат. Це дозволить користувачеві групувати фінансові операції за певними ознаками та отримувати детальну статистику щодо структури власних витрат.

Окремою функціональною вимогою є формування статистичної інформації. Застосунок повинен автоматично розраховувати сумарні доходи, витрати та поточний баланс користувача. Для зручності сприйняття інформація повинна відображатися у вигляді графіків і діаграм.

Крім цього, система повинна підтримувати механізм планування бюджету. Користувач матиме можливість встановлювати фінансові ліміти на певний період часу та контролювати рівень їх виконання.



Рис 2.1 – Основні функціональні вимоги до мобільного застосунку

Для систематизації вимог сформовано таблицю функціональних можливостей майбутнього застосунку.

Таблиця 2.1 – Основні функціональні вимоги

№	Вимога	Призначення
1	Реєстрація користувача	Створення облікового запису
2	Авторизація	Вхід до системи
3	Додавання доходів	Внесення інформації про надходження коштів
4	Додавання витрат	Внесення інформації про витрати
5	Керування категоріями	Створення та редагування категорій
6	Перегляд історії операцій	Аналіз попередніх транзакцій
7	Формування статистики	Відображення фінансових показників
8	Планування бюджету	Контроль витрат користувача
9	Пошук транзакцій	Фільтрація записів
10	Збереження даних	Робота з базою SQLite

Окрім функціональних вимог, система повинна відповідати низці нефункціональних вимог. До них належать висока швидкодія, надійність роботи, простота інтерфейсу, безпечне зберігання даних та можливість подальшого масштабування програмного забезпечення.

Інтерфейс застосунку повинен бути інтуїтивно зрозумілим для користувача та забезпечувати швидке виконання основних операцій. Час відгуку системи на дії користувача повинен бути мінімальним, а навігація між екранами — логічною та послідовною.

Таким чином, сформований набір функціональних і нефункціональних вимог визначає основу для подальшого проєктування архітектури системи, структури бази даних та користувацького інтерфейсу мобільного застосунку.

2.2 Проєктування структури даних

Ефективна організація даних є важливою складовою розробки мобільного застосунку для відстеження особистих фінансів. Від структури даних залежить швидкість роботи системи, коректність збереження інформації та можливість подальшого розширення функціоналу. На етапі проєктування необхідно визначити основні сутності предметної області, їхні атрибути та взаємозв'язки.

Для реалізації мобільного застосунку було виділено чотири основні сутності: користувач, фінансова операція, категорія та бюджет. Саме ці об'єкти забезпечують виконання всіх основних функцій системи та дозволяють організувати зберігання інформації про фінансову діяльність користувача.

Сутність «Користувач» призначена для зберігання інформації про зареєстрованих користувачів системи. Для кожного користувача зберігається унікальний ідентифікатор, ім'я, електронна адреса та пароль.

Сутність «Транзакція» використовується для обліку доходів і витрат. Кожна фінансова операція містить інформацію про суму, дату проведення, категорію та короткий опис. Саме ця таблиця є основним джерелом даних для формування статистики та аналітичних звітів.

Сутність «Категорія» використовується для групування фінансових операцій. Наявність категорій дозволяє користувачу отримувати більш детальну інформацію про структуру витрат та доходів.

Для контролю фінансових обмежень передбачено сутність «Бюджет». Вона містить інформацію про встановлені користувачем ліміти витрат та дозволяє контролювати рівень їх виконання протягом обраного періоду.

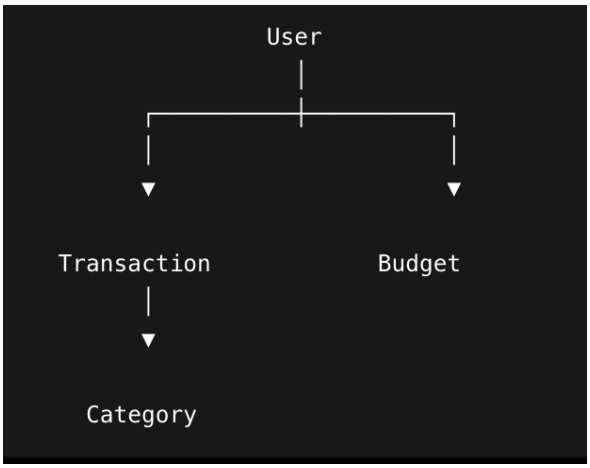

Рис 2.2 – Структура даних мобільного застосунку

Основні сутності системи наведено у таблиці 2.2.

Таблиця 2.2 – Основні сутності бази даних

Сутність	**Призначення**
User	Зберігання інформації про користувача
Transaction	Зберігання фінансових операцій
Category	Категорії доходів і витрат
Budget	Планування бюджету

Для забезпечення цілісності даних між сутностями встановлюються відповідні зв'язки. Один користувач може мати багато фінансових операцій та декілька бюджетів. Кожна транзакція належить до певної категорії, що забезпечує можливість подальшого аналізу фінансової інформації.

Логічна структура взаємозв'язків між сутностями представлена на рисунку 2.2. Такий підхід дозволяє уникнути дублювання даних, спростити процес їх обробки та забезпечити можливість масштабування системи в майбутньому.

Для реалізації структури даних у Flutter-застосунку передбачається використання окремих моделей, які відображають сутності предметної області. Кожна модель міститиме набір полів, конструкторів та методів для роботи з даними.

Приклад основних моделей системи:

class User {

```
  int? id;
  String name;
  String email;
  String password;
}
class Transaction {
  int? id;
  String title;
  double amount;
  String category;
  String type;
  String date;
}
class Budget {
  int? id;
  double limit;
  String month;
}
```

Завдяки такій структурі даних забезпечується зручне збереження інформації у базі SQLite та ефективна взаємодія між окремими модулями застосунку.

Отже, проєктування структури даних дозволило визначити основні сутності предметної області, встановити взаємозв'язки між ними та підготувати основу для подальшого проєктування бази даних і програмної реалізації мобільного застосунку.

2.3 UML-моделювання системи

Під час проєктування програмного забезпечення важливим етапом є створення UML-моделей, які дозволяють візуалізувати структуру системи, взаємодію користувачів із програмним продуктом та зв'язки між основними

компонентами. Використання UML забезпечує більш чітке розуміння архітектури майбутнього застосунку та спрощує процес його подальшої реалізації.

Для мобільного застосунку відстеження особистих фінансів було використано два основні типи UML-діаграм: діаграму варіантів використання (Use Case Diagram) та діаграму класів (Class Diagram). Перша діаграма дозволяє описати взаємодію користувача із системою, а друга — відображає структуру основних програмних об'єктів та їх взаємозв'язки.

Діаграма варіантів використання демонструє функціональні можливості системи та основні сценарії взаємодії користувача із застосунком. Основним учасником системи є користувач, який виконує всі операції з управління особистими фінансами.

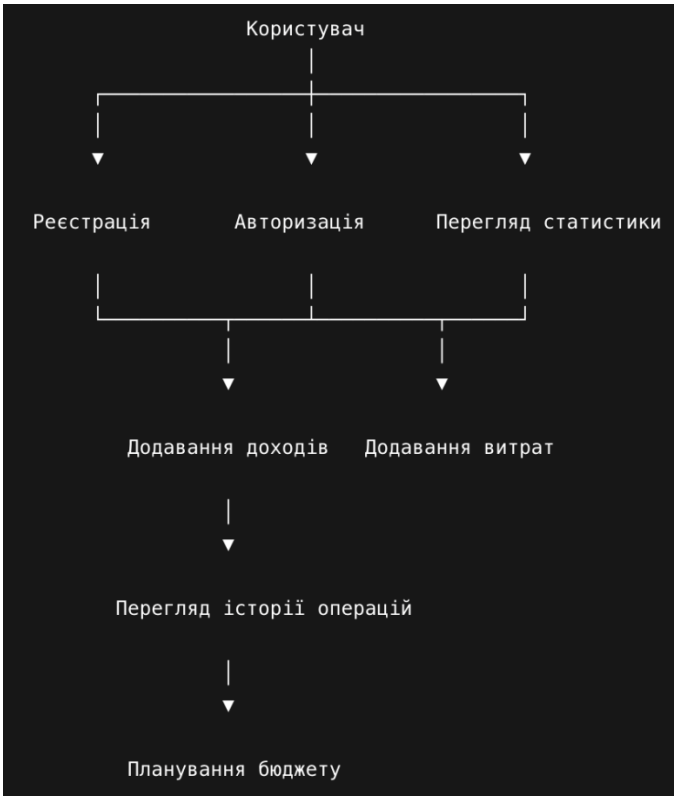

Рис 2.3 – UML Use Case Diagram

До основних сценаріїв використання системи належать:

– реєстрація користувача;

– авторизація в системі;

– додавання доходів;

– додавання витрат;

– перегляд історії фінансових операцій;

– формування статистичних звітів;

– планування бюджету;

– перегляд аналітичної інформації.

Застосування діаграми варіантів використання дозволяє визначити межі системи та сформувати перелік функцій, які повинні бути реалізовані в програмному продукті.

Наступним етапом UML-моделювання є створення діаграми класів. Дана діаграма відображає структуру програмного забезпечення та взаємозв'язки між основними сутностями предметної області.

Основними класами системи є User, Transaction, Category та Budget. Клас User містить інформацію про користувача системи та забезпечує роботу механізмів авторизації. Клас Transaction використовується для зберігання інформації про фінансові операції. Клас Category забезпечує групування доходів і витрат, а клас Budget використовується для реалізації механізму контролю бюджету.

Для кращого розуміння структури класів основні атрибути наведено в таблиці 2.3.

Таблиця 2.3 – Основні класи системи

Клас	**Основні атрибути**
User	id, name, email, password
Transaction	id, title, amount, date, type
Category	id, name, icon
Budget	id, limit, month

У програмній реалізації кожен клас буде представлений окремою моделлю Dart. Такий підхід забезпечує модульність програмного коду, спрощує підтримку застосунку та дозволяє легко масштабувати систему шляхом додавання нових функцій.

Використання UML-моделювання дозволило сформувати цілісне уявлення про структуру майбутнього програмного продукту та підготувати основу для проєктування бази даних і розробки користувацького інтерфейсу.

Таким чином, побудовані UML-діаграми відображають основні сценарії використання системи та її внутрішню структуру, що є необхідною умовою для якісної реалізації мобільного застосунку відстеження особистих фінансів.

2.4 Проєктування бази даних SQLite

Для забезпечення збереження інформації про користувачів та їх фінансові операції у мобільному застосунку необхідно спроєктувати ефективну структуру бази даних. Враховуючи специфіку роботи програмного продукту та необхідність локального зберігання інформації, для реалізації системи було обрано SQLite. Дана система керування базами даних є вбудованою, не потребує встановлення додаткового серверного програмного забезпечення та забезпечує достатню продуктивність для мобільних застосунків.

Основним завданням бази даних є збереження інформації про користувачів, доходи, витрати, категорії фінансових операцій та бюджетні обмеження. Структура бази даних повинна забезпечувати швидкий доступ до інформації, підтримувати цілісність даних та дозволяти виконувати аналітичні операції над фінансовими записами.

У межах проєкту було спроєктовано чотири основні таблиці: Users, Transactions, Categories та Budgets. Кожна таблиця відповідає окремій сутності предметної області та містить необхідний набір атрибутів для збереження інформації.

Таблиця Users використовується для зберігання облікових записів користувачів. Вона містить унікальний ідентифікатор користувача, ім'я, електронну адресу та пароль.

Таблиця Transactions є центральним елементом бази даних і призначена для збереження інформації про всі фінансові операції користувача. Для кожної

транзакції зберігаються сума, дата проведення операції, категорія та тип операції.

Таблиця Categories використовується для зберігання категорій доходів та витрат. Використання окремої таблиці дозволяє уникнути дублювання даних та спрощує формування статистичних звітів.

Таблиця Budgets містить інформацію про встановлені бюджетні обмеження та використовується для контролю фінансових витрат користувача.



Рис 2.5 – Структура бази даних SQLite

Основні таблиці бази даних наведено у таблиці 2.4.

Таблиця 2.4 – Структура таблиць бази даних

Таблиця	**Призначення**
Users	Зберігання інформації про користувачів
Transactions	Зберігання доходів та витрат
Categories	Категорії фінансових операцій
Budgets	Контроль бюджетних обмежень

Для реалізації бази даних у Flutter використовується пакет sqflite, який забезпечує взаємодію застосунку з SQLite за допомогою SQL-запитів.

Створення таблиці користувачів виконується наступним запитом:

```
CREATE TABLE users(
 id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
 name TEXT,
 email TEXT,
 password TEXT
);
```

Таблиця фінансових операцій створюється за допомогою такого SQL-запиту:

```
CREATE TABLE transactions(
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 title TEXT,
 amount REAL,
 type TEXT,
 category TEXT,
 date TEXT
);
```

Для збереження категорій використовується окрема таблиця:

```
CREATE TABLE categories(
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 name TEXT,
 icon TEXT
);
```

Структура таблиці бюджетів має вигляд:

```
CREATE TABLE budgets(
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 amount REAL,
 month TEXT
);
```

Використання такої структури бази даних забезпечує логічне розділення інформації та дозволяє ефективно виконувати операції додавання, редагування, видалення та пошуку даних.

Під час проєктування бази даних також враховано можливість подальшого розширення функціональності застосунку. У майбутньому структура може бути доповнена таблицями для зберігання фінансових цілей, нагадувань, валют та історії синхронізації даних.

Таким чином, спроєктована база даних SQLite забезпечує зручне та надійне зберігання фінансової інформації користувача і створює основу для реалізації основних функцій мобільного застосунку відстеження особистих фінансів.

РОЗДІЛ 3. РОЗРОБКА ТА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ЗАСТОСУНКУ

Після завершення етапів аналізу предметної області та проєктування системи було розпочато реалізацію мобільного застосунку для відстеження особистих фінансів. Основною метою даного етапу є створення програмного продукту, який забезпечує ведення обліку доходів і витрат, формування статистичної інформації та контроль особистого бюджету користувача.

Для розробки застосунку було обрано фреймворк Flutter та мову програмування Dart. Використання Flutter дозволяє створювати кросплатформні мобільні застосунки з єдиною кодовою базою для операційних систем Android та iOS. Додатковою перевагою є наявність великої кількості готових компонентів інтерфейсу та можливість швидкого тестування програмного забезпечення.

Для зберігання фінансової інформації використовується локальна база даних SQLite. Доступ до бази даних реалізовано за допомогою пакета sqflite, який забезпечує виконання SQL-запитів безпосередньо із програмного коду. Такий підхід дозволяє зберігати дані локально на пристрої користувача та забезпечує автономну роботу застосунку без необхідності підключення до мережі Інтернет.

Під час розробки програмного забезпечення використано архітектурний підхід, який передбачає поділ системи на окремі модулі. Основними складовими програмного продукту є модуль користувацького інтерфейсу, модуль роботи з базою даних, модуль обробки фінансових операцій та модуль формування статистики.

Важливим завданням під час реалізації стало забезпечення зручної взаємодії користувача із системою. Для цього було створено інтуїтивно зрозумілий інтерфейс, який дозволяє швидко додавати фінансові операції, переглядати історію транзакцій та аналізувати статистичні показники.

У межах даного розділу буде розглянуто архітектуру мобільного застосунку, реалізацію бази даних SQLite, механізми обробки фінансових

операцій, побудову статистичних звітів, реалізацію інтерфейсу користувача та результати тестування програмного забезпечення.

3.1 Архітектура Flutter-застосунку

Архітектура програмного забезпечення визначає структуру застосунку, взаємодію між його компонентами та способи обробки даних. Правильно спроєктована архітектура забезпечує зручність підтримки програмного коду, можливість масштабування функціоналу та підвищує загальну якість програмного продукту.

Для реалізації мобільного застосунку відстеження особистих фінансів було використано архітектурний підхід, який передбачає поділ програмного забезпечення на окремі логічні модулі. Така структура дозволяє розмежувати функціональність системи та забезпечує незалежність окремих компонентів.

Основними складовими архітектури є модулі інтерфейсу користувача, керування станом застосунку, роботи з базою даних та обробки фінансової інформації. Взаємодія між цими компонентами здійснюється через спеціальні моделі даних та провайдери стану.(рисунок 3.1.)

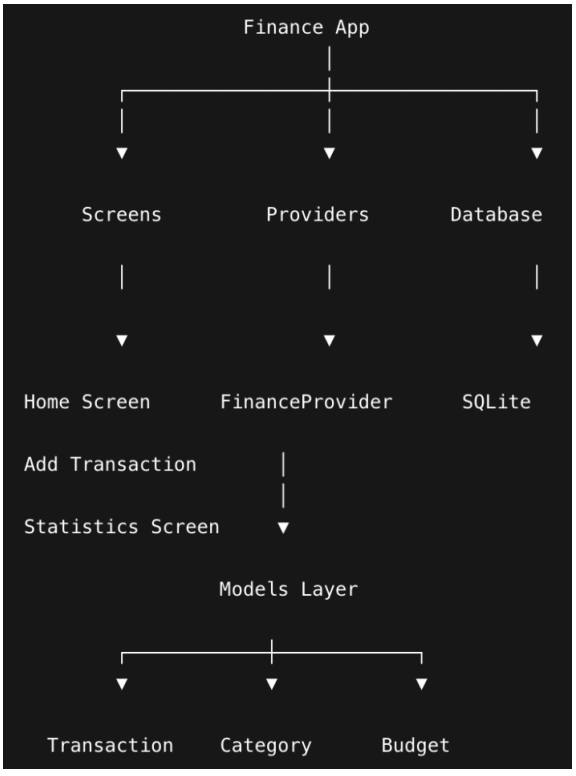

Рис 3.1 – Архітектура Flutter-застосунку

У процесі розробки було використано наступну структуру проєкту:

```
lib/
main.dart
models/
 transaction.dart
 category.dart
 budget.dart
 user.dart

database/
 database_helper.dart

providers/
 finance_provider.dart

screens/
 login_screen.dart
 home_screen.dart
 add_transaction_screen.dart
 statistics_screen.dart

widgets/
 transaction_card.dart
 chart_widget.dart
       custom_button.dart
```

Файл main.dart є точкою входу до програми та відповідає за запуск застосунку.

```
void main() {
 runApp(
   const FinanceApp()
 );}
```

Головний компонент застосунку реалізується через MaterialApp.

```
class FinanceApp extends StatelessWidget {
 const FinanceApp({super.key});
@override
 Widget build(BuildContext context) {
   return MaterialApp(
     debugShowCheckedModeBanner: false,
     title: 'Finance Tracker',
     home: LoginScreen(),
   );
 }
}
```

Для керування даними використовується пакет Provider. Він дозволяє централізовано зберігати стан застосунку та забезпечує доступ до фінансової інформації з будь-якого екрана.

Підключення провайдера виконується наступним чином:

```
runApp(
 ChangeNotifierProvider(
   create: (_) => FinanceProvider(),
   child: const FinanceApp(),
 ),
);
```

Основним класом для роботи з фінансовими операціями є FinanceProvider.

```
class FinanceProvider
 extends ChangeNotifier {

 List<Transaction> transactions = [];

 void addTransaction(
   Transaction transaction
```

```
 ) {
   transactions.add(transaction);
   notifyListeners();
 }
}
```

Після додавання нової транзакції автоматично викликається метод notifyListeners(), який оновлює всі компоненти інтерфейсу, що використовують дані провайдера.

Для представлення фінансових операцій використовується модель Transaction.

```
class Transaction {
 int? id;
 String title;
 double amount;
 String type;
 String category;
 String date;

 Transaction({
   this.id,
   required this.title,
   required this.amount,
   required this.type,
   required this.category,
   required this.date,
 });
}
```

Розроблена архітектура забезпечує чітке розмежування між інтерфейсом користувача, бізнес-логікою та механізмами збереження даних. Такий підхід

спрощує подальший розвиток застосунку та дозволяє легко додавати нові функціональні можливості без зміни існуючої структури програмного забезпечення.

Таким чином, використання Flutter, Provider та модульної структури проєкту дозволило створити гнучку архітектуру мобільного застосунку, яка відповідає сучасним вимогам розробки мобільного програмного забезпечення.

3.2 Реалізація бази даних SQLite

Для забезпечення збереження фінансової інформації в мобільному застосунку використовується локальна база даних SQLite. Дане рішення дозволяє працювати із застосунком без постійного підключення до мережі Інтернет, забезпечує високу швидкість доступу до даних та не потребує використання окремого серверного програмного забезпечення.

У середовищі Flutter для взаємодії з SQLite використовується пакет sqflite. Він надає набір інструментів для створення бази даних, виконання SQL-запитів та обробки отриманих результатів.

Для організації роботи з базою даних було створено окремий клас DatabaseHelper, який реалізує основні операції створення таблиць, додавання записів, отримання інформації та видалення даних.

Створення екземпляра бази даних виконується за допомогою патерна Singleton.

```
class DatabaseHelper {

 static final DatabaseHelper
 instance = DatabaseHelper._init();

 static Database? _database;

 DatabaseHelper._init();
```

```
}
```

Такий підхід забезпечує використання єдиного екземпляра бази даних протягом роботи всього застосунку.

Для отримання доступу до бази даних реалізовано метод database.

```
Future<Database> get database
async {

 if (_database != null)
 {
   return _database!;
 }

 _database =
 await _initDB("finance.db");

 return _database!;
}
```

Під час першого запуску застосунку автоматично створюється файл бази даних та необхідні таблиці.

```
Future<Database> _initDB(
 String filePath
) async {

 final dbPath =
 await getDatabasesPath();

 final path =
 join(dbPath, filePath);
```

```
 return await openDatabase(
   path,
   version: 1,
   onCreate: _createDB,
 );
}
```

Метод _createDB відповідає за створення структури бази даних.

```
Future _createDB(
 Database db,
 int version
) async {

 await db.execute('''
 CREATE TABLE transactions(
   id INTEGER PRIMARY KEY AUTOINCREMENT,
   title TEXT NOT NULL,
   amount REAL NOT NULL,
   category TEXT NOT NULL,
   type TEXT NOT NULL,
   date TEXT NOT NULL
 )
 ''');
}
```

Створена таблиця transactions використовується для збереження інформації про всі доходи та витрати користувача.

Для додавання нової фінансової операції використовується метод insertTransaction.

```
Future<int>
insertTransaction(
```

```
 Transaction transaction
) async {

 final db =
 await instance.database;

 return await db.insert(
   "transactions",
   transaction.toMap(),
 );
}
```

Після виклику цього методу інформація автоматично записується до локальної бази даних.

Для отримання списку всіх транзакцій реалізовано окремий метод.

```
Future<List<Transaction>>
getTransactions()
async {

 final db =
 await instance.database;

 final result =
 await db.query(
   "transactions",
   orderBy: "date DESC"
 );
```

```
 return result.map(
   (json) =>
   Transaction.fromMap(json)
 ).toList();
}
```

Отримані дані використовуються для формування історії фінансових операцій та побудови статистичних звітів.

Для оновлення інформації про транзакцію використовується SQL-операція UPDATE.

```
Future<int>
updateTransaction(
 Transaction transaction
) async {

 final db =
 await instance.database;

 return await db.update(
   "transactions",
   transaction.toMap(),
   where: "id = ?",
   whereArgs: [transaction.id],
 );
}
```

Видалення запису реалізовано за допомогою SQL-команди DELETE.

```
Future<int>
deleteTransaction(
```

```
 int id
) async {

 final db =
 await instance.database;

 return await db.delete(
   "transactions",
   where: "id = ?",
   whereArgs: [id],
 );
}
```

Для зручної роботи з базою даних у класі Transaction реалізовано методи перетворення об'єктів у формат Map та навпаки.

```
Map<String, dynamic>
toMap() {

 return {
   "id": id,
   "title": title,
   "amount": amount,
   "category": category,
   "type": type,
   "date": date,
 };
}
factory Transaction.fromMap(
```

```
 Map<String, dynamic> json
) {

 return Transaction(
   id: json["id"],
   title: json["title"],
   amount: json["amount"],
   category: json["category"],
   type: json["type"],
   date: json["date"],
 );
}
```

Використання SQLite дозволяє ефективно зберігати фінансову інформацію користувача, забезпечує швидке виконання операцій вибірки та не потребує додаткових серверних ресурсів. Завдяки локальному зберіганню даних застосунок може працювати автономно навіть за відсутності доступу до мережі Інтернет.

Таким чином, реалізована база даних SQLite забезпечує надійне збереження фінансових операцій та створює основу для функціонування всіх основних модулів мобільного застосунку.

3.3 Реалізація екрана доходів та витрат

Однією з основних функцій мобільного застосунку є можливість додавання та обліку фінансових операцій. Саме тому важливим етапом розробки стала реалізація екрана внесення доходів і витрат. Даний модуль забезпечує введення користувачем інформації про фінансові операції та подальше збереження даних у базі SQLite.

Основними вимогами до інтерфейсу екрана є простота використання, швидкість внесення даних та зручна навігація між елементами форми.

Користувач повинен мати можливість додати нову транзакцію за мінімальну кількість дій.

Для реалізації екрана використано StatefulWidget, що дозволяє керувати станом форми та оновлювати інтерфейс під час введення даних.

Створення екрана додавання транзакції виконується наступним чином:

```
class AddTransactionScreen
extends StatefulWidget {

 const AddTransactionScreen({
  super.key
 });

 @override
 State<AddTransactionScreen>
 createState() =>
 _AddTransactionScreenState();
}
```

Для зберігання введених даних використовуються контролери текстових полів.

```
final titleController =
TextEditingController();

final amountController =
TextEditingController();
```

Контролери дозволяють отримувати значення, введені користувачем у відповідні поля форми.

Для введення назви фінансової операції використовується компонент TextField.

```
TextField(
 controller: titleController,
 decoration: InputDecoration(
  labelText:
  "Назва операції",
 ),
)
```

Введення суми виконується через окреме текстове поле з числовою клавіатурою.

```
TextField(
 controller: amountController,
 keyboardType:
 TextInputType.number,
 decoration:
 InputDecoration(
  labelText: "Сума",
 ),
)
```

Для вибору категорії реалізовано випадаючий список DropdownButton.

```
DropdownButton<String>(
 value: selectedCategory,
 items: categories.map(
 (category) {
  return DropdownMenuItem(
   value: category,
   child: Text(category),
  );
```

```
 }).toList(),
 onChanged: (value) {
  setState(() {
   selectedCategory = value!;
  });
 },
)
```

Користувач може вибрати одну із заздалегідь створених категорій доходів або витрат.

Після заповнення всіх полів користувач натискає кнопку збереження.

```
ElevatedButton(
 onPressed: saveTransaction,
 child: Text("Зберегти"),
)
```

Метод saveTransaction відповідає за створення нового об'єкта транзакції та його збереження в базі даних.

```
Future<void>
saveTransaction() async {

 final transaction =
 Transaction(
  title:
  titleController.text,

  amount:
  double.parse(
   amountController.text
```

```
  ),

  category:
  selectedCategory,

  type:
  selectedType,

  date:
  DateTime.now()
  .toString(),
 );

 await DatabaseHelper
 .instance
 .insertTransaction(
  transaction
 );

 Navigator.pop(context);
}
```

Після успішного збереження користувач автоматично повертається на головний екран застосунку.

Для відображення історії фінансових операцій використовується компонент ListView.builder.

```
ListView.builder(
 itemCount:
```

```
 transactions.length,

 itemBuilder:
 (context, index) {

  final transaction =
  transactions[index];

  return TransactionCard(
   transaction:
   transaction,
  );
 },
    )
```

Кожна транзакція відображається у вигляді окремої картки.

```
Card(
 child: ListTile(
  title:
  Text(transaction.title),

  subtitle:
  Text(
   transaction.category
  ),

  trailing:
  Text(
```

```
   "${transaction.amount} ₴"
  ),
 ),
)
```

Такий підхід забезпечує зручний перегляд історії фінансових операцій та дозволяє швидко знаходити необхідну інформацію.

Для покращення користувацького досвіду реалізовано кольорове позначення доходів та витрат.

```
Color getTransactionColor() {

 if(transaction.type
 == "income")
 {
  return Colors.green;
 }

 return Colors.red;
}
```

Завдяки цьому користувач може швидко візуально відрізняти надходження коштів від фінансових витрат.

Реалізований екран забезпечує швидке додавання нових фінансових операцій, перегляд історії транзакцій та взаємодію з локальною базою даних. Його функціональність є основою роботи всього мобільного застосунку, оскільки саме фінансові записи використовуються для формування статистики та аналізу особистого бюджету користувача.

3.4 Реалізація категорій фінансових операцій

Для ефективного аналізу особистих фінансів важливим є групування доходів та витрат за певними категоріями. Використання категорій дозволяє

користувачу отримувати детальну інформацію про структуру власних витрат, визначати найбільш витратні напрямки та приймати більш обґрунтовані фінансові рішення.

У розробленому мобільному застосунку реалізовано систему категорій, яка дозволяє класифікувати фінансові операції за різними типами. Для витрат передбачено категорії «Харчування», «Транспорт», «Розваги», «Комунальні послуги», «Одяг» та інші. Для доходів використовуються категорії «Заробітна плата», «Премія», «Подарунок», «Інвестиції» та «Інші доходи».

Для представлення категорій у програмному коді створено окрему модель Category.

```
class Category {

 final int? id;
 final String name;
 final String icon;

 Category({
  this.id,
  required this.name,
  required this.icon,
 });
}
```

Модель містить ідентифікатор категорії, її назву та назву іконки, яка використовується для візуального відображення в інтерфейсі застосунку.

Для зберігання категорій у базі даних створено окрему таблицю categories.

```
CREATE TABLE categories(
 id INTEGER PRIMARY KEY AUTOINCREMENT,
 name TEXT NOT NULL,
```

```
 icon TEXT NOT NULL
);
```

Зчитування категорій із бази даних реалізовано через метод getCategories().

```
Future<List<Category>>
getCategories() async {

 final db =
 await instance.database;

 final result =
 await db.query("categories");

 return result.map(
  (json) =>
  Category(
   id: json["id"] as int,
   name: json["name"] as String,
   icon: json["icon"] as String,
  ),
 ).toList();
}
```

Для відображення категорій на екрані використовується список ListView.

```
ListView.builder(
 itemCount: categories.length,

 itemBuilder:
 (context, index) {
```

```
  return ListTile(
   title:
   Text(
    categories[index].name,
   ),
  );
 },
)
```

Під час створення нової фінансової операції користувач може вибрати категорію зі спеціального випадаючого списку.

```
DropdownButton<String>(
 value: selectedCategory,

 items: categories.map(
 (category) {

  return DropdownMenuItem(
   value: category.name,
   child:
   Text(category.name),
  );

 }).toList(),

 onChanged: (value) {
```

```
  setState(() {
   selectedCategory = value!;
  });

 },
)
```

Використання випадаючого списку дозволяє уникнути помилок введення та забезпечує уніфіковане зберігання фінансових записів.

Для підвищення зручності використання кожній категорії призначається відповідна іконка.

```
final categories = [

 Category(
  name: "Харчування",
  icon: "restaurant",
 ),

 Category(
  name: "Транспорт",
  icon: "directions_car",
 ),

 Category(
  name: "Розваги",
  icon: "sports_esports",
 ),
```

```
];
```

Завдяки використанню іконок користувач може швидше орієнтуватися у списку категорій та знаходити необхідний пункт.

Для відображення категорій на головному екрані реалізовано окремий віджет CategoryCard.

```
Card(
 child: ListTile(
  leading: Icon(
   Icons.restaurant
  ),

  title:
  Text("Харчування"),
 ),
)
```

Крім стандартного набору категорій, система може бути розширена шляхом додавання нових типів доходів та витрат. Такий підхід забезпечує гнучкість застосунку та дозволяє адаптувати його до потреб різних користувачів.

Таким чином, реалізована система категорій забезпечує структуроване збереження фінансових операцій, покращує якість статистичного аналізу та підвищує зручність використання мобільного застосунку.

3.5 Реалізація статистики та графіків

Однією з ключових функцій мобільного застосунку для відстеження особистих фінансів є можливість аналізу фінансової активності користувача. Для цього в системі реалізовано модуль статистики, який дозволяє візуалізувати доходи та витрати за допомогою графіків і діаграм. Використання графічного подання інформації значно спрощує аналіз фінансових показників та дозволяє швидко оцінити поточний стан особистого бюджету.

Для побудови графіків у Flutter використовується пакет fl_chart, який надає набір готових компонентів для створення діаграм різних типів. У межах розробленого застосунку реалізовано кругову діаграму розподілу витрат за категоріями та стовпчиковий графік співвідношення доходів і витрат.

Для підключення бібліотеки до проєкту необхідно додати залежність до файлу pubspec.yaml.

```
dependencies:
 flutter:
  sdk: flutter


 fl_chart: ^0.68.0
```

Перед побудовою графіків виконується обробка даних, отриманих із бази SQLite. Для цього розраховується загальна сума доходів та витрат.

```
double totalIncome = 0;
double totalExpense = 0;

for (var transaction
 in transactions)
{
 if(transaction.type
 == "income")
 {
  totalIncome +=
  transaction.amount;
 }
 else
 {
```

```
  totalExpense +=
  transaction.amount;
 }
}
```

Отримані значення використовуються для побудови статистичних звітів.

Для реалізації кругової діаграми використовується компонент PieChart.

```
PieChart(
 PieChartData(
  sections: [

   PieChartSectionData(
    value: totalIncome,
    title: "Доходи",
    radius: 90,
   ),

   PieChartSectionData(
    value: totalExpense,
    title: "Витрати",
    radius: 90,
   ),

  ],
 ),
)
```

Кругова діаграма дозволяє швидко оцінити співвідношення між доходами та витратами користувача за обраний період.

Для більш детального аналізу реалізовано стовпчиковий графік доходів і витрат.

```
BarChart(
 BarChartData(
  barGroups: [

   BarChartGroupData(
    x: 0,
    barRods: [
     BarChartRodData(
      toY: totalIncome,
     ),
    ],
   ),

   BarChartGroupData(
    x: 1,
    barRods: [
     BarChartRodData(
      toY: totalExpense,
     ),
    ],
   ),

  ],
 ),
)
```

Стовпчиковий графік дозволяє користувачеві швидко порівнювати обсяги фінансових надходжень та витрат.

Для формування статистики за категоріями реалізовано групування транзакцій.

```
Map<String, double>
categoryTotals = {};

for(var transaction
 in transactions)
{
 categoryTotals.update(
  transaction.category,

  (value) =>
  value +
  transaction.amount,

  ifAbsent: () =>
  transaction.amount,
 );
}
```

Результати групування використовуються для побудови окремої діаграми структури витрат.

Для відображення загального балансу реалізовано спеціальний інформаційний блок.

```
double balance =
totalIncome -
```

```
totalExpense;
Text(
 "Баланс: "
 "${balance.toStringAsFixed(2)} ₴",
)
```

Отримані показники автоматично оновлюються після додавання нових фінансових операцій, що забезпечує актуальність статистичної інформації.

Для зручності користувача статистика формується в реальному часі без необхідності ручного оновлення сторінки. Використання Provider забезпечує автоматичне оновлення інтерфейсу після зміни даних у базі SQLite.

```
Consumer<FinanceProvider>(
 builder:
 (context, provider, child)
 {
  return StatisticsScreen(
   transactions:
   provider.transactions,
  );
 },
)
```

Реалізований модуль статистики дозволяє користувачеві аналізувати фінансову активність, оцінювати структуру витрат та контролювати особистий бюджет. Візуальне представлення даних у вигляді графіків значно покращує сприйняття інформації та спрощує процес прийняття фінансових рішень.

Таким чином, використання бібліотеки fl_chart та автоматичного формування статистичних показників забезпечує ефективний інструмент аналізу особистих фінансів у мобільному застосунку.

ВИСНОВКИ

У кваліфікаційній роботі було виконано проєктування та розробку прототипу мобільного застосунку для відстеження особистих фінансів. У процесі виконання роботи досліджено особливості персонального фінансового менеджменту, проаналізовано сучасні мобільні застосунки для обліку доходів і витрат та визначено основні вимоги до програмного продукту.

У першому розділі проведено аналіз предметної області мобільних фінансових застосунків. Розглянуто основні принципи персонального фінансового менеджменту та досліджено функціональні можливості популярних програмних продуктів, зокрема Wallet, Monefy, CoinKeeper і Money Manager. На основі проведеного аналізу сформовано перелік функціональних і нефункціональних вимог до майбутнього мобільного застосунку та обґрунтовано вибір технологій розробки.

У другому розділі виконано проєктування системи. Визначено структуру даних, розроблено UML-діаграми, спроєктовано базу даних SQLite та створено прототип інтерфейсу користувача. Проведене моделювання дозволило сформувати логічну структуру програмного продукту та підготувати основу для подальшої реалізації застосунку.

У третьому розділі виконано програмну реалізацію мобільного застосунку засобами Flutter. Розроблено архітектуру програмного забезпечення, реалізовано модуль роботи з базою даних SQLite, механізми додавання та збереження фінансових операцій, систему категорій доходів і витрат, а також модуль статистичного аналізу. Для візуалізації фінансових показників використано графіки та діаграми, що дозволяють користувачеві швидко оцінювати власний фінансовий стан.

У процесі розробки було використано сучасні технології мобільної розробки, зокрема Flutter, Dart, SQLite, Provider та бібліотеку fl_chart. Застосування зазначених інструментів дозволило створити кросплатформний

мобільний застосунок із зручним користувацьким інтерфейсом та можливістю локального зберігання даних.

Розроблений застосунок забезпечує виконання основних завдань персонального фінансового менеджменту, а саме ведення обліку доходів і витрат, перегляд історії фінансових операцій, аналіз структури витрат, контроль бюджету та формування статистичних звітів. Реалізований функціонал дозволяє користувачам більш ефективно контролювати власні фінанси та приймати обґрунтовані фінансові рішення.

Практична цінність роботи полягає у створенні програмного продукту, який може бути використаний як основа для подальшої розробки повноцінної системи управління особистими фінансами. У майбутньому функціональні можливості застосунку можуть бути розширені шляхом додавання хмарної синхронізації, інтеграції з банківськими сервісами, підтримки декількох валют, автоматичного формування фінансових прогнозів та використання інструментів штучного інтелекту для аналізу фінансової поведінки користувачів.

Отже, поставлена мета роботи досягнута, усі визначені завдання виконані в повному обсязі, а розроблений мобільний застосунок відповідає сучасним вимогам до програмних засобів управління особистими фінансами та може бути використаний як основа для подальшого розвитку і вдосконалення фінансових інформаційних систем.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Биков В. Ю. Інформаційні технології в сучасному суспільстві. Київ : Либідь, 2020. 368 с.
2. Гужва В. М. Інформаційні системи і технології на підприємствах. Київ : КНЕУ, 2021. 400 с.
3. Копняк К. В. Інформаційні системи та технології. Київ : Центр учбової літератури, 2022. 312 с.
4. Морзе Н. В., Вембер В. П. Основи програмної інженерії. Київ : Академія, 2021. 256 с.
5. Федорченко О. В. Розробка мобільних застосунків для Android та iOS. Київ : Видавництво Ліра-К, 2023. 286 с.
6. Шаховська Н. Б., Пасічник В. В. Проєктування інформаційних систем. Львів : Магнолія 2006, 2021. 384 с.
7. Android Developers. URL: https://developer.android.com/ (дата звернення: 13.06.2026).
8. Apple Developer Documentation. URL: https://developer.apple.com/documentation/ (дата звернення: 13.06.2026).
9. Banks A., Porcello E. Learning React. Sebastopol : O'Reilly Media, 2023. 394 p.
10. CoinKeeper Application. URL: https://coinkeeper.me/ (дата звернення: 13.06.2026).
11. Dart Documentation. URL: https://dart.dev/guides (дата звернення: 13.06.2026).
12. Figma Documentation. URL: https://help.figma.com/ (дата звернення: 13.06.2026).
13. Firebase Documentation. URL: https://firebase.google.com/docs (дата звернення: 13.06.2026).
14. Flutter Documentation. URL: https://docs.flutter.dev/ (дата звернення: 13.06.2026).

15. Flutter Package fl_chart. URL: https://pub.dev/packages/fl_chart (дата звернення: 13.06.2026).
16. Flutter Package Provider. URL: https://pub.dev/packages/provider (дата звернення: 13.06.2026).
17. Flutter Package sqflite. URL: https://pub.dev/packages/sqflite (дата звернення: 13.06.2026).
18. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd Edition. Boston : Addison-Wesley, 2018. 208 p.
19. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 2019. 416 p.
20. Garrett J. The Elements of User Experience. Berkeley: New Riders, 2021. 192p.
21. Google Developers. Flutter Architecture Guide. URL: https://docs.flutter.dev/app-architecture (дата звернення: 13.06.2026).
22. Google Material Design Guidelines. URL: https://m3.material.io/ (дата звернення: 13.06.2026).
23. ISO 9241-210:2019 Ergonomics of Human-System Interaction — Human-Centred Design for Interactive Systems.
24. Krug S. Don't Make Me Think: Revisited. Berkeley : New Riders, 2022. 216 p.
25. Lewis C. Irresistible Apps: Motivational Design Patterns for Apps, Games and Web-Based Communities. New York : Apress, 2020. 320 p.
26. Martin R. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2018. 432 p.
27. Martin R. Clean Code: A Handbook of Agile Software Craftsmanship. Boston : Prentice Hall, 2019. 464 p.
28. Monefy Application. URL: https://monefy.me/ (дата звернення: 13.06.2026).
29. Money Manager Application. URL: https://realbyteapps.com/ (дата звернення: 13.06.2026).
30. Nielsen J. Usability Engineering. San Francisco : Morgan Kaufmann, 2021. 362 p.

31. Powers D. PHP Solutions: Dynamic Web Design Made Easy. New York : Apress, 2022. 528 p.
32. Pressman R., Maxim B. Software Engineering: A Practitioner's Approach. 9th Edition. New York : McGraw-Hill, 2020. 880 p.
33. Sommerville I. Software Engineering. 10th Edition. Boston : Pearson, 2021. 816 p.
34. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 13.06.2026).
35. Wallet Application. URL: https://budgetbakers.com/ (дата звернення: 13.06.2026).

ДОДАТОК

```
lib/

main.dart

models/
 user.dart
 transaction.dart
 category.dart
 budget.dart

database/
 database_helper.dart

providers/
 finance_provider.dart

screens/
 login_screen.dart
 register_screen.dart
 home_screen.dart
 add_transaction_screen.dart
 statistics_screen.dart

widgets/
 transaction_card.dart
```

```
 chart_widget.dart
 category_card.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';

import 'providers/finance_provider.dart';
import 'screens/login_screen.dart';

void main() {
  runApp(
    ChangeNotifierProvider(
      create: (_) => FinanceProvider(),
      child: const FinanceApp(),
    ),
  );
}

class FinanceApp extends StatelessWidget {
  const FinanceApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Finance Tracker',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
```

```
        primarySwatch: Colors.green,
        useMaterial3: true,
      ),
      home: const LoginScreen(),
    );
  }
}
class User {
  int? id;
  String name;
  String email;
  String password;

  User({
    this.id,
    required this.name,
    required this.email,
    required this.password,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'name': name,
      'email': email,
      'password': password,
```

```
    };
  }

  factory User.fromMap(Map<String, dynamic> map) {
    return User(
      id: map['id'],
      name: map['name'],
      email: map['email'],
      password: map['password'],
    );
  }
}
class TransactionModel {
  int? id;
  String title;
  double amount;
  String category;
  String type;
  String date;

  TransactionModel({
    this.id,
    required this.title,
    required this.amount,
    required this.category,
    required this.type,
```

```
    required this.date,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'title': title,
      'amount': amount,
      'category': category,
      'type': type,
      'date': date,
    };
  }

  factory TransactionModel.fromMap(
      Map<String, dynamic> map) {
    return TransactionModel(
      id: map['id'],
      title: map['title'],
      amount: map['amount'],
      category: map['category'],
      type: map['type'],
      date: map['date'],
    );
  }
}
```