НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
__інформаційних систем і технологій_
(назва кафедри)

_______________ / Швиденко М. З. /
(підпис) (ПІБ)

“___” __________ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

«Мобільна платформа для ефективного управління особистими фінансами»

Спеціальність 122 – «Комп’ютерні науки»

Гарант освітньої програми
_________д. ек. н, професор_________ ________________ _____ Руденський Р.А______
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи
___________ к. ек. н.____________ ______________ __Стариченко Є. М.________
(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _________________ ___________Капінус Денис Олексійович__________
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ
УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри
__інформаційних систем і технологій __
(назва кафедри)

___к.ек. н, доцент ____________ Швиденко М. З.
(науковий ступінь, вчене звання) (підпис) (ПІБ)
“___” __________ 2025 р.

З А В Д А Н Н Я

на виконання бакалаврської кваліфікаційної роботи студенту
Капінус Денис Олексійович

Спеціальність 122 – «Комп’ютерні науки»

1. Тема бакалаврської кваліфікаційної роботи «Мобільна платформа для ефективного управління особистими фінансами» затверджена наказом ректора ректора НУБіП Украї від ід 25.04.2025 № 699 “С”
2. Термін подання завершеної роботи на кафедру________02.06.2025 р._______
(рік, місяць, число)
3. Вихідні дані: надання інформації про облік власних доходів та витрат у вигляді категорій, а також аналіз своїх фінансових даних у вигляді звітності.
4. Перелік питань, що розглядаються:
- Аналіз проблемної області
- Моделювання предметної області
- Проєктування програмної системи
- Впровадження та експлуатація системи

Дата видачі завдання “25”_квітня_ 2025 р.

Керівник бакалаврської кваліфікаційної роботи __________________/ Стариченко Є. М./
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання:______________________________/ Капінус Д.О./
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У бакалаврській роботі представлено розробку мобільної платформи, спрямованої на спрощення та покращення управління особистими фінансами. Додаток розроблено за допомогою Flutter та мови програмування Dart, що забезпечує кросплатформну сумісність та безперебійний користувацький досвід. Дані зберігаються локально за допомогою SQLite, що дозволяє додатку функціонувати без необхідності підключення до Інтернету.

Система дозволяє користувачам створювати та керувати категоріями доходів та витрат, записувати фінансові операції та візуалізувати моделі витрат за допомогою інтерактивних діаграм та зведень. Акцент робиться на інтуїтивно зрозумілому інтерфейсі та ефективній організації інформації, щоб допомогти користувачам приймати обґрунтовані фінансові рішення.

Ця робота включає аналіз існуючих програм для управління фінансами, проєктування архітектури системи та структури бази даних, а також реалізацію ключових функцій, таких як відстеження транзакцій та візуалізація даних.

ABSTRACT

This bachelor's thesis presents the development of a mobile platform aimed at simplifying and improving the management of personal finances. The application is developed using Flutter and the Dart programming language, ensuring cross-platform compatibility and a smooth user experience. Data is stored locally using SQLite, allowing the application to function without the need for an internet connection.

The system allows users to create and manage income and expense categories, record financial transactions, and visualize spending patterns through interactive charts and summaries. Emphasis is placed on an intuitive interface and efficient organization of information to support users in making informed financial decisions.

This work includes an analysis of existing financial management applications, the design of the system architecture and database structure, and the implementation of key features such as transaction tracking and data visualization.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання бакалаврської роботи	Строк виконання етапів бакалаврської роботи	Примітка
1	Отримання завдання	25 квітня 2025	
2	Аналіз предметної області	квітень 2025	
3	Моделювання предметної області	травень 2025	
4	Проєктування програмної системи	травень 2025	
5	Розгортання та експлуатація програмного забезпечення	травень 2025	
6	Економічне дослідження розробки та експлуатації програмної системи	квітень- травень 2025	
7	Оформлення записки	травень 2025	
8	Перевірка на плагіат	травень 2025	
9	Проходження нормо контроля	травень 2025	
10	Проходження передзахисту	2 червня 2025	
11	Захист роботи	червень 2025	

Студент ____________ Капінус Д.О.

(підпис) (ПІБ)

Керівник бакалаврської кваліфікаційної роботи

к.ек.н, доцент ____________ Стариченко Є. М.

(науковий ступінь та вчене звання) (підпис) (ПІБ)

ЗМІСТ

ВСТУП ... 4
1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ... 6
1.1 Опис предметної області ... 6
1.2 Огляд існуючих рішень ... 9
1.3 Постановка завдання ... 12
1.4 Функціональні та нефункціональні вимоги ... 14
1.5 Вимоги до інтерфейсу користувача ... 15
2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ... 18
2.1 Загальні відомості ... 18
2.2 Об’єктне та функціональне моделювання ... 20
2.3 Абстракції предметної області ... 30
2.4 Діаграма класів ... 32
3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ ... 36
3.1 Логічна модель даних ... 36
3.2 Вибір системи управління базою даних та її реалізація ... 42
3.3 Архітектура програмного забезпечення ... 46
3.4 Організаційна структура програмного забезпечення ... 49
3.5 Вибір інструментарію для створення програмного забезпечення ... 51
4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ ... 53
4.1 Вимоги до апаратного та програмного забезпечення ... 53
4.2 Тестування системи ... 56
ВИСНОВКИ ... 61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ... 63
ДОДАТОК А ... 66
ДОДАТОК Б ... 68

ВСТУП

У сучасному світі ефективне управління фінансами є ключовим компонентом стабільного та успішного життя. Зі зростанням кількості фінансових транзакцій, як у цифровому, так і у фізичному вигляді, люди стикаються з проблемою організації та моніторингу своїх доходів і витрат. Розвиток цифрових технологій, особливо мобільних платформ, відкрив нові можливості для створення доступних та зручних інструментів для управління особистими фінансами.

Швидкий розвиток технологій відкрив нові можливості для створення інтуїтивно зрозумілих та доступних інструментів, що допомагають у фінансовому самокеруванні. У відповідь на цю зростаючу потребу, у цій роботі представлено проєктування та впровадження системи управління особистими фінансами. Система надає користувачам централізовану платформу для відстеження їхніх доходів та витрат, управління категоріями транзакцій та аналізу їхніх фінансових звичок з часом.

Основною **метою** цієї бакалаврської роботи є допомога користувачам краще контролювати свої особисті фінанси, розробивши мобільний додаток для ефективного управління фінансами. Система дозволяє користувачам створювати категорії доходів і витрат, фіксувати фінансові операції та аналізувати їх за допомогою статистики та візуальних представлень. Додаток реалізовано за допомогою Flutter та мови програмування Dart, що забезпечує кросплатформну сумісність, та зберігає дані локально за допомогою SQLite, що забезпечує швидкий та офлайн-доступ до фінансових записів.

Актуальність теми визначається зростаючою важливістю фінансової грамотності та потребою в доступних інструментах для контролю особистих бюджетів. Багато існуючих рішень перевантажені функціями, вимагають постійного підключення до Інтернету або зберігають конфіденційні дані в хмарі, що може не відповідати потребам користувачів, які цінують простоту,

швидкість та конфіденційність. Тому легкий та інтуїтивно зрозумілий мобільний додаток, який працює офлайн та пропонує базовий функціонал для відстеження доходів і витрат, є дуже актуальним у сучасному контексті.

Предметом цього дослідження є процес розробки мобільних додатків для управління фінансами. Це включає вивчення технологій мобільної розробки (Flutter, SQLite), дизайну інтерфейсу користувача для інструментів, пов'язаних з фінансами, інтеграції з локальними базами даних та методів ефективної організації та відображення фінансових даних.

Об'єктом цього дослідження є процес управління особистими фінансами, реалізований за допомогою мобільного програмного продукту. Він охоплює способи взаємодії користувачів з фінансовими інструментами, організації своїх грошових потоків та прийняття рішень на основі фінансової інформації, що надається системою.

Основні цілі системи:

- забезпечити безпечний та зручний інтерфейс для запису фінансових транзакцій;
- дозволити користувачам визначати власні категорії доходів та витрат;
- забезпечити детальний аналіз та візуалізацію фінансових даних;
- підтримка прийняття обґрунтованих фінансових рішень шляхом відстеження бюджету та зведень.

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис предметної області

В епоху, коли цифрові платежі та онлайн-банкінг стали повсякденними нормами, управління особистими фінансами більше не обмежується лише збереженням квитанцій або перевіркою банківських виписок наприкінці місяця. Фінансова обізнаність є важливою, проте багатьом людям досі бракує ефективних інструментів для відстеження своїх доходів та витрат у структурований та зручний для користувача спосіб. Ця прогалина спричинила зростаючий попит на мобільні рішення, які можуть допомогти людям контролювати своє фінансове життя, не покладаючись на традиційні або надмірно складні системи.

Сьогоднішній ринок пропонує різноманітні програми, розроблені для управління особистими фінансами. Серед найпопулярніших – Wallet, Monefy, Spendee та Money Manager. Ці програми надають корисні функції, такі як відстеження витрат, планування бюджету, фінансова аналітика та навіть інтеграція з банком. Однак вони також мають суттєві недоліки. Багато з них вимагають реєстрації користувача та постійного доступу до Інтернету, що може відлякати тих, хто турбується про конфіденційність або бажає використовувати програму офлайн. Інші накладають обмеження на безкоштовне використання, блокуючи розширені функції за платним доступом, а деякі перевантажують користувачів захаращеними інтерфейсами та непотрібною складністю [1].

Ці недоліки підкреслюють явну потребу в альтернативному підході – такому, що наголошує на простоті, доступності та контролі з боку користувача. Більшість людей не потребують розширеного фінансового прогнозування чи синхронізації з кількома обліковими записами. Натомість вони отримують більше користі від простої системи, яка дозволяє їм легко додавати щоденні витрати, створювати персоналізовані категорії та

переглядати візуальні зведення своїх фінансів. Більше того, надійний офлайн-режим є критично важливим для користувачів, які або не мають постійного доступу до Інтернету, або вважають за краще зберігати свої фінансові дані локально на своїх пристроях.

Розробка мобільної платформи за допомогою Flutter та SQLite вирішує багато з цих проблем. Flutter пропонує плавний та адаптивний кросплатформний інтерфейс, дозволяючи додатку працювати як на Android, так і на iOS з єдиною кодовою базою. SQLite, як легка вбудована база даних, забезпечує швидке та безпечне локальне зберігання даних, усуваючи залежність від хмарних сервісів та гарантуючи конфіденційність користувачів. Разом ці технології дозволяють створити мобільний додаток, який є не тільки ефективним та практичним, але й доступним для широкої аудиторії, включаючи тих, хто має старіші або менш потужні пристрої.

Зрештою, аналіз існуючих інструментів та потреб користувачів підкреслює важливість цілеспрямованого, мінімалістичного рішення для управління особистими фінансами — такого, яке поєднує функціональність з простотою використання та надає користувачеві повний контроль над власними фінансовими даними.

Управління особистими фінансами відіграє вирішальну роль у забезпеченні фінансової стабільності та благополуччя людей. Зі зростанням кількості та складності фінансових операцій багато людей стикаються з труднощами у відстеженні своїх витрат, управлінні доходами та підтримці збалансованого бюджету. Предметна галузь управління особистими фінансами зосереджена на інструментах та методах, які допомагають користувачам приймати обґрунтовані рішення щодо того, як вони заробляють, витрачають та заощаджують гроші [2].

Традиційно люди керували своїми фінансами за допомогою блокнотів, електронних таблиць або простих калькуляторів. Однак з розвитком цифрових технологій мобільні платформи стали найдоступнішим та найефективнішим засобом виконання повсякденних фінансових завдань. Смартфони зараз

широко використовуються всіма віковими групами, пропонуючи користувачам можливість контролювати свої фінанси в дорозі, будь-коли та будь-де. Цей зсув створив значний попит на мобільні додатки, які надають інтуїтивно зрозумілі та надійні фінансові інструменти, адаптовані до індивідуальних потреб.

У контексті цієї предметної області основна увага приділяється основним функціям, необхідним типовому користувачеві: можливості створювати та налаштовувати категорії доходів і витрат, швидко записувати транзакції та переглядати зведення або статистику для кращого розуміння своєї фінансової поведінки. Ці основні функції слугують основою для формування хороших фінансових звичок та підвищення фінансової грамотності з часом.

З технічної точки зору, розробка такого застосунку передбачає поєднання дизайну інтерфейсу користувача, інтеграції з локальною базою даних та плавного користувацького досвіду. Flutter, сучасний фреймворк, розроблений Google, дозволяє створювати кросплатформні мобільні застосунки з єдиною кодовою базою, забезпечуючи високу продуктивність та візуальну узгодженість як на пристроях Android, так і на iOS. SQLite використовується для локального зберігання даних, що дозволяє користувачам отримувати доступ до своєї фінансової інформації навіть без підключення до Інтернету — функція, яка не тільки підвищує конфіденційність, але й підвищує надійність.

Тематична галузь також охоплює такі аспекти, як візуалізація фінансових даних, організація категорій та аналіз транзакцій. Ці елементи допомагають користувачам виявляти закономірності у своїх звичках доходів та витрат, виявляти потенційні перевитрата та вносити відповідні корективи. Спрощуючи ці завдання за допомогою мобільного застосунку, користувачі отримують кращий контроль над своїми фінансами та мають можливість приймати більш розумні фінансові рішення [3].

Підсумовуючи, тематична галузь мобільного фінансового менеджменту об'єднує принципи особистих фінансів та сучасні технології для вирішення реальних життєвих проблем. Вона спрямована на створення рішень, орієнтованих на користувача, безпечних та адаптивних — все це є важливими якостями в сучасному швидкоплинному, цифровому світі.

1.2 Огляд існуючих рішень

В останні роки попит на додатки для управління особистими фінансами значно зріс, що зумовлено потребою людей краще керувати своїми грошима у все більш безготівковому та цифровому світі. З'явився широкий спектр мобільних рішень, кожне з яких пропонує різні підходи до відстеження доходів і витрат, складання бюджету та фінансового аналізу. Ці додатки спрямовані на спрощення процесу управління особистими фінансами, хоча їхня ефективність залежить від дизайну, функціональності та взаємодії з користувачем.

Wallet від BudgetBakers – це широко використовуваний додаток для особистих фінансів, розроблений, щоб допомогти користувачам керувати своїми доходами, витратами та загальним бюджетуванням. Позиціонований як розумний помічник для особистого фінансового планування, Wallet пропонує різноманітні функції, такі як відстеження транзакцій, створення бюджету, постановка цілей та фінансова аналітика на основі поведінки користувача. Однією з його ключових переваг є можливість синхронізації з банківськими рахунками, автоматичний імпорт транзакцій з кількох фінансових установ, що спрощує процес моніторингу [4].

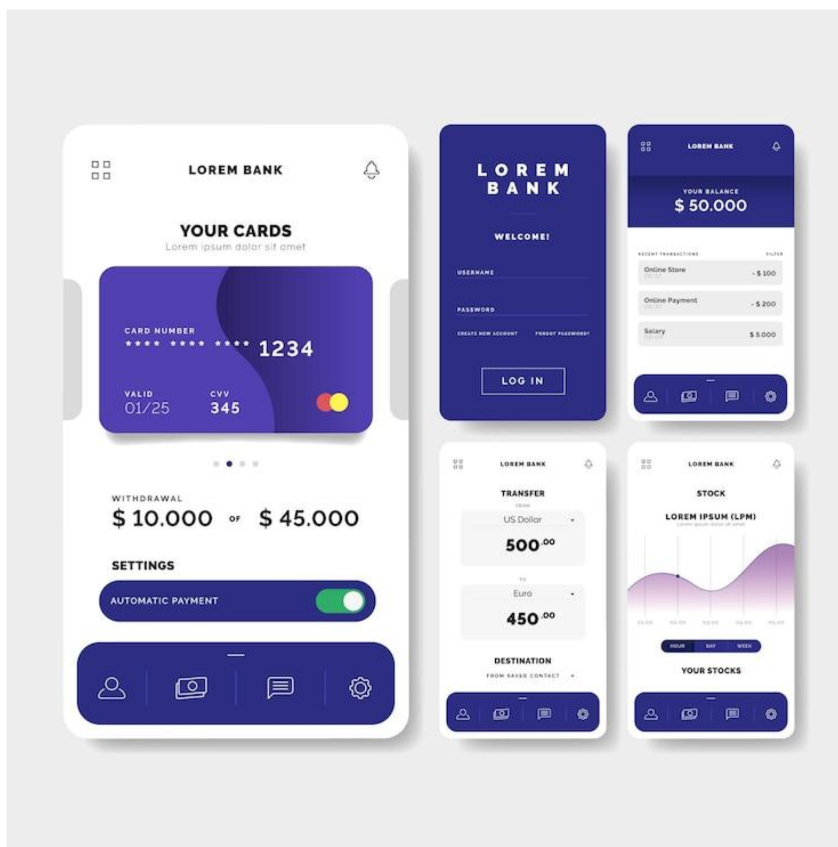

Рис. 1.1 – Мобільний додаток “Wallet”

Додаток підтримує операції з кількома валютами, спільні бюджети для сімей або пар та детальні діаграми для фінансового аналізу. Користувачі можуть класифікувати витрати, планувати майбутні платежі та встановлювати цілі заощаджень. Усі дані можна зберігати в хмарі, що дозволяє доступ до них з кількох пристроїв, включаючи смартфони, планшети та веб-браузери.

Однак Wallet має деякі обмеження. Більшість розширених функцій, таких як автоматична синхронізація банківських рахунків, розширені звіти та резервне копіювання даних у режимі реального часу, доступні лише в преміум-версії. Крім того, для повної функціональності потрібне постійне підключення до Інтернету, що може бути не ідеальним для користувачів, які віддають перевагу офлайн-доступу або стурбовані безпекою своїх фінансових даних, що зберігаються на зовнішніх серверах [4].

Незважаючи на ці недоліки, Wallet залишається популярним та потужним інструментом для управління особистими фінансами, особливо для користувачів, які цінують автоматизацію, синхронізацію та детальну

аналітику. Його зручний інтерфейс та широкий функціонал роблять його сильним конкурентом на ринку фінансових додатків, хоча він може більше підійти для користувачів зі складними фінансовими потребами та готовністю інвестувати в підписку.

Розглянемо інше програмне рішення на ринку.

Monefy — це легкий додаток для особистих фінансів, відомий своєю простотою, швидкістю та зручним дизайном. Розроблений з урахуванням потреб повсякденних користувачів, він зосереджений на тому, щоб зробити відстеження витрат і доходів максимально швидким та інтуїтивно зрозумілим. Основна функціональність Monefy полягає в ручному введенні транзакцій, які одразу відображаються на візуальних діаграмах, що відображають витрати за категоріями [5].

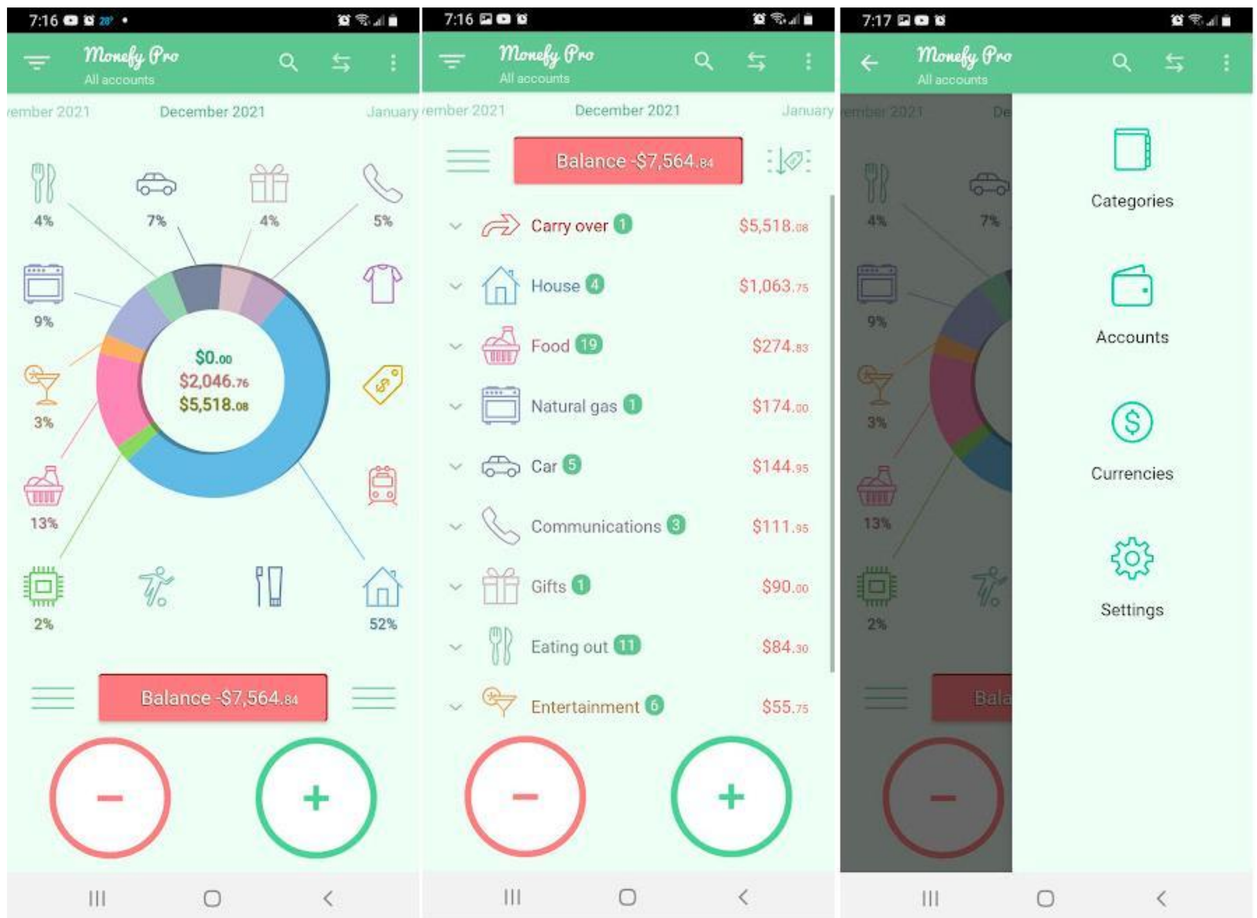

Рис. 1.2 – Мобільний додаток “Monefy”

Додаток має чистий та мінімалістичний інтерфейс, що дозволяє користувачам легко вибирати категорії, вводити суми та переглядати

фінансові звіти. Monefy підтримує налаштовувані категорії, використання кількох валют, захист паролем та синхронізацію даних через хмарні сервіси, такі як Google Drive або Dropbox. Він сумісний з платформами Android та iOS і пропонує віджет для швидшого доступу на головному екрані.

Однією з сильних сторін Monefy є акцент на конфіденційності та доступності офлайн. Усі дані за замовчуванням зберігаються локально на пристрої, що робить їх доступними без підключення до Інтернету та зменшує залежність від зовнішніх серверів. Це робить його привабливим варіантом для користувачів, які стурбовані безпекою даних та підключенням [5].

Незважаючи на свою простоту, Monefy має обмеження. Йому бракує розширених інструментів для складання бюджету, він не пропонує інтеграцію з банківськими рахунками та надає лише базову статистичну інформацію. Ці обмеження можуть зробити його менш придатним для користувачів, які шукають комплексні функції фінансового планування. Крім того, деякі функції, такі як детальні звіти, резервне копіювання та редагування категорій, доступні лише в платній версії.

Загалом, Monefy приваблює людей, які шукають простий, швидкий та конфіденційний спосіб управління особистими фінансами без зайвої складності. Його зосередженість на основних функціях та інтуїтивно зрозумілий інтерфейс відрізняють його від альтернатив з більш насиченим набором функцій, що робить його ідеальним для користувачів з базовими потребами у фінансовому відстеженні.

1.3 Постановка завдання

Щоб продемонструвати можливості розробленої мобільної платформи, розглянемо відділ, де керівник відповідає за відстеження певних показників ефективності, таких як споживання калорій, споживання води та фізична активність, включаючи пройдені кроки. Хоча ці показники зазвичай пов'язані з моніторингом здоров'я, принцип залишається актуальним: необхідність

збору, зберігання та аналізу персональних даних для прийняття більш обґрунтованих рішень. У нашому випадку ця парадигма застосовується до фінансового менеджменту.

Система, що розробляється, функціонує як цифровий помічник, який дозволяє користувачам відстежувати свої фінансові показники, такі як потоки доходів, витрати за категоріями та загальні заощадження, персоналізованим та інтерактивним способом. Для цього додаток повинен створити надійну локальну базу даних, яка зберігає всі фінансові записи та дозволяє взаємодіяти в режимі реального часу без необхідності постійного доступу до Інтернету.

Розширена функціональність платформи надає користувачам миттєвий доступ до ключової фінансової інформації, включаючи:

- ефективність витрат за різними категоріями (наприклад, продукти харчування, розваги, транспорт);
- точність та регулярність обліку доходів;
- загальні витрати порівняно з бюджетними лімітами;
- профіцит або дефіцит порівняно із запланованими цілями заощаджень.

Програма працює в діалоговому інтерфейсі, що дозволяє користувачеві переміщатися між опціями меню. Ці опції керують користувачем у процесі реєстрації транзакцій, перегляду фінансових звітів та коригування категорій або бюджетів. Крім того, користувачі мають можливість переглядати та видаляти раніше введені записи за необхідності, зберігаючи точні та актуальні фінансові дані.

На цьому прикладі стає очевидним, що мобільна платформа, орієнтована на ефективне управління фінансами, повинна не лише зберігати та обробляти інформацію, але й забезпечувати зручне середовище, яке підтримує постановку цілей, відстеження прогресу та прийняття рішень на основі зворотного зв'язку в режимі реального часу.

1.4 Функціональні та нефункціональні вимоги

Функціональні вимоги:

1. додаток повинен дозволяти користувачам створювати обліковий запис, безпечно входити в систему та відновлювати забуті облікові дані;
2. користувачі повинні мати можливість додавати, редагувати та видаляти записи про доходи. Кожен запис повинен містити дату, суму, категорію та додаткові примітки;
3. система повинна забезпечувати швидкий запис витрат з такими деталями, як сума, дата, категорія та додатковий опис;
4. користувачі повинні мати можливість створювати, перейменовувати та видаляти власні категорії доходів і витрат для кращої персоналізації;
5. додаток повинен генерувати зведення та візуальні представлення (наприклад, кругові діаграми або стовпчикові діаграми), щоб показати структуру витрат, розподіл доходів та заощадження;
6. користувачі повинні мати можливість встановлювати щомісячні бюджети для певних категорій та отримувати попередження або сповіщення, коли вони наближаються до лімітів або перевищують їх;
7. повинна існувати функція пошуку та фільтрації записів за датою, сумою або категорією;
8. додаток повинен підтримувати експорт та імпорт даних для резервного копіювання та передачі, бажано з використанням локального сховища або додаткової хмарної інтеграції;
9. усі основні функції повинні бути доступні офлайн, а зміни синхронізуються з хмарою (якщо використовується) після відновлення з'єднання.

Нефункціональні вимоги:

1. додаток повинен забезпечувати плавний та адаптивний інтерфейс користувача, навіть на пристроях з обмеженими ресурсами;

2. система повинна мати можливість обробляти зростаючу кількість записів та користувачів без погіршення продуктивності;
3. інтерфейс має бути інтуїтивно зрозумілим та доступним для користувачів різного віку та рівня цифрової грамотності. Він має відповідати сучасним практикам UX/UI дизайну та рекомендаціям Flutter;
4. оскільки додаток розроблено на Flutter, він має бути сумісний як з платформами Android, так і з iOS;
5. усі персональні та фінансові дані повинні зберігатися безпечно з використанням шифрування, де це необхідно. Механізми автентифікації повинні запобігати несанкціонованому доступу;
6. система має бути модульною та добре документованою, щоб майбутні розробники могли легко оновлювати або розширювати її функціональність;
7. додаток повинен коректно обробляти недійсні вхідні дані та неочікувану поведінку без втрати даних або збоїв;
8. перевірка вхідних даних та обробка помилок повинні бути реалізовані для запобігання дублюванню або пошкодженню записів.

Чітко визначаючи та дотримуючись цих функціональних та нефункціональних вимог, платформа може задовольнити очікування кінцевих користувачів, забезпечуючи довгострокову стабільність, зручність використання та розширюваність.

1.5 Вимоги до інтерфейсу користувача

Добре розроблений інтерфейс користувача (UI) є ключовим компонентом будь-якого успішного мобільного застосунку, особливо того, що орієнтований на управління фінансами. Оскільки користувачі покладаються на цей інструмент для відстеження доходів, витрат та цілей бюджетування, інтерфейс має бути інтуїтивно зрозумілим, адаптивним та візуально цілісним. Основна мета — створити середовище, де користувачі можуть легко

орієнтуватися, ефективно взаємодіяти з функціями та інтерпретувати фінансові дані без плутанини чи відволікань.

Застосунок повинен підтримувати чисту та сучасну естетику, яка сприяє зосередженню та мінімізує когнітивне навантаження. Візуальна узгодженість на всіх екранах, включаючи однакове використання шрифтів, кольорів та піктограми, допомагає створити передбачуваний та комфортний досвід. Адаптивність не менш важлива; інтерфейс користувача повинен миттєво реагувати на введення користувача, пропонуючи візуальне підтвердження завершених дій або надсилання форм. Це зміцнює довіру до системи та забезпечує безперебійний робочий процес.

Оскільки застосунок створено за допомогою Flutter, інтерфейс повинен безперешкодно адаптуватися до різних розмірів та роздільних здатностей екранів як на платформах Android, так і на iOS. Оптимізація для мобільних пристроїв не обмежується дисплеєм, але також включає сенсорний досвід, що вимагає кнопок відповідного розміру, інтуїтивно зрозумілих жестів та продуманого розташування елементів [6].

Навігація в застосунку має бути простою. Користувачі повинні мати можливість плавно переміщатися між такими розділами, як панель інструментів, історія транзакцій, менеджер категорій, статистика та налаштування. Незалежно від того, чи використовується нижня панель навігації чи бічна шухляда, обрана структура повинна залишатися доступною та логічною. Кожен розділ повинен бути природним чином пов'язаний з іншими, що усуває будь-які здогадки щодо розташування певної функції. Крім того, всі екрани повинні підтримувати легке повернення до попередніх переглядів за допомогою стандартних або налаштованих шаблонів навігації.

Процес введення даних, таких як запис доходів або витрат, має бути максимально безперешкодним. Форми повинні бути мінімальними та інтелектуально розробленими, пропонуючи користувачам лише необхідні поля, такі як сума, дата, категорія та додаткові примітки. Перевірка введених даних повинна запобігати помилкам перед надсиланням, а чіткі контекстні

повідомлення про помилки допомагають користувачеві вносити виправлення без плутанини.

Щодо розширення можливостей користувачів, додаток повинен забезпечувати персоналізовану категоризацію. Користувачі повинні мати можливість створювати та керувати власними категоріями доходів та витрат, коригуючи їх з часом, щоб відображати потреби, що змінюються. Аналогічно, функції бюджетування повинні дозволяти користувачам встановлювати щомісячні ліміти для різних категорій, з чіткими візуальними індикаторами, що показують, наскільки вони близькі до досягнення своїх цілей.

Інтерфейс повинен підтримувати чіткі механізми зворотного зв'язку. Це включає ненав'язливі сповіщення, які попереджають користувачів, коли вони наближаються до бюджету або перевищують його, а також підтвердження успішного збереження або оновлення даних. Кожен аспект інтерфейсу повинен м'яко, але впевнено вести користувача через його фінансову подорож.

Завдяки впровадженню цих принципів дизайну та орієнтованих на користувача вимог, платформа фінансового управління запропонує надійний та доступний інструмент, який не лише відповідає потребам користувачів, але й заохочує постійну залученість та фінансову обізнаність.

2 МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Загальні відомості

Уніфікована мова моделювання (UML) – це стандартизована мова моделювання, яка широко використовується в програмній інженерії для візуального представлення структури, поведінки та взаємодії всередині системи. Вона служить важливим інструментом для розробників, зацікавлених сторін та керівників проєктів, щоб чітко та зрозуміло представити дизайн системи. UML відіграє вирішальну роль у процесі розробки, особливо під час планування складної системи, такої як мобільна платформа управління фінансами, гарантуючи, що всі компоненти чітко визначені, а їхні зв'язки чітко зрозумілі до початку кодування [7].

UML пропонує різноманітні типи діаграм, кожна з яких розроблена для охоплення різних аспектів системи. Ці діаграми можна загалом розділити на структурні, поведінкові та діаграми взаємодії. Використовуючи UML, команди можуть візуалізувати, як організована система, як взаємодіють компоненти та як система розвиватиметься з точки зору функціональності та поведінки.

Серед найважливіших діаграм UML для цього проєкту є діаграми варіантів використання, діаграми класів, діаграми послідовностей, діаграми діяльності та діаграми станів. Кожен з цих типів діаграм служить окремій меті, ілюструючи, як функціонуватиме мобільна платформа управління фінансами та як користувачі взаємодіятимуть з нею.

Наприклад, діаграма варіантів використання представляє функціональні аспекти системи, зосереджуючись на взаємодії між користувачами та платформою. Вона окреслює ключові дії користувача, такі як додавання транзакцій, встановлення бюджетів або перегляд фінансових звітів. Діаграма варіантів використання допомагає зацікавленим сторонам

зрозуміти високорівневу функціональність програми з точки зору користувача.

Діаграма класів, з іншого боку, надає план статичної структури системи. Вона визначає основні сутності системи, такі як користувачі, транзакції, категорії та бюджети, та визначає їхні атрибути, методи та зв'язки. Відображаючи ці класи, діаграма забезпечує узгодженість структури бази даних програми та об'єктно-орієнтованого дизайну, що полегшує розробку та майбутні модифікації [8].

Діаграма послідовності демонструє потік взаємодій між різними об'єктами з плином часу. Вона особливо корисна для візуалізації послідовності дій, що беруть участь у певному процесі, такому як запис витрат. Ця діаграма показує, як система взаємодіє з користувачем, перевіряє вхідні дані, оновлює базу даних та надає зворотний зв'язок, пропонуючи детальний огляд динамічної поведінки платформи.

Діаграми активності ілюструють робочий процес у системі, показуючи послідовність завдань та рішень, що виникають під час певних процесів. Наприклад, коли користувач налаштовує бюджет, діаграма активності зображує всі кроки, від введення суми бюджету до отримання сповіщення про перевищення ліміту. Ці діаграми є важливими для відображення логіки системи та забезпечення ефективної роботи програми.

Діаграми станів зосереджені на тому, як об'єкти системи змінюють стани у відповідь на дії користувача або системні події. Наприклад, транзакція може переходити через такі стани, як «Очікує», «Завершено» або «Скасовано», залежно від її статусу та взаємодії користувача з нею. Діаграми станів допомагають забезпечити передбачувану та логічну поведінку системи з чітко визначеними переходами.

Використання UML у розробці мобільної платформи управління фінансами має кілька переваг. Це дозволяє всім членам команди та зацікавленим сторонам досягти спільного розуміння дизайну системи, зменшуючи ризик непорозумінь. Діаграми UML також служать

документацією, надаючи цінний довідник як для команди розробників, так і для майбутніх розробників системи. Виявляючи потенційні проблеми на ранній стадії проєктування, UML допомагає запобігти дороговартісним помилкам під час розробки та гарантує, що функції та архітектура застосунку є надійними та відповідають потребам користувача.

На завершення, UML є незамінним інструментом у розробці мобільної системи управління фінансами. Він забезпечує чітке візуальне представлення структури та поведінки системи, підтримуючи створення функціонального та ефективного застосунку. У міру розвитку проєкту ці діаграми UML будуть розвиватися, адаптуючись до нових вимог та вдосконалень, гарантуючи, що платформа залишається зручною для користувача та ефективною у досягненні цілей управління фінансами.

2.2 Об’єктне та функціональне моделювання

2.2.1 Діаграма прецедентів. Діаграма варіантів використання – це важливий візуальний інструмент, який використовується для представлення функціональності системи з точки зору її користувачів. Вона служить для фіксації взаємодії між користувачами, відомими як учасники, та самою системою, ілюструючи, як користувачі взаємодіятимуть із системою та які дії вони можуть виконувати. Цей тип діаграми є важливим на ранніх етапах розробки програмного забезпечення, оскільки він допомагає визначити ключові функції системи та вимоги користувачів [9].

На діаграмі варіантів використання учасники – це сутності, які взаємодіють із системою. Ці учасники можуть бути користувачами-людьми, зовнішніми системами або навіть іншими програмними компонентами, які взаємодіють з платформою. Наприклад, у контексті мобільної платформи управління фінансами типовими учасниками можуть бути «Користувач», «Адміністратор» або «Система».

Випадки використання на діаграмі представляють конкретні дії або послуги, які система надає у відповідь на вхідні дані користувача. Ці дії можуть включати такі завдання, як «Запис доходу», «Перегляд фінансового звіту», «Встановлення бюджету» або «Створення звітів». Кожен варіант використання відповідає певному елементу функціональності, який система пропонуватиме своїм користувачам.

Для ілюстрації зв'язків між акторами та функціями системи, на діаграмі використовується кілька типів зв'язків. Найпоширенішими є:

- асоціація: цей зв'язок пов'язує актора з варіантом використання, вказуючи на те, що актор взаємодіє з цим конкретним варіантом використання. Наприклад, актор «Користувач» може бути пов'язаний з варіантом використання «Запис доходу»;
- include: цей зв'язок показує, що один варіант використання містить інший як частину свого процесу. Він використовується, коли певна дія завжди передбачає іншу як необхідну частину її виконання;
- extend: розширений варіант використання надає додаткову функціональність до базового варіанту використання, але це розширення є необов'язковим і не завжди може бути використане.

Межа системи є ще одним важливим компонентом діаграми варіантів використання. Вона представлена прямокутником, який охоплює варіанти використання, позначаючи область дії системи та відрізняючи її від зовнішніх елементів. Ця межа уточнює, які функціональні можливості є частиною системи, а які знаходяться поза її областю дії.

Розроблена діаграма прецедентів використання представлена на рис. 2.1.

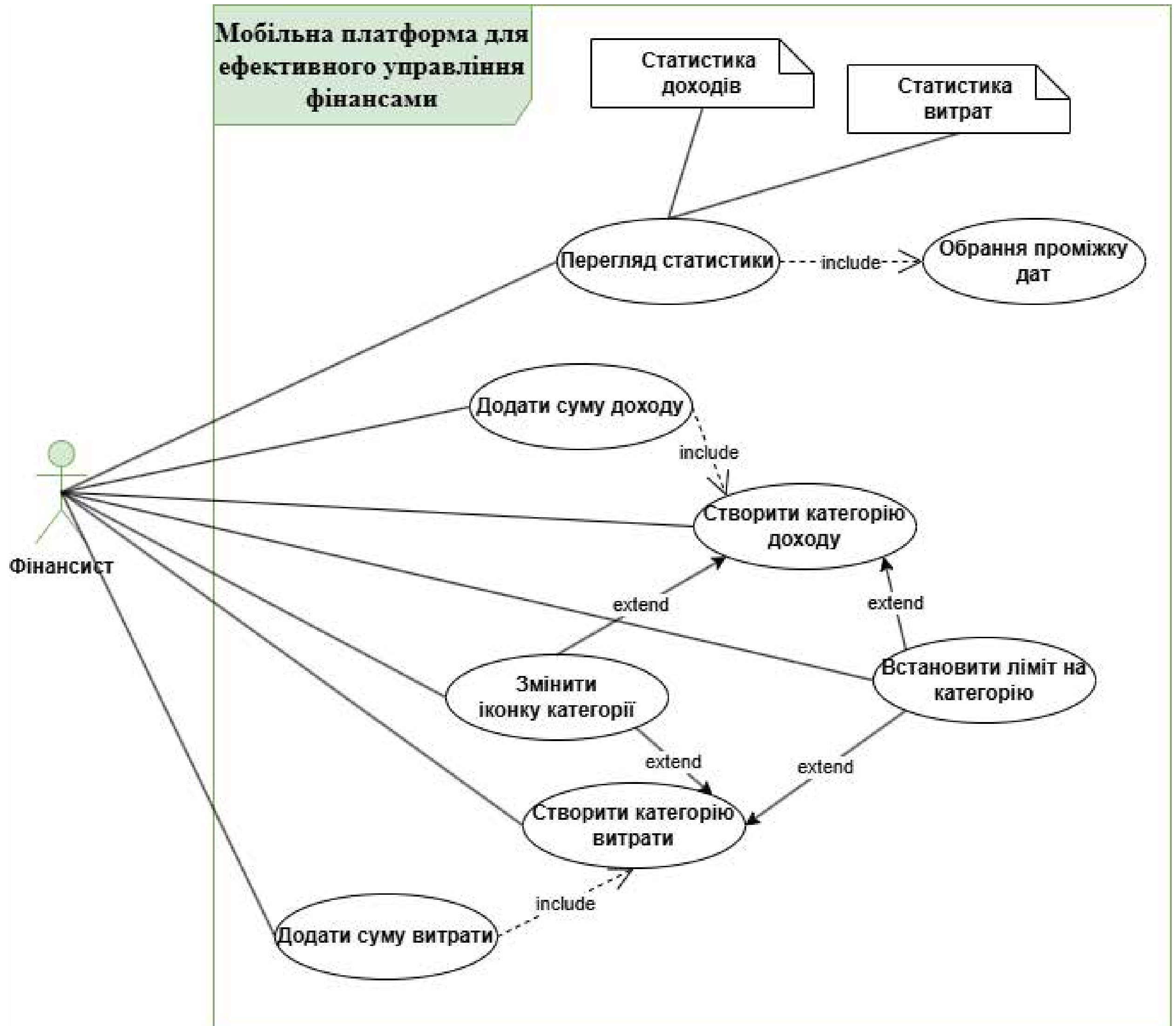

Рис. 2.1 – Діаграма прецедентів

Створена діаграма прецедентів містить акторів:

- “Фінансист”.

Актор «Фінансист» включає такі прецеденти:

- створити категорію доходу;
- додати суму доходу;
- змінити іконку категорії;
- встановити ліміт на категорію;
- створити категорію витрати;
- додати суму витрати;
- перегляд статистики;

- обрання проміжку дат.

Прецеденти певним чином залежать одне від одного.

Розглянемо детальніше вищеописані прецеденти.

Назва випадку використання: Створення категорії доходу

Актор: Фінансист

Опис: Цей варіант використання дозволяє фінансисту створити нову категорію доходу на мобільній платформі фінансового управління. Фінансист визначить категорію, призначить відповідні деталі, такі як назва та опис, і забезпечить її правильне додавання до системи для подальшого використання під час обліку доходу. Ця функціональність допомагає ефективніше організовувати та класифікувати джерела доходу, що дозволяє краще відстежувати фінанси та звітувати.

Основний потік:

1. фінансист входить у систему на платформі та переходить до розділу «Категорії доходу» в головному меню;
2. система відображає список існуючих категорій доходу разом з опцією «Створити нову категорію»;
3. фінансист вибирає опцію «Створити нову категорію»;
4. система пропонує фінансисту ввести назву для нової категорії доходу та необов'язковий опис;
5. фінансист вводить необхідні дані (назва категорії, опис) та надсилає форму;
6. система перевіряє введені дані, щоб переконатися, що назва категорії унікальна та не містить недійсних символів;
7. після успішної перевірки система зберігає нову категорію доходу в базі даних;
8. система підтверджує створення нової категорії доходу та оновлює список доступних категорій доходу;
9. фінансист отримує підтвердження про успішне створення нової категорії доходу.

Альтернативні потоки:

1. якщо Фінансист вводить недійсне ім'я (наприклад, вже існуюче ім'я або спеціальні символи), система відображає повідомлення про помилку та пропонує фінансисту виправити введені дані;
2. якщо Фінансист залишає поле імені або опису порожнім, система пропонує Фінансисту заповнити необхідні дані перед надсиланням;
3. якщо під час створення категорії виникає технічна проблема (наприклад, проблема з базою даних), система повідомить Фінансиста про помилку та запропонує спробувати ще раз пізніше.

Випадок використання «Створення категорії доходу» успішно дозволяє Фінансисту додати нову категорію доходу до системи. Це гарантує, що платформа може ефективно класифікувати джерела доходу, допомагаючи в організації фінансових записів. Дотримуючись основного потоку та належним чином обробляючи будь-які альтернативні потоки (наприклад, недійсні введені дані або системні помилки), система забезпечує безперебійну роботу для Фінансиста. Ця функція розширює можливості платформи щодо організації та відстеження різних джерел доходу.

Розглянемо ще один сценарій використання.

Назва випадку використання: Запис витрати

Актор: Фінансист

Опис: Цей варіант використання дозволяє фінансисту записати витрати на мобільній платформі фінансового управління. Фінансист класифікує витрати, вводить суму, надає опис та вибирає спосіб оплати. Ця функціональність допомагає відстежувати та керувати фінансовими витратами, забезпечуючи точну реєстрацію витрат для подальшого аналізу та складання бюджету.

Основний потік:

1. фінансист входить у мобільну платформу та переходить до розділу «Витрати» з головної панелі інструментів;
2. система відображає список зареєстрованих витрат разом з опцією «Записати нові витрати»;
3. фінансист вибирає опцію «Записати нові витрати»;
4. система пропонує фінансисту заповнити поля;
5. фінансист вводить необхідні дані для витрат і надсилає форму;
6. система перевіряє введені дані, гарантуючи правильність заповнення всіх полів і додатне число в сумі витрат;
7. після успішної перевірки система реєструє витрати в базі даних та оновлює загальний фінансовий звіт;
8. система підтверджує успішне реєстрування витрат і оновлює список зареєстрованих витрат;
9. фінансист отримує підтвердження про успішне реєстрування витрат.

Альтернативні потоки:

1. якщо Фінансист вводить від'ємну суму або нечислове значення, система відображає повідомлення про помилку та просить Фінансиста виправити суму;
2. якщо Фінансист залишає будь-яке обов'язкове поле (наприклад, суму або категорію) порожнім, система відображає повідомлення про помилку, що спонукає Фінансиста заповнити форму перед надсиланням;
3. якщо під час реєстрації витрат виникає технічна проблема (наприклад, помилка бази даних), система відображає повідомлення, щоб повідомити Фінансиста про помилку та запропонувати повторити спробу пізніше.

Варіант використання «Запис витрати» дозволяє фінансисту точно реєструвати та класифікувати фінансові витрати на платформі. Дотримуючись основного потоку та вирішуючи потенційні проблеми за допомогою

альтернативних потоків, система забезпечує відстеження витрат у режимі реального часу, що полегшує фінансисту моніторинг витрат та управління бюджетами. Цей процес підтримує точну фінансову звітність та допомагає підтримувати організований фінансовий огляд, сприяючи ефективному фінансовому управлінню на платформі.

2.2.2 Діаграма послідовності. Діаграма послідовності — це тип діаграми взаємодії в UML (Уніфікована мова моделювання), яка ілюструє, як різні об'єкти в системі взаємодіють один з одним з часом. Зазвичай вона використовується для представлення потоку повідомлень або дій між компонентами або акторами під час виконання конкретного випадку використання [10].

На діаграмі послідовності різні елементи об'єднуються, щоб показати процес взаємодії. Актори, які представляють зовнішні сутності, такі як користувачі або системи, ініціюють та отримують повідомлення. Це можуть бути, наприклад, фінансист, система або зовнішні постачальники послуг у рамках платформи фінансового управління. Поряд з акторами, діаграма також включає об'єкти в системі, такі як служба витрат, база даних або відстеження доходів, які беруть участь у потоці зв'язку.

Повідомлення, що обмінюються між об'єктами, зображуються у вигляді стрілок, які представляють виклики методів, передачу даних або запити на дії. Об'єкти представлені вертикальними пунктирними лініями, які називаються лініями життя, що означають їх активну участь протягом усієї взаємодії. Смуги активації — це невеликі прямокутники, розміщені на цих лініях життя, що вказують періоди, коли об'єкт виконує операцію. Для позначення повернення повідомлення або дії використовуються пунктирні стрілки, що показують результати, надіслані назад до викликаючого об'єкта. Діаграма послідовності відповідає часовій шкалі, де взаємодії відбуваються зверху вниз. Це означає, що діаграма відображає, як кожен актор або об'єкт крок за кроком взаємодіє з системою, що призводить до послідовності операцій. Наприклад, коли фінансист починає процес, такий як «Запис

витрат», він надсилає повідомлення системі, щоб розпочати дію. Потім система перевіряє вхідні дані, обробляє інформацію та взаємодіє з базою даних або службою витрат для оновлення записів. Після завершення транзакції система надсилає підтвердження фінансисту.

Діаграма послідовності, зображена на рис. 2.2, включає наступні об'єкти:

- “Фінансист”;
- “Доходи”;
- “Витрати”;
- “Ліміт”.

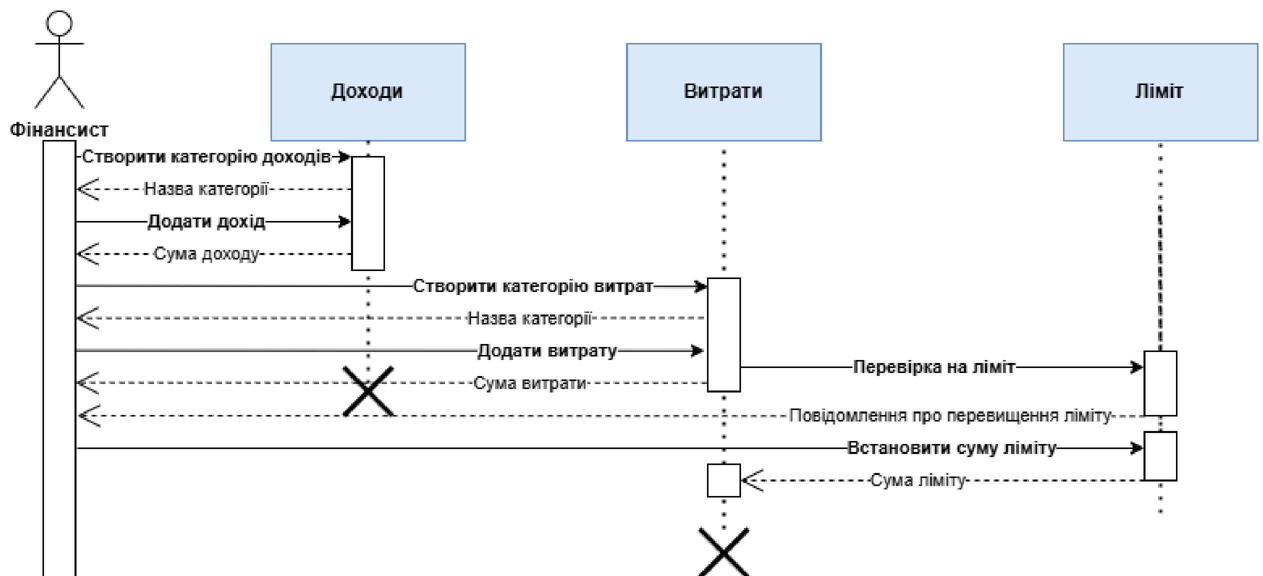

Рис. 2.2 – Діаграма послідовності

Діаграма послідовності ілюструє, як управляються фінансові процеси, включаючи доходи, витрати та ліміти витрат. Вона зображує фінансового експерта, який взаємодіє з трьома ключовими елементами: Дохід, Витрати та Ліміт.

Спочатку фінансовий експерт встановлює категорії доходів, призначаючи їм назви та вказуючи суми. Такий структурований підхід забезпечує чітке відстеження потоків доходів. Аналогічно, створюються категорії витрат із записом назв та сум, що дозволяє систематично організувати витрати.

Діаграма також розглядає ліміти витрат, забезпечуючи механізм моніторингу та коригування фінансових обмежень. Якщо попередньо визначений ліміт перевищено, спрацьовує сповіщення, що спонукає до переоцінки та потенційної зміни допустимого порогу витрат.

Окреслюючи ці взаємодії, діаграма пропонує комплексне уявлення про обробку фінансових даних, що може бути важливим для автоматизації транзакцій та покращення фінансового нагляду.

2.2.3 Діаграма активності. Діаграма діяльності – це тип діаграми UML (Уніфікованої мови моделювання), яка зосереджена на моделюванні динамічної поведінки системи, представляючи потік керування або даних у різних процесах. Ці діаграми ілюструють, як дії, завдання або дії виконуються крок за кроком, а також переходи між ними. Вони особливо корисні для візуалізації робочих процесів, бізнес-процесів та послідовності операцій, необхідних для досягнення певної мети [11].

На діаграмі діяльності дії є основними елементами та представлені у вигляді прямокутників із заокругленими кутами. Кожна дія означає завдання або операцію, наприклад, «Оплата процесу» або «Перевірка введених користувачем даних». Ці дії з'єднані переходами, які є стрілками, що показують перехід від однієї дії до іншої. Переходи запускаються умовами, подіями або діями користувача та допомагають визначити послідовність кроків у процесі.

На початку діаграми діяльності зазвичай є початковий вузол, символізований заповненим чорним колом. Цей вузол вказує на початкову точку процесу. На іншому кінці процес досягає кінцевого вузла, зображеного у вигляді заповненого чорного кола з облямівкою навколо нього, що позначає завершення потоку.

Коли потік керування потребує розгалуження на основі певних умов, вступають у гру вузли прийняття рішень. Вони представлені ромбами та спрямовують потік різними шляхами залежно від оцінки умов. Натомість

вузли злиття використовуються для об'єднання кількох шляхів в один потік, спрощуючи діаграму, коли кілька умов сходяться в певній точці.

Розроблена діаграма активності представлена на рис. 2.3

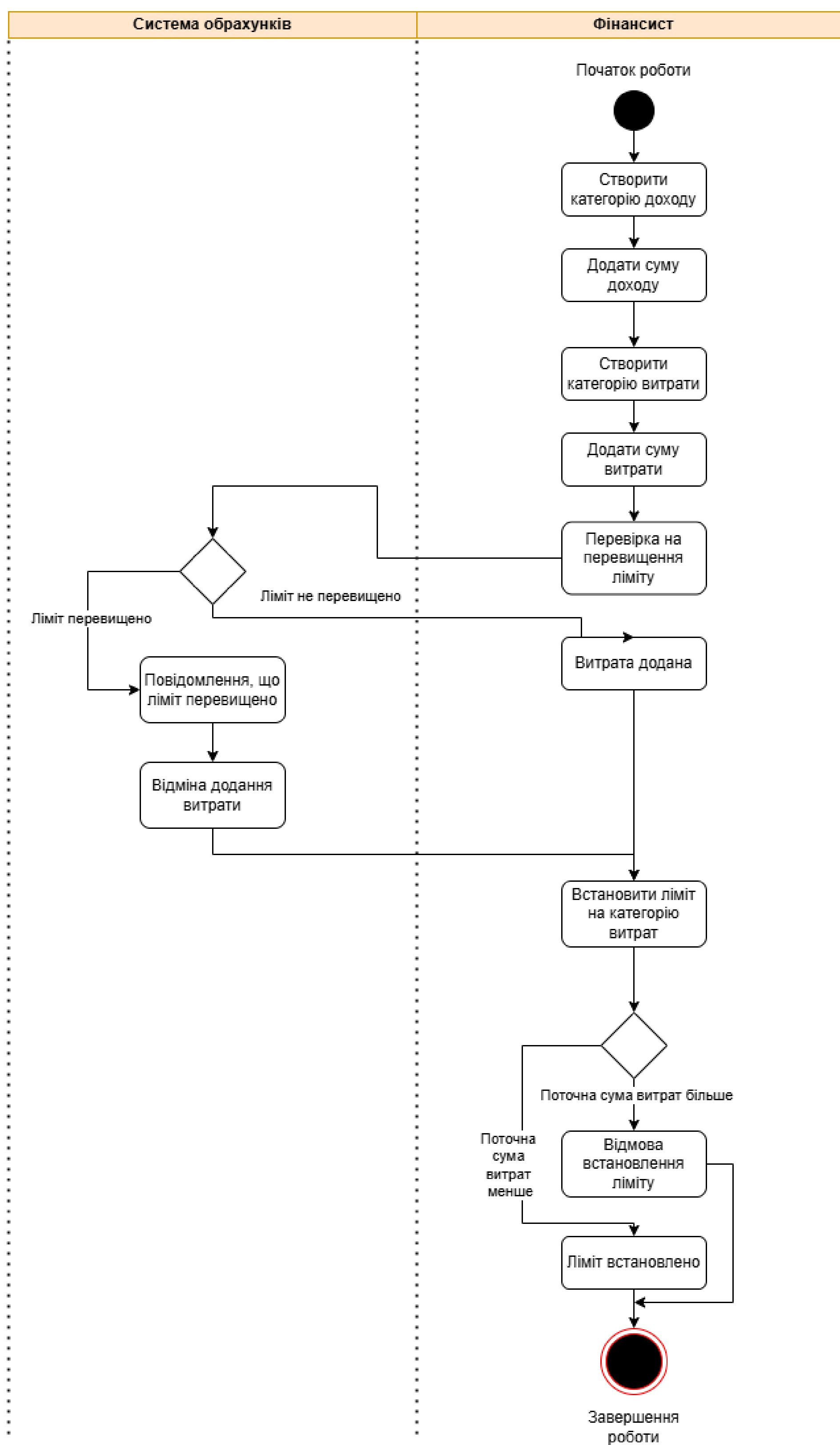

Рис. 2.3 – Діаграма активності

Ця діаграма активності відображає процес управління фінансами, зосереджуючись на додаванні доходів, витрат та контролі лімітів. Вона складається з двох основних областей: Система обрахунків і Фінансист, які взаємодіють для забезпечення фінансового контролю.

Процес починається з визначення доходів: створюються відповідні категорії та вводяться суми. Аналогічно налаштовуються витрати – додаються категорії та призначаються суми витрат. Як тільки дані про витрати внесені, система перевіряє, чи не перевищено встановлений ліміт. У разі перевищення користувач отримує повідомлення, і додавання витрати скасовується. Якщо ж ліміт не перевищено, витрата успішно фіксується.

Діаграма також демонструє механізм встановлення обмежень на витрати. Якщо загальна сума витрат вже перевищує пропонований ліміт, його встановлення блокується. Якщо ж поточні витрати нижчі за обмеження, система успішно застосовує встановлене значення. Завершуючи всі операції, процес управління фінансами приходить до логічного завершення.

Ця діаграма допомагає візуалізувати ключові кроки фінансового менеджменту та оптимізувати автоматизацію цих процесів.

2.3 Абстракції предметної області

Абстракції предметної області стосуються процесу спрощення та представлення складних концепцій, процесів або систем таким чином, щоб їх було легше зрозуміти та керувати ними. У контексті проєктування систем абстракція допомагає зосередитися на основних характеристиках системи, ігноруючи непотрібні деталі. Це особливо корисно для розуміння складних систем, таких як мобільна платформа для управління фінансами, де є багато компонентів та взаємодій [12].

У мобільній платформі для управління фінансами абстракції використовуються для визначення ключових сутностей, процесів та взаємодій

у системі. Ці абстракції можуть включати моделі, структури даних та елементи інтерфейсу користувача, які представляють реальні концепції фінансів, такі як категорії доходів, категорії витрат, транзакції та фінансові звіти.

Наприклад, однією з основних абстракцій у системі управління фінансами є концепція транзакцій. Ця абстракція дозволяє системі представляти різні типи фінансових дій, такі як доходи та витрати, без необхідності турбуватися про специфіку контексту кожної транзакції. Аналогічно, категорії доходів та витрат забезпечують спосіб організації фінансових даних у змістовні групи, що полегшує користувачам відстеження та аналіз їхніх фінансів.

Ще однією важливою абстракцією є користувач у системі. Користувача можна розглядати як центральну сутність, яка взаємодіє з платформою, або як особу, яка керує своїми фінансами, або як адміністратора, який контролює загальну функціональність системи. Роль користувача абстрагується на різні типи, такі як звичайний користувач або фінансовий оператор, кожен з яких має певні дозволи та рівні доступу (рис. 2.4).

Абстракція: Транзакція
Властивості: Дата/Час створення Тип транзакції Сума
Обов'язки: Зробити транзакцію Редагувати транзакцію Видалити транзакцію Перевірка на ліміт

Абстракція: Категорія
Назва категорії Тип категорії Іконка Поточна сума Список транзакцій
Обов'язки: Створити категорію Налаштувати категорію Додати транзакцію Встановити ліміт

Рис. 2.4 – Абстракції предметної області

У сфері фінансового менеджменту звіти та статистика є ключовими абстракціями, що відображають результат аналізу фінансових даних. Ці звіти

дають користувачам уявлення про їхнє фінансове здоров'я, висвітлюючи тенденції доходів, витрат та заощаджень. Вони абстрагують складні розрахунки та аналізи, що відбуваються за лаштунками, надаючи користувачам легкі для розуміння візуалізації або зведення їхнього фінансового стану.

Ця схема (рис. 2.4) представляє дві ключові абстракції предметної області: Транзакція та Категорія, кожна з яких має власні властивості та обов’язки.

Абстракція "Транзакція" містить базові атрибути, необхідні для обліку фінансових операцій: дату та час створення, тип транзакції та її суму. Основні функції, які вона виконує, включають здійснення, редагування та видалення транзакції, а також перевірку на відповідність встановленому ліміту.

Абстракція "Категорія" використовується для класифікації фінансових потоків. Вона містить назву та тип категорії, іконку, поточну суму та список транзакцій, що входять до неї. Її основні функції включають створення та налаштування категорії, додавання транзакцій та встановлення ліміту.

Завдяки таким абстракціям можна ефективно структурувати фінансові дані та забезпечити їхню керованість.

2.4 Діаграма класів

Діаграма класів – це основний компонент уніфікованої мови моделювання (UML), який використовується для зображення статичної структури системи шляхом ілюстрації її класів, атрибутів, методів та зв'язків між ними. Діаграми класів відіграють життєво важливу роль в об'єктно-орієнтованому проєктуванні, пропонуючи чітке представлення архітектури та структури системи. Це допомагає як розробникам, так і зацікавленим сторонам зрозуміти, як різні сутності в системі взаємопов'язані [13].

Клас є фундаментальною одиницею діаграми класів і служить планом для створення об'єктів. Кожен клас зазвичай представлений у вигляді

прямокутника, розділеного на три секції: верхня секція містить назву класу, середня секція містить його атрибути, а нижня секція показує його методи або операції. Наприклад, у програмі для управління фінансами клас Transaction може мати такі атрибути, як сума, дата та категорія, а також методи, такі як addTransaction() або generateReport().

Атрибути визначають характеристики або дані, пов'язані з класом. Наприклад, клас User може мати такі атрибути, як ім'я, електронна пошта та баланс. Ці атрибути представляють стан об'єкта, створеного з класу. Методи, з іншого боку, представляють поведінку або операції, які може виконувати об'єкт. Вони визначають, як можна маніпулювати даними або взаємодіяти з ними, наприклад, обробка транзакцій або створення звітів у випадку платформи фінансового управління.

Діаграми класів також ілюструють зв'язки між різними класами. Одним з найпоширеніших зв'язків є асоціація, яка вказує на простий зв'язок між двома класами. Наприклад, клас User може мати асоціацію з класом Transaction, що означає, що користувач може мати кілька транзакцій. Окрім асоціацій, існують інші типи зв'язків, такі як агрегація та композиція, які описують, як класи є частинами один одного. Агрегація представлена порожнистим ромбом і вказує на те, що один клас є частиною іншого, але може існувати незалежно. Композиція, позначена заповненим ромбом, означає сильніший зв'язок, де частина не може існувати без цілого.

Ще одним ключовим зв'язком є узагальнення (або успадкування), яке моделює зв'язок "є-а" між класами. Це корисно для визначення спільних характеристик у батьківському класі, які можуть бути успадковані підкласом. Наприклад, клас RegularUser може успадковуватися від класу User, отримуючи властивості та методи батьківського класу, а також додаючи його специфічну функціональність. Крім того, реалізація використовується для показу зв'язку між класом та інтерфейсом, зазвичай у випадках, коли клас реалізує поведінку, визначену інтерфейсом.

Для цього проєкту було створено спеціальну діаграму класів з використанням об’єктно-орієнтованого підходу (рис. 2.5).

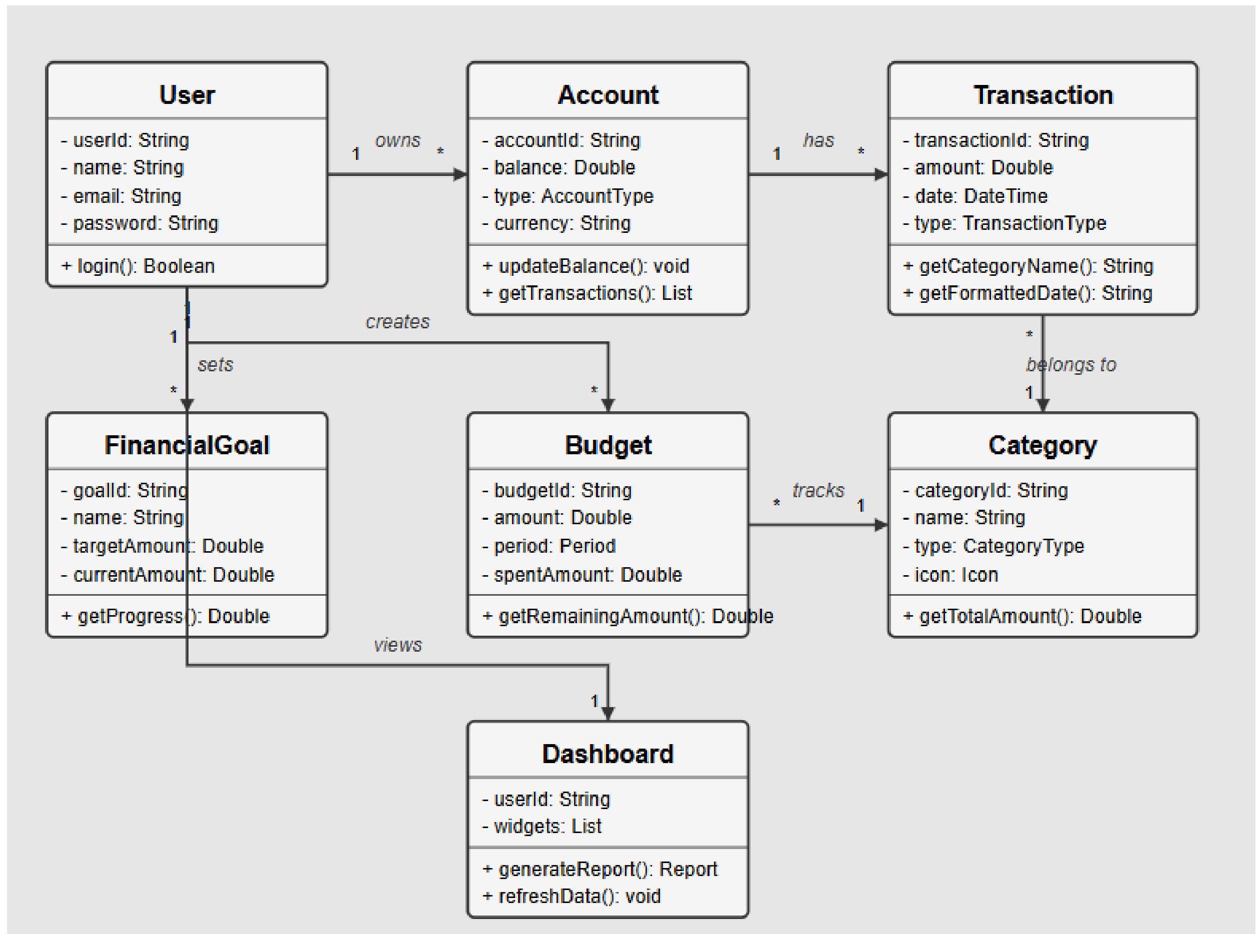

Рис. 2.5 – Діаграма класів

Ця діаграма класів моделює структуру фінансової системи, представляючи ключові об'єкти та їхні взаємозв’язки. Вона містить основні класи, включаючи Транзакцію, Категорію та Користувача, кожен з яких має відповідні властивості та методи для управління фінансами.

Клас Транзакція відповідає за збереження фінансових операцій, містить атрибути, такі як сума, тип, дата та час створення. Він забезпечує базові функції, включаючи додавання, редагування та видалення транзакцій, а також перевірку їх відповідності встановленим лімітам.

Клас Категорія служить для організації фінансових потоків, включаючи доходи та витрати. Він містить інформацію про назву категорії,

тип, іконку та пов’язані транзакції. Основні методи дозволяють створювати, оновлювати та встановлювати ліміти для категорій.

Клас Користувач представляє особу, яка керує фінансами. Він має властивості, такі як ім'я, баланс рахунку та список категорій. Користувач може взаємодіяти з транзакціями, налаштовувати бюджет і встановлювати фінансові обмеження.

Завдяки цій структурі забезпечується чітке моделювання предметної області, що дозволяє ефективно обробляти фінансові дані та автоматизувати процеси управління бюджетом.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Логічна модель даних

Логічна модель даних слугує планом для організації та структурування даних у системі, зосереджуючись на зв'язках та розташуванні сутностей даних без прив'язки до будь-якої конкретної технології впровадження. Ця модель надає концептуальний огляд того, як дані повинні бути організовані, та гарантує, що структура системи відповідає бізнес-вимогам. Вона відображає спосіб логічного зв'язку даних, забезпечуючи основу для більш детального проєктування та впровадження бази даних [14].

По суті, логічна модель даних визначає сутності або об'єкти, які система оброблятиме. Ці сутності, такі як Користувач, Транзакція, Категорія доходу та Категорія витрат, представляють ключові об'єкти в системі. Кожна сутність пов'язана з атрибутами, які визначають характеристики сутності. Наприклад, сутність Користувач може мати такі атрибути, як user_id, ім'я, електронна пошта та баланс. Ці атрибути є фундаментальними точками даних, які будуть зберігатися для кожної сутності.

Логічна модель даних також визначає зв'язки між сутностями. Ці зв'язки описують, як сутності взаємодіють або пов'язані одна з одною. Наприклад, Користувач може мати кілька Транзакцій, що пов'язують дві сутності разом. Аналогічно, Транзакція може належати до Категорії доходів або Категорії витрат, встановлюючи зв'язок категоризації. Зв'язки можуть бути різних типів, таких як один до одного, один до багатьох або багато до багатьох, залежно від конкретних потреб бізнесу.

Первинні ключі призначаються кожній сутності для унікальної ідентифікації записів у цій сутності. Наприклад, сутність Користувач матиме унікальний user_id, який відрізняє кожного користувача. Аналогічно, кожна транзакція може мати унікальний transaction_id. Ці первинні ключі необхідні

для підтримки цілісності даних та встановлення чітких зв'язків між різними сутностями.

Окрім первинних ключів, модель також використовує зовнішні ключі для зв'язку сутностей між собою. Зовнішній ключ в одній сутності посилається на первинний ключ іншої сутності, допомагаючи підтримувати зв'язок між пов'язаними даними. Наприклад, сутність Транзакція може включати user_id як зовнішній ключ для посилання на пов'язаного користувача, який здійснив транзакцію [14].

Ще одним важливим аспектом логічної моделі даних є нормалізація, процес організації даних для зменшення надлишковості та уникнення аномалій даних. Нормалізація забезпечує ефективність структури бази даних та підтримку цілісності даних, гарантуючи, що пов'язані дані зберігаються в окремих логічних таблицях. Ця практика допомагає оптимізувати зберігання та пошук даних, тим самим покращуючи загальну продуктивність бази даних.

Логічна модель даних відіграє життєво важливу роль у розробці системи баз даних, пропонуючи чітке представлення сутностей, їхніх атрибутів та того, як вони взаємодіють. Вона забезпечує загальне уявлення про структуру даних, що є вирішальним для узгодження дизайну системи з бізнес-цілями. Хоча вона не заглиблюється в деталі фізичної реалізації бази даних, логічна модель гарантує, що дані структуровані таким чином, щоб підтримувати функціональні потреби системи.

Наприклад, у контексті платформи фінансового управління логічна модель даних визначатиме ключові сутності, такі як Користувач, Транзакція, Категорія Доходу, Категорія Витрат та Бюджет. Вона визначатиме, як користувач може створювати та класифікувати доходи та витрати, а також як ці записи пов'язані між собою. Модель також окреслюватиме взаємозв'язки, наприклад, як кожна транзакція пов'язана з певним користувачем та класифікується за певною категорією доходів або витрат.

Перевага логічної моделі даних полягає в тому, що вона забезпечує основу для організації даних таким чином, щоб вони могли масштабуватися та

розвиватися відповідно до змінних потреб бізнесу. Вона дозволяє гнучко та ефективно проєктувати систему, гарантуючи, що база даних може підтримувати свої функціональні вимоги, зберігаючи при цьому цілісність та узгодженість даних.

Підсумовуючи, логічна модель даних виступає як план для структурування даних у системі. Вона забезпечує чіткий огляд сутностей, їх атрибутів та зв'язків між ними, гарантуючи, що система організована таким чином, що відповідає бізнес-цілям. Закладаючи цю основу, модель допомагає керувати наступними етапами проєктування бази даних, гарантуючи, що система є ефективною, масштабованою та здатною відповідати як поточним, так і майбутнім вимогам.

Логічна модель системи представлена на рисунку 3.1.

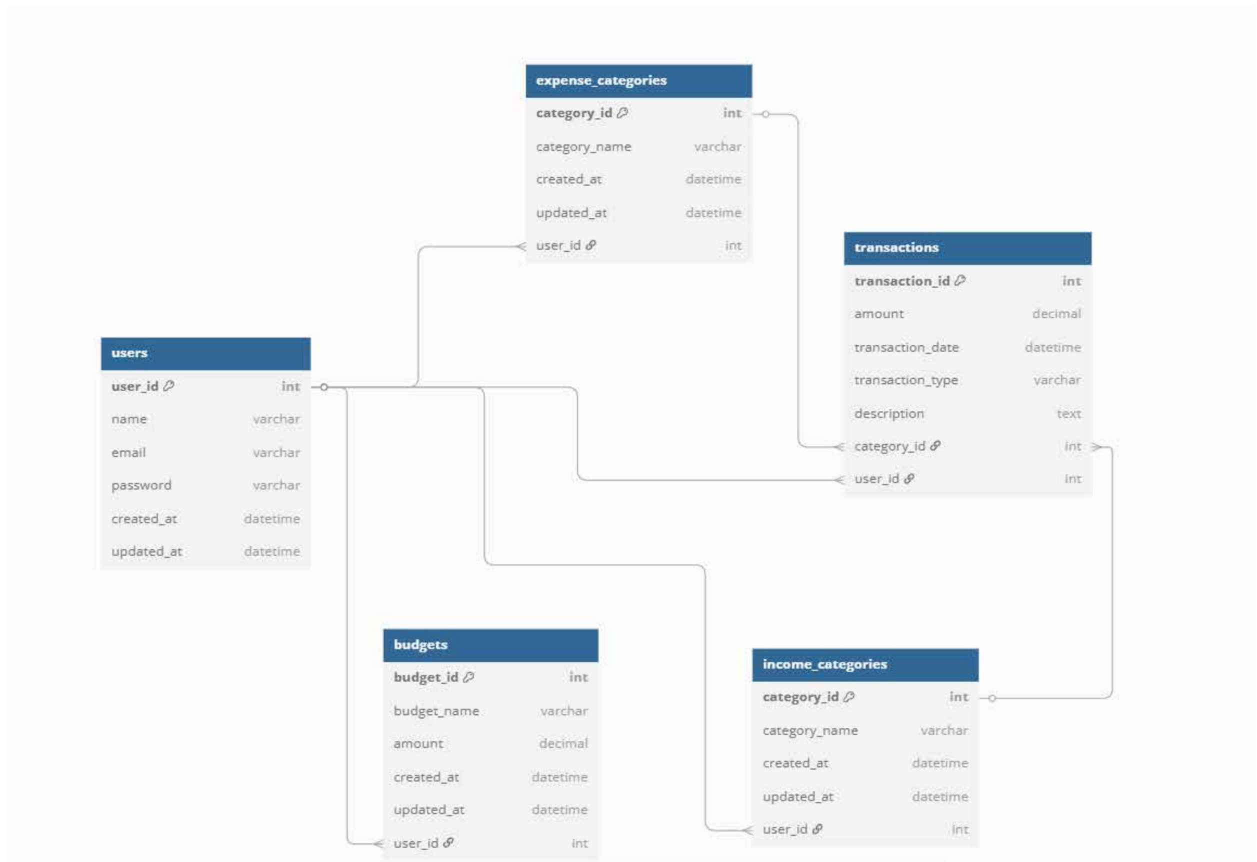

Рис. 3.1 – ER-діаграма

Проєкт бази даних для мобільної платформи для ефективного фінансового управління складається з п'яти ключових таблиць: Користувачі, Категорії доходів, Категорії витрат, Транзакції та Бюджети. Кожна з цих

таблиць структурована для зберігання даних, пов'язаних з фінансовою діяльністю користувача та управлінням бюджетом. Зв'язки між таблицями визначаються за допомогою зовнішніх ключів, що забезпечує цілісність даних та належне зв'язування пов'язаних записів.

Таблиця **User**

Ця таблиця зберігає інформацію про користувачів платформи. Кожен користувач має унікальний обліковий запис з пов'язаними з ним даними, такими як ім'я, електронна пошта та пароль. Ця таблиця є центральною для системи, оскільки вона пов'язує всі інші сутності (категорії, транзакції, бюджети) з окремими користувачами.

Поля:

1. user_id (int, Первинний ключ): Унікальний ідентифікатор для кожного користувача;
2. name (varchar): Повне ім'я користувача;
3. email (varchar): Адреса електронної пошти користувача;
4. password (varchar): Пароль користувача (у хешованій формі);
5. created_at (datetime): Дата та час створення облікового запису користувача;
6. updated_at (datetime): Дата та час останнього оновлення даних облікового запису користувача.

Таблиця **IncomeCategories**

Ця таблиця зберігає категорії доходів. Вона дозволяє користувачам визначати конкретні типи доходів, такі як зарплата, фріланс, інвестиції тощо. Кожна категорія доходів пов'язана з певним користувачем.

Поля:

1. category_id (int, Первинний ключ): Унікальний ідентифікатор для кожної категорії доходів;
2. category_name (varchar): Назва категорії доходів (наприклад, "Зарплата", "Фріланс");
3. created_at (datetime): Дата та час створення категорії;

4. updated_at (datetime): Дата та час останнього оновлення категорії;
5. user_id (int, Зовнішній ключ): Посилання на user_id у таблиці Користувачі, що вказує, до якого користувача належить ця категорія.

Таблиця **ExpenseCategories**

Ця таблиця зберігає категорії витрат. Вона дозволяє користувачам класифікувати свої витрати, такі як продукти харчування, оренда, комунальні послуги тощо. Як і категорії доходів, кожна категорія витрат пов'язана з користувачем.

Поля:

1. category_id (int, Primary Key): Унікальний ідентифікатор для кожної категорії витрат;
2. category_name (varchar): Назва категорії витрат (наприклад, "Продукти", "Оренда");
3. created_at (datetime): Дата та час створення категорії;
4. updated_at (datetime): Дата та час останнього оновлення категорії;
5. user_id (int, Foreign Key): Посилання на user_id у таблиці Users, що вказує, до якого користувача належить ця категорія.

Таблиця **Transactions**

У цій таблиці зберігаються фінансові транзакції, здійснені користувачами. Транзакція може бути як доходом, так і витратами, залежно від transaction_type. Кожна транзакція пов'язана з категорією (доходом або витратами) та користувачем.

Поля:

1. transaction_id (int, Primary Key): Унікальний ідентифікатор для кожної транзакції;
2. amount (decimal): Грошова сума для транзакції (додатна для доходів, від'ємна для витрат);
3. transaction_date (datetime): Дата та час здійснення транзакції;

4. transaction_type (varchar): Визначає, чи є транзакція доходом, чи витратою (може мати значення типу "дохід" або "витрати");
5. description (text): Опис транзакції (необов'язково);
6. category_id (int, Foreign Key): Посилання на category_id у таблиці "Категорії доходів" або "Категорії витрат", залежно від типу транзакції;
7. user_id (int, Foreign Key): Посилання на user_id у таблиці "Користувачі", що пов'язує транзакцію з користувачем.

Таблиця **Budgets**

Ця таблиця дозволяє користувачам встановлювати та відстежувати бюджети. Кожен бюджет пов'язаний з певним користувачем і містить відомості про назву та суму бюджету.

Поля:

1. budget_id (int, Permain Key): Унікальний ідентифікатор для кожного бюджету;
2. budget_name (varchar): Назва бюджету (наприклад, "Щомісячний бюджет");
3. amount (decimal): Сума грошей, виділена на бюджет;
4. created_at (datetime): Дата та час створення бюджету;
5. updated_at (datetime): Дата та час останнього оновлення бюджету;
6. user_id (int, Foreign Key): Посилання на user_id у таблиці Users, що вказує, якому користувачеві належить цей бюджет.

Зв'язки між різними таблицями в базі даних розроблені для встановлення чітких зв'язків між користувачами та їхніми фінансовими даними. Наприклад, між користувачами та категоріями доходів існує зв'язок «один до багатьох». Це означає, що користувач може створювати кілька категорій доходів, але кожна категорія пов'язана лише з одним користувачем. Цей зв'язок підтримується через зовнішній ключ, де user_id у таблиці «Категорії доходів» посилається на user_id у таблиці «Користувачі».

Аналогічно, зв'язок між користувачами та категоріями витрат також є «один до багатьох». Користувач може мати багато категорій витрат, але кожна категорія витрат пов'язана з певним користувачем. Це досягається за допомогою зовнішнього ключа, де user_id у таблиці «Категорії витрат» посилається на user_id у таблиці «Користувачі».

Що стосується транзакцій, база даних встановлює ще один зв'язок «один до багатьох» з користувачами. Це означає, що кожен користувач може мати кілька транзакцій, але кожна окрема транзакція належить лише одному користувачеві. Зовнішнім ключем, який встановлює цей зв'язок, є user_id у таблиці «Транзакції», який вказує на user_id у таблиці «Користувачі».

Що стосується транзакцій, то як категорії доходів, так і категорії витрат пов'язані з таблицею «Транзакції» за принципом «один до багатьох». Кожна транзакція пов'язана з однією категорією (або доходом, або витратами), але кожна категорія може бути пов'язана з кількома транзакціями. Цей зв'язок підтримується через category_id у таблиці «Транзакції», який посилається на відповідний ідентифікатор категорії в таблицях «Категорії доходів» або «Категорії витрат», залежно від характеру транзакції.

Зв'язок між користувачами та бюджетами також є «один до багатьох». Користувач може створювати кілька бюджетів, але кожен бюджет пов'язаний з одним користувачем. Це представлено зовнішнім ключем, де user_id у таблиці «Бюджети» посилається на user_id у таблиці «Користувачі».

Ці зв'язки створюють добре організовану структуру, яка пов'язує дані користувача в різних таблицях, забезпечуючи ефективне керування та отримання фінансової інформації.

3.2 Вибір системи управління базою даних та її реалізація

Система керування базами даних (СКБД) – це спеціалізоване програмне забезпечення, призначене для ефективного створення, зберігання, керування та маніпулювання базами даних. Вона діє як посередник між

користувачами та фактичними даними, забезпечуючи стабільний доступ до інформації під час обробки різних операцій, таких як організація даних, запити, оновлення та безпечний багатокористувацький доступ. Крім того, СКБД забезпечує цілісність даних, резервне копіювання та відновлення, а також контроль паралельності для запобігання конфліктам під час одночасних операцій з даними.

СКБД поділяються на дві широкі категорії: реляційні та нереляційні системи. Основна відмінність між ними полягає в тому, як вони організовують та отримують дані [15].

Реляційні системи керування базами даних (РСБД) упорядковують дані у структуровані таблиці, що складаються з рядків і стовпців, де кожна таблиця представляє певну сутність. Зв'язки між таблицями встановлюються за допомогою ключів, таких як первинні та зовнішні ключі. Ці системи працюють на основі заздалегідь визначеної схеми, яка визначає структуру даних. SQL (структурована мова запитів) – це стандартна мова, що використовується в РСБД для визначення, маніпулювання та запитів даних. СУБД (системи управління реляційними базами даних) відомі підтримкою властивостей ACID (атомірність, узгодженість, ізоляція, довговічність), які забезпечують надійні та узгоджені транзакції. Такі системи, як MySQL, PostgreSQL та Oracle, є популярними прикладами цього типу. Вони добре підходять для застосувань, де дані мають високий рівень структурованості та необхідні узгоджені зв'язки, таких як фінанси, системи управління запасами та корпоративні додатки.

З іншого боку, нереляційні СУБД, які зазвичай називають NoSQL-базами даних, використовують більш гнучкі структури для зберігання даних, включаючи пари ключ-значення, документи, сховища з широкими стовпцями або графічні моделі. Ці системи не мають схем або мають динамічні схеми, що дозволяє кожному запису мати різні структури. Вони особливо ефективні для обробки великих обсягів напівструктурованих або неструктурованих даних і чудово справляються з ситуаціями, що вимагають високої продуктивності, масштабованості та гнучких моделей даних. На відміну від СУБД, NoSQL-

бази даних часто надають перевагу кінцевій узгодженості (на відміну від суворої відповідності ACID), що забезпечує кращу продуктивність та доступність у розподілених системах. Прикладами NoSQL баз даних є MongoDB (орієнтована на документи), Redis (сховище ключ-значення), Cassandra (сховище сімейств стовпців) та Neo4j (графова база даних) [16].

Загалом, реляційні бази даних забезпечують структуроване та надійне середовище для керування узгодженими даними зі складними зв'язками, що робить їх ідеальними для традиційних бізнес-додатків. На противагу цьому, нереляційні бази даних пропонують гнучкість, масштабованість та швидкість, що є важливими для сучасних веб-додатків, рішень для великих даних та платформ аналітики в реальному часі. Вибір між ними залежить від конкретних потреб проєкту, характеру даних та вимог до продуктивності.

У контексті розробки мобільної платформи для ефективного управління фінансами, вибір відповідної системи керування базами даних (СКБД) є вирішальним рішенням, яке безпосередньо впливає на продуктивність, масштабованість та взаємодію з користувачем застосунку. Оскільки застосунок розроблено з використанням Flutter та Dart і орієнтований на мобільні пристрої, важливо обрати легке, швидке та офлайн-рішення.

Одним із найбільш підходящих варіантів для цього сценарію є SQLite. Це вбудований реляційний механізм баз даних, який широко використовується в мобільній розробці завдяки своєму невеликому розміру, високій ефективності та легкій інтеграції з Flutter за допомогою таких пакетів, як sqflite.

SQLite пропонує кілька ключових переваг для мобільного застосунку для управління фінансами:

- повністю працює на пристрої, що гарантує, що користувачі можуть записувати та керувати своїми фінансами навіть без підключення до Інтернету;

- для однокористувацьких мобільних застосунків SQLite забезпечує швидкі операції читання/запису з мінімальною затримкою;
- він не потребує окремого серверного процесу, що спрощує розгортання та керування в мобільному застосунку;
- плавно інтегрується з Flutter через пакети, що підтримуються спільнотою, такі як sqflite, що спрощує операції з базою даних за допомогою Dart.

Схема бази даних розроблена для зберігання та керування важливими фінансовими даними користувачів, такими як доходи, витрати, категорії та бюджети. Реалізація починається з визначення таблиць для користувачів, категорій доходів, категорій витрат, транзакцій та бюджетів. Кожна таблиця містить відповідні поля, такі як імена, описи, суми та позначки часу. Обмеження зовнішнього ключа використовуються для підтримки цілісності посилань між пов'язаними сутностями.

За допомогою плагіна sqflite команди SQL для створення та запитів таблиць пишуться безпосередньо в Dart. Ці команди зазвичай обгорнуті в класи сервісів, які інкапсулюють логіку для вставки, оновлення, видалення та отримання даних. Наприклад, клас TransactionService оброблятиме всю логіку, пов'язану з транзакціями користувачів, забезпечуючи розділення завдань та зручність обслуговування.

Вибір SQLite як СУБД для цього мобільного застосунку пропонує оптимальний баланс між продуктивністю, простотою та функціональністю. Він підтримує потребу застосунку в локальному сховищі, дозволяє надійно обробляти реляційні дані та бездоганно вписується в екосистему Flutter. Такий підхід гарантує, що платформа є надійною, адаптивною та доступною для користувачів, які керують своїми фінансами в дорозі.

3.3 Архітектура програмного забезпечення

Архітектура програмного забезпечення являє собою фундаментальний дизайн програмної системи, що окреслює, як її компоненти структуровані, взаємодіють та працюють разом для виконання як функціональних, так і нефункціональних вимог. Вона виступає як план, який керує процесом розробки, визначаючи ключові елементи системи, їхні обов'язки та механізми зв'язку між ними [17].

По суті, архітектура програмного забезпечення визначає, як завдання розподіляються між модулями, як дані проходять через систему та як різні частини інтегровані для досягнення загальної продуктивності, надійності, масштабованості та зручності обслуговування. Так само, як структура вимагає ретельно продуманого архітектурного плану для забезпечення стабільності та зручності використання, програмна система потребує цілісної структури, щоб забезпечити її ефективну роботу та плавний розвиток з часом.

Ефективна архітектура відіграє вирішальну роль у підтримці масштабованості системи, дозволяючи додавати нові функції без порушення існуючої функціональності. Вона також сприяє зручності обслуговування, дозволяючи розробникам ізолювати та вирішувати проблеми з мінімальним впливом на решту системи. Крім того, вона підвищує безпеку системи, гарантуючи захист даних та операцій від несанкціонованого доступу, та оптимізує використання ресурсів для забезпечення високої продуктивності.

Зазвичай використовується кілька архітектурних стилів, залежно від цілей проєкту. До них належать монолітні системи, де вся функціональність міститься в одній кодовій базі; клієнт-серверні моделі, які відокремлюють постачальників послуг від споживачів; багаторівневі архітектури, які організовують код на різних рівнях, таких як представлення, бізнес-логіка та управління даними; мікросервіси, які розділяють програму на незалежні сервіси, що взаємодіють через API; та подієво-керовані проєкти, де компоненти реагують на згенеровані події.

Підсумовуючи, архітектура програмного забезпечення — це не просто впорядкування коду; це стратегічна дисципліна, яка закладає основу для успіху, адаптивності та довгострокової цінності системи. Вона гарантує, що програмне забезпечення може зростати разом з потребами користувачів, залишатися безпечним та ефективним, а також бути простішим у розумінні та обслуговуванні протягом усього його життєвого циклу.

Рішення про впровадження «чистої архітектури» для розробки мобільної платформи для ефективного управління фінансами було прийнято на основі комплексної оцінки цілей проєкту, майбутньої масштабованості та потреби в підтримуваному коді. Цей архітектурний шаблон пропонує високоструктурований та модульний підхід до проєктування програмного забезпечення, який ідеально відповідає функціональним та нефункціональним вимогам програми.

«Чиста архітектура» сприяє розділенню обов'язків, розділяючи кодову базу на окремі шари: презентація, домен та дані. Кожен з цих шарів має чітко визначену відповідальність та взаємодіє з іншими через чітко визначені інтерфейси. Для програми для управління фінансами, яка включає складну бізнес-логіку, таку як розрахунок бюджетів, категоризація транзакцій та створення звітів, таке розділення гарантує, що основна логіка залишається незалежною від інтерфейсу користувача та технології бази даних [18].

Однією з ключових причин вибору «чистої архітектури» є її масштабованість та гнучкість. У міру розвитку програми можуть знадобитися додаткові функції, такі як хмарна синхронізація, підтримка кількох пристроїв або розширена аналітика. «Чиста архітектура» дозволяє впроваджувати ці зміни без порушення існуючої кодової бази завдяки своєму слабо пов'язаному дизайну.

Крім того, тестованість була критичним фактором у цьому рішенні. Завдяки «Чистій архітектурі» бізнес-правила та варіанти використання ізольовані від зовнішніх залежностей, що спрощує написання модульних тестів для логіки застосунку. Це особливо важливо у фінансовому

програмному забезпеченні, де точність та надійність є надзвичайно важливими.

Крім того, використання Flutter як фронтенд-фреймворку добре поєднується з «Чистою архітектурою». Реактивна модель інтерфейсу користувача Flutter доповнює структуру «Чистої архітектури», дозволяючи використовувати сучасні інструменти управління станом, такі як Provider, Riverpod або Bloc, які допомагають чітко обробляти оновлення інтерфейсу користувача та взаємодію з користувачем.

Підсумовуючи, «Чиста архітектура» була обрана за її здатність забезпечити надійну, тестовану, зручну в обслуговуванні та перспективну основу для створення застосунку для управління фінансами. Вона забезпечує високоякісну структуру програмного забезпечення, яка підтримує як поточні вимоги, так і майбутні вдосконалення з мінімальним технічним боргом.

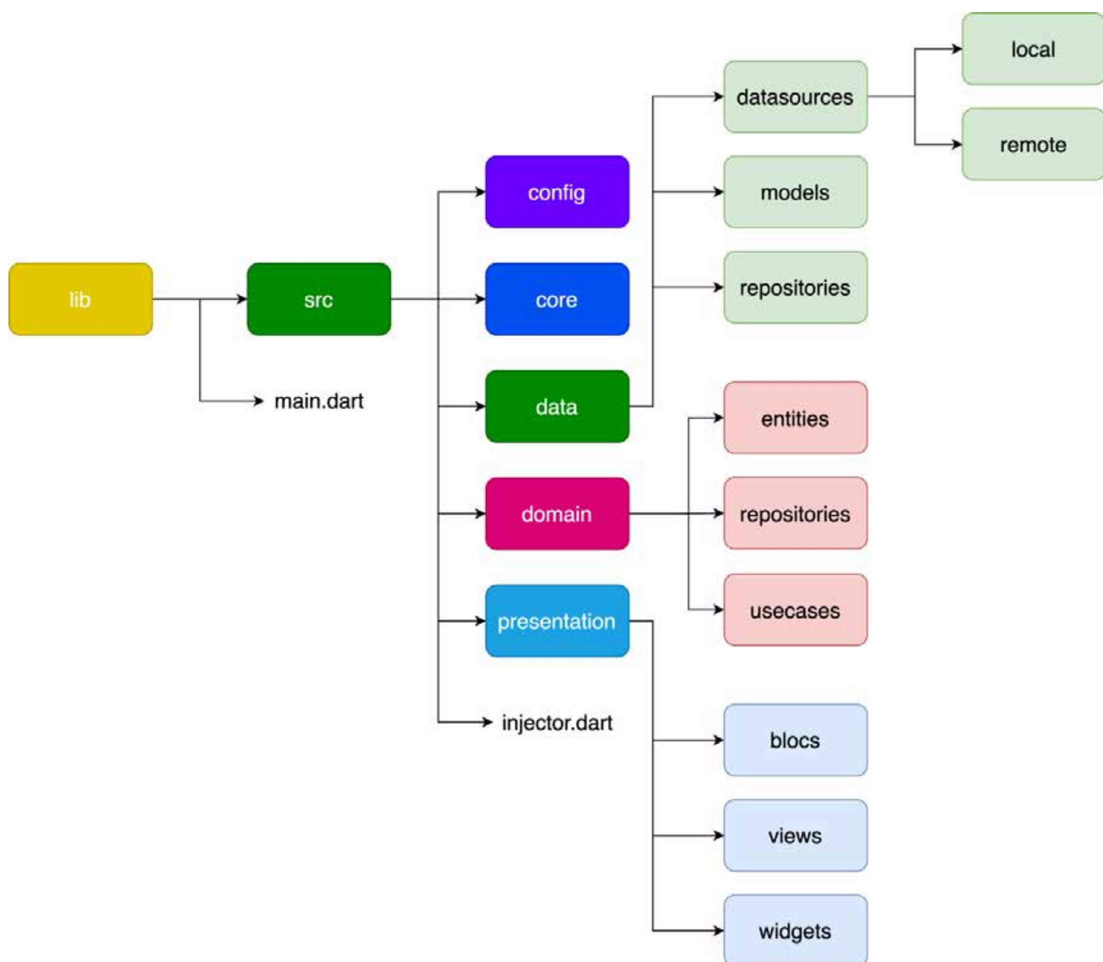

Рис. 3.2 – Чиста архітектура ПЗ

3.4 Організаційна структура програмного забезпечення

3.4.1 Діаграма пакетів. Діаграма пакета — це структурне представлення, що використовується в UML для візуалізації модульної організації програмної системи. Вона показує, як система розділена на окремі пакети, які виступають логічними контейнерами для пов'язаних класів, інтерфейсів та компонентів. Ці пакети допомагають спростити складність кодової бази, організовуючи її в керовані розділи, що робить систему легшою для розуміння, підтримки та спільної розробки [18].

У контексті мобільної платформи для фінансового управління, розробленої за допомогою Flutter та Clean Architecture, діаграма пакета ілюструє, як додаток структурований на високому рівні та як взаємодіють різні частини системи. Дизайн зазвичай обертається навколо кількох основних пакетів.

Рівень представлення включає елементи інтерфейсу користувача, такі як екрани, віджети та логіка управління станом. Він обробляє взаємодію з користувачем та взаємодіє з рівнем домену для отримання або відображення даних. Рівень домену знаходиться в основі архітектури. Він містить основну бізнес-логіку, включаючи сутності, класи варіантів використання та інтерфейси репозиторіїв. Цей рівень залишається незалежним від зовнішніх фреймворків, забезпечуючи цілісність та можливість повторного використання логіки додатку.

Це підтримує рівень даних, який відповідає за управління джерелами даних додатку. Він обробляє взаємодію з локальними базами даних, такими як SQLite, та зовнішніми API, а також реалізує інтерфейси, визначені на рівні домену. Крім того, основний пакет може використовуватися для зберігання спільних ресурсів, таких як утиліти, константи або інструменти обробки помилок, до яких здійснюється доступ на кількох рівнях. Нарешті, зовнішній пакет містить усі сторонні залежності, інтегровані в систему, такі як пакети для доступу до бази даних, мереж або аналітики.

У цій архітектурі залежності рухаються всередину — зовнішні рівні можуть залежати від внутрішніх, але не навпаки. Це забезпечує чіткий розподіл обов'язків та підтримує модульну, масштабовану розробку. Таким чином, діаграма пакета забезпечує чіткий та логічний огляд того, як організовано програмне забезпечення та як розподіляються обов'язки по всій системі.

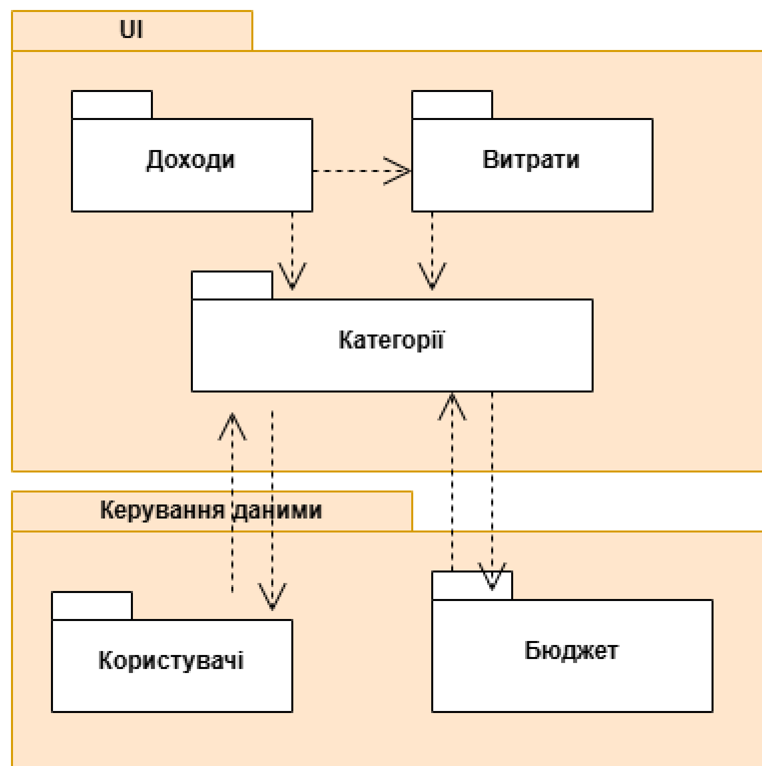

Рис. 3.3 – Діаграма пакетів

Ця діаграма пакетів ілюструє структуру фінансової системи, розділяючи її на логічні модулі для кращого управління. Вона складається з двох головних пакетів: Система обрахунків та Фінансист, які взаємодіють між собою для контролю фінансів.

Фінансист ініціює основні операції, такі як створення категорій доходів і витрат та введення відповідних сум. Система обрахунків приймає ці дані, обробляє фінансові потоки та перевіряє дотримання встановлених лімітів. Якщо витрати перевищують допустимий рівень, вона надсилає попередження і блокує операцію, запобігаючи перевищенню бюджету.

Крім того, система відповідає за встановлення обмежень на витрати. Якщо загальна сума перевищує заданий ліміт, встановлення нових обмежень

блокується. В іншому випадку система дозволяє застосування фінансових коригувань. Завершуючи всі розрахунки, процес управління фінансами приходить до логічного завершення.

Ця структура забезпечує ефективну організацію фінансових операцій, покращуючи автоматизацію та контроль над бюджетом.

3.5 Вибір інструментарію для створення програмного забезпечення

Розробка мобільної платформи для ефективного управління фінансами вимагає ретельно продуманого вибору інструментів та технологій для забезпечення ефективності, зручності обслуговування та зручності використання. Інструменти, обрані для цього проєкту — Dart, Flutter, SQLite, Ruby та Git — були обрані на основі їхньої сумісності, простоти інтеграції та відповідності принципам чистої архітектури.

Dart слугує основною мовою програмування для цього застосунку. Розроблений Google, Dart оптимізований для створення швидких та адаптивних інтерфейсів користувача, особливо при використанні з Flutter. Він пропонує чітке введення тексту, можливості асинхронного програмування та синтаксис, який легко читати та підтримувати, що робить його ідеальним вибором для розробки мобільних додатків.

Flutter, популярний набір інструментів для інтерфейсу користувача з відкритим кодом, також розроблений Google, — це фреймворк, який використовується для створення кросплатформного мобільного інтерфейсу. Його структура на основі віджетів та функція гарячого перезавантаження забезпечують швидке створення прототипів та ефективний рендеринг інтерфейсу користувача. За допомогою Flutter застосунок можна розгорнути як на платформах Android, так і на iOS з єдиної кодової бази, що значно скорочує час розробки та витрати.

SQLite використовується для локального зберігання даних. Він забезпечує легку вбудовану реляційну систему баз даних, яка добре підходить для мобільних пристроїв. SQLite гарантує, що користувачі можуть безпечно зберігати записи про доходи та витрати на своїх пристроях, навіть у автономному режимі. Його простота, надійність та інтеграція з Dart/Flutter через такі пакети, як sqflite, роблять його природним вибором для керування постійними даними в цій програмі.

Ruby включено як додатковий інструмент, головним чином для завдань сценаріїв, створення прототипів на серверній частині або автоматизації, таких як підготовка наборів даних або створення звітів. Його зрозумілий синтаксис та простота використання можуть підтримувати процеси розробки, де потрібне легке сценарне написання.

Git використовується як система контролю версій, що забезпечує ефективну співпрацю та відстеження змін протягом усього життєвого циклу розробки. Він дозволяє розробникам керувати версіями коду, розгалужувати функції та повертатися до попередніх станів, якщо необхідно. Використання таких платформ, як GitHub або GitLab, разом з Git гарантує безпечне резервне копіювання коду та полегшує командну співпрацю, навіть у розподілених середовищах.

Разом ці інструменти забезпечують міцну технічну основу для розробки надійної, масштабованої та орієнтованої на користувача платформи фінансового управління. Кожна технологія відіграє певну роль у підтримці архітектури застосунку, забезпечуючи ефективне виконання як функціональних, так і нефункціональних вимог.

4 ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ СИСТЕМИ

4.1 Вимоги до апаратного та програмного забезпечення

Для забезпечення безперебійної розробки та оптимального користувацького досвіду для мобільної платформи, орієнтованої на управління фінансами, необхідно дотримуватися певних вимог до апаратного та програмного забезпечення.

Для середовища розробки необхідний багатоядерний процесор, такий як Intel i5 або i7, або еквівалентний AMD Ryzen, для виконання багатозадачних та ресурсомістких завдань, таких як компіляція та тестування програми. Розробникам знадобиться щонайменше 8 ГБ оперативної пам'яті для ефективного керування середовищем розробки, емуляторами та іншими вимогливими програмами. Крім того, рекомендується мати 256 ГБ SSD-накопичувача для зберігання файлів розробки, програм та даних тестування, оскільки SSD-накопичувачі забезпечують вищу швидкість читання та запису. Хоча спеціальна відеокарта не є обов'язковою для більшості завдань, розробники, які працюють над ресурсомісткими функціями, такими як складна анімація або графічні елементи, можуть скористатися перевагами середнього класу графічного процесора.

Для тестування розробникам важливо мати доступ до сучасних смартфонів Android або iOS, бажано з щонайменше 4 ГБ оперативної пам'яті та з останніми стабільними версіями операційної системи (Android 10+ або iOS 14+). Ці пристрої дозволять проводити реальне тестування та оптимізацію функцій та продуктивності програми.

З боку користувача, мобільна платформа в першу чергу орієнтована на смартфони. Додаток розроблено для безперебійної роботи на пристроях з щонайменше 2 ГБ оперативної пам'яті, процесором з тактовою частотою 1,5

ГГц+ та щонайменше 16 ГБ пам'яті. Він сумісний з Android 6.0+ та iOS 12+, що забезпечує широкий охоплення на різних пристроях.

Щодо програмного середовища, розробники використовуватимуть Dart, основну мову програмування для написання бізнес-логіки та функціональності застосунку. Такі функції Dart, як сувора типізація, безпека використання null-значень та підтримка асинхронного програмування, роблять його ідеальним для забезпечення високої продуктивності в мобільних застосунках. Flutter SDK буде основною платформою для створення власно компільованих застосунків на мобільних, веб- та настільних платформах з єдиної кодової бази. Flutter забезпечує узгоджений, високоякісний інтерфейс користувача на платформах Android та iOS, що є важливим для безперебійної роботи.

Android Studio або IntelliJ IDEA – рекомендовані IDE для розробки на Flutter, які пропонують необхідні інструменти для написання коду, налагодження та тестування. Для розробки на iOS для створення та розгортання застосунку на пристроях Apple потрібен Xcode. Розробники також використовуватимуть Git для контролю версій, забезпечення належного управління змінами коду, співпраці з членами команди та ефективного відстеження прогресу проєкту. Такі платформи, як GitHub, GitLab або Bitbucket, є важливими для розміщення репозиторіїв та сприяння спільним робочим процесам.

SQLite слугуватиме локальною системою керування базами даних для зберігання даних у додатку. Вона легка, ефективна та добре підходить для мобільних платформ, гарантуючи, що користувачі можуть зберігати фінансові дані та отримувати до них доступ офлайн. Для хмарних сервісів, таких як автентифікація, функції баз даних у реальному часі та аналітика, Firebase можна інтегрувати в платформу, забезпечуючи надійне бекенд-рішення.

Щоб забезпечити ретельне тестування та налагодження, розробники використовуватимуть такі інструменти, як Postman, для тестування API та Visual Studio Code (VSCode) для написання та керування кодом. Екосистема

Flutter включає різні плагіни та пакети, які покращують функціональність додатка, такі як обробка платежів, управління станом та компоненти візуалізації даних.

Інструменти тестування, такі як емулятори та симулятори в Android Studio та Xcode, необхідні для перевірки продуктивності додатка на різних пристроях та конфігураціях. Перед офіційним запуском бета-тестування можна буде проводити за допомогою таких платформ, як TestFlight для iOS, що дозволить розробникам збирати цінні відгуки користувачів. Після цього додаток буде готовий до розгортання в Google Play Store та Apple App Store, для чого потрібен обліковий запис Google Play Console та обліковий запис розробника Apple.

Підтримка стабільного інтернет-з’єднання важлива для розробників, щоб мати доступ до хмарних сервісів та оновлювати залежності. Програмне забезпечення безпеки має вирішальне значення для захисту кодової бази та даних користувачів від несанкціонованого доступу або кіберзагроз. Регулярне резервне копіювання файлів розробки, баз даних та даних додатків також буде важливим для захисту від втрати даних під час розробки та після розгортання.

Відповідаючи цим вимогам до апаратного та програмного забезпечення, мобільна платформа для управління фінансами буде оснащена необхідними ресурсами для безперебійного функціонування як під час розробки, так і під час розгортання, пропонуючи користувачам надійний та високопродуктивний інструмент для управління своїми фінансами.

Також було зроблено діаграму розгортання для візуального огляду деплою системи (рис. 4.1).

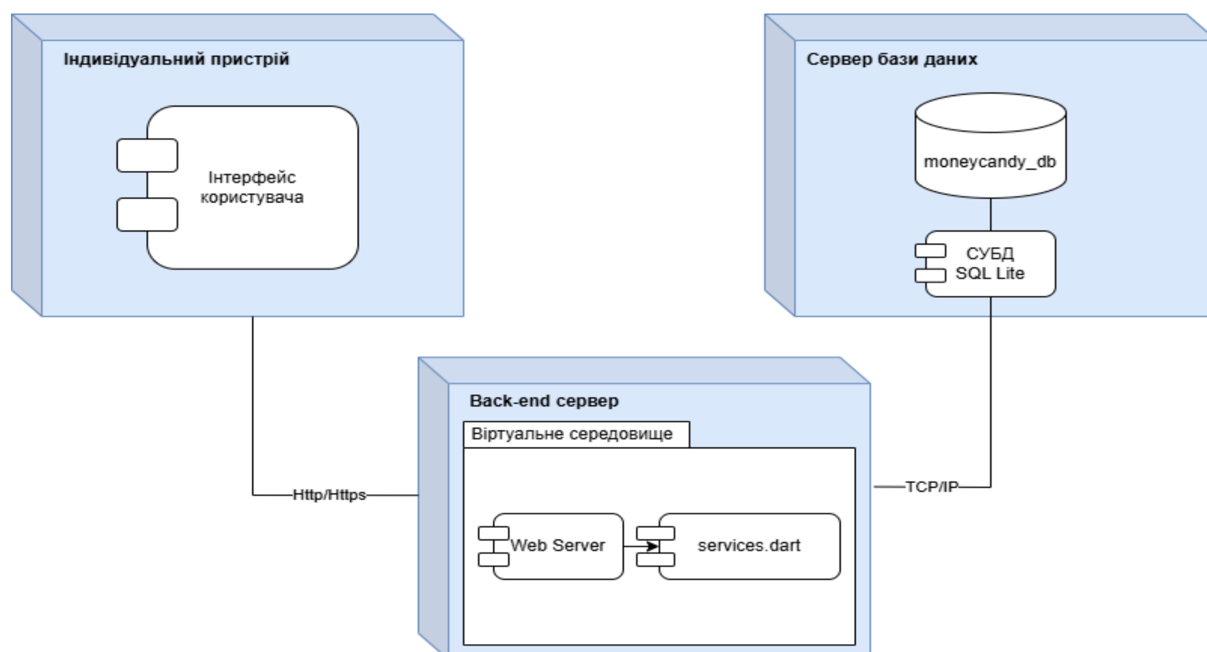

Рис. 4.1 – Діаграма розгортання

4.2 Тестування системи

Запустивши систему, потрапляємо на форму авторизації, тут нам треба ввести логін та пароль користувача (рис. 4.2 - 4.3).

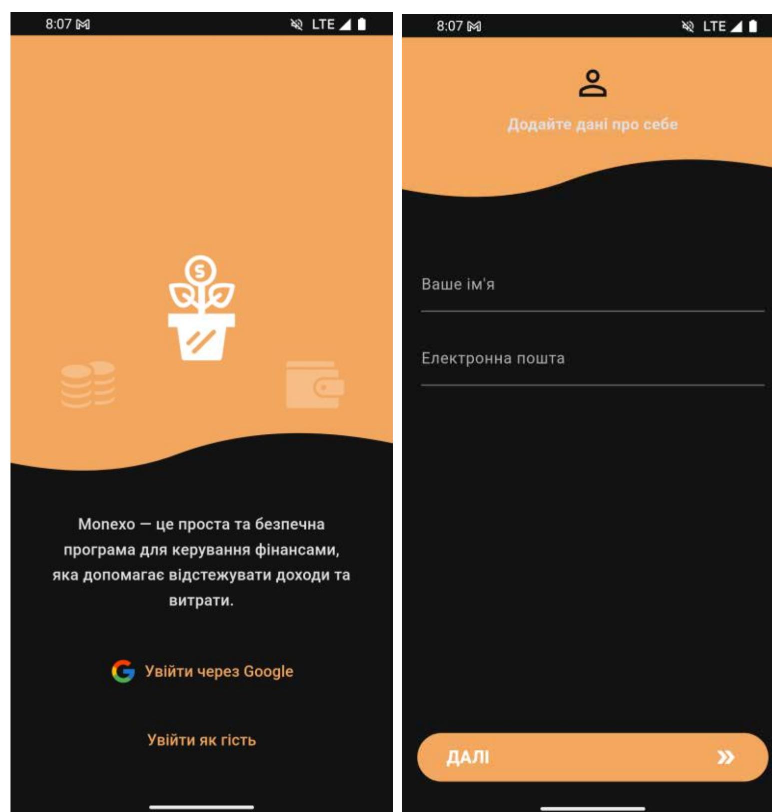

Рис. 4.2 – 4.3 Форма авторизації

Нам необхідно створити рахунок для надходження доходів. Для цього вводимо назву рахунку, вибираємо валюту і, якщо хочемо, то й зазначимо одразу бюджет (Рис. 4.4).

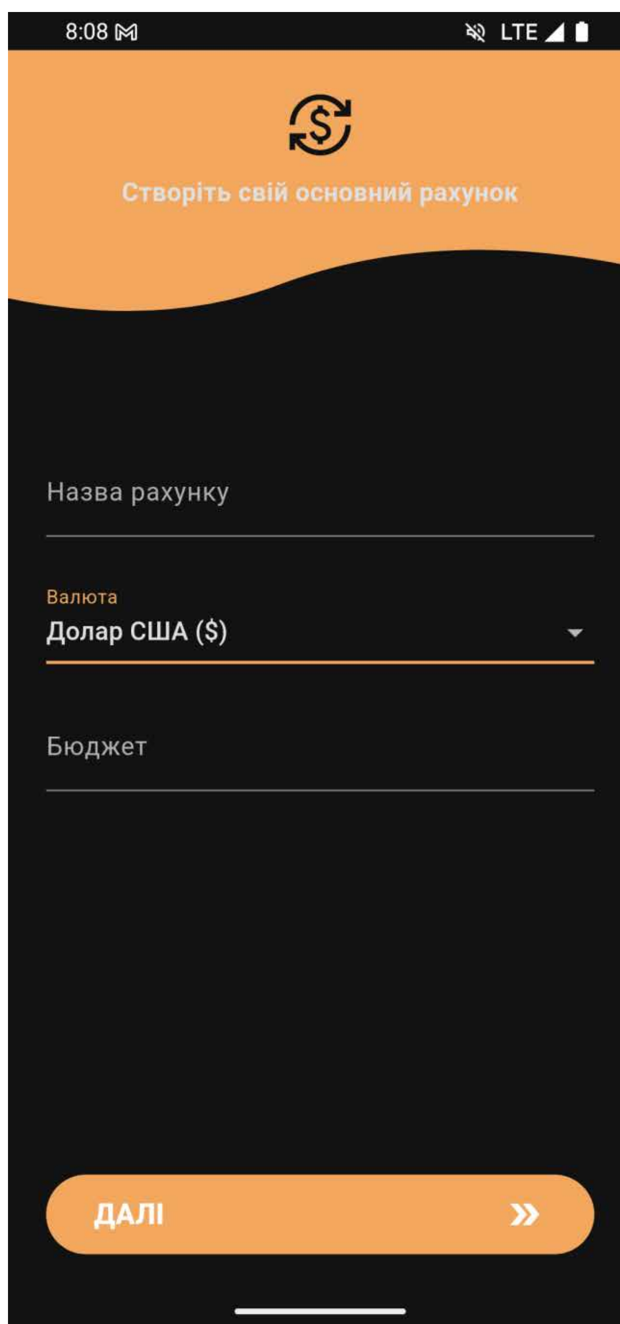

Рис. 4.4 – Створення рахунку

Після цього потрапляємо на основну сторінку нашого рахунку. Тут ми бачимо наш баланс на рахунку та всі останні транзакції (Рис. 4.4 – 4.5).

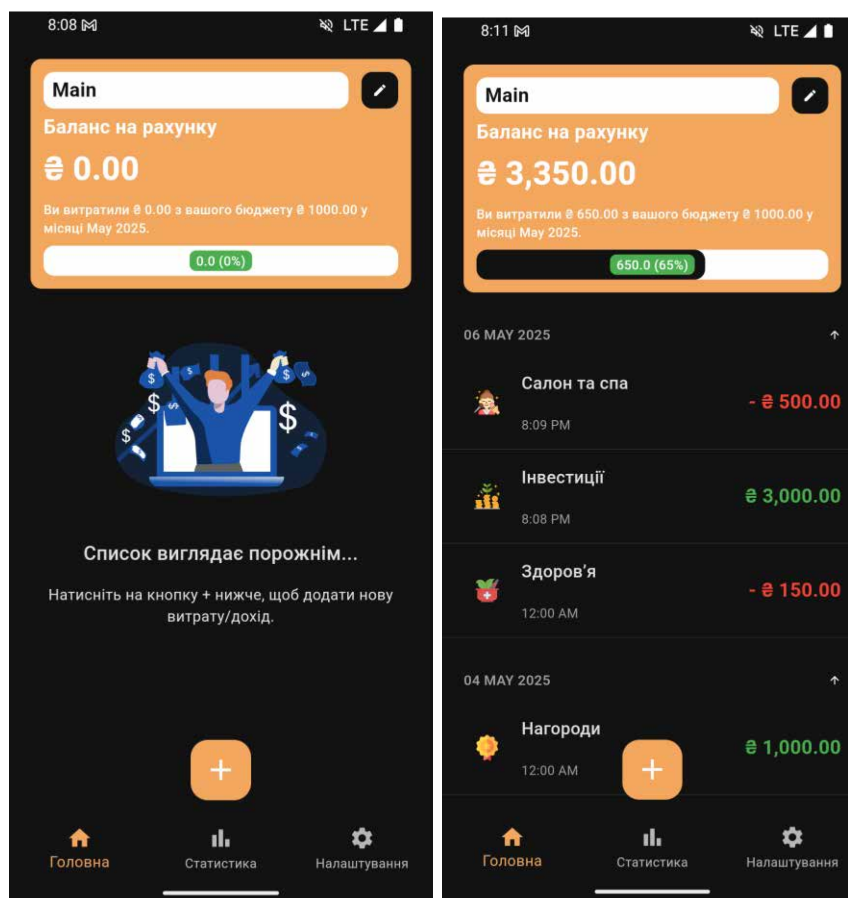

Рис. 4.5 – 4.6 Основна сторінка

Переходимо на вкладку "Статистика", на якій можна подивитися кругообіг фінансів за доходами та витратами на день/тиждень/місяць. Дані подаються у вигляді графіка та числових операцій (Рис. 4.7 – 4.9).

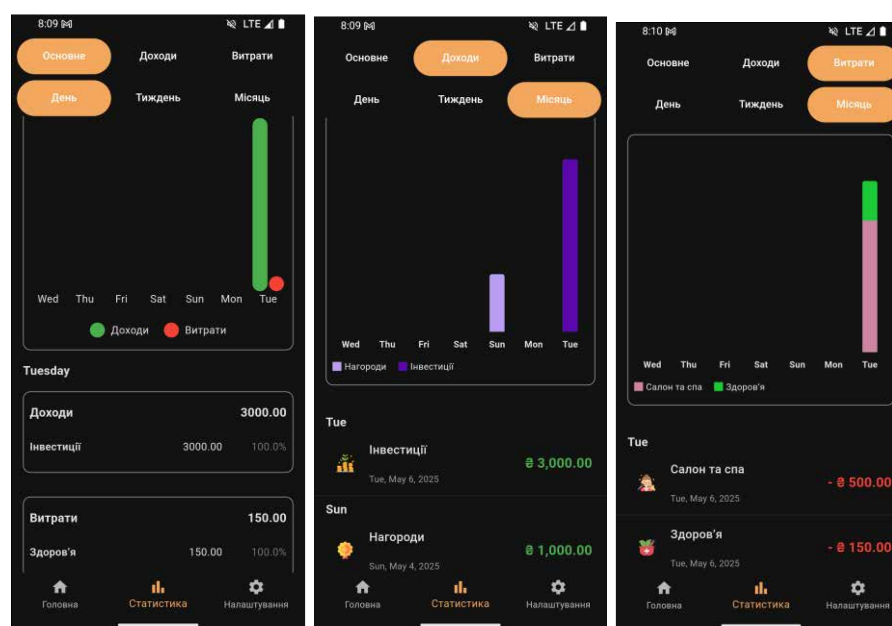

Рис. 4.7 – 4.9 Статистика (групування по дохід/витрати)

Спробуємо створити нову категорію, переходимо на сторінку категорій, вибираємо якусь саме - доходів або витрат і вибираємо назву та колір іконки (Рис. 4.9 - 4.10).

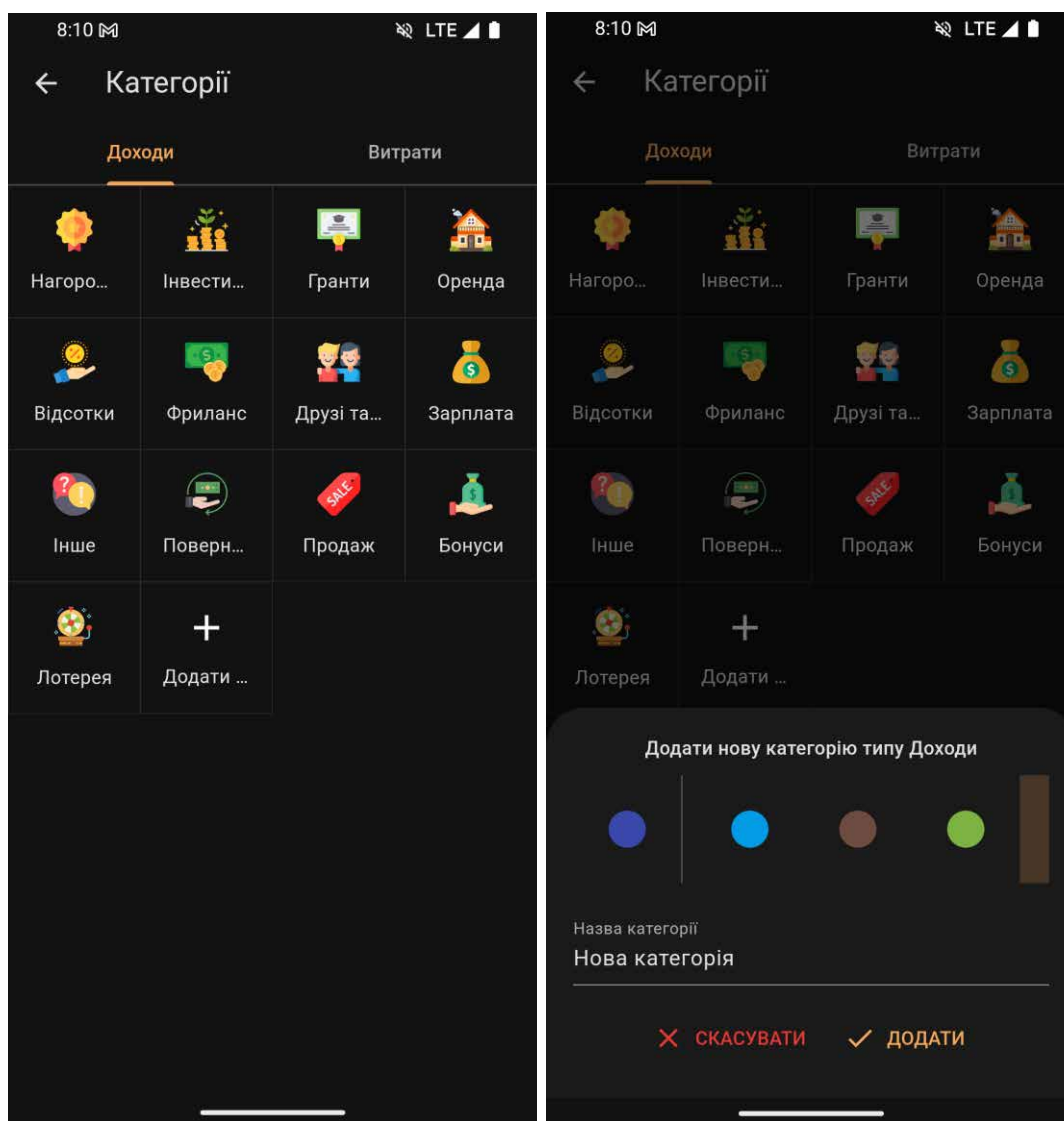

Рис. 4.9 – 4.10 Список категорій

І так само зробимо нову транзакцію, для цього натискає на кнопку створення транзакції, вибираємо категорію доходу або витрати, вибираємо на що витрачатиме, суму, за бажанням опис, дата запишеться автоматично і підтверджуємо. Після цього транзакція буде успішно додана до обраної категорії (Рис. 4.11 - 4.12).

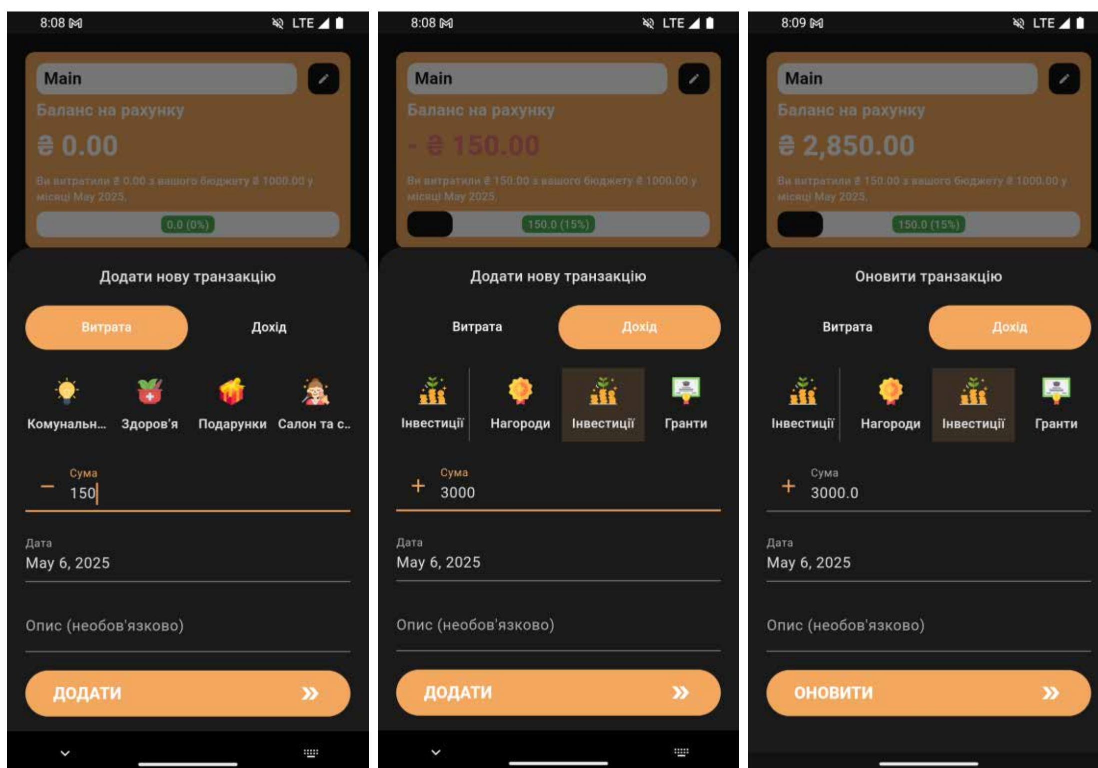

Рис. 4.11 – 4.12 Пейджі створення транзакцій

Продемонстрована функціональність допомогає користувачам краще контролювати свої особисті фінанси, відстежувати свої витрати та доходи, здійснювати категоризацію фінансових операцій, а також планувати фінансові цілі.

ВИСНОВКИ

Розроблена мобільна платформа для ефективного управління фінансами пропонує комплексне та ефективне рішення для людей, щоб вони могли легко керувати своїми особистими фінансами. Завдяки використанню сучасних технологій, таких як Dart, Flutter та SQLite, ця платформа здатна забезпечити безперебійний користувацький досвід, як на пристроях Android, так і на iOS. Вона відповідає на зростаючу потребу в цифрових фінансових інструментах, які дозволяють користувачам відстежувати доходи, витрати та встановлювати бюджети, водночас надаючи цінну інформацію для покращення фінансових звичок.

Рішення впровадити чисту архітектуру в дизайн системи забезпечує масштабованість, зручність обслуговування та розділення обов'язків, що дозволяє додатку розвиватися з часом з мінімальними перебоями. Обравши Flutter як основну основу для розробки, платформа здатна запропонувати послідовний та високоякісний інтерфейс на кількох пристроях, скорочуючи час розробки та забезпечуючи ефективне використання ресурсів. Крім того, включення SQLite як локальної бази даних забезпечує надійне зберігання даних, водночас забезпечуючи офлайн-функціональність для користувачів, яким потрібен доступ до своєї фінансової інформації в дорозі.

У процесі розробки пріоритет було надано ключовим міркуванням, таким як безпека, конфіденційність користувачів та цілісність даних. Такі функції, як налаштовувані категорії доходів і витрат, відстеження транзакцій та управління бюджетом, дозволяють користувачам краще контролювати своє фінансове становище. Інтеграція хмарних сервісів, таких як Firebase, ще більше розширює можливості платформи, пропонуючи синхронізацію даних у режимі реального часу, безпечну автентифікацію та резервне копіювання даних.

Зрештою, ця мобільна платформа надає практичний інструмент для всіх, хто прагне отримати кращу фінансову обізнаність та приймати обґрунтовані рішення щодо своїх витратних звичок. Завдяки зручному інтерфейсу та потужній архітектурі серверної частини, вона має потенціал для значного покращення управління особистими фінансами та допомоги користувачам у досягненні їхніх фінансових цілей. У майбутньому майбутні вдосконалення можуть включати додавання розширеної аналітики, персоналізованих фінансових рекомендацій та глибшу інтеграцію з банківськими системами, що дозволить платформі залишатися на передовій технологій управління особистими фінансами.

Крім того, гнучкість платформи дозволяє їй адаптуватися до потреб користувачів, що постійно змінюються, оскільки її можна легко розширити новими функціями та можливостями в майбутніх оновленнях. Ця адаптивність є вирішальною у швидкозмінному фінансовому ландшафті, де нові тенденції, фінансові інструменти та вимоги користувачів постійно формують ринок. Модульна архітектура застосунку гарантує ефективне впровадження цих майбутніх змін без порушення основної функціональності, що робить його перспективним рішенням для управління особистими фінансами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сміт, Дж., Робертс, Л. Фінансовий менеджмент у цифрову епоху: дослідження мобільних рішень. — Wiley, 2020.
2. Браун, М., Грін, Т. Розробка мобільних додатків: найкращі практики для Flutter та Dart. — Pearson, 2019.
3. Вільямс, Х. Додатки для особистих фінансів: дослідження тенденцій та майбутніх перспектив. — McGraw-Hill, 2021.
4. Wallet – [Електронний ресурс] – Режим доступу: https://web.budgetbakers.com
5. Monefy – [Електронний ресурс] – Режим доступу: https://monefy.com/faq
6. Патель, С., Шарма, Р. Фінансові технології та мобільні додатки: трансформація управління особистими фінансами. — Springer, 2022.
7. Андерсон, П., Тейлор, А. SQLite для розробників мобільних додатків: практичний посібник. — O'Reilly Media, 2021.
8. Джонсон, Д. Чиста архітектура: вичерпний посібник зі створення масштабованих мобільних додатків. — Addison-Wesley, 2020.
9. Робертс, С. Створення фінансових додатків для iOS та Android за допомогою Dart та Flutter. — Apress, 2021.
10. Міллер, Дж., Едвардс, П. Посібник з дизайну користувацького досвіду для мобільних додатків. — CRC Press, 2020.
11. Гарріс, Р., Томпсон, Л. Конфіденційність та безпека даних у мобільних фінансових додатках. — Elsevier, 2019.
12. Вілсон, Т., Гарсія, М. Майбутнє фінансового менеджменту: мобільні платформи та інновації. — Routledge, 2021.
13. Документація Flutter – [Електронний ресурс] – Режим доступу: https://flutter.dev/docs

14. Документація з мови програмування Dart – [Електронний ресурс] – Режим доступу: https://dart.dev/guides
15. Документація з SQLite – [Електронний ресурс] – Режим доступу: https://www.sqlite.org/docs.html
16. Документація з Git – [Електронний ресурс] – Режим доступу: https://git-scm.com/doc
17. Чиста архітектура Роберта К. Мартіна – [Електронний ресурс] – Режим доступу: https://blog.cleancoder.com/
18. Flutter та Dart для початківців: Навчіться створювати додатки для iOS та Android – [Електронний ресурс] – Режим доступу: https://www.udemy.com/course/flutter-dart-the-complete-guide/
19. Посібник з SQLLite для початківців – [Електронний ресурс] – Режим доступу: https://www.sqlitetutorial.net/
20. Мобільні додатки для управління фінансами: Порівняльне дослідження – [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/329467345
21. Створення фінансових додатків за допомогою Flutter – [Електронний ресурс] – Режим доступу: https://www.packtpub.com/product/building-financial-applications-with-flutter/9781800562770
22. Дизайн інтерфейсу користувача мобільного додатку – [Електронний ресурс] – Режим доступу: https://uxdesign.cc/
23. Керівні принципи дизайну матеріалів Google – [Електронний ресурс] – Режим доступу: https://material.io/design
24. Інтеграція бази даних SQLite у Flutter – [Електронний ресурс] – Режим доступу: https://medium.flutterdevs.com/
25. Найкращі практики безпеки мобільних додатків – [Електронний ресурс] – Режим доступу: https://www.owasp.org/
26. Керування станом Flutter: глибокий огляд – [Електронний ресурс] – Режим доступу: https://bloclibrary.dev/

27. Вступ до чистої архітектури мобільних додатків – [Електронний ресурс] – Режим доступу: https://medium.com/swlh/
28. Фінансовий менеджмент у застосунках для особистих фінансів – [Електронний ресурс] – Режим доступу: https://www.forbes.com/sites/
29. Впровадження бази даних SQLite у застосунках Flutter – [Електронний ресурс] – Режим доступу: https://www.flutterbeads.com/
30. Найкращі бібліотеки Dart та Flutter для фінансових застосунків – [Електронний ресурс] – Режим доступу: https://pub.dev/packages
31. Тестування та налагодження мобільних застосунків у Flutter – [Електронний ресурс] – Режим доступу: https://flutter.dev/docs/testing
32. SQLite проти Firebase для застосунків для фінансового менеджменту – [Електронний ресурс] – Режим доступу: https://www.turing.com/

ДОДАТОК А

Фрагменти програмного коду. Створення провайдеру для категорій транзакцій

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:monexo/models/models.dart';

/// Provider для роботи з категоріями транзакцій
///
/// Для кожного нового користувача створюються нові категорії транзакцій
/// на стороні бази даних Firebase - тому їх можна додавати і видаляти
final class CategoryProvider {
  static final CategoryProvider _instance = CategoryProvider._internal();

  factory CategoryProvider() {
    return _instance;
  }

  CategoryProvider._internal();


  final FirebaseFirestore _db = FirebaseFirestore.instance;
  late CollectionReference<Map<String, dynamic>> _categoriesCollection;

  init(AppUser user) {
    _categoriesCollection =
        _db.collection('users').doc(user.uid).collection('categories');
  }

  Future<void> setupCategories() async {
    for (final category in baseIncomeCategories) {
      add(category);
    }
    for (final category in baseExpenseCategories) {
      add(category);
    }
  }

  Future<List<CategoryModel>> getCategories(String accountId) async {
    final snapshot = await _categoriesCollection
        .get();

    return snapshot.docs.map((doc) {
      return CategoryModel.fromJson(doc.data());
    }).toList();
  }

  Future<void> add(CategoryModel category) async {
    var doc = await _categoriesCollection.add(category.toJson());
    return doc.update({'id': doc.id});
```

```
  }

  Future<void> update(CategoryModel category) {
    return _categoriesCollection.doc(category.id).update(category.toJson());
  }

  Future<void> delete(CategoryModel category) async {
    await _categoriesCollection.doc(category.id).delete();
  }
}

final List<CategoryModel> baseIncomeCategories = [
  CategoryModel(
    id: 'awards',
    name: 'Нагороди',
    icon: 'income/awards',
    type: CategoryType.income,
  ),
  CategoryModel(
    id: 'bonus',
    name: 'Бонуси',
    icon: 'income/bonus',
    type: CategoryType.income,
  ),
  CategoryModel(
    id: 'freelance',
    name: 'Фриланс',
    icon: 'income/freelance',
    type: CategoryType.income,
  ),
  CategoryModel(
    id: 'friends_and_family',
    name: 'Друзі та сім’я',
    icon: 'income/friends',
    type: CategoryType.income,
  ),
CategoryModel(
    id: 'rent',
    name: 'Оренда',
    icon: 'income/rent',
    type: CategoryType.income,
  ),
  CategoryModel(
    id: 'salary',
    name: 'Зарплата',
    icon: 'income/salary',
    type: CategoryType.income,
  ),
];
```

ДОДАТОК Б

Фрагменти програмного коду. Функціонал створення транзакцій

```
import 'package:cloud_firestore/cloud_firestore.dart' hide Transaction;
import 'package:collection/collection.dart';
import 'package:monexo/models/models.dart';

class TransactionProvider {
  static final TransactionProvider _instance =
      TransactionProvider._internal();

  factory TransactionProvider() {
    return _instance;
  }

  TransactionProvider._internal();

  final FirebaseFirestore _db = FirebaseFirestore.instance;
  late CollectionReference<Map<String, dynamic>> _transactionCollection;

  init(AppUser user) {
    _transactionCollection =
        _db.collection('users').doc(user.uid).collection('transactions');
  }


  Future<void> addTransaction(TransactionModel transaction) async {
    var doc = await _transactionCollection.add(transaction.toJson());
    return doc.update({'id': doc.id});
  }

  Future<void> updateTransaction(TransactionModel transaction) {
    return _transactionCollection
        .doc(transaction.id)
        .update(transaction.toJson());
  }

  Future<void> deleteTransaction(TransactionModel transaction) async {
    await _transactionCollection.doc(transaction.id).delete();
  }

  Map<DateTime, List<TransactionModel>> groupTransactionsByDate(
    List<TransactionModel> transactions,
  ) {
    return groupBy<TransactionModel, DateTime>(transactions, (txn) {
      return DateTime(txn.timestamp.year, txn.timestamp.month, txn.timestamp.day);
    });
  }

  Future<List<TransactionModel>> getAccountTransactions(String accountId) async {
```

```
    final snapshot = await _transactionCollection
        .where('accountId', isEqualTo: accountId)
        .orderBy('timestamp', descending: true)
        .get();

    return snapshot.docs.map((doc) {
      return TransactionModel.fromJson(doc.data());
    }).toList();
  }

  double calculateAccountBalance(List<TransactionModel> transactions) {
    if (transactions.isEmpty) {
      return 0;
    }
    return transactions.map((item) => item.amount).reduce((a, b) => a + b);
  }

  Stream<List<TransactionModel>> accountTransactionsStream(String accountId) =>
      _transactionCollection
          .where('accountId', isEqualTo: accountId)
          .orderBy('timestamp', descending: true)
          .snapshots()
          .map((snapshot) {
        return snapshot.docs.map((doc) {
          return TransactionModel.fromJson(doc.data());
        }).toList();
      });

}
```