МІНІСТЕРСТВО ВНУТРІШНІХ СПРАВ УКРАЇНИ

ЛЬВІВСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ ВНУТРІШНІХ СПРАВ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ УПРАВЛІННЯ, ПСИХОЛОГІЇ ТА БЕЗПЕКИ

Кафедра інформаційних технологій

РОЗРОБКА ПЕРСОНАЛЬНОГО ФІНАНСОВОГО АСИСТЕНТА

кваліфікаційна робота
здобувача вищої освіти
4 курсу денної форми навчання
Аліни САПРУН

Науковий керівник:
старший викладач кафедри
інформаційних систем та технологій
Наталія ГАЛАЙКО

Рецензент:

_____
вчене звання, науковий ступінь

_____
(Ім'я ПРІЗВИЩЕ рецензента)

Кваліфікаційна робота допущена до захисту
«___» _____ 2025 р., протокол № ____

Завідувач кафедри інформаційних технологій
_____Ігор ФЕДЧАК_
(підпис)

Львів
2025

АНОТАЦІЯ

Сапрун Аліна. Розробка персонального фінансового асистента. – Рукопис.

Дослідження на здобуття освітнього ступеня «бакалавр» за спеціальністю 126 «Інформаційні системи та технології». – Львівський державний університет внутрішніх справ, МВС України, Львів, 2025.

Дипломна робота присвячена розробці персонального фінансового асистента у вигляді мобільного застосунку «Фінансовий менеджер» для платформи Android. У межах дослідження проведено детальний аналіз існуючих мобільних застосунків-аналогів, виявлено їхні переваги та недоліки, визначено функціональні вимоги до нового продукту. Обґрунтовано вибір інструментів і технологій для реалізації проєкту, зокрема використання фреймворку React Native для кросплатформенної розробки, бібліотек React Navigation та Redux для навігації та керування станом застосунку, а також бази даних SQLite для локального збереження даних. У застосунку реалізовано функціонал додавання, редагування та перегляду доходів і витрат користувача, формування звітів і візуалізації даних. Проведено тестування застосунку на різних пристроях.

Ключові слова: персональний фінансовий асистент, мобільний застосунок, фінансовий менеджер, Android, управління фінансами, React Navigation, Redux, SQLite.

ABSTRACT

Saprun Alina. Development of a Personal Financial Assistant. Manuscript.

Research on the bachelor's degree in specialty 126 « Information systems and technologies». Lviv State University of Internal Affairs, MIA of Ukraine, Lviv, 2025.

The thesis is dedicated to the development of a personal financial assistant in the form of a mobile application called "Financial Manager" for the Android platform. As part of the research, a detailed analysis of existing mobile application analogues was conducted, identifying their advantages and shortcomings, defining the functional requirements for the new product. The choice of tools and technologies for the project was justified, including the use of the React Native framework for cross-platform development, React Navigation and Redux libraries for navigation and state management, and the SQLite database for local data storage. The application implements functionality for adding, editing, and viewing user income and expenses, generating reports, and visualizing financial data. The application was tested on various devices.

Keywords: personal financial assistant, mobile application, financial manager, Android, financial management, React Navigation, Redux, SQLite.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасні інформаційні технології стрімко інтегруються в різні сфери життя, зокрема у фінансову діяльність. У зв’язку зі зростанням обсягів фінансових операцій та необхідністю їх точного й своєчасного обліку, автоматизація управління персональними фінансами стає важливою складовою ефективного фінансового планування. Персональні фінансові асистенти надають користувачам можливість зручно відстежувати доходи та витрати, здійснювати аналіз фінансових показників, а також приймати обґрунтовані управлінські рішення. Це сприяє підвищенню фінансової грамотності користувачів і допомагає уникати необґрунтованих витрат.

Саме тому розробка персонального фінансового асистента є актуальним напрямом, що відповідає сучасним потребам у спрощенні управління фінансами, підвищенні прозорості витрат, зменшенні ризиків і покращенні якості фінансового планування.

Аналіз останніх досліджень. Питання автоматизації фінансового обліку широко висвітлюється в науковій літературі та практичних дослідженнях. Дослідники, зокрема Савченко В., Кононенко Л. та Гай О. [18], аналізують застосування статистичних методів і фінансового аналізу для інформаційного забезпечення суб’єктів фінансового ринку. Шиш А. М. [23] досліджує роль хмарних технологій у бухгалтерському обліку та фінансовому аналізі, акцентуючи увагу на їхній адаптації до місцевих умов. Рудовіл Є. А. зосереджується на розробці веб-додатків для управління фінансами із застосуванням сучасних технологій, тоді як Смирнов Є. Т. [19] вивчає методи персистентності даних у мобільних фінансових застосунках. У працях Шаповалової А. В. розглядається створення автоматизованої інформаційної системи обліку доходів і витрат, що підкреслює важливість інтеграції таких рішень у мобільні платформи. Окремі дослідження зосереджені на проєктуванні інтелектуальних систем фінансового

моніторингу, використанні технологій аналізу великих даних і забезпеченні інформаційної безпеки. Особливу увагу приділено також створенню адаптивних інтерфейсів і забезпеченню сумісності з сучасними мобільними платформами (Rayhan R. [7]). Проте, попри значні досягнення в цій сфері, зберігається потреба у створенні більш гнучкого, функціонального та інтуїтивно зрозумілого фінансового асистента, здатного забезпечити комплексну аналітику й автоматизований контроль за фінансовими потоками.

Мета дослідження. Розробка персонального фінансового асистента, який забезпечуватиме автоматизацію обліку фінансових операцій, підвищення ефективності управління особистими фінансами, а також створення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів. Додатково передбачено впровадження аналітичних можливостей для глибокого аналізу витрат і доходів, прогнозування фінансового стану та надання персоналізованих рекомендацій щодо оптимізації фінансових ресурсів.

Завдання дослідження:

- проаналізувати сучасні персональні фінансові асистенти та їх функціональні можливості;
- визначити вимоги до персонального фінансового асистента;
- сформулювати технічне завдання для розробки системи;
- обґрунтувати вибір технологій для створення фінансового асистента;
- розробити архітектуру застосунку та структуру бази даних;
- реалізувати мобільний застосунок для платформи Android з використанням React Navigation і Redux;
- забезпечити збереження фінансових даних у локальній базі SQLite;
- реалізувати засоби візуалізації фінансових даних;
- провести тестування розробленого застосунку;

– оцінити ефективність запропонованого рішення та можливості його подальшого вдосконалення.

Об’єкт дослідження: процеси, пов’язані з автоматизацією персонального фінансового обліку.

Предмет дослідження: методи, підходи та програмні засоби, що застосовуються для розробки персонального фінансового асистента, призначеного для автоматизації обліку та аналізу фінансових операцій з метою підвищення ефективності управління особистими фінансами.

Методи дослідження. У процесі дослідження було використано комплекс методів, зокрема: аналіз наукових джерел та існуючих програмних рішень – для визначення функціональних та нефункціональних вимог до системи; методи системного аналізу – для розробки архітектури програмного застосунку; методи програмної інженерії – для реалізації програмного забезпечення; методи тестування – для перевірки функціональності, надійності та відповідності системи поставленим вимогам.

Практичне значення роботи. Створення прототипу мобільного застосунку «Фінансовий менеджер», призначеного для автоматизації персонального фінансового обліку. Результати дослідження можуть бути використані як основа для подальшої розробки та впровадження подібних рішень у фінансових установах, а також як навчальний матеріал для здобувачів спеціальностей, пов’язаних з інформаційними технологіями. Окремі напрацювання, отримані в ході реалізації проєкту, були представлені на науково-практичній конференції з інформаційних технологій.

Структура роботи. Дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Загальний обсяг роботи становить 74 сторінки, включаючи 20 ілюстрацій, 1 таблицю та 23 джерела літератури. Повний обсяг основного тексту складає 45 сторінок.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ТА ВИМОГ ДО ФІНАНСОВИХ АСИСТЕНТІВ

1.1. Огляд сучасних персональних фінансових асистентів

Сучасний розвиток мобільних технологій відкриває широкі можливості для вдосконалення процесу управління особистими фінансами, роблячи фінансове планування та контроль за витратами більш доступними, зручними та ефективними. Завдяки інтеграції штучного інтелекту, аналізу великих даних і хмарних технологій, мобільні фінансові асистенти можуть автоматизувати облік доходів і витрат, надавати персоналізовані рекомендації щодо бюджету та сприяти підвищенню фінансової грамотності користувачів.

Існує чимало популярних додатків, які можуть бути використані як аналоги для створеної системи контролю та моніторингу фінансових операцій. Вони забезпечують широкий функціонал, включаючи синхронізацію з банківськими рахунками, автоматичну категоризацію витрат, побудову аналітичних звітів і налаштування фінансових цілей. Серед найбільш відомих рішень виділяються такі застосунки, як Mint, який пропонує комплексний аналіз фінансового стану користувача та синхронізується з банківськими рахунками; YNAB (You Need A Budget), що базується на принципах планування бюджету та заохочує користувачів до розумного розподілу коштів; PocketGuard, який допомагає контролювати витрати та знаходити можливості для заощаджень. Ці додатки демонструють ефективність автоматизованого управління особистими фінансами та слугують орієнтиром для створення нових інноваційних рішень у цій сфері.Mint – один із найпоширеніших інструментів для ведення фінансового обліку, який забезпечує автоматичну інтеграцію з банківськими рахунками,

категоризацію витрат та детальний аналіз фінансових потоків. Додаток пропонує зручний інтерфейс, можливість встановлення бюджетних обмежень та отримання звітів у реальному часі. Крім того, Mint автоматично оновлює інформацію про фінансові операції, що значно спрощує процес контролю за особистими фінансами. Проте, цей додаток має певні обмеження, зокрема недостатню гнучкість у налаштуваннях, обмежені можливості для кастомізації категорій витрат, а також обов'язкове підключення до банківських установ, що може створювати труднощі для користувачів, які бажають вручну керувати фінансовими записами або працювати з альтернативними джерелами фінансової інформації [1].

YNAB (You Need a Budget) спеціалізується на детальному плануванні бюджету та надає користувачам інструменти для ефективного розподілу фінансових ресурсів між різними категоріями витрат. Основний принцип роботи додатка полягає у наданні кожному долару певного призначення, що сприяє більш усвідомленому управлінню особистими фінансами. Крім базових функцій бюджетування, YNAB пропонує можливість автоматичного імпорту банківських транзакцій, аналітику витрат, а також навчальні матеріали для покращення фінансової грамотності. Однак, основним недоліком цього додатка є висока вартість підписки, що може стати перешкодою для користувачів, які шукають безкоштовні або більш доступні альтернативи. Також певні обмеження накладає необхідність регулярного введення даних вручну, що може здатися незручним для деяких користувачів [4].

PocketGuard є зручним інструментом для моніторингу витрат у реальному часі, що допомагає користувачам контролювати фінансові потоки та аналізувати витрати відповідно до встановленого бюджету. Завдяки автоматичному відстеженню транзакцій та інтеграції з банківськими рахунками, застосунок дозволяє отримувати актуальну фінансову аналітику та визначати доступний баланс після врахування регулярних платежів і запланованих витрат. Водночас, його безкоштовна версія має обмежений

функціонал, що може створювати незручності для користувачів, які потребують розширених можливостей фінансового планування. Додатково, наявність реклами у безкоштовній версії може знижувати комфорт користування додатком, відволікаючи від аналізу фінансової інформації та управління бюджетом [2].

Для більш детального аналізу особливостей кожного з розглянутих застосунків їх основні переваги та недоліки представлені у вигляді порівняльної таблиці (таблиця 1.1).

Таблиця 1.1

Порівняльний аналіз додатків

	Mint	YNAB	PocketGuard
Автоматична синхронізація з різними банками та кредитними картками	1	0	0
Велика кількість доступних функцій	1	1	1
Аналітика та звіти про фінансовий стан	1	1	1
Можливість встановлення додаткових цілей	0	1	1
Безкоштовний доступ	1	0	0
Сумарний коефіцієнт	4	3	3

Джерело: складено автором за матеріалами [18; 19; 4]

З огляду на сильні та слабкі сторони існуючих рішень, були сформовані функціональні можливості майбутнього застосунку. Його основне призначення – допомагати користувачам у раціональному управлінні особистими фінансами. Застосунок автоматизуватиме процеси фінансового обліку та надаватиме зручні інструменти для планування, контролю й аналізу грошових потоків.

Серед ключових можливостей додатка буде:

– ведення обліку доходів і витрат з можливістю планування;

– аналіз фінансової поведінки користувача;

– створення, налаштування та контроль бюджетів;

– адаптивність функціоналу відповідно до індивідуальних потреб;

– надання персоналізованих рекомендацій для покращення фінансової

ситуації;

– нагадування про важливі фінансові події та зобов’язання;

– постановка та моніторинг фінансових цілей.

Такий функціонал забезпечить комплексний підхід до управління особистими фінансами, допомагаючи користувачам приймати обґрунтовані фінансові рішення та досягати поставлених цілей.

Майбутній мобільний застосунок для контролю та управління особистими фінансами поєднуватиме широкий спектр функцій, спрямованих на оптимізацію фінансового планування користувачів. Застосунок забезпечуватиме зручний інтерфейс для ведення бюджету, обліку доходів і витрат, аналізу фінансової поведінки та формування фінансових цілей. Він надаватиме користувачам можливість детально відстежувати рух коштів, ухвалювати обґрунтовані фінансові рішення та ефективно досягати поставлених цілей.

Використання цього застосунку сприятиме підвищенню фінансової обізнаності користувачів, допоможе оптимізувати витрати та покращити загальний фінансовий стан. Завдяки розширеним можливостям індивідуального налаштування, застосунок адаптуватиметься до специфічних потреб кожного користувача, що підвищуватиме його практичну цінність і конкурентоспроможність на ринку.

1.2. Вимоги до персонального фінансового асистента

Персональний фінансовий асистент повинен виконувати кілька ключових функцій, щоб забезпечити ефективне управління фінансами користувача. Однією з основних є облік доходів та витрат, який дозволяє систематизувати всі фінансові операції, даючи користувачеві чітке уявлення про його поточний фінансовий стан. Це дозволяє вчасно виявляти фінансові труднощі, контролювати бюджет та планувати витрати. Іншою важливою функцією є статистика, яка дозволяє користувачеві аналізувати свої

фінансові звички, визначати тенденції та оцінювати ефективність своїх фінансових рішень. Завдяки цьому можна виявити найбільш витратні категорії і прийняти відповідні заходи для оптимізації бюджету. Прогнозування є наступною важливою функцією, яке дозволяє користувачеві на основі поточних даних та минулих тенденцій передбачати майбутні доходи та витрати. Це забезпечує можливість планування фінансових цілей, таких як накопичення на великі покупки чи забезпечення фінансової стабільності в довгостроковій перспективі [6].

Вимоги до безпеки та захисту даних є критичними для персонального фінансового асистента, адже він оперує з конфіденційною інформацією про фінанси користувача. Важливо, щоб ці дані були захищені від несанкціонованого доступу, зловмисників або витоку. Для цього повинні бути застосовані сучасні методи шифрування, багатофакторна аутентифікація, регулярні оновлення безпеки програмного забезпечення та заходи для захисту від потенційних атак. Лише за умови високого рівня безпеки користувач може довіряти фінансовому асистенту та використовувати його функції без побоювань за конфіденційність своїх даних [20].

1.3. Постановка задачі для розробки фінансового асистента

Основною метою розробки фінансового асистента є автоматизація обліку фінансових операцій, що дозволить знизити час, необхідний для виконання повсякденних фінансових задач, а також мінімізувати помилки, пов’язані з ручним введенням даних. Для досягнення цієї мети система повинна забезпечувати автоматичне введення, обробку та збереження фінансових операцій, включаючи доходи, витрати, інвестиції та інші фінансові показники, що дозволяє користувачам легко відстежувати свої фінансові потоки. Однією з ключових складових проекту є створення зручного інтерфейсу, який має бути інтуїтивно зрозумілим і легким у

використанні для користувачів будь-якого рівня технічної підготовки [22].

Інтерфейс повинен включати можливості швидкого доступу до основних функцій, таких як перегляд балансу, створення звітів, планування бюджету та моніторинг витрат. Важливо забезпечити інтуїтивно зрозумілий та адаптивний дизайн, який дозволить користувачам легко орієнтуватися у застосунку незалежно від рівня їхньої фінансової грамотності. Зручна навігація, персоналізація інтерфейсу та гнучкість налаштувань мають сприяти ефективному управлінню фінансами. Однією з ключових складових системи є інтеграція аналітики, яка дозволить користувачам отримувати глибокий аналіз їхньої фінансової ситуації. Це включає автоматичне розпізнавання фінансових шаблонів, прогнозування витрат і доходів на основі попередніх даних та рекомендації щодо оптимізації бюджету. Аналітичний модуль повинен підтримувати динамічні звіти з можливістю деталізації інформації, що допоможе користувачам приймати обґрунтовані фінансові рішення.

Система повинна пропонувати графічне відображення фінансових даних, зокрема інтерактивні діаграми, графіки та інфографіку, що дозволить швидко оцінювати стан бюджету та виявляти тенденції у фінансових потоках. Важливим аспектом є можливість налаштування параметрів візуалізації відповідно до уподобань користувача, наприклад, вибір кольорової схеми або формату представлення інформації. Забезпечення високої надійності та безпеки даних є одним із ключових завдань системи, оскільки фінансова інформація є надзвичайно чутливою. У зв’язку з цим передбачається реалізація багаторівневого захисту, включаючи шифрування даних, двофакторну аутентифікацію, автоматичне резервне копіювання та механізми виявлення підозрілих транзакцій.

Додатково, система повинна передбачати можливість гнучкого налаштування прав доступу, що дозволить обмежувати доступ до певних функцій або даних залежно від ролі користувача. Це особливо важливо для багатокористувацьких середовищ, де одна фінансова система може

використовуватися сім’єю, компанією або фінансовим консультантом для управління фінансами клієнтів.

Крім того, застосунок має підтримувати функціональність імпорту та експорту даних у різних форматах для інтеграції з іншими бухгалтерськими або фінансовими системами. Це дозволить користувачам легко переносити свої фінансові дані між різними платформами та сервісами, забезпечуючи безперервність роботи та зручність аналізу інформації.

Вимоги до продуктивності системи передбачають швидке виконання всіх операцій, обробку великих обсягів даних без значних затримок та оптимізацію використання ресурсів пристрою. Оптимізація запитів до бази даних, використання кешування та мінімізація навантаження на процесор і оперативну пам’ять дозволять забезпечити стабільну роботу застосунку навіть на пристроях із середніми характеристиками. Враховуючи сучасні тенденції та потреби користувачів, система повинна бути масштабованою та адаптованою до різних платформ, включаючи десктопні та мобільні версії. Це забезпечить універсальність застосунку, дозволяючи користувачам взаємодіяти з фінансовими даними у будь-який зручний для них спосіб. Підтримка синхронізації між пристроями через хмарні сервіси дозволить користувачам отримувати актуальну інформацію в режимі реального часу незалежно від того, який пристрій вони використовують.

Висновки до першого розділу

У першому розділі було здійснено ґрунтовний аналіз сучасного стану у сфері персональних фінансових асистентів. Проведений огляд існуючих рішень дозволив визначити основні функціональні можливості таких систем, зокрема: ведення обліку доходів і витрат, побудову бюджетів, нагадування про фінансові події, автоматичну класифікацію транзакцій, а також надання рекомендацій щодо оптимізації витрат.

На основі аналізу були сформульовані основні вимоги до ефективного персонального фінансового асистента. До них належать: інтуїтивно

зрозумілий інтерфейс, захист персональних даних, адаптивність до змін фінансової поведінки користувача, можливість інтеграції з банківськими системами та іншими сервісами, наявність аналітичних інструментів та механізмів візуалізації фінансової інформації.

У результаті аналізу проблемної області було чітко сформульовано постановку задачі для подальшої розробки персонального фінансового асистента. Зокрема, визначено, що система має бути зручною у користуванні, підтримувати автоматизований облік фінансових операцій, мати можливість прогнозування та аналізу витрат, а також функціонал персоналізованих порад для користувача. Отримані результати створюють необхідне підґрунтя для подальшого проєктування та реалізації ефективного програмного рішення.

РОЗДІЛ 2
ВИБІР ЗАСОБІВ ДЛЯ РЕАЛІЗАЦІЇ ПЕРСОНАЛЬНОГО ФІНАНСОВОГО АСИСТЕНТА

2.1. Обґрунтування вибору технологій

При розробці фінансового додатка було прийнято рішення реалізувати його як кросплатформене рішення, сумісне з операційними системами Android та iOS. Такий підхід обрано з огляду на високу популярність обох платформ серед користувачів, що дозволяє значно розширити потенційну аудиторію, зробити додаток доступним для більшості мобільних пристроїв та забезпечити стабільний користувацький досвід незалежно від обраної операційної системи. Кросплатформний підхід сприяє оптимізації витрат на розробку та технічну підтримку, оскільки усуває необхідність створювати дві окремі версії додатка для різних операційних систем. Це не лише скорочує часові та фінансові витрати на програмування, тестування й оновлення, а й дозволяє команді розробників зосередитися на поліпшенні функціоналу та якості застосунку.

Однією з головних переваг цього підходу є можливість використання єдиного коду для обох платформ, що спрощує розгортання нових версій та оновлень, а також забезпечує швидке впровадження нових функцій. Крім того, використання кросплатформених технологій сприяє покращенню продуктивності та узгодженості візуального оформлення додатка, що позитивно впливає на його зручність і привабливість для користувачів.

Завдяки кросплатформній розробці також спрощується інтеграція додатка з різними сервісами та API, що розширює його функціональні можливості. Окрім цього, забезпечується єдина логіка обробки даних та синхронізація між пристроями, що дозволяє користувачам безперешкодно працювати з фінансовою інформацією на різних гаджетах. Таким чином, вибір кросплатформеного підходу при розробці фінансового асистента

дозволяє не лише забезпечити ширший доступ користувачів до додатка, але й значно покращити ефективність його розробки, технічної підтримки та подальшого масштабування [21].

Для реалізації кросплатформеного рішення було обрано React Native — один із найпопулярніших фреймворків для розробки мобільних застосунків. Основна перевага React Native полягає у можливості написання єдиного коду, який працює на обох платформах, що дозволяє значно скоротити час і ресурси на розробку. На відміну від гібридних рішень, React Native працює безпосередньо з нативними компонентами, що забезпечує високу продуктивність та плавність інтерфейсу, наближену до нативних застосунків.

Ключовим критерієм вибору React Native став баланс між продуктивністю, зручністю розробки та можливістю масштабування. Основні переваги використання React Native у межах даного проєкту наведено нижче.

Кросплатформеність. React Native дозволяє створювати застосунки для iOS та Android на основі одного коду, що суттєво зменшує витрати на розробку та час на тестування.

Висока продуктивність. На відміну від веб-застосунків, React Native працює безпосередньо з нативними компонентами платформи, що забезпечує високу швидкість та плавність роботи застосунку.

Велика екосистема бібліотек. React Native має велику спільноту розробників та велику кількість готових рішень (бібліотек і плагінів), що прискорює процес розробки та дозволяє легко додавати нові функції.

Оновлення в реальному часі. Завдяки використанню Hot Reloading, розробник може вносити зміни до коду без перезавантаження застосунку, що пришвидшує процес налагодження.

Зручність у розробці. React Native дозволяє писати код мовою JavaScript, що робить розробку доступною для більшої кількості програмістів та спрощує процес підтримки коду.

Також важливим фактором є можливість інтеграції React Native з популярними бібліотеками для керування станом застосунку, такими як

Redux та Context API. Це забезпечує стабільну роботу застосунку навіть за умови великої кількості користувачів та високого навантаження на сервер.

Важливим аспектом роботи є забезпечення зручності та інтуїтивності інтерфейсу – користувач має можливість легко додавати та редагувати записи, переглядати графіки витрат, фільтрувати транзакції за категоріями та отримувати аналітику у зручному вигляді.

Таким чином, вибір системи розробки React Native для реалізації фінансового менеджера є логічним і обґрунтованим рішенням, оскільки він поєднує у собі високу продуктивність, кросплатформеність та легкість у налаштуванні та розробці. Створений застосунок допоможе користувачам контролювати свої фінанси, оптимізувати витрати та досягати фінансової стабільності. Застосунок надасть користувачеві всі необхідні інструменти для ефективного управління бюджетом у зручному мобільному форматі, що є надзвичайно актуальним у сучасному швидкому ритмі життя [15].

Фреймворк підтримує використання JavaScript – популярної та гнучкої мови програмування, що полегшує розробку, а велика екосистема бібліотек та інструментів дає можливість швидко додавати нові функціональні можливості. Важливим фактором при виборі React Native стало також використання механізму Hot Reloading, який дозволяє миттєво вносити зміни до коду без перезавантаження додатка, що суттєво прискорює процес розробки та налагодження [23]. Крім того, React Native має активну спільноту розробників та підтримку від Facebook, що гарантує регулярні оновлення, покращення продуктивності та інтеграцію з новими функціями мобільних платформ. Завдяки цим перевагам використання React Native є оптимальним вибором для створення кросплатформеного додатка «персональний фінансовий помічник», оскільки він забезпечує ефективність розробки, продуктивність, гнучкість у додаванні нових функцій та зручність у подальшій підтримці.

2.2. Вибір архітектури та засобів управління даними

Для ефективного управління станом у мобільному застосунку «Персональний фінансовий асистент» було обрано Redux Toolkit, що забезпечує зручний та оптимізований підхід до роботи з глобальним станом програми. Це інструмент, який дозволяє легко інтегрувати концепцію управління станом у додаток, що є особливо важливим для масштабованих і складних мобільних додатків, таких як фінансові застосунки. Основною перевагою Redux Toolkit є те, що він значно спрощує процес створення та підтримки сховища стану, автоматизуючи рутинні задачі, такі як створення дій (actions) та редукторів (reducers), що, в свою чергу, дозволяє знизити складність коду. Однією з основних переваг використання Redux Toolkit є його здатність зменшувати обсяг шаблонного коду, що зазвичай супроводжує роботу з Redux, таким чином підвищуючи читабельність і полегшуючи підтримку та розширення додатка. Замість того щоб писати великі обсяги кодових блоків для кожної дії, Redux Toolkit дозволяє швидко створювати дії та редуктори за допомогою зручних утиліт, таких як `createSlice`. Це забезпечує більш чистий та зрозумілий код, що зменшує ймовірність помилок.

Крім того, Redux Toolkit дозволяє ефективно працювати з асинхронними запитами, що є важливим для фінансових додатків, де необхідно обробляти запити до зовнішніх джерел даних або зберігати/отримувати транзакції з локальної бази даних. Це досягається завдяки вбудованій підтримці middleware, зокрема Redux Thunk, який дає змогу обробляти асинхронні дії. Наприклад, це дозволяє створити складні сценарії обробки запитів до API, таких як синхронізація даних між сервером і локальним сховищем, що дуже важливо для забезпечення актуальності фінансових транзакцій у мобільному додатку [5].

Завдяки централізованому управлінню станом, яке пропонує Redux Toolkit, інформація в додатку завжди актуальна і синхронізована на всіх

екранах. Це особливо важливо для програм, що мають складну логіку або вимагають збереження даних у різних частинах інтерфейсу. Всі фінансові операції, незалежно від того, де відбувається зміна даних (наприклад, на екрані обробки транзакцій або екрані балансу), автоматично оновлюються на інших екранах. Це забезпечує зручність користувача, який отримує синхронізовану та актуальну інформацію про свої фінанси на будь-якому етапі взаємодії з додатком.

Інтеграція з Redux DevTools є ще однією важливою перевагою, яку надає Redux Toolkit. Цей інструмент дозволяє легко моніторити та налагоджувати стан додатка, надаючи можливість переглядати всі зміни стану та дії в реальному часі. Це значно спрощує процес розробки та тестування, оскільки розробники можуть відслідковувати, як стан змінюється під час взаємодії користувача з додатком. Також, можливість здійснювати «rewind» та «fast-forward» в DevTools дозволяє легко відновлювати попередні стани додатка, що корисно для діагностики помилок або перевірки різних сценаріїв. Загалом, використання Redux Toolkit дозволяє значно покращити ефективність розробки мобільного застосунку «Персональний фінансовий асистент», забезпечуючи зручне управління глобальним станом, легке керування асинхронними операціями та забезпечення зручної та ефективної інтеграції з інструментами для налагодження і тестування [17].

Для збереження даних офлайн у застосунку використовується SQLite, оскільки ця база даних є легкою, швидкою та ідеально підходить для мобільних пристроїв. Важливою перевагою SQLite є можливість зберігання даних без необхідності підключення до інтернету, що забезпечує автономну роботу застосунку. Це особливо актуально для фінансового асистента, оскільки користувачі повинні мати постійний доступ до історії транзакцій, категорій витрат та бюджетів, незалежно від наявності мережі. SQLite забезпечує високу продуктивність завдяки оптимізованій обробці SQL-запитів, що дозволяє швидко зчитувати, змінювати та видаляти дані. Використання цієї бази також дозволяє організувати збереження даних у

структурованому вигляді, забезпечуючи цілісність і консистентність інформації. Крім того, SQLite споживає мінімальні ресурси пристрою, що робить його оптимальним рішенням для мобільних застосунків, які потребують зберігання великого обсягу даних при низькому енергоспоживанні [1, с. 139-141]. Використання локальної бази даних SQLite у мобільному застосунку забезпечує кілька ключових переваг:

Локальне збереження даних – Всі транзакції, категорії витрат та бюджети зберігаються безпосередньо на пристрої користувача, що дозволяє отримати доступ до них навіть за відсутності підключення до інтернету. Це особливо важливо для користувачів, які хочуть мати доступ до своєї фінансової інформації у будь-який момент, незалежно від стану мережі.

Висока швидкість обробки – SQLite оптимізована для швидкого виконання запитів, що дозволяє оперативно отримувати дані про транзакції та бюджети, а також швидко вносити зміни в базу даних без помітних затримок.

Невелике споживання ресурсів – SQLite є легкою базою даних із низькими вимогами до пам’яті та обчислювальних ресурсів пристрою. Це дозволяє ефективно працювати навіть на пристроях зі скромними характеристиками без суттєвого навантаження на систему.

Зручне використання – Бібліотека expo-sqlite надає простий і зрозумілий інтерфейс для взаємодії з базою даних. Вона дозволяє легко створювати таблиці, виконувати SQL-запити та керувати даними за допомогою звичайних JavaScript-функцій [3].

Вибір SQLite у поєднанні з expo-sqlite дозволяє розробити стабільну та надійну систему управління даними, яка зберігає всі фінансові записи на пристрої користувача, забезпечуючи швидкий доступ і можливість резервного копіювання. Така архітектура гарантує безперебійну роботу додатка в будь-яких умовах, підвищуючи його надійність та зручність використання.

2.3. Інструменти для розробки та тестування

React Native дозволяє використовувати єдиний код для обох платформ – Android та iOS, що значно знижує витрати на розробку та підтримку додатка. Цей підхід є надзвичайно ефективним, оскільки розробники не повинні писати окремий код для кожної з платформ, що дозволяє зекономити значний час і ресурси на етапах розробки, тестування та підтримки продукту. Завдяки використанню одного коду для двох платформ, компанії можуть швидше виводити додатки на ринок, зменшуючи витрати на утримання окремих команд для кожної операційної системи. Це, в свою чергу, дозволяє швидше адаптуватися до змін на ринку, знижуючи витрати на оновлення та удосконалення продукту. Крім того, React Native дозволяє використовувати JavaScript для написання бізнес-логіки додатка. Це дає розробникам можливість працювати з потужною та популярною мовою програмування, яка має велику кількість бібліотек та інструментів для вирішення різноманітних задач. Важливою перевагою є те, що через JavaScript можна інтегрувати нативні компоненти для кожної платформи, що дає можливість швидко адаптуватися до нових функцій і оновлень кожної операційної системи. Це забезпечує високу гнучкість та дозволяє підтримувати додаток на актуальному рівні без необхідності повної переробки коду. Завдяки своїй здатності комбінувати нативні компоненти з бізнес-логікою, написаною на JavaScript, React Native пропонує оптимальний баланс між швидкістю розробки та продуктивністю. Це особливо важливо для мобільних додатків, де швидкість виконання і зручність користування є критичними для успіху продукту. Всі ці фактори роблять React Native одним із найбільш ефективних та популярних інструментів для крос-платформної розробки мобільних додатків.

Ця технологія активно використовує компоненти, що спільно

використовуються на обох платформах, зокрема такі як Text, View, ScrollView та інші (рис. 2.1), що значно полегшує процес інтеграції і дозволяє швидше створювати функціональні та зручні додатки.

```
return (
  <ScrollView style={style.container}>
    <View style={style.inputContainer}>
      <TextInput
        style={style.input}
        placeholder="Search for your transactions"
        onChangeText={(text) => setSearch(text)}
        inputMode="search"
      />
    </View>

    <MyButton
      title={isSearching ? "Searching ..." : "Search"}
      variant="primary"
      onPress={handleSearch}
    />

    <Text style={style.searchMessage}>{searchMessage}</Text>

    <ScrollView>
      {searchResult.length > 0 &&
        searchResult.map((eachTx) => (
          <EachSearchResult search={eachTx} key={eachTx.id} />
        ))}
    </ScrollView>
  </ScrollView>
); ← #51-77 return
```

Рис. 2.1. Компоненти

Для розробки мобільного застосунку було обрано платформу Expo (рис. 2.2), що є потужним набором інструментів та бібліотек для розробки додатків на основі React Native. Expo надає розробникам готові рішення для інтеграції з такими можливостями мобільних пристроїв, як геолокація, камери, датчики, бази даних, push-сповіщення та інші функції, що дозволяють швидко створювати високоякісні додатки.

Рис. 2.2 Expo.

Expo дозволяє значно спростити налаштування середовища для розробки, зокрема за допомогою інтеграції з Expo CLI – інтерфейсом командного рядка, який забезпечує швидке створення проектів, зручну налагоджувальну інфраструктуру і просту інтеграцію з хмарними сервісами для розгортання додатків. Це дозволяє зменшити час на налаштування середовища розробки та зосередитися безпосередньо на розробці функціоналу додатка, що є критично важливим для досягнення швидких результатів.

Також за допомогою Expo можна швидко тестувати додаток на мобільних пристроях завдяки функції Expo Go, що дозволяє запускати додаток без необхідності налаштовувати складне середовище для емуляторів або реальних пристроїв, що значно полегшує процес тестування і виявлення помилок на ранніх етапах розробки.

Для реалізації навігації між екранами у мобільному застосунку «персональний фінансовий асистент» було обрано бібліотеку React Navigation, оскільки вона є стандартом для організації маршрутизації у застосунках, розроблених на базі React Native. Основною перевагою React Navigation є її гнучкість та можливість створення різних типів навігації, таких як Stack Navigation, Tab Navigation і Drawer Navigation. Це дозволяє забезпечити зручний перехід між екранами, зберігати історію переглядів та легко налаштовувати вкладки або бічне меню. React Navigation має модульну структуру, що дає можливість масштабувати додаток, додаючи нові екрани та функціональні можливості без значних змін у коді. Додаткова перевага полягає у підтримці анімацій під час переходів, що покращує користувацький досвід, роблячи навігацію плавною та інтуїтивною. Крім того, бібліотека дозволяє налаштовувати заголовки, стилізацію інтерфейсу та взаємодію з нативними можливостями платформи, що робить її ідеальним вибором для створення зручного та функціонального мобільного застосунку [13].

Для налагодження та тестування застосунку використовується Redux DevTools, який є потужним інструментом для відстеження та керування

станом програми. Однією з головних переваг Redux DevTools є можливість покрокового перегляду змін у глобальному стані під час роботи застосунку, що значно спрощує процес виявлення та виправлення помилок. Цей інструмент дозволяє зберігати та відтворювати історію змін, здійснювати "time-travel debugging", що дає змогу повернутися до будь-якого попереднього стану застосунку та проаналізувати, як саме зміни вплинули на його роботу. Завдяки цьому розробник може швидко знайти та усунути проблеми у логіці програми, особливо в частині обробки фінансових операцій та відображення статистики [3]. Використання Redux DevTools також покращує продуктивність розробки, оскільки дає змогу тестувати складні сценарії, відстежувати асинхронні дії та контролювати потік даних між компонентами. Поєднання Redux Toolkit та Redux DevTools забезпечує стабільність роботи застосунку, спрощує налагодження та дає змогу ефективно масштабувати систему, додаючи нові функціональні можливості.

Висновки до другого розділу

У другому розділі було здійснено комплексний аналіз сучасних технологічних рішень з метою обґрунтування вибору оптимальної технологічної основи для розробки персонального фінансового асистента. Визначено набір інструментів, мов програмування, фреймворків і платформ, які відповідають вимогам до продуктивності, масштабованості, безпеки та зручності використання.

Обґрунтовано доцільність застосування архітектурного підходу, що поєднує переваги мікросервісної структури з централізованим управлінням даними, що дозволяє забезпечити гнучкість і надійність системи.

Окрему увагу приділено вибору систем управління базами даних та інструментів для ефективної розробки, тестування і налагодження застосунку. Застосування бази даних SQLite у комбінації з бібліотекою expo-sqlite є обґрунтованим вибором для реалізації локального зберігання даних у

мобільному застосунку персонального фінансового асистента. Така технологічна зв’язка дозволяє зберігати всі фінансові записи безпосередньо на пристрої користувача, забезпечуючи при цьому швидкий доступ до даних, їхню цілісність і можливість резервного копіювання. Це рішення мінімізує залежність від мережевого з’єднання, підвищує стабільність роботи застосунку та сприяє збереженню конфіденційності персональної інформації.

Визначено інструменти, необхідні для розробки, тестування та налагодження системи. Зокрема, акцент зроблено на інструментах, що дозволяють забезпечити гнучкість розробки, ефективне тестування функціоналу та виявлення потенційних помилок на ранніх етапах розробки.

Таким чином, результати цього етапу дослідження створюють міцне підґрунтя для переходу до практичної реалізації персонального фінансового асистента, забезпечуючи збалансованість між функціональністю, стабільністю та зручністю користування.

РОЗДІЛ 3.
ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Розроблення застосунку «Фінансовий менеджер» на системі Android

У сучасному світі фінансова грамотність та ефективне управління особистими коштами є невід'ємними чинниками стабільного та передбачуваного життя. Багато людей стикаються з труднощами при веденні особистого бюджету, що зумовлено недостатнім контролем над доходами і витратами, наявністю кредитних зобов'язань, а також непередбаченими фінансовими ситуаціями. Такі фактори часто призводять до фінансової нестабільності, боргових проблем та неможливості досягнення запланованих цілей.

З розвитком цифрових технологій і широким розповсюдженням мобільних пристроїв усе більше людей обирають мобільні застосунки як інструмент для вирішення повсякденних задач, зокрема — для управління особистими фінансами. Подібні застосунки сприяють підвищенню фінансової дисципліни, дозволяють користувачам планувати витрати, здійснювати облік доходів і заощаджень, контролювати боргові зобов'язання та аналізувати власний фінансовий стан.

Попри високу популярність таких сервісів, значна частина наявних застосунків має низку недоліків, серед яких — обмежений функціонал, відсутність персоналізованої аналітики або складний для користувача інтерфейс. Це знижує ефективність їх використання та не задовольняє запити широкої аудиторії.

У зв'язку з цим виникає потреба у створенні універсального, інтуїтивно зрозумілого та функціонально потужного мобільного застосунку для автоматизованого управління особистими фінансами. Такий застосунок має забезпечувати користувачу можливість зручно фіксувати доходи й витрати,

класифікувати їх за категоріями, формувати наочну фінансову статистику, а також отримувати персоналізовані рекомендації щодо оптимізації витрат. У результаті це сприятиме підвищенню ефективності планування бюджету, зростанню фінансової дисципліни та досягненню користувачем довгострокових фінансових цілей.

Для реалізації цього застосунку було обрано фреймворк React Native (рис. 3.1). Це популярний інструмент для створення кросплатформених мобільних застосунків, який дозволяє писати один код для двох основних мобільних платформ – Android та iOS. Таким чином, створений застосунок стане надійним інструментом для управління фінансами, який допоможе користувачам підвищити рівень фінансової грамотності, контролювати свої витрати та оптимізувати фінансові рішення. React Native забезпечить високу продуктивність і зручність у використанні, що зробить застосунок доступним для широкої аудиторії користувачів.

React Navigation

Для організації навігації між екранами в застосунку було обрано бібліотеку React Navigation, яка є стандартом для навігації в React Native. React Navigation забезпечує зручний, модульний та гнучкий інтерфейс для побудови навігаційних маршрутів, що дозволяє ефективно організувати взаємодію користувача з додатком. У фінансовому менеджері застосовується кілька типів навігації, що забезпечують зручний доступ до основних функцій додатка та оптимізують користувацький досвід [18].

Stack Navigation є основним механізмом навігації, що дозволяє створювати типову лінійну навігацію, де екрани додаються в стек один за одним, а перехід між екранами здійснюється за допомогою функцій push і pop. Коли користувач переходить на новий екран, він додається до стеку, і коли він натискає кнопку "Назад", попередній екран виводиться з верхньої частини стека. Це дозволяє створювати зручні та інтуїтивно зрозумілі маршрути для переходу між екранами, наприклад, перехід до детального перегляду транзакцій або налаштувань.

Tab Navigation використовується для реалізації нижніх та верхніх вкладок, що дає змогу ефективно організувати навігацію між основними категоріями додатка. Вкладки дозволяють користувачам швидко перемикатися між різними функціями додатку без необхідності повертатися до головного екрана. У фінансовому менеджері ці вкладки відповідають за різні функціональні блоки: баланс, витрати, звітність, що полегшує користувачу доступ до потрібної інформації [14].

```
  return (
    <NavigationContainer>
      <Tab.Navigator
        activeColor={allColors.tabActive}
        inactiveColor={allColors.tabInactive}
        barStyle={{ backgroundColor: allColors.tabBackground }}
      >
        <Tab.Screen
          name="TransactionScreen"
          component={MainTransactionScreen}
          options={{
            title: "Transactions",
            tabBarIcon: ({ color }) => (
              <MaterialIcons name="grid-view" size={28} color={color} />
            ),
          }} ← #84-89 options=
        />
        <Tab.Screen
          name="MainSummaryScreen"
          component={MainSummaryScreen}
          options={{
            title: "Summary",
            tabBarIcon: ({ color }) => (
              <MaterialIcons
                name="account-balance-wallet"
                size={28}
                color={color}
              />
            ), ← #96-102 tabBarIcon: ({ color }) =>
          }} ← #94-103 options=
        />


      </Tab.Navigator>
    </NavigationContainer>
  ); ← #74-109 return
}; ← #30-110 const Main = () =>

export default Main;
```

Рис. 3.1. Навігація Tab Navigation.

Вкладки у нижній частині екрана є популярним і зручним елементом інтерфейсу в мобільних додатках, оскільки вони дозволяють забезпечити швидкий доступ до найбільш використовуваних функцій.

Ще однією важливою особливістю навігації у фінансовому менеджері є використання бібліотеки Material Top and Bottom Tabs. Вона надає можливість реалізувати вкладки, які відповідають стандартам Material Design, популярного дизайну для мобільних додатків, особливо на платформі Android. Цей тип навігації не тільки естетично привабливий, але й забезпечує користувачам зручний інтерфейс для перемикання між різними екранами [8].

Material Design фокусується на наданні користувачеві зручного та інтуїтивно зрозумілого інтерфейсу, що дозволяє легко орієнтуватися в додатку. Вкладки в стилі Material Design забезпечують гарний візуальний вигляд і допомагають користувачам орієнтуватися в додатку, що підвищує загальний користувацький досвід.

React Navigation дозволяє гнучко комбінувати ці типи навігації, що дозволяє створити зручну і логічну структуру навігації в додатку, що відповідає потребам користувачів. Крім того, бібліотека надає широкий набір параметрів для налаштування анімацій переходів, що підвищує привабливість і функціональність застосунку.

3.2. Управління станом за допомогою Redux

Для централізованого зберігання та управління станом додатка в проекті було використано бібліотеку Redux, що дозволяє зберігати і керувати глобальним станом на одному місці. У застосунку Фінансовий менеджер це включає в себе збереження даних про транзакції, бюджетні ліміти, історію витрат та доходів, що забезпечує консистентність даних між різними екранами.

Redux дозволяє зберігати всі дані застосунку в єдиному store (рис. 3.2), що дозволяє у будь-який момент часу отримати актуальну інформацію з будь-якого компонента. Це особливо корисно для додатків, які мають складну бізнес-логіку або потребують синхронізації між різними частинами інтерфейсу. Наприклад, в фінансовому менеджері користувач може мати доступ до змін в балансі або витратах на різних екранах одночасно, і Redux гарантує, що ці дані будуть оновлюватися в реальному часі, коли користувач взаємодіє з додатком.

```
import { configureStore } from "@reduxjs/toolkit";
import incomeCategoryReducer from "../features/incomeCategory/incomeCategorySlice";
import expensesCategoryReducer from "../features/expensesCategory/expensesCategorySlice";
import accountReducer from "../features/account/accountSlice";
import incomeReducer from "../features/income/incomeSlice";
import expensesReducer from "../features/expenses/expensesSlice";
import selectedMonthReducer from "../features/selectedMonth/selectedMonthSlice";
import transactionsCounterReducer from "../features/transactionsCounter/transactionsCounterSlice";

export const store = configureStore({
  reducer: {
    incomeCategory: incomeCategoryReducer,
    expensesCategory: expensesCategoryReducer,
    account: accountReducer,
    income: incomeReducer,
    expenses: expensesReducer,
    selectedMonth: selectedMonthReducer,
    transactionsCounter: transactionsCounterReducer,
  }, ← #11-19 reducer:
});| ← #10-20 export const store = configureStore
```

Рис. 3.2. Store.

У фінансовому менеджері Redux відповідає за зберігання важливих даних, таких як історія витрат, категорії витрат, бюджети, інші налаштування та параметри користувача. Це дає змогу застосунку бути більш ефективним і динамічним, оскільки стан застосунку оновлюється в реальному часі та без

Для спрощення роботи з Redux було використано Redux Toolkit, що надає зручні інструменти для обробки асинхронних запитів та створення редукторів і сторів без необхідності написання великої кількості шаблонного коду. Redux Toolkit включає функції, які дозволяють легко створювати редуктори, автоматично генерувати екшн-креатори та керувати асинхронними запитами без необхідності ручного налаштування [19].

Однією з основних переваг Redux Toolkit є інтеграція з Thunk для асинхронних дій, що дозволяє здійснювати запити до API або виконувати інші асинхронні операції без необхідності додаткових налаштувань. Наприклад, запити для отримання списку транзакцій або оновлення балансу можуть бути оброблені за допомогою Redux Toolkit, що робить код більш чистим і зрозумілим.

Використання Redux значно спрощує процес тестування, оскільки всі зміни стану проходять через єдиний глобальний стан. Це дозволяє створювати unit-тести для редукторів і перевіряти, чи правильно оновлюється стан в різних сценаріях. Всі зміни в додатку можна відслідковувати через Redux DevTools, що забезпечує зручну можливість дебаггінгу та відстеження

історії змін стану.

Крім того, Redux допомагає зробити додаток масштабованим. Коли додаток росте і з'являються нові функції, можна додавати нові slice (рис. 3.3), в Redux store, що дозволяє зберігати і керувати новими даними, не впливаючи на інші частини програми. Це також дозволяє підтримувати чистоту коду, оскільки кожен новий шматок даних і його обробка будуть чітко розділені на окремі частини програми [12].

```
const expensesCategorySlice = createSlice({
  name: "expensesCategory",
  initialState,
  reducers: {
    initExpensesCategoryFromDb: (state, action) => {
      let category = [];

      for (eachElement of action.payload) {
        const { id, value } = eachElement;

        if (!!id && !!value) {
          category = [...category, { id, value }];
        }
      } ← #17-23 for (eachElement of action.payload)
      return category;
    }, ← #14-25 initExpensesCategoryFromDb: (state, action) =>

    addExpensesCategory: (state, action) => {
      const { id, value } = action.payload;

      if (!!id && !!value) {
        return [...state, { id, value }];
      }
      return state;
    }, ← #27-34 addExpensesCategory: (state, action) =>

    updateExpensesCategory: (state, action) => {
      const { id, value } = action.payload;

      if (!!id && !!value) {
        return state.map((expensesCategory) =>
          expensesCategory.id === id ? expensesCategory : { id, value }
        );
      } ← #39-43 if (!!id && !!value)

      return state;
    }, ← #36-46 updateExpensesCategory: (state, action) =>

    deleteExpensesCategory: (state, action) => {
      const { id } = action.payload;

      if (id) {
        return state.map((expensesCategory) => expensesCategory.id === id);
      }

      return state;
    }, ← #48-56 deleteExpensesCategory: (state, action) =>
  }, ← #13-57 reducers:
}); ← #10-58 const expensesCategorySlice = createSlice
```

Рис. 3.3. Slice

Таким чином, використання Redux в комбінації з Redux Toolkit забезпечує ефективне управління станом і дозволяє створювати додатки, які зручні для тестування, масштабування та підтримки в довгостроковій перспективі.

3.3. Збереження даних за допомогою SQLite

Для реалізації функціональності збереження даних про транзакції, категорії витрат та бюджети у мобільному застосунку було використано expo-sqlite – бібліотеку, що дозволяє взаємодіяти з локальною базою даних SQLite безпосередньо на пристрої користувача. SQLite (рис. 3.4), є однією з найпопулярніших вбудованих баз даних, яка забезпечує високу швидкість обробки запитів і надійне зберігання великих обсягів інформації при мінімальних витратах на ресурси пристрою.

```
import * as SQLite from "expo-sqlite";

const database = SQLite.openDatabase("mmdata.db");    "mmdata": Unknown word.

// table initialization
export const initializeAccountTable = () => {
  const promise = new Promise((resolve, reject) => {
    database.transaction((tx) => {
      tx.executeSql(
        `
          CREATE TABLE IF NOT EXISTS account (
            id TEXT PRIMARY KEY,
            value TEXT NOT NULL
          )
        `,
        [],
        () => {
          resolve();
        },
        (_, error) => {
          reject(error);
        }
      ); ← #9-23 tx.executeSql
    }); ← #8-24 database.transaction
  }); ← #7-25 const promise = new Promise

  return promise;
}; ← #6-28 export const initializeAccountTable = () =>

export const initializeExpensesCategoryTable = () => {
  const promise = new Promise((resolve, reject) => {
    database.transaction((tx) => {
      tx.executeSql(
        `CREATE TABLE IF NOT EXISTS expensesCategory (
          id TEXT PRIMARY KEY,
          value TEXT NOT NULL
        )`,
        [],
        () => {
          resolve();
        },
        (_, error) => {
          reject(error);
        }
      ); ← #33-45 tx.executeSql
    }); ← #32-46 database.transaction
  }); ← #31-47 const promise = new Promise

  return promise;
}; ← #30-50 export const initializeExpensesCategoryTable = () =>
```

Рис. 3.4. SQLite

У базі даних створено кілька основних таблиць:

Transactions – таблиця для зберігання інформації про фінансові операції

користувача. Вона містить поля для суми транзакції, дати, категорії та типу (наприклад, дохід або витрата).

Categories – таблиця для зберігання категорій витрат і доходів. Кожна категорія має унікальний ідентифікатор, назву та колір для візуальної ідентифікації в інтерфейсі застосунку.

Budgets – таблиця для зберігання інформації про бюджети користувача, включаючи суму бюджету, період дії та відповідні категорії.

Під час ініціалізації бази даних застосунок створює необхідні таблиці за допомогою SQL-запитів, наприклад, створення таблиці для транзакцій. Взаємодія з базою даних для внесення нових транзакцій у базу використовується метод executeSql (рис. 3.5), що дозволяє виконувати SQL-запити.

```
export const initializeExpensesTable = () => {
  const promise = new Promise((resolve, reject) => {
    database.transaction((tx) => {
      tx.executeSql(
        `
        CREATE TABLE IF NOT EXISTS expenses (
          id TEXT PRIMARY KEY,
          amount INTEGER,
          date TEXT,
          note TEXT,
          type TEXT,
          account TEXT,
          category TEXT,
          FOREIGN KEY (category) REFERENCES expensesCategory (id) ON DELETE SET NULL,
          FOREIGN KEY (account) REFERENCES account (id) ON DELETE SET NULL
        )
      `,
        [],
        () => {
          resolve();
        },
        (_, error) => {
          reject(error);
        }
      ); ← #108-129 tx.executeSql
    }); ← #107-130 database.transaction
  }); ← #106-131 const promise = new Promise

  return promise;
}; ← #105-134 export const initializeExpensesTable = () =>
```

Рис. 3.5. Створення таблиці

У базі даних створюється кілька таблиць, кожна з яких відповідає за певний тип інформації.

Користувач може створювати різні рахунки або акаунти для відстеження своїх коштів, наприклад, банківський рахунок, готівку чи кредитну картку. Під час створення рахунку користувач вказує назву та

початковий баланс. Якщо потрібно, рахунок можна відредагувати або видалити. Усі пов’язані з рахунком транзакції автоматично оновлюються після внесення змін. Якщо користувач хоче видалити рахунок із прив’язаними транзакціями, застосунок попередить про це та запропонує спочатку видалити пов’язані записи або перенести їх на інший рахунок. Це дозволяє уникнути помилок і втрати даних.

Для кращого розуміння фінансових потоків користувач може створювати категорії доходів і витрат. Наприклад, серед доходів можна створити категорії «Зарплата», «Подарунки», «Інвестиції», а серед витрат – «Їжа», «Транспорт», «Оренда». Під час створення транзакції користувач обирає відповідну категорію, що допомагає у подальшому аналізі фінансового стану. Усі категорії можна редагувати або видаляти у будь-який момент, і зміни автоматично відобразяться у всіх пов’язаних транзакціях (рис. 3.6).

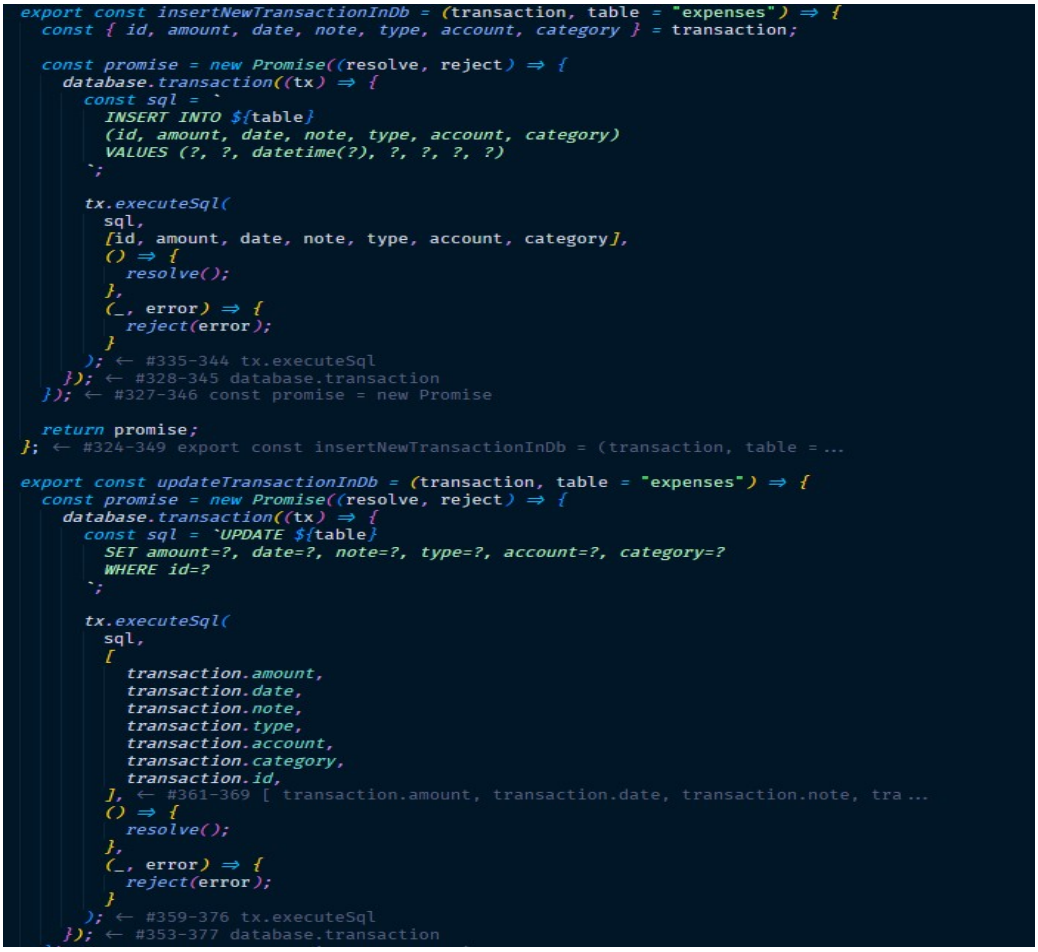


Рис. 3.6 Додавання оновлення транзакцій

Додавання транзакцій у застосунку виконується максимально просто. Якщо користувач отримав зарплату у розмірі 15 000 грн, він може створити транзакцію, зазначивши рахунок "Банк", категорію "Зарплата" і додавши коментар "зарплата за березень". Сума автоматично додається до загального балансу рахунку. У випадку витрат, наприклад, на їжу, користувач створює транзакцію на суму 500 грн із зазначенням категорії "Їжа" та рахунку "Картка", і ця сума автоматично віднімається з балансу рахунку. Якщо користувач помилився або хоче оновити інформацію, транзакцію можна відредагувати або видалити.

Застосунок автоматично підсумовує всі доходи та витрати і відображає поточний баланс по кожному рахунку. Якщо користувач має рахунок "Банк" із балансом 10 000 грн і рахунок "Готівка" із 5 000 грн, застосунок покаже загальний баланс у розмірі 15 000 грн. При кожній новій транзакції баланс оновлюється миттєво, що дозволяє користувачу завжди бачити актуальний стан фінансів.

Користувач може переглядати доходи та витрати за будь-який період – день, тиждень, місяць або рік. Наприклад, якщо потрібно дізнатися, скільки було витрачено на їжу у березні, застосунок відфільтрує відповідні транзакції та покаже підсумкову суму. Це дозволяє побачити, куди саме йдуть гроші, і зрозуміти, на чому можна зекономити. Для кращої наочності застосунок також відображає статистику у вигляді кругових або стовпчастих діаграм, де можна побачити співвідношення доходів і витрат за категоріями.

Якщо користувач додає нову транзакцію або редагує існуючу, баланс рахунку оновлюється одразу, без необхідності перезапуску застосунку. Наприклад, якщо користувач додає дохід у розмірі 1 000 грн на рахунок "Картка", баланс рахунку одразу зміниться і оновиться загальний баланс. Це дозволяє користувачу швидко реагувати на зміни у фінансовому стані.

Застосунок має зручний механізм видалення рахунків та транзакцій (рис. 3.7). Якщо користувач хоче видалити рахунок із прив’язаними транзакціями, застосунок попередить про можливу втрату даних і запропонує

перенести транзакції на інший рахунок або видалити їх. Це запобігає випадковій втраті важливої інформації. Те саме стосується транзакцій – перед видаленням застосунок запитує підтвердження, щоб уникнути помилкових дій.

```
export const deleteTransactionFromDb = (id, table = "expenses") => {
  const promise = new Promise((resolve, reject) => {
    const sql = `DELETE FROM ${table} WHERE id = ?`;

    database.transaction((tx) => {
      tx.executeSql(
        sql,
        [id],
        () => {
          resolve();
        },
        (_, error) => {
          reject(error);
        }
      ); ← #388-397 tx.executeSql
    }); ← #387-398 database.transaction
  }); ← #384-399 const promise = new Promise

  return promise;
}; ← #383-402 export const deleteTransactionFromDb = (id, table = "expenses ...

export const dropTableFromDB = (table = "income") => {
  const promise = new Promise((resolve, reject) => {
    database.transaction((tx) => {
      const sql = `DROP TABLE ${table}`;
      console.log("sql :", sql);
      tx.executeSql(
        sql,

        [],
        (_, result) => {
          resolve(result);
        },
        (_, error) => {
          reject(error);
        }
      ); ← #409-419 tx.executeSql
    }); ← #406-420 database.transaction
  }); ← #405-421 const promise = new Promise

  return promise;
}; ← #404-424 export const dropTableFromDB = (table = "income") =>
```

Рис.3.7. Видалення таблиць

Користувач може змінювати назви рахунків та категорій у будь-який момент. Якщо, наприклад, користувач хоче перейменувати категорію «Транспорт» на «Поїздки», застосунок автоматично оновить усі пов’язані транзакції. Якщо змінюється назва рахунку, всі відповідні транзакції також оновлюються автоматично. Це допомагає підтримувати консистентність даних та уникати плутанини.

Для запобігання випадковій втраті даних застосунок створює резервні

копії бази даних. Користувач також може вручну створити резервну копію та відновити її у разі необхідності. Це забезпечує захист інформації навіть у випадку помилок користувача чи технічних проблем.

Застосунок дозволяє користувачу переглядати підсумкові дані за будь-який період. Якщо потрібно побачити підсумок за місяць, застосунок відобразить загальні доходи та витрати, а також різницю між ними. Якщо доходи перевищують витрати, результат відобразиться зеленим кольором, якщо витрати більші — червоним. Це дозволяє користувачу швидко оцінити стан своїх фінансів і прийняти відповідні рішення.

Щоб полегшити роботу з даними, застосунок дозволяє сортувати транзакції за датою, сумою та категорією (рис. 3.8). Також можна застосовувати фільтри для відображення лише доходів, витрат або певного типу транзакцій. Наприклад, користувач може переглянути лише витрати на їжу за останній місяць або тільки доходи на рахунок "Картка" за поточний тиждень.

```
export const searchInDb = (searchText, table = "income") => {
  const promise = new Promise((resolve, reject) => {
    const sql = `SELECT * FROM ${table} WHERE note LIKE '%${searchText}%' `;
    database.transaction((tx) => {
      tx.executeSql(
        sql,
        [],
        (_, result) => {
          const data = result.rows._array;
          resolve(data);
        },
        (_, error) => reject(error)
      ); ← #460-468 tx.executeSql
    }); ← #459-469 database.transaction
  }); ← #457-470 const promise = new Promise

  return promise;
}; ← #456-473 export const searchInDb = (searchText, table = "income") =>
```

Рис. 3.8. Сортування таблиць

Застосунок працює без інтернету, оскільки всі дані зберігаються локально у файлі бази даних SQLite. Якщо користувач перемістить файл на інший пристрій, застосунок зможе зчитати інформацію і відновити всі транзакції та налаштування без додаткових дій.

Таким чином, застосунок надає користувачу зручний та

функціональний інструмент для управління фінансами. Він дозволяє легко додавати та редагувати рахунки, вести облік доходів і витрат, переглядати підсумкові дані та статистику, а також контролювати стан фінансів у реальному часі. Завдяки інтуїтивному інтерфейсу та миттєвому оновленню балансу користувач отримує повний контроль над своїми фінансами та можливість приймати зважені фінансові рішення.

3.4. Візуалізація фінансових даних

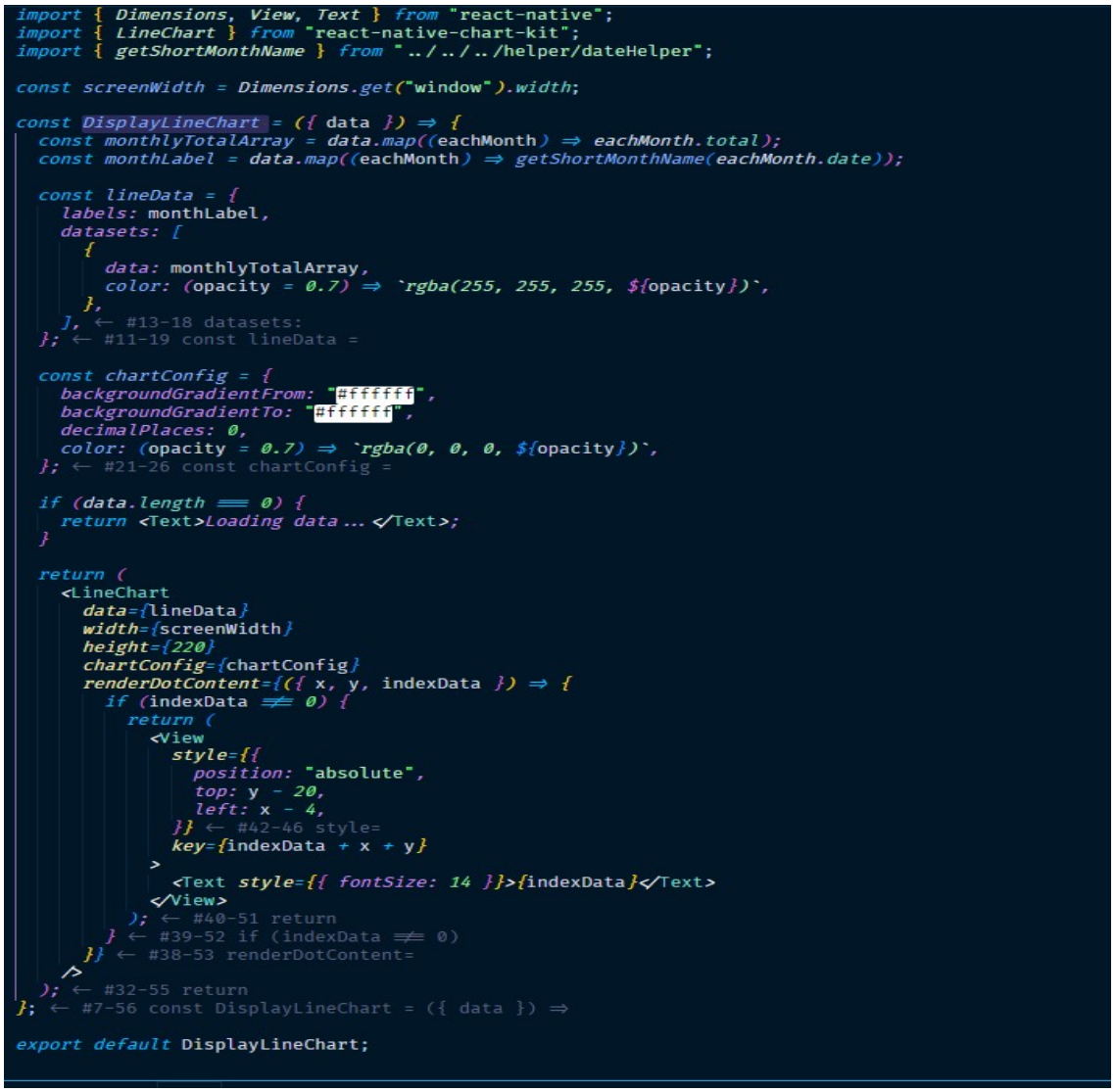
Візуалізація фінансових даних відіграє ключову роль у покращенні користувацького досвіду, оскільки дозволяє користувачам швидко і наочно аналізувати стан своїх фінансів. У застосунку реалізована система інтерактивних графіків, яка допомагає користувачу краще розуміти структуру своїх доходів і витрат, спостерігати за тенденціями та приймати зважені фінансові рішення. Для побудови графіків використовується бібліотека react-native-chart-kit, яка забезпечує широкі можливості для створення різних типів діаграм та дозволяє створювати привабливі, динамічні й зручні у використанні графіки.

Однією з основних переваг використання графіків є можливість швидко побачити загальну картину фінансового стану. Наприклад, якщо користувач хоче зрозуміти, на що витрачається більша частина бюджету, він може переглянути кругову діаграму (рис. 3.9), яка наочно покаже, який відсоток загальних витрат припадає на певні категорії — наприклад, їжу, житло, транспорт тощо. Такий візуальний підхід дозволяє користувачу миттєво побачити, яка категорія споживає найбільше коштів, і відповідно прийняти рішення щодо оптимізації витрат.

Рис. 3.9. Кругова діаграма

Окрім кругових діаграм, застосунок також дозволяє створювати лінійні графіки (рис. 3.10), які відображають динаміку змін фінансового стану з плином часу. Наприклад, користувач може побачити, як змінювалися його витрати або доходи протягом останнього місяця, тижня або навіть року. Лінійний графік допомагає виявити певні закономірності та тренди — наприклад, користувач може помітити, що його витрати на їжу значно зростають у вихідні або що дохід збільшується на початку місяця після надходження зарплати.

Візуалізація фінансових даних у застосунку є інтерактивною, що значно покращує користувацький досвід. Користувач може взаємодіяти з графіками — наприклад, натискати на певний сегмент кругової діаграми, щоб побачити деталі щодо конкретної категорії витрат або доходів. На лінійних графіках можна переглядати точні значення для кожного дня або місяця, просто переміщаючи палець по екрану. Це дозволяє користувачу отримувати більше інформації без необхідності переходити на інші екрани або відкривати додаткові розділи.



```
import { Dimensions, View, Text } from "react-native";
import { LineChart } from "react-native-chart-kit";
import { getShortMonthName } from "../../../helper/dateHelper";

const screenWidth = Dimensions.get("window").width;

const DisplayLineChart = ({ data }) => {
  const monthlyTotalArray = data.map((eachMonth) => eachMonth.total);
  const monthLabel = data.map((eachMonth) => getShortMonthName(eachMonth.date));

  const lineData = {
    labels: monthLabel,
    datasets: [
      {
        data: monthlyTotalArray,
        color: (opacity = 0.7) => `rgba(255, 255, 255, ${opacity})`,
      },
    ], ← #13-18 datasets:
  }; ← #11-19 const lineData =

  const chartConfig = {
    backgroundGradientFrom: "#ffffff",
    backgroundGradientTo: "#ffffff",
    decimalPlaces: 0,
    color: (opacity = 0.7) => `rgba(0, 0, 0, ${opacity})`,
  }; ← #21-26 const chartConfig =

  if (data.length === 0) {
    return <Text>Loading data ...</Text>;
  }

  return (
    <LineChart
      data={lineData}
      width={screenWidth}
      height={220}
      chartConfig={chartConfig}
      renderDotContent={({ x, y, indexData }) => {
        if (indexData !== 0) {
          return (
            <View
              style={{
                position: "absolute",
                top: y - 20,
                left: x - 4,
              }} ← #42-46 style=
              key={indexData + x + y}
            >
              <Text style={{ fontSize: 14 }}>{indexData}</Text>
            </View>
          ); ← #40-51 return
        } ← #39-52 if (indexData !== 0)
      }} ← #38-53 renderDotContent=
    />
  ); ← #32-55 return
}; ← #7-56 const DisplayLineChart = ({ data }) =>

export default DisplayLineChart;
```

Рис. 3.10. Лінійна діаграма

Інтерактивність також реалізована у вигляді можливості фільтрування даних. Користувач може налаштувати графіки так, щоб вони показували лише певний період або певні категорії. Наприклад, якщо користувач хоче переглянути лише витрати на транспорт і харчування за останній квартал, він може швидко застосувати відповідний фільтр і отримати оновлений графік. Це дозволяє легко і швидко аналізувати фінансові дані без необхідності вручну обчислювати показники чи створювати окремі звіти.

Завдяки використанню бібліотеки react-native-chart-kit графіки виглядають привабливо і чітко навіть на невеликих екранах мобільних пристроїв. Висока роздільна здатність і плавна анімація роблять процес

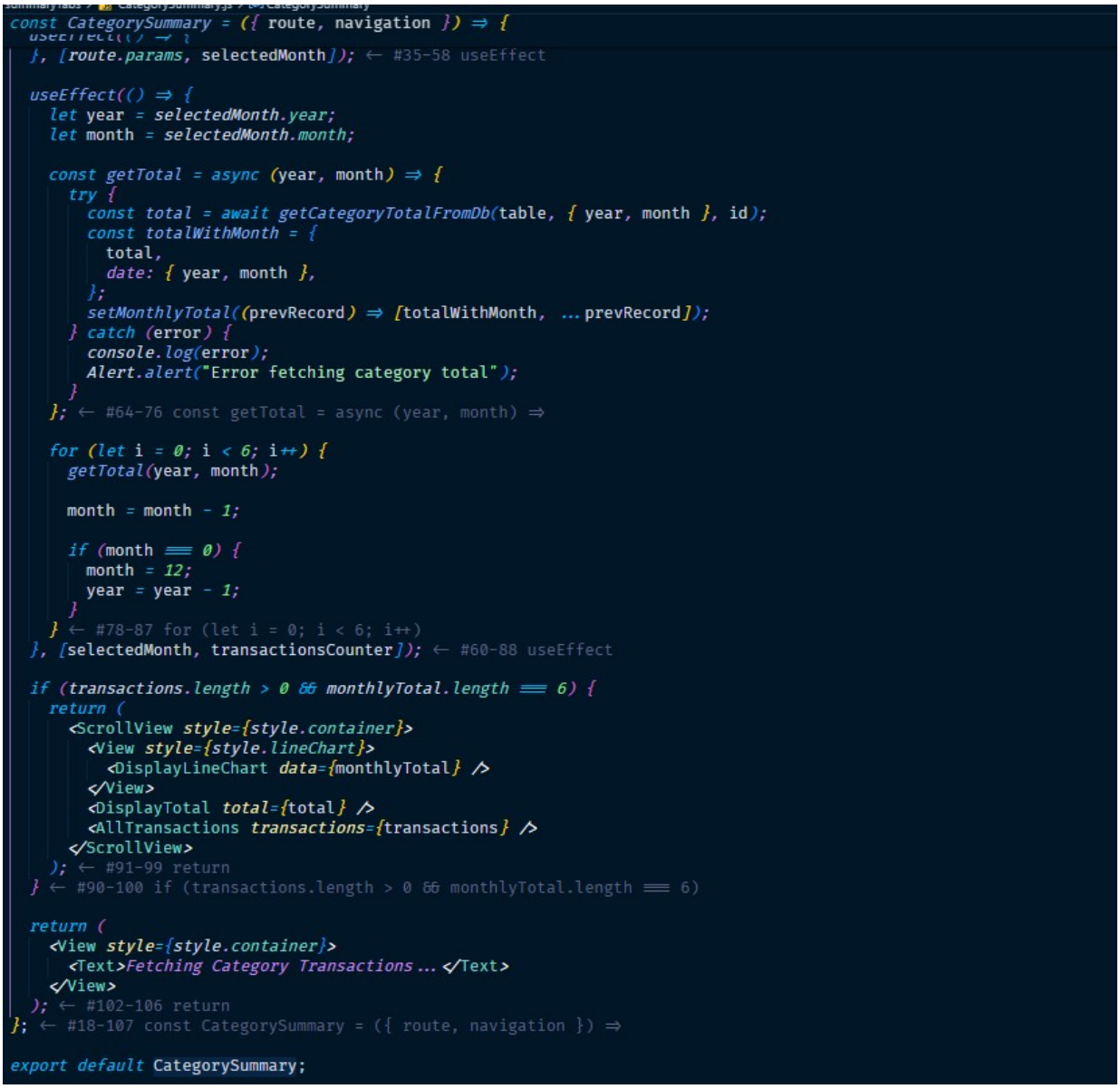
взаємодії з графіками зручним і приємним для користувача. Бібліотека дозволяє налаштовувати кольори, шрифти, розміри і стиль графіків, що дає змогу адаптувати їх під загальний дизайн застосунку. Крім того, користувач може перемикатися між світлою і темною темою інтерфейсу, і графіки автоматично змінюватимуть стиль відповідно до вибраної теми.

Додатковою перевагою візуалізації є можливість збереження графіків і статистики у вигляді зображень або звітів. Наприклад, користувач може створити кругову діаграму витрат за місяць, зберегти її у форматі PNG і поділитися з членами сім'ї або фінансовим консультантом. Це дозволяє не лише контролювати свої витрати, але й спільно планувати бюджет та ухвалювати більш обґрунтовані рішення.

Користувач може переглядати підсумкові графіки за будь-який період – день, тиждень, місяць або рік. Якщо користувач хоче побачити динаміку змін витрат на їжу протягом останнього місяця, застосунок створить лінійний графік із відповідною кривою, яка покаже дні зі зростанням або зниженням витрат. У разі різких змін користувач зможе проаналізувати причини такого зростання та внести корективи до свого бюджету.

Для полегшення аналізу даних застосунок автоматично підсумовує доходи та витрати у кожній категорії та виводить ці значення у вигляді числових підписів на графіку (рис. 3.11). Якщо на круговій діаграмі сегмент «Їжа» займає 30%, користувач також побачить підпис із точною сумою витрат на їжу за вибраний період. Це дозволяє поєднувати візуальне сприйняття з точними цифрами, що значно полегшує аналіз фінансів.

Якщо користувач хоче порівняти кілька періодів або категорій між собою, застосунок дозволяє відобразити декілька графіків одночасно. Наприклад, можна вивести стовпчастий графік із витратами на їжу за три останні місяці та одночасно відобразити лінійний графік із загальним балансом за цей же період. Це допомагає побачити взаємозв'язок між доходами та витратами та приймати більш зважені фінансові рішення.



```
const CategorySummary = ({ route, navigation }) => {
  }, [route.params, selectedMonth]); ← #35-58 useEffect

  useEffect(() => {
    let year = selectedMonth.year;
    let month = selectedMonth.month;

    const getTotal = async (year, month) => {
      try {
        const total = await getCategoryTotalFromDb(table, { year, month }, id);
        const totalWithMonth = {
          total,
          date: { year, month },
        };
        setMonthlyTotal((prevRecord) => [totalWithMonth, ...prevRecord]);
      } catch (error) {
        console.log(error);
        Alert.alert("Error fetching category total");
      }
    }; ← #64-76 const getTotal = async (year, month) =>

    for (let i = 0; i < 6; i++) {
      getTotal(year, month);

      month = month - 1;

      if (month === 0) {
        month = 12;
        year = year - 1;
      }
    } ← #78-87 for (let i = 0; i < 6; i++)
  }, [selectedMonth, transactionsCounter]); ← #60-88 useEffect

  if (transactions.length > 0 && monthlyTotal.length === 6) {
    return (
      <ScrollView style={style.container}>
        <View style={style.lineChart}>
          <DisplayLineChart data={monthlyTotal} />
        </View>
        <DisplayTotal total={total} />
        <AllTransactions transactions={transactions} />
      </ScrollView>
    ); ← #91-99 return
  } ← #90-100 if (transactions.length > 0 && monthlyTotal.length === 6)

  return (
    <View style={style.container}>
      <Text>Fetching Category Transactions...</Text>
    </View>
  ); ← #102-106 return
}; ← #18-107 const CategorySummary = ({ route, navigation }) =>

export default CategorySummary;
```

Рис. 3.11. Підбиття підсумків

Таким чином, використання інтерактивної візуалізації у вигляді кругових, лінійних графіків дозволяє користувачу краще розуміти свою фінансову ситуацію, аналізувати витрати та доходи, виявляти закономірності та ухвалювати більш ефективні рішення. Інтуїтивний інтерфейс, інтерактивність та можливість налаштування під потреби користувача роблять аналіз фінансових даних простим, наочним і ефективним.

3.5. Тестування застосунку «Фінансовий менеджер»

На початковій заставці додатку зображено стартову заставку

мобільного застосунку «Фінансовий менеджер». У центрі екрана розміщено ілюстрацію з банкнотами, монетами та графіком на тлі таблиці, що символізує фінансову звітність. Фон виконано в градієнтних зеленуватих тонах, що асоціюються з грошима та стабільністю. Нижче розташовано назву застосунку, написану декоративним шрифтом українською мовою — «Фінансовий менеджер» (рис. 3.12).

Рис. 3.12. Початкова заставка

На наступному рисунку продемонстровано два режими відображення фінансових даних у застосунку «Фінансовий менеджер» за березень 2025 року. Ліворуч представлено режим «Month» (місячний огляд), де показано підсумкові значення доходів, витрат і загального залишку по кожному місяцю в табличному форматі. Праворуч зображено режим «Calendar» (календар), у якому щоденні транзакції відображаються у вигляді значень

безпосередньо на відповідних датах календаря (рис. 3.13.). Зеленим кольором позначено доходи, червоним – витрати. Обидва режими мають зручну навігацію та кнопки додавання нової транзакції у вигляді зеленої кнопки з іконкою плюса внизу екрана.

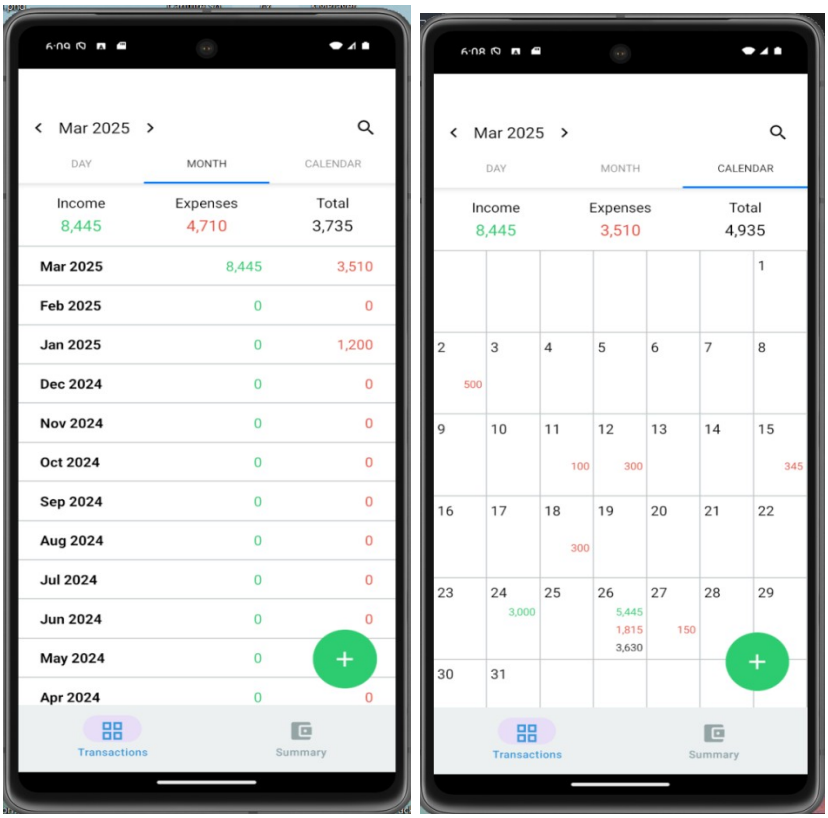


Рис. 3.13. Відображення даних в календарі та помісячно

На рисунку 3.14 зображено функціонал відображення щоденних транзакцій і пошуку по записах у застосунку «Фінансовий менеджер». Ліворуч представлено режим «Day» (щоденний огляд) для березня 2025 року, де користувач може переглядати доходи та витрати за кожен день. Транзакції згруповані за категоріями (наприклад, «Eat», «Clothes», «Salary») і супроводжуються сумами, іменами та примітками. Праворуч демонструється функція пошуку: за запитом «clot» знайдено 8 записів, які містять відповідне слово. У результатах пошуку наведено дату, категорію, опис і суму витрат, що дає змогу швидко знаходити необхідні дані.

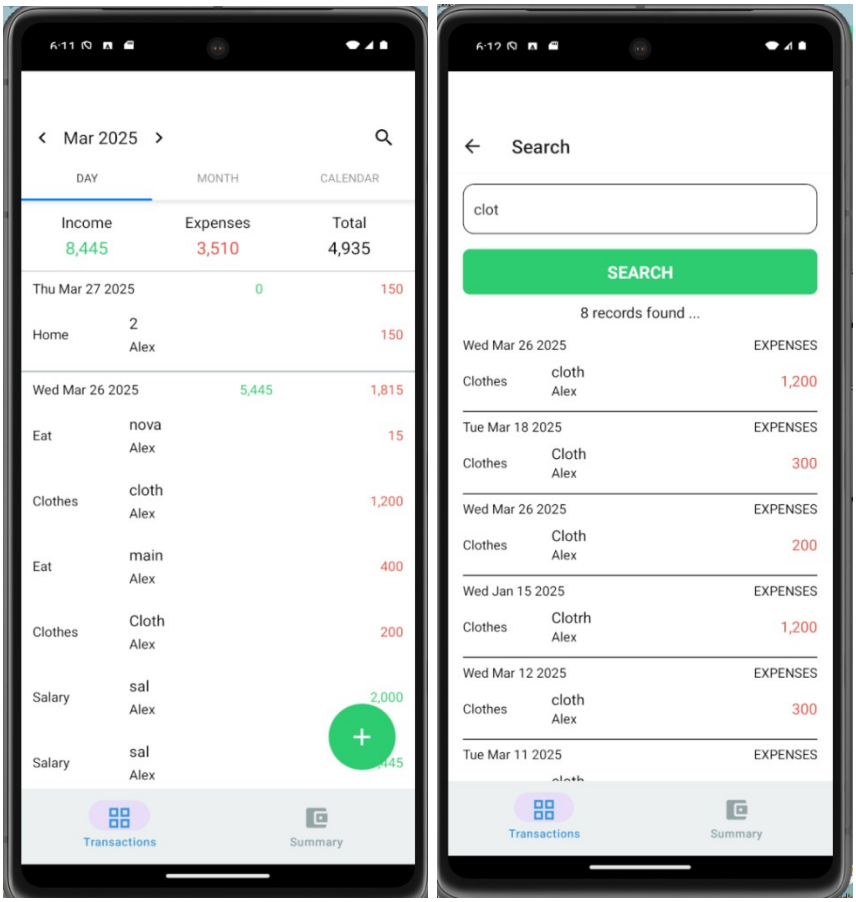


Рис. 3.14 Відображення транзакцій та пошук по ним.

Розглянемо процес створення нової транзакції у мобільному застосунку «Фінансовий менеджер». Ліворуч зверху представлено інтерфейс введення даних для додавання витрати: користувач обирає тип операції (дохід або витрата), дату, акаунт, категорію, суму та за потреби залишає примітку. Також можна швидко вибрати одну з наявних категорій. Праворуч зверху демонструється календар, який відкривається для вибору дати транзакції. Продемонструємо заповнення форми для витрати (внизу ліворуч): користувач вводить дані про акаунт, категорію (Clothes), суму (5000) та примітку (Cloth). Внизу праворуч зображено аналогічну форму для введення доходу з категорією Salary (рис. 3.15.). В обох випадках присутні кнопки перемикання між типами транзакцій (дохід або витрата), а також можливість додавати нові категорії. Це забезпечує користувачеві зручний спосіб фіксації та швидкість внесення фінансових операцій.

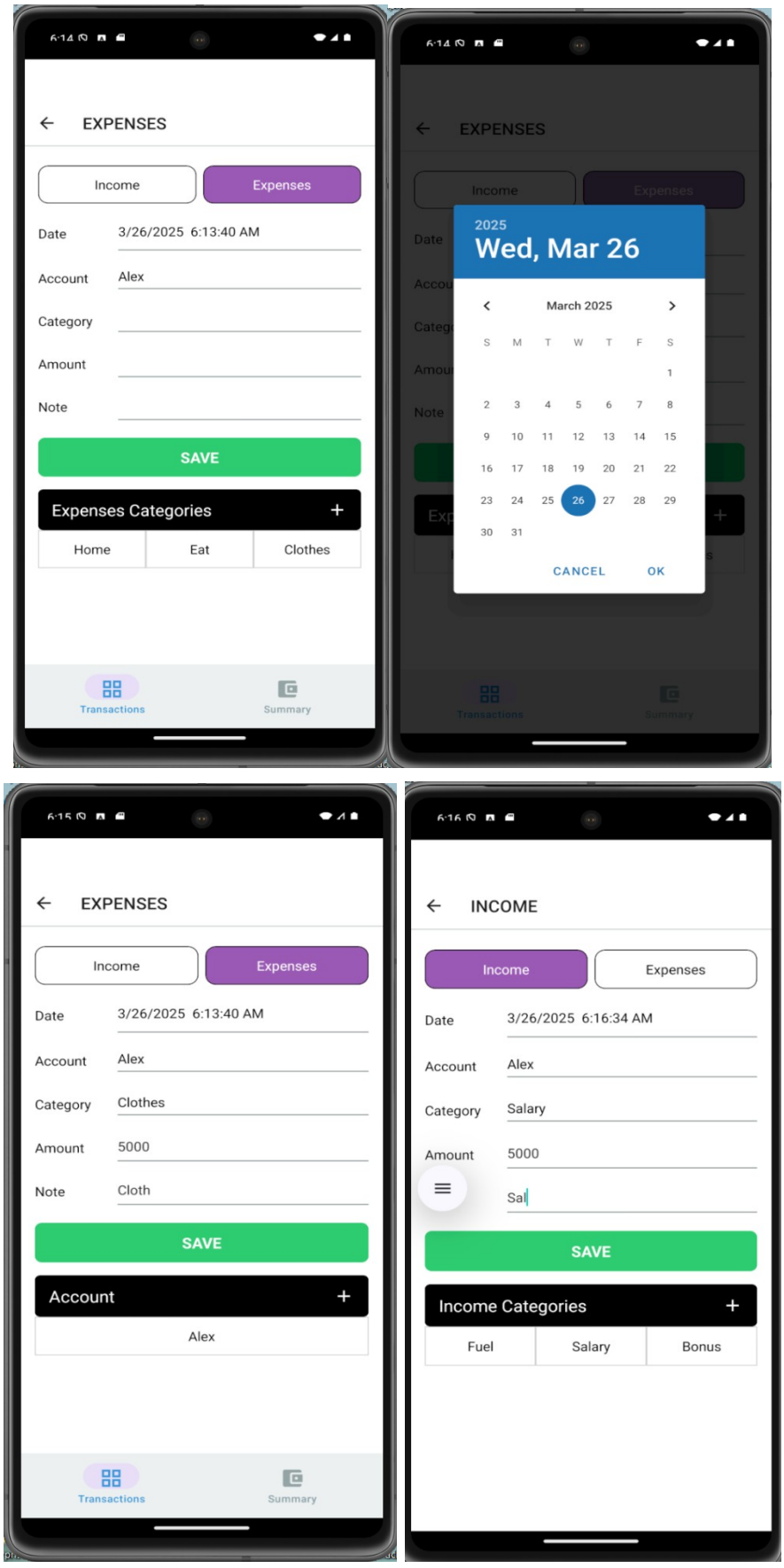


Рис. 3.15. Створення транзакцій

Далі розглянемо інтерфейс керування категоріями доходів у мобільному застосунку (рис. 3.16.). Ліворуч показано список наявних категорій доходів (Fuel, Salary, Bonus) з можливістю редагування кожної з них. У нижній частині екрана розташована кнопка додавання нової категорії. Праворуч зображено екран створення нової категорії доходу, де користувач вводить назву нової категорії в текстове поле та натискає кнопку «SAVE» для збереження. Це дозволяє користувачам адаптувати категорії під свої потреби.

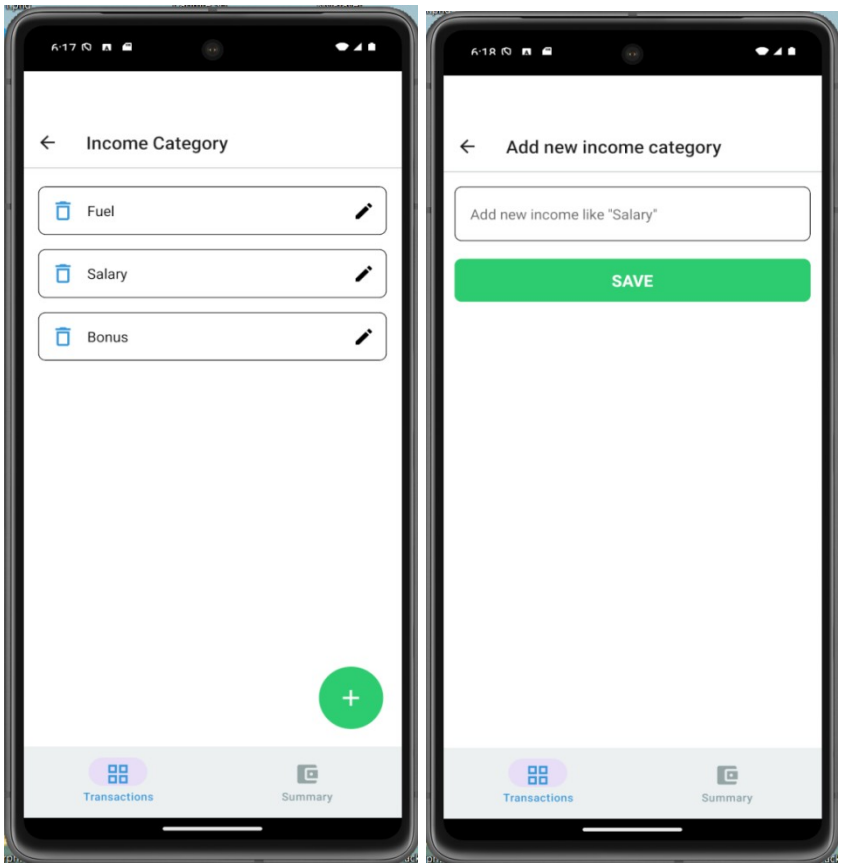


Рис. 3.16. Створення та вибір категорій

На зображенні 3.17 представлено два екрани мобільного застосунку для управління фінансами, які ілюструють процес оновлення раніше створеної транзакції. Ліва частина рисунка демонструє інтерфейс редагування існуючої транзакції типу «Expenses» (витрати). У верхній частині екрану розміщені перемикачі між типами транзакцій: «Income» (дохід) та «Expenses» (витрати), з активним положенням на «Expenses». Нижче відображається інформація про транзакцію:

– Date – дата і час транзакції: 26 березня 2025 року о 4:43:08 ранку;

– Account – акаунт, до якого належить транзакція: «Alex»;
– Category – категорія витрат: «Clothes»;
– Amount – сума витрат: 200;
– Note – примітка до транзакції: «Cloth».
– Під формою введення даних розташовані дві кнопки:
– UPDATE (зелена) – зберігає внесені зміни до транзакції;
– DELETE (червона) – дозволяє видалити транзакцію.

Також у нижній частині екрана відображено список акаунтів з активним акаунтом «Alex». Права частина рисунка показує екран вибору місяця та року, що використовується для фільтрації або навігації по періодах у додатку. Користувач може обрати конкретний місяць і рік, після чого натиснути кнопку CONFIRM для підтвердження вибору. У прикладі вибрано березень (Mar) 2025 року. Цей інтерфейс дозволяє зручно оновлювати фінансові записи, видаляти непотрібні транзакції, а також швидко перемикатися між місяцями для перегляду звітів або транзакцій за обраний період.

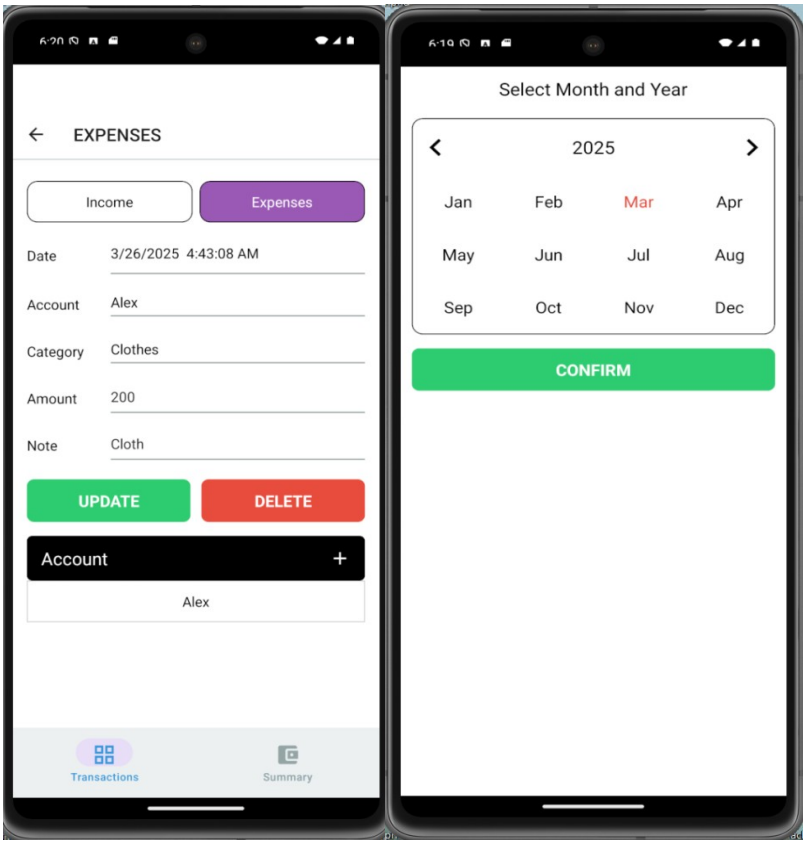


Рис. 3.17. Оновлення транзакцій

Рисунок 3.18 демонструє візуалізацію фінансових даних у мобільному додатку «Фінансовий менеджер». На першому знімку екрана, що складається з двох частин, представлено підсумкову інформацію про доходи та витрати за березень 2025 року. У лівій частині екрана відображено вкладку «ДОХОДИ», де у вигляді кругової діаграми візуалізовано розподіл джерел доходів: 64,48% припадає на зарплату (5 445 одиниць валюти), а 35,52% – на бонуси (3 000 одиниць). У правій частині наведено вкладку «ВИТРАТИ», де також за допомогою кругової діаграми відображено структуру витрат: основну частину (83,90%) займають витрати на одяг (2 945 одиниць), 11,82% – на харчування (415 одиниць), і 4,27% – на житло (150 одиниць). Нижня панель додатку містить дві іконки для перемикання між вкладками «Transactions» (транзакції) та «Summary» (підсумки), з активною останньою.

Другий знімок екрана відображає деталізовану інформацію про витрати на одяг у листопаді 2023 року. У верхній частині екрана представлено графік зміни залишку коштів за місяцями, з позначеннями числових значень на кожен місяць – з червня по листопад, де видно коливання залишку: пік у серпні (7 500), спад у жовтні (5 500) та зростання до 7 300 в листопаді. Під графіком зазначено загальний залишок – 7 300. Далі наведено список витрат з точними датами, категорією, способом оплати та сумою. Наприклад, 25 листопада була придбана куртка за 2 800 одиниць за допомогою картки, 18 листопада – взуття через мобільний банкінг за 1 700, 16 листопада – футболка за готівку (1 200), а 6 листопада – джинси за 1 600. Всі витрати чітко відображені червоним кольором праворуч, що підкреслює списання коштів.

Ця візуалізація дозволяє користувачеві швидко оцінити структуру доходів і витрат, а також проаналізувати деталі фінансової активності в певній категорії за обраний період.

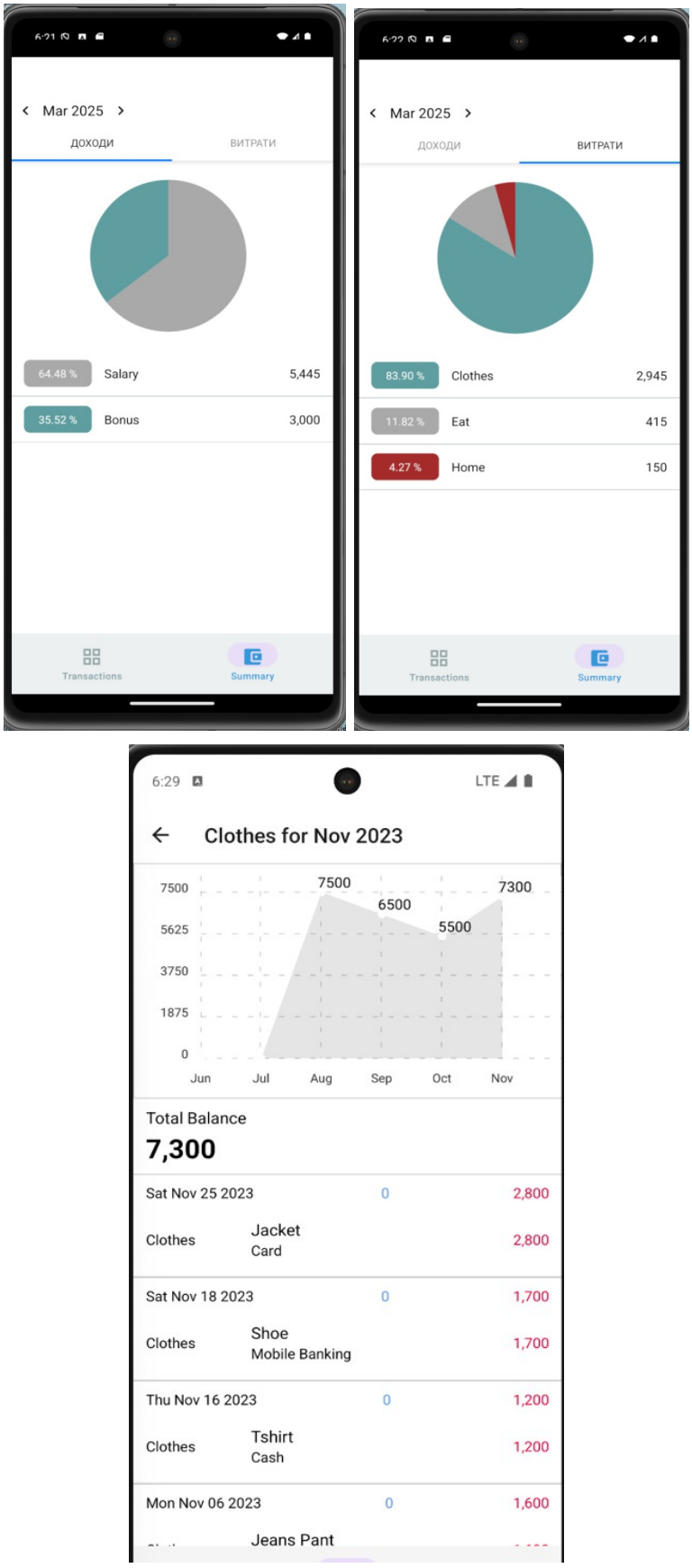


Рис. 3.18 Візуалізація даних у графіках

Висновки до третього розділу

У третьому розділі було розглянуто повний цикл розроблення мобільного застосунку «Фінансовий менеджер» для платформи Android, що призначений для обліку, аналізу та візуалізації особистих фінансів користувача.

На першому етапі було спроєктовано функціональність та інтерфейс додатку відповідно до потреб кінцевого користувача. Для реалізації навігації між екранами використано бібліотеку React Navigation, яка забезпечує зручну архітектуру переходів і логіку маршрутизації. Управління станом застосунку реалізовано з використанням Redux, що дозволило централізовано обробляти дані, зменшити кількість дублювання коду та забезпечити масштабованість рішення. Для локального збереження фінансових даних обрано SQLite, що дало змогу створити незалежну та надійну систему зберігання без потреби у зовнішньому сервері.

Особливу увагу було приділено візуалізації даних, що підвищує зручність користування застосунком: фінансова аналітика подається у вигляді графіків та діаграм, що дозволяє швидко оцінити витрати, доходи та загальну фінансову динаміку. На завершальному етапі було проведено перевірку працездатності застосунку, в результаті чого встановлено його стабільну роботу, коректне збереження та обробку даних, а також зручність взаємодії з користувачем. Загалом, розроблений застосунок відповідає поставленим вимогам та може бути використаний як ефективний інструмент для управління особистими фінансами

ВИСНОВКИ

У процесі дослідження на тему «Розробка персонального фінансового асистента» було досягнуто поставленої мети – створення програмного застосунку для ефективного управління персональними фінансами. Успішно реалізовано всі поставлені завдання. Проведено глибокий аналіз сучасних персональних фінансових асистентів, вивчено їхні функціональні можливості, виявлено ключові переваги й недоліки наявних рішень. На основі отриманих результатів сформульовано вимоги до нового фінансового асистента, які включають зручний облік доходів і витрат, аналітику у вигляді графіків, локальне зберігання даних та можливість подальшого розширення функціоналу.

Складено технічне завдання, яке визначає основні функціональні компоненти, вимоги до інтерфейсу, структуру даних і технічні характеристики. На основі порівняльного аналізу інструментів розробки обґрунтовано вибір: платформи Android як найбільш поширеної мобільної екосистеми, React Navigation – для реалізації навігації між екранами, Redux – для централізованого управління станом, SQLite – як ефективного рішення для локального зберігання даних.

Розроблено архітектуру застосунку з чітким поділом функціональних модулів, що забезпечує масштабованість, простоту підтримки та можливість подальшого розвитку. Спроєктовано структуру бази даних для ефективного зберігання інформації про доходи, витрати та категорії операцій.

Створено повнофункціональний мобільний застосунок «Фінансовий менеджер» для Android, що забезпечує фіксацію фінансових операцій, їх категоризацію, перегляд статистики, та має зручну багатосторінкову навігацію завдяки React Navigation. Redux забезпечує стабільне управління станом, а локальна база даних SQLite – автономність роботи та безпеку персональних фінансових даних.

Додатково реалізовано графічну візуалізацію даних, що значно

покращує сприйняття інформації користувачем і сприяє прийняттю обґрунтованих фінансових рішень. Комплексне тестування засвідчило коректність роботи всіх ключових функцій, стабільність на різних пристроях та відповідність визначеним вимогам.

Оцінка ефективності показала, що застосунок відповідає очікуванням користувачів щодо зручного інструменту для обліку фінансів. Застосунок має потенціал для комерційного впровадження, а очікуваний економічний ефект полягає у підвищенні фінансової обізнаності користувачів, оптимізації витрат та покращенні контролю за особистими фінансами. Подальший розвиток проекту може включати інтеграцію з банківськими API для автоматичного відстеження транзакцій та розширення функціоналу за рахунок використання технологій машинного навчання для прогнозування фінансових тенденцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mint. URL: https://mint.intuit.com (дата звернення: 15.02.2025).
2. PocketGuard. URL: https://pocketguard.com
3. Rayhan R. Social Media Web App REST APIs Implementation: Bachelor's Thesis, 2024. URL: https://www.theseus.fi/bitstream/handle/10024/818944/Rayhan_Rana.pdf?sequence=2&isAllowed=y (дата звернення: 20.02.2025).
4. YNAB. URL: https://www.ynab.com (дата звернення: 17.02.2025).
5. Бармутов Р., Онищенко Б. Засоби генерації програмного коду для роботи з базами даних у мобільних андроїд-застосунках. *Ricerche scientifiche e metodi della loro realizzazione: esperienza mondiale e realtà domestiche*: VI International Scientific and Practical Conference. *Collection of scientific papers «ΛΟΓΟΣ»*. 2024. Т. 15 (15 листопада 2024 р., Болонья, Італія). С. 139–141.
6. Бойчук О. А. Застосунок для фінансового планування, аналізу та оптимізації витрат: кваліфікаційна робота. 2023.
7. Гнатюк-Шаповал Д. Створення застосунку для відображення вмісту бази даних на другому екрані // *Інформаційно-комунікаційні технології в освіті* (26 грудня 2024р.). 2024. №13. URL: https://e-journals.udu.edu.ua/index.php/ikt/article/view/1465 (дата звернення: 21.03.2025).
8. Дакаленко Д. О. Розробка Android-додатку моніторингу витрат користувача: кваліфікаційна робота. 2024.
9. Зачек О. І. Методичні вказівки до виконання бакалаврської кваліфікаційної роботи для здобувачів освітнього ступеня «бакалавр» галузі знань 12 «Інформаційні технології» спеціальність 126 «Інформаційні системи і технології» факультету № 2 Інституту з підготовки фахівців для підрозділів Національної поліції, Центру післядипломної освіти, дистанційного та заочного навчання. Львів: Львівський державний університет внутрішніх справ, 2024. 52 с.
10. Калабухін Ю. Є., Губар С. О. Обгрунтування необхідності розробки ефективного мобільного додатку з персонального фінансового обліку. *Вісник*

економіки транспорту і промисловості. 2023. №81-82. С.345-350. URL: http://lib.kart.edu.ua/handle/123456789/21011 (дата звернення: 27.02.2025).

11. Кириленко В. С. Система фінансового контролю власних коштів шляхом зберігання чеків в електронному вигляді: кваліфікаційна робота. 2023. URL: https://krs.chmnu.edu.ua/jspui/bitstream/123456789/2965/1/%d0%9a%d0%b8%d1%80%d0%b8%d0%bb%d0%b5%d0%bd%d0%ba%d0%be%20%d0%92%d0%bb%d0%b0%d0%b4%d0%b8%d1%81%d0%bb%d0%b0%d0%b2%20%d0%91%d0%9a%d0%a0.pdf (дата звернення: 10.03.2025).

12. Козолуп П., Любчак В. Огляд методів та інструментів для розробки інформаційного сервісу обліку особистих активів. *Інформаційні технології та суспільство*. 2023. № 3 (9), С. 47-53. URL: https://journals.maup.com.ua/index.php/it/article/view/2794/3253 (дата звернення: 16.03.2025).

13. Костенко Ю. О., Іваненко В. О., Козаченко А. Ю. Інтеграція штучного інтелекту до бухгалтерських систем для оптимізації фінансового аналізу. *Ефективна економіка.* 2024. № 11. URL: https://nayka.com.ua/index.php/ee/article/view/5093/5137 (дата звернення: 17.03.2025).

14. Мельник Степан, Шевченко Наталія, Висоцька Інна. Банківська система: навчальний посібник у схемах і таблицях. Львів: Львівський державний університет внутрішніх справ, 2023. 184 с.

15. Прокопенко О. Є. Розробка навчальної системи для збору персональних даних з використанням штучного інтелекту та соціальної інженерії: Manuscript. Master's thesis. 2025.

16. Розробка веб-додатків, мобільних додатків та порталів. URL: http://ittel.com.ua/informacijnitexnologiyi/rozrobka-mobilnih-dodatkiv/ (дата звернення: 16.03.2025).

17. Рудовіл, Є. А. Розробка веб-додатку для управління фінансами з використанням Python і Flask: кваліфікаційна робота. 2024.

18. Савченко, В., Кононенко, Л., Гай, О. Використання статистичних

методів та фінансового аналізу для інформаційного забезпечення суб'єктів фінансового ринку. *Наука і техніка*. 2024. №2(30). URL: http://perspectives.pp.ua/index.php/nts/article/view/9393/9446 (дата звернення: 17.02.2025).

19. Смирнов, Є. Т. Дослідження технологій персистентності даних Android-застосунків: кваліфікаційна робота. 2024.

20. Технології створення мобільних додатків. URL: http://group-global.org/publication/63343-tehnologiisozdaniyamobilnyh-prilozheniy (дата звернення: 18.03.2025).

21. Чорна Т., Богданова Є., Малахов С. Огляд технік перехоплення сеансів веб-користувачів для компрометації їх особистих даних. *Глобалізація наукових знань: міжнародна співпраця та інтеграція галузей наук*: матеріали IV Міжнародної студентська наукова конференція конференцій (17 лютого 2023 р., м. Тернопіль). С. 205-208. URL: https://archive.liga.science/index.php/conference-proceedings/article/view/232 (дата звернення: 18.03.2025).

22. Шаповалова, А. В. Розробка автоматизованої інформаційної системи ведення книги доходів та витрат: магістерська робота. Сумський державний університет, 2024.

23. Шиш, А. М. Хмарні технології у бухгалтерському обліку та фінансовому аналізі в Україні: аналіз відмінностей та стратегії адаптації до місцевого контексту. *Здобутки економіки: перспективи та інновації*. 2024. № 2. С. 45-51. DOI: https://doi.org/10.57125/econp.2024.01.29.02 (дата звернення: 15.02.2025).

ДОДАТКИ

Додаток А

Код застосунку у середовищі React-native

```
import { View, StyleSheet } from "react-native";
import { createMaterialTopTabNavigator } from "@react-navigation/material-top-tabs";
import IncomeTab from "./summaryTabs/IncomeTab";
import ExpensesTab from "./summaryTabs/ExpensesTab";
import SelectedMonthAndYear from "./components/SelectedMonthAndYear";
const TopTab = createMaterialTopTabNavigator();
const SummaryScreen = () => {
 return (
  <View style={style.container}>
   <View style={style.monthAndYearContainer}>
    <SelectedMonthAndYear />
   </View>
   <View style={style.navigationContainer}>
    <TopTab.Navigator>
     <TopTab.Screen
      name="IncomeTab"
      component={IncomeTab}
      options={{
       title: "Р”РѕС…РѕРґРё",
      }}
     />
     <TopTab.Screen
      name="ExpensesTab"
      component={ExpensesTab}
      options={{
       title: "Р’РёС‚СЂР°С‚Рё",
```

```
          }}
         />
       </TopTab.Navigator>
     </View>
   </View>
  );
};
export default SummaryScreen;
const style = StyleSheet.create({
  container: {
    flex: 1,
  },
  navigationContainer: {
    flex: 1,
  },
  monthAndYearContainer: {
    backgroundColor: "white",
  },
});
import {
  View,
  Text,
  StyleSheet,
  ScrollView,
  Linking,
  Pressable,
} from "react-native";
import IconText from "./components/IconText";
import { allColors } from "../Colors";
const MoreScreen = () => {
```

```
  return (
    <ScrollView style={style.mainContainer}>
      <Text style={style.title}>Settings</Text>
      <ScrollView style={style.container}>
        <View>
          <View style={style.flexRow}>
            <IconText icon="settings" text="Configuration" />
            <IconText icon="vpn-key" text="Passcode" />
          </View>
          <View style={style.flexRow}>
            <IconText icon="style" text="Theme" />
            <IconText icon="help-outline" text="Help" />
          </View>
          <View style={style.flexRow}>
            <IconText icon="backup" text="Backup" />
            <IconText icon="thumb-up-off-alt" text="Recommend" />
          </View>
        </View>
      </ScrollView>

    </ScrollView>
  );
};
export default MoreScreen;
const style = StyleSheet.create({
  mainContainer: {
    flex: 1,
    backgroundColor: "white",
  },
```

```
container: {
  flex: 1,
  backgroundColor: "white",
  marginBottom: 12,
},
flexRow: {
  flexDirection: "row",
},
title: {
  fontSize: 28,
  fontWeight: "bold",
  paddingTop: 16,
  paddingBottom: 30,
  textAlign: "center",
},
creditContainer: {
  flexDirection: "row",
  justifyContent: "center",
  alignItems: "center",
  marginBottom: 8,
},
creditText: {
  fontSize: 16,
  textAlign: "center",
  fontWeight: "300",
},
developer: {
  fontSize: 18,
  fontWeight: "bold",
  color: allColors.developer,
```

```
    textDecorationLine: "underline",
  },
});

import { createStackNavigator } from "@react-navigation/stack";
import SummaryScreen from "./SummaryScreen";
import CategorySummary from "./summaryTabs/CategorySummary";
const Stack = createStackNavigator();
const MainSummaryScreen = () => {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name="IncomeExpensesSummary"
        component={SummaryScreen}
        options={{ headerShown: false }}
      />
                                    <Stack.Screen      name="CategorySummary"
component={CategorySummary} />
    </Stack.Navigator>
  );
};
export default MainSummaryScreen;

import { createStackNavigator } from "@react-navigation/stack";
import                    TransactionScreen                    from
"./mainTransactionScreen/TransactionScreen";
import                AddIncomeExpensesScreen                  from
"./mainTransactionScreen/AddIncomeExpensesScreen";
import                    EditOptionsScreen                    from
"./mainTransactionScreen/EditOptionsScreen";
```

```
import EditOptionsFormScreen from "./mainTransactionScreen/editOptionsScreen/EditOptionsFormScreen";
import SearchScreen from "./mainTransactionScreen/SearchScreen";
const Stack = createStackNavigator();
const MainTransactionScreen = () => {
 return (
  <Stack.Navigator
   screenOptions={{
    headerBackTitleVisible: false,
   }}
  >
   <Stack.Screen
    name="AllTransactions"
    component={TransactionScreen}
    options={{
     headerShown: false,
    }}
   />
   <Stack.Screen
    name="AddIncomeExpenses"
    component={AddIncomeExpensesScreen}
   />
   <Stack.Screen
    name="EditOptions"
    component={EditOptionsScreen}
    options={{
     headerTitle: "Setting",
    }}
   />
   <Stack.Screen
```

```
        name="EditOptionsForm"
        component={EditOptionsFormScreen}
        options={{
          headerTitle: "Update Category",
        }}
      />
      <Stack.Screen name="Search" component={SearchScreen} />
    </Stack.Navigator>
  );
};
export default MainTransactionScreen;

import { Text, StyleSheet, Pressable, View } from "react-native"
import { useSelector } from "react-redux";
import { useNavigation } from "@react-navigation/native";

import { allColors } from "../../../Colors";

import { mainStyle } from "../../../mainStyle";

const EachSearchResult = ({ search }) => {
  const incomeCategory = useSelector((state) => state.incomeCategory);
  const expensesCategory = useSelector((state) => state.expensesCategory);

  const transactionCategory =
    incomeCategory.find((category) => category.id === search.category) ||
    expensesCategory.find((category) => category.id === search.category);

  const transactionAccount = useSelector((state) =>
    state.account.find((eachAccount) => eachAccount.id === search.account)
```

```
    );

    const navigation = useNavigation();

    return (
        <Pressable style={style.container} android_ripple={{ color: allColors.lightGray }}
          onPress={() => navigation.navigate("AddIncomeExpenses", { transaction: search })}>
      <View style={style.dateAndTypeContainer}>
       <Text>
         {(new Date(search.date)).toDateString()}
       </Text>

       <Text>
         {(search.type).toUpperCase()}
       </Text>
      </View>

      <View
       style={style.transaction}

      >
       <View style={style.category}>
         <Text>{transactionCategory?.value}</Text>
       </View>

       <View style={style.note}>
         <Text style={style.noteText}>{search?.note}</Text>
         <Text>{transactionAccount?.value}</Text>
```

```
        </View>

        <View style={style.amount}>
          <Text
            style={[
              style.amountText,
              search.type === "income"
                ? mainStyle.incomeColor
                : mainStyle.expensesColor,
            ]}
          >
            {Number(search?.amount).toLocaleString()}
          </Text>
        </View>
      </View>
    </Pressable>
  )
}

const style = StyleSheet.create({
  container: {
    paddingTop: 6
  },
  transaction: {
    paddingBottom: 12,
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
    borderBottomWidth: 1,
    borderBottomColor: 'black',
```

```
  },
  category: {
    flex: 1,
  },
  note: {
    flex: 2,
  },
  noteText: {
    fontSize: 16,
  },
  amount: {
    flex: 1,
  },
  amountText: {
    textAlign: "right",
    fontSize: 16
  },
  dateAndTypeContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    marginBottom: 8
  }

});

export default EachSearchResult

import { useEffect, useState } from "react";
import {
  View,
```

```
  Text,
  TextInput,
  StyleSheet,
  ScrollView,
  Alert,
} from "react-native";
import MyButton from "../../components/MyButton";
import { searchInDb } from "../../util/database";
import EachSearchResult from "./searchScreen/EachSearchResult";
const SearchScreen = () => {
  const [search, setSearch] = useState("");
  const [isSearching, setIsSearching] = useState(false);
  const [searchMessage, setSearchMessage] = useState("");
  const [searchResult, setSearchResult] = useState([]);
  const handleSearch = () => {
    setIsSearching(true);
  };
  useEffect(() => {
    const searchTransaction = async () => {
      try {
        const incomeSearch = await searchInDb(search, "income");
        const expensesSearch = await searchInDb(search, "expenses");
        const result = [...incomeSearch, ...expensesSearch];
        setSearchResult(result);
        setSearchMessage(
          result.length
            ? `${result.length} records found ...`
            : "No transaction found !"
        );
      } catch (error) {
```

```
      Alert.alert("Error searching...");
      console.log(error);
    } finally {
      setIsSearching(false);
    }
  };
  if (isSearching && search) {
    searchTransaction();
  }
}, [isSearching]);
return (
  <ScrollView style={style.container}>
    <View style={style.inputContainer}>
      <TextInput
        style={style.input}
        placeholder="Search for your transactions"
        onChangeText={(text) => setSearch(text)}
        inputMode="search"
      />
    </View>
    <MyButton
      title={isSearching ? "Searching..." : "Search"}
      variant="primary"
      onPress={handleSearch}
    />
    <Text style={style.searchMessage}>{searchMessage}</Text>
    <ScrollView>
      {searchResult.length > 0 &&
        searchResult.map((eachTx) => (
          <EachSearchResult search={eachTx} key={eachTx.id} />
```

```
          ))}
        </ScrollView>
      </ScrollView>
    );
};
export default SearchScreen;
const style = StyleSheet.create({
  container: {
    paddingVertical: 12,
    paddingHorizontal: 16,
    backgroundColor: "white",
  },
  inputContainer: {
    marginBottom: 16,
  },
  input: {
    borderWidth: 1,
    borderColor: "black",
    borderRadius: 12,
    paddingHorizontal: 12,
    paddingVertical: 12,
    fontSize: 16,
  },
  searchMessage: {
    fontSize: 16,
    textAlign: "center",
    marginVertical: 8,
  },
});
```

```
import { View, StyleSheet, Pressable } from "react-native";
import { createMaterialTopTabNavigator } from "@react-navigation/material-top-tabs";
import CalendarTab from "./transactionScreen/CalendarScreen";
import DayTab from "./transactionScreen/DayScreen";
import MonthTab from "./transactionScreen/MonthScreen";
import SelectedMonthAndYear from "../components/SelectedMonthAndYear";
import { mainStyle } from "../../mainStyle";
import { MaterialIcons } from "@expo/vector-icons";
import CircularAddIcon from "../../components/CircularAddIcon";
const TopTab = createMaterialTopTabNavigator();
const TransactionScreen = ({ navigation }) => {
 const handleNewTransaction = () => {
  navigation.navigate("AddIncomeExpenses");
 };
 return (
  <View style={mainStyle.fullArea}>
   <View style={style.headerContainer}>
    <SelectedMonthAndYear />
    <Pressable
     style={style.search}
     onPress={() => navigation.navigate("Search")}
    >
     <MaterialIcons name="search" size={24} color="black" />
    </Pressable>
   </View>
   <View style={[mainStyle.fullArea]}>
    <TopTab.Navigator
     screenOptions={{
```

```
            tabBarLabelStyle: { fontSize: 12 },
          }}
        >
          <TopTab.Screen
            name="DayTab"
            component={DayTab}
            options={{ title: "Day" }}
          />
          <TopTab.Screen
            name="MonthTab"
            component={MonthTab}
            options={{ title: "Month" }}
          />
          <TopTab.Screen
            name="CalenderTab"
            component={CalendarTab}
            options={{ title: "Calendar" }}
          />
        </TopTab.Navigator>
      </View>
                                    <Pressable      style={style.addIconContainer}
onPress={handleNewTransaction}>
          <CircularAddIcon />
        </Pressable>
      </View>
    );
  };
  export default TransactionScreen;
  const style = StyleSheet.create({
    headerContainer: {
```

```
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
    paddingHorizontal: 4,
    backgroundColor: "white",
  },
  search: {
    marginRight: 12,
    padding: 4,
  },
  addIconContainer: {
    position: "absolute",
    bottom: 20,
    right: 20,
  },
});
```