Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Фінансовий трекер для малого бізнесу: розробка Android-застосунку для обліку доходів і витрат

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-21-1

_____ /К. О. Сивожелєзов/

Керівник
кваліфікаційної роботи _____ / А. В. Козиков /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій
Кафедра: Моделювання та програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А. М. Стрюк
«___» _____ 20__ р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-21-1 Сивожелєзову Костянтину Олександровичу

1. Тема: Фінансовий трекер для малого бізнесу: розробка Android-застосунку для обліку доходів і витрат
   затверджено наказом по КНУ № __ від «__» лютого 2025 р.
2. Термін подання студентом закінченої роботи: «__» червня 2025р.
3. Вихідні дані по роботі: Система має забезпечувати облік доходів і витрат, підтримку кількох рахунків і категорій, а також роботу з хмарними даними через Firebase.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Провести аналіз задачі обліку фінансів у малому бізнесі та існуючих мобільних рішень. Визначити технології реалізації та спроєктувати архітектуру застосунку. Розробити базу даних у Firestore, інтерфейс користувача та реалізувати основний функціонал (операції, рахунки, категорії, аналітика). Провести тестування застосунку.
5. Перелік ілюстративного матеріалу: екран входу і реєстрації користувача, всі екрани застосунку.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Аналіз предметної області та існуючих аналогів	27.02.2025 – 29.02.2025
2	Формулювання вимог та постановка задачі	30.02.2025 – 09.03.2025
3	Вибір технологій, архітектури та інструментів	10.03.2025– 18.03.2025
4	Проєктування структури бази даних	19.03.2025 – 23.03.2025
5	Розробка інтерфейсу користувача	24.03.2025– 30.03.2025
6	Реалізація функціоналу транзакцій, рахунків, категорій	1.04.2025– 04.04.2025
7	Синхронізація з Firebase та побудова аналітики	05.04.2025– 13.05.2025
8	Тестування застосунку	14.05.2025 – 20.05.2025
9	Оцінка відповідності вимогам, усунення недоліків	21.05.2025 – 26.05.2025
10	Оформлення пояснювальної записки	27.05.2025– 05.06.2025

Дата видачі завдання: «15» лютого 2025 р.

Студент _____ / К. О. Сивожелєзов /

Керівник роботи _____ / А. В. Козиков /

РЕФЕРАТ

ФІНАНСИ, АНДРОЇД-ЗАСТОСУНОК, МАЛИЙ БІЗНЕС, FIREBASE, JETPACK COMPOSE, MVVM, АНАЛІТИКА.

Пояснювальна записка: с.107, табл.0, рис.2, дод.2, джерел.9

Метою кваліфікаційної роботи є розробка Android-застосунку для обліку фінансів у малому бізнесі з використанням сучасних технологій, таких як Kotlin, Jetpack Compose, Firebase та архітектура MVVM.

У роботі виконано аналіз існуючих мобільних рішень для фінансового обліку, порівняно їх функціональні можливості, зручність використання та технічні обмеження. На основі цього аналізу обґрунтовано доцільність створення власного застосунку з урахуванням потреб малого бізнесу, таких як простота, багатомовність та інтеграція з хмарними сервісами.

Розроблено архітектуру застосунку, спроєктовано базу даних у Firebase Firestore, реалізовано облік доходів і витрат, підтримку рахунків і категорій, а також візуалізацію фінансових показників. Впроваджено підтримку трьох мов інтерфейсу (українська, російська, англійська) та адаптацію до темної/світлої теми.

Здійснено програмну реалізацію системи та її тестування на функціональність, стабільність і зручність використання.

Основні результати кваліфікаційної роботи можуть бути використані для подальшого розвитку застосунків фінансового обліку на мобільних платформах.

ABSTRACT

FINANCE, ANDROID APPLICATION, SMALL BUSINESS, FIREBASE, JETPACK COMPOSE, MVVM, ANALYTICS.

Thesis in p.107, tab.0, fig.2, app.2, references.9

The aim of this qualification work is to develop an Android application for financial tracking tailored to the needs of small businesses, using modern technologies such as Kotlin, Jetpack Compose, Firebase, and MVVM architecture.

The work includes an analysis of existing mobile financial tracking applications, comparing their functionality, usability, and limitations. Based on this analysis, the necessity of developing a custom application has been justified — one that emphasizes simplicity, multilingual support, and cloud integration.

The system architecture was designed, a cloud database in Firebase Firestore was structured, and functionality for income and expense tracking, account and category management, and visualization of financial statistics was implemented. The application supports three languages (Ukrainian, Russian, English) and adapts to both dark and light themes.

The system was fully developed and tested for functionality, stability, and usability.

The main results of this qualification work can be used for further development of mobile financial tracking tools.

ЗМІСТ

Вступ

У сучасних умовах розвитку цифрових технологій, автоматизація процесів управління бізнесом стає важливою складовою ефективної підприємницької діяльності. Особливу роль у цьому процесі відіграє сфера фінансового обліку, що є основою для прийняття управлінських рішень, планування витрат, аналізу прибутковості та оцінки загального стану підприємства. Для малого бізнесу, де часто відсутні окремі спеціалісти з фінансів, надзвичайно важливо мати зручні та доступні інструменти, які дозволяють вести фінансовий облік без залучення складних облікових систем.

Мобільні застосунки стали невід'ємною частиною повсякденного життя користувачів, у тому числі й підприємців. Сучасні мобільні рішення дозволяють забезпечити облік фінансових операцій у режимі реального часу, незалежно від місцезнаходження користувача. Проте більшість наявних на ринку фінансових застосунків орієнтовані на особисте користування або не враховують специфіку потреб малого бізнесу — зокрема, ведення обліку за кількома рахунками, підтримку категорій витрат і доходів, зберігання даних у хмарі, інтеграцію з актуальними інструментами візуалізації тощо.

Метою цієї кваліфікаційної роботи є розробка Android-застосунку для малого бізнесу, який дозволяє зручно вести облік фінансових операцій, здійснювати категоризацію доходів і витрат, управляти рахунками та переглядати аналітичні звіти. У процесі реалізації буде використано сучасні технології мобільної розробки — мову Kotlin, фреймворк Jetpack Compose для створення інтерфейсу, архітектурну модель MVVM, а також хмарне сховище даних Firebase Firestore.

Об'єктом дослідження є процеси обліку фінансових операцій у середовищі малого бізнесу. Предметом дослідження — методи та інструменти реалізації мобільного Android-застосунку для обліку доходів і витрат.

Робота включає в себе аналіз предметної області, огляд існуючих рішень, формування технічних вимог, проєктування архітектури програмного забезпечення, реалізацію функціоналу, а також тестування та аналіз результатів.

1. Аналіз предметної області та постановка задачі

1.1.Опис предметної області

Фінансовий облік є однією з найважливіших функцій у діяльності будь-якого підприємства, зокрема й малого бізнесу. Ефективне управління фінансовими потоками дозволяє контролювати дохідну та витратну частини бюджету, своєчасно реагувати на зміни в ринковому середовищі, виявляти фінансові ризики та ухвалювати обґрунтовані управлінські рішення. На відміну від великих компаній, малі підприємства зазвичай не мають доступу до складних бухгалтерських систем або найнятих спеціалістів, тому потребують зручних, доступних і зрозумілих інструментів для ведення фінансового обліку.

У сучасних умовах цифровізації все більшої популярності набувають мобільні програмні засоби, які дозволяють користувачам взаємодіяти з системою без прив’язки до робочого місця. Рішення у вигляді Android-застосунків забезпечують постійний доступ до даних, гнучкість та зручність використання. Мобільні трекери фінансів здатні забезпечити оперативний облік витрат і доходів, ведення звітності, аналіз фінансової активності, що особливо корисно для підприємців, які працюють у динамічному середовищі.

У предметну область дослідження входить побудова програмного забезпечення, яке забезпечує функціональність обліку транзакцій, категоризацію фінансових операцій, створення та редагування рахунків, а також формування аналітичної інформації у вигляді графіків, таблиць і звітів. Програмний продукт повинен мати зручний графічний інтерфейс, бути адаптованим до різних типів пристроїв і мати можливість зберігати дані у хмарному середовищі з використанням Firebase Firestore. Для зручності роботи та гнучкості в реалізації логіки застосунку доцільно використовувати архітектурний підхід MVVM, який забезпечує чітке розділення між логікою бізнес-процесів, інтерфейсом користувача та обробкою станів.

Ключовими елементами системи є: транзакції (що містять дату, тип, суму, опис, категорію та рахунок), рахунки (із власною назвою та балансом), категорії (що дозволяють групувати операції за типами — наприклад, «харчування», «оренда», «дохід від продажу»), а також модулі для візуалізації та аналізу фінансових даних за обраний період.

Таким чином, предметна область розробки охоплює питання побудови мобільного інструменту, який дозволяє автоматизувати базові елементи фінансового обліку малого бізнесу, забезпечує мобільність, інтеграцію з

хмарними сервісами та використання сучасних підходів до реалізації інтерфейсів і логіки взаємодії з користувачем.

1.2.Аналіз існуючих мобільних застосунків для фінансового обліку

На сучасному ринку програмного забезпечення існує велика кількість мобільних застосунків, що надають користувачам можливість вести облік особистих фінансів. Серед найбільш відомих рішень можна виокремити такі застосунки, як Money Manager, Monefy, Spendee, Wallet, CoinKeeper та ZenMoney. Ці програми охоплюють широкий спектр функціональності — від простого додавання доходів і витрат до формування докладної фінансової аналітики, ведення бюджету та синхронізації з банківськими рахунками.

Більшість таких застосунків мають привабливий інтерфейс користувача, підтримку категорій, валют, облік за рахунками та побудову графіків. Водночас вони часто орієнтовані на особисте користування, а не на потреби малого бізнесу. Деякі функції, зокрема експорт звітів, мультирахунки або зберігання в хмарі, доступні лише у платних версіях. Це обмежує можливості використання застосунків у повсякденній роботі підприємців, які шукають безкоштовні або гнучкіші рішення.

Також варто зазначити, що деякі програми не дозволяють редагувати структуру категорій або рахунків відповідно до специфіки бізнесу, не підтримують кастомізацію інтерфейсу чи мають складну навігацію, що ускладнює роботу непідготовлених користувачів. Інша важлива проблема — недостатня інтеграція з хмарними сервісами без необхідності створення окремого облікового запису або підписки, що знижує зручність синхронізації даних між пристроями.

Ще однією характерною ознакою більшості наявних застосунків є їхній фокус на іноземного користувача. Часто відсутня підтримка локальних мов інтерфейсу, а також не враховуються потреби підприємств, які працюють із локальними податковими або обліковими особливостями.

На основі проведеного аналізу можна дійти висновку, що існує потреба у створенні простого, адаптивного й водночас потужного фінансового інструменту у вигляді мобільного Android-застосунку, який би враховував потреби малого бізнесу. Такий застосунок повинен забезпечувати швидке введення операцій, підтримку кількох рахунків і категорій, зберігання даних у хмарі, зручну аналітику та працювати без обмежень базового функціоналу у безкоштовній версії. Це дозволить заповнити існуючу нішу програмних

рішень для підприємців, які шукають ефективний і водночас простий інструмент фінансового обліку.

1.3.Формулювання вимог до розроблюваного програмного забезпечення

На основі аналізу предметної області, а також вивчення функціональних можливостей існуючих мобільних застосунків для обліку фінансів, сформульовано вимоги до мобільного програмного забезпечення, яке розробляється в межах цієї кваліфікаційної роботи.

Метою створення застосунку є надання зручного інструменту для малого бізнесу, який дозволяє здійснювати повсякденний облік фінансових операцій без потреби у складних бухгалтерських системах або спеціальних знаннях. Розроблюване програмне забезпечення повинно бути простим у використанні, функціональним, мобільним і надійним.

Загальні вимоги до системи передбачають:

Функціональні вимоги:

a) можливість створення, редагування та видалення фінансових операцій (доходів і витрат);
b) підтримка класифікації транзакцій за категоріями (наприклад, "Оренда", "Продукти", "Продаж");
c) облік кількох фінансових рахунків (готівка, банківська карта, рахунок у банку тощо);
d) автоматичне оновлення балансу рахунків після додавання або видалення транзакцій;відображення зведеної статистики доходів та витрат у вигляді графіків та таблиць;
e) збереження та синхронізація даних через хмарну базу Firebase Firestore;

Нефункціональні вимоги:

a) застосунок має бути розроблений для операційної системи Android (рекомендована підтримка з Android 9.0 та вище);
b) інтерфейс повинен бути інтуїтивно зрозумілим, адаптованим до різних розмірів екранів, підтримувати світлу і темну теми;
c) застосунок повинен забезпечувати високу продуктивність і швидкий відгук на дії користувача;
d) збереження даних має бути безпечним, з використанням перевірених хмарних сервісів;

e) архітектура додатка повинна базуватись на сучасному шаблоні MVVM, що спрощує підтримку й масштабування коду.

Ці вимоги становлять основу технічного завдання для створення мобільного застосунку фінансового трекера. Вони визначають основний функціонал, який повинен бути реалізований у межах кваліфікаційної роботи, та орієнтують на створення програмного продукту, що відповідає сучасним стандартам Android-розробки й задовольняє реальні потреби користувача з числа представників малого бізнесу.

2.Проєктування програмного забезпечення

2.1.Вибір інструментів розробки та архітектури системи

Для розробки Android-застосунку фінансового трекера було обрано сучасний стек технологій, що дозволяє створити ефективний, масштабований та підтримуваний програмний продукт із зручним інтерфейсом і стабільною роботою.

Основою проєкту є мова програмування Kotlin — сучасна, статично типізована мова, яка офіційно підтримується Google для Android-розробки. Використання Kotlin забезпечує скорочення шаблонного коду, покращену безпеку та інтеграцію з екосистемою Jetpack.

Для побудови користувацького інтерфейсу застосовано Jetpack Compose — декларативний UI-фреймворк, що дозволяє створювати інтерфейс у вигляді модульних, реактивних компонентів. Compose забезпечує простоту у розробці, легкість підтримки та швидку адаптацію UI до змін стану додатку.

Архітектура застосунку базується на патерні MVVM (Model-View-ViewModel), який розділяє відповідальність між трьома шарами: UI (View), логікою представлення (ViewModel) та роботою з даними (Model). Це сприяє підвищенню якості коду, його тестуванню та масштабованості.

Структура проєкту поділена на такі основні шари:

a) UI (Presentation Layer): відповідає за відображення інформації користувачу, реалізований за допомогою Jetpack Compose. Включає екрани, компоненти, навігацію між ними, а також підтримку темізації (світла і темна тема) і локалізації.
b) ViewModel (Presentation Logic Layer): керує станом UI, обробляє події користувача, виконує бізнес-логіку, забезпечує зв’язок між UI та даними. Для кожного екрану чи функціонального блоку створюється окремий ViewModel.
c) Data (Data Layer): містить репозиторії, що відповідають за отримання та збереження даних, зокрема через Firebase Firestore та локальні налаштування користувача, реалізовані за допомогою DataStore або SharedPreferences. Також містить моделі даних, які відображають структуру фінансових операцій, рахунків та категорій.

Для впровадження залежностей у систему використовується бібліотека Hilt, що спрощує ін’єкцію залежностей у ViewModel, репозиторії та інші компоненти, покращує модульність та тестованість коду.

Навігація між екранами здійснюється за допомогою Navigation Compose — компонента Jetpack, що інтегрується з Compose і дозволяє легко налаштовувати маршрути переходів у додатку.

Для збереження користувацьких налаштувань, таких як тема та мова інтерфейсу, обрано DataStore, що є сучасною альтернативою SharedPreferences і забезпечує асинхронне, надійне збереження даних.

Таким чином, потік даних у додатку можна представити як послідовність: UI взаємодіє з ViewModel, яка через репозиторій отримує або оновлює дані у DataSource — це може бути як локальне сховище (DataStore), так і хмарна база (Firebase Firestore).

Файлова структура проєкту поділена логічно за функціональними зонами: активність MainActivity, модулі DI (di), робота з даними (data), моделі (model), інтерфейс користувача (ui), утиліти (utils) тощо. Така організація полегшує навігацію по коду, робить проект більш зрозумілим і зручним для підтримки.

Вибір цих інструментів і архітектурних рішень обумовлений потребою створити надійний, гнучкий та ефективний застосунок, який відповідатиме сучасним стандартам розробки Android, а також забезпечить користувачу інтуїтивно зрозумілий і приємний у використанні інтерфейс.

2.2.Структура бази даних у Firebase Firestore

Для зберігання даних у мобільному застосунку фінансового трекера обрано хмарну базу даних Firebase Firestore — це документоорієнтована NoSQL-база даних, що дозволяє зберігати структуровану інформацію у вигляді колекцій та документів, з можливістю масштабування, синхронізації в реальному часі та доступом з різних пристроїв.

Основною перевагою Firestore є її висока гнучкість, автоматичне масштабування, офлайн-підтримка, а також глибока інтеграція з Android SDK. Завдяки підтримці оновлення даних у реальному часі Firestore забезпечує актуальність фінансової інформації для користувача без необхідності ручного оновлення інтерфейсу.

База даних структурована ієрархічно, з урахуванням поділу даних між користувачами. Усі дані згруповані за унікальним ідентифікатором

користувача (userId), що дозволяє кожному користувачу працювати лише зі своїм набором інформації.

Основні колекції та підколекції:

a) Колекція users — містить документи, що відповідають зареєстрованим користувачам. Кожен документ має власний унікальний ідентифікатор userId, який використовується для доступу до відповідних фінансових даних. У документі зберігається електронна пошта, дата створення профілю та, за потреби, хеш пароля (у випадку, якщо не використовується Firebase Authentication).
b) Підколекція accounts — зберігає фінансові рахунки користувача (наприклад, «готівка», «банківська карта»). Кожен рахунок має власний ідентифікатор, назву, поточний баланс, валюту та дату створення. Ці дані використовуються для групування транзакцій за джерелами фінансів.
c) Підколекція categories — містить категорії, до яких відносяться доходи та витрати. Категорії поділяються на типи (INCOME, EXPENSE), мають унікальну назву, іконку та пов’язані з конкретним користувачем. Це дозволяє користувачеві систематизувати свої транзакції за призначенням.
d) Підколекція transactions — містить записи про всі фінансові операції користувача. Кожна транзакція зберігає суму, тип (дохід або витрата), дату, посилання на рахунок і категорію, опис, а також унікальні назви категорії та рахунку, необхідні для відображення у UI без додаткових запитів.

Приклад структури документів:

users/{userId}

accounts/{accountId}

categories/{categoryId}

transactions/{transactionId}

Переваги такої структури:

Безпека. Доступ до даних здійснюється за допомогою Firebase Security Rules, які гарантують, що користувач може працювати лише з власною інформацією.

Гнучкість. Можлива адаптація структури до нових потреб (наприклад, додавання тегів до транзакцій) без повної перебудови моделі даних.

Масштабованість. Кожна колекція може містити необмежену кількість документів, що дозволяє зберігати великі обсяги транзакцій.

Синхронізація. Підтримка оновлень у реальному часі дозволяє користувачу бачити зміни одразу після їх внесення.

Таким чином, структура бази даних у Firebase Firestore дозволяє ефективно зберігати, організовувати та обробляти дані про фінансову активність користувача, забезпечуючи при цьому масштабованість, безпеку й зручність доступу з мобільного пристрою.

2.3.Модель даних: транзакції, рахунки, категорії

У межах реалізації фінансового трекера для малого бізнесу було спроєктовано й реалізовано основні моделі даних, що відображають ключові сутності застосунку: рахунки, категорії та транзакції. Дані моделі формують основу логіки зберігання, обробки та візуалізації фінансової інформації користувача.

Модель рахунку (Account)

Модель Account описує фінансовий рахунок користувача — це може бути готівковий рахунок, банківська карта або будь-яке інше джерело коштів. Вона містить такі поля:

a) id: унікальний ідентифікатор рахунку (генерується Firestore або клієнтом),
b) name: назва рахунку, задана користувачем,
c) balance: поточний баланс рахунку,
d) userId: ідентифікатор користувача, якому належить цей рахунок.

Модель дозволяє користувачеві створювати необмежену кількість рахунків, а також відслідковувати залишок коштів на кожному з них.

Модель категорії (Category)

Модель Category використовується для класифікації транзакцій і забезпечує можливість групування витрат і доходів за напрямками. Кожна категорія має такі атрибути:

a) id: унікальний ідентифікатор категорії,
b) name: назва категорії (наприклад, «Їжа», «Зарплата»),
c) icon: графічне позначення категорії, представлене у вигляді назви ресурсу або іконки,
d) type: тип категорії — дохід (INCOME) або витрата (EXPENSE),

e) userId: ідентифікатор користувача, якому належить ця категорія.

Тип CategoryType реалізований у вигляді перерахування (enum), що дозволяє явно розділити категорії за функціональним призначенням — доходи або витрати.

Модель транзакції (Transaction)

Модель Transaction описує фінансову операцію користувача — надходження коштів або їх витрату. Ця модель є центральною у системі, оскільки об'єднує інформацію з рахунків і категорій. Вона включає:

a) id: унікальний ідентифікатор транзакції,
b) amount: сума операції (позитивне число),
c) categoryId: ідентифікатор обраної категорії,
d) categoryName: збережене ім'я категорії (для швидкого відображення без додаткових запитів),
e) type: тип транзакції (INCOME або EXPENSE),
f) description: короткий опис або коментар до транзакції,
g) date: дата та час виконання операції,
h) accountId: ідентифікатор рахунку, з якого або на який здійснюється операція,
i) accountName: назва рахунку, збережена для відображення,
j) userId: ідентифікатор користувача, який створив транзакцію.

Тип TransactionType, подібно до категорій, реалізований у вигляді перерахування (enum), що спрощує логіку фільтрації та обробки фінансових даних.

Загальна структура

Усі моделі містять поле userId, що забезпечує прив'язку даних до конкретного користувача. Це дозволяє ефективно використовувати Firebase Firestore для зберігання даних у вигляді підколекцій всередині users/{userId}/.

Таке моделювання даних дозволяє гнучко керувати фінансовою інформацією користувача, будувати аналітику, формувати звіти, а також легко масштабувати додаток у разі розширення функціоналу.

2.4.Проєктування навігації та користувацького інтерфейсу (UI)

Проєктування користувацького інтерфейсу мобільного застосунку здійснювалося з урахуванням принципів зручності, інтуїтивності та відповідності сучасним гайдлайнам Material Design. Для реалізації

інтерфейсу використовувався фреймворк Jetpack Compose, який дозволяє створювати декларативний UI із динамічним управлінням станом.

Архітектура інтерфейсу

Інтерфейс застосунку реалізовано відповідно до архітектурного підходу MVVM (Model–View–ViewModel). Кожен екран має окрему ViewModel, яка відповідає за логіку, обробку подій, стан і взаємодію з відповідними use-case та репозиторіями.

Структура інтерфейсу згідно з файлами проєкту:

a) MainActivity.kt – головна активність, яка ініціалізує навігацію, теми та головний контейнер UI.
b) AppNavHost.kt – компонент, що відповідає за реалізацію навігації між екранами, використовуючи Jetpack Navigation Compose.
c) BottomNavigationBar.kt – компонент нижньої панелі навігації, який забезпечує доступ до основних розділів застосунку.
d) MainScreen.kt – контейнер для основної навігації та взаємодії з нижньою навігаційною панеллю.

Основні екрани застосунку

Кожна функціональна частина реалізована у вигляді окремого екрану в пакеті presentation.screens:

a) AuthScreen.kt / AuthViewModel.kt – екран входу або реєстрації користувача (див. рис. 1.1). Забезпечує первинний доступ до системи.
b) DashboardScreen.kt / DashboardViewModel.kt – головна інформаційна панель із графіком або короткою статистикою витрат/доходів.
c) AccountsScreen.kt / AccountsViewModel.kt – екран керування рахунками користувача (створення, редагування, перегляд балансу).
d) CategoriesScreen.kt / CategoriesViewModel.kt – екран для перегляду та керування категоріями доходів і витрат.
e) TransactionsScreen.kt / TransactionsViewModel.kt – екран перегляду фінансових операцій, фільтрація за категоріями, рахунками, датами.
f) SettingsScreen.kt – екран налаштувань користувача (тема, мова, вихід).

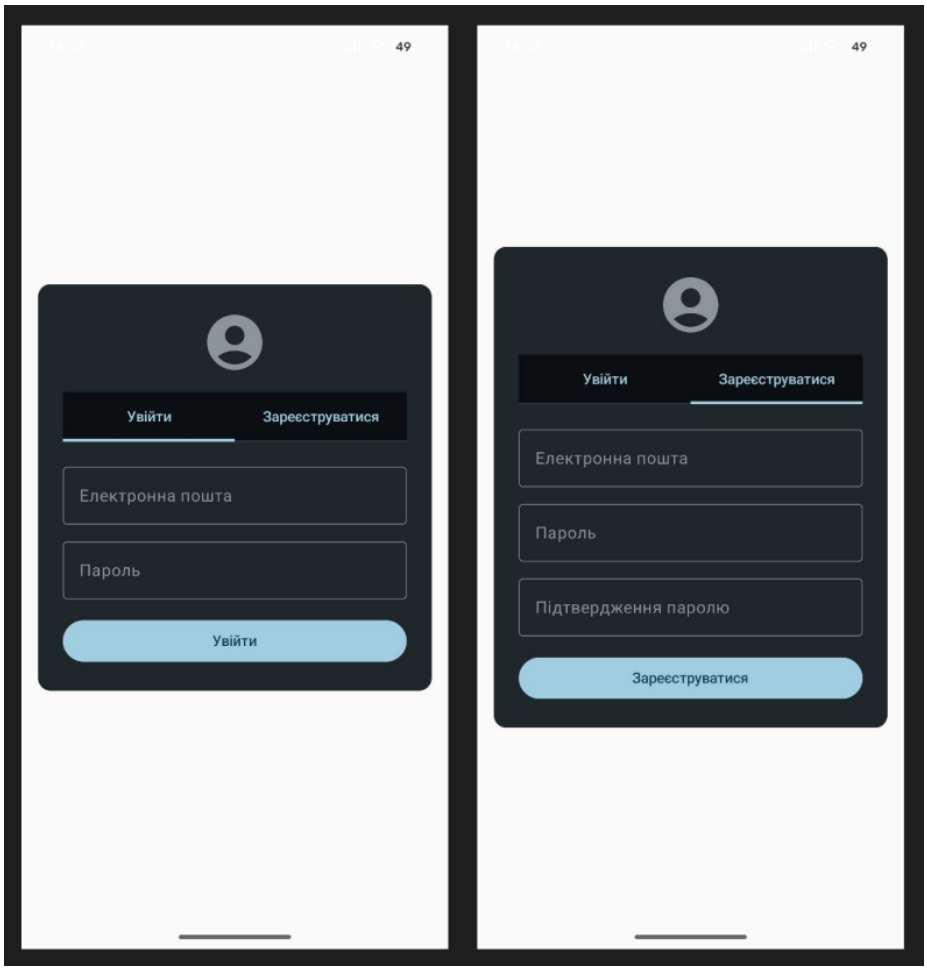

Рисунок 1.1 – екран входу і реєстрації користувача

Навігація

Уся навігація централізовано організована через компонент AppNavHost.kt. Вона побудована на основі декларативного підходу Jetpack Navigation Compose. Роутинг між екранами відбувається за унікальними маршрутами (routes), які зберігаються у вигляді констант.

Нижня навігаційна панель (BottomNavigationBar.kt) використовується на головних екранах після авторизації і забезпечує швидкий перехід між (див. рис. 1.2):

a) Дашборд,
b) Транзакціями,
c) Рахунками,
d) Категоріями,
e) Налаштуваннями.

Всі візуальні елементи створено з використанням бібліотеки Jetpack Compose. Темізація додатку реалізована в пакеті theme/, де задаються основні

кольори, шрифти та стилі компонентів згідно з Material Design 3. Застосунок підтримує темну та світлу теми, які можуть перемикатися користувачем у налаштуваннях.

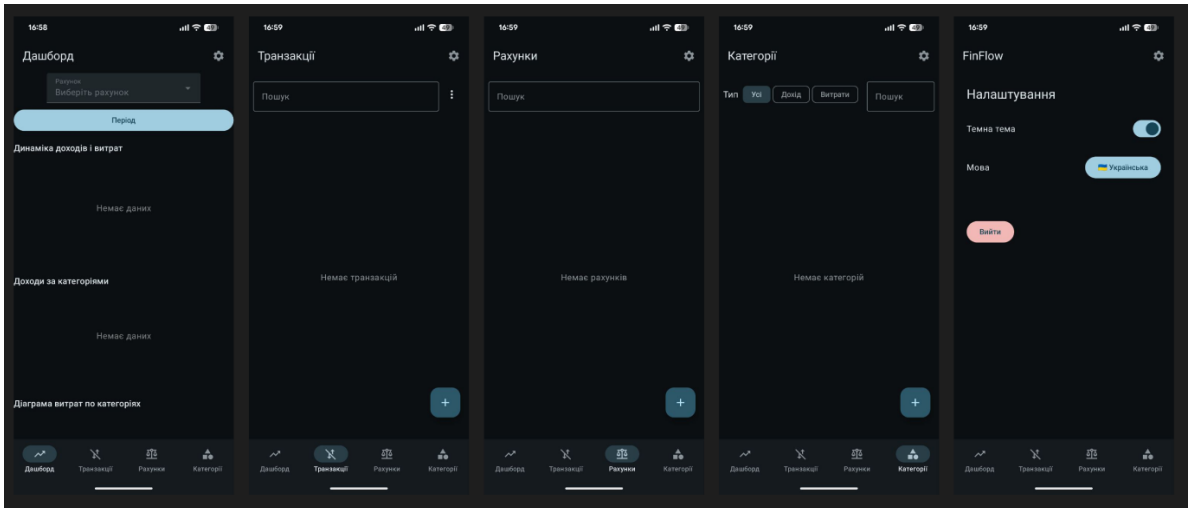

Рисунок 1.2 – Всі екрани застосунку

2.5.Визначення основних бізнес-процесів системи

Розроблювана система фінансового обліку призначена для полегшення ведення особистих або малих бізнес-фінансів шляхом автоматизації основних операцій, пов’язаних з доходами, витратами, категоризацією транзакцій, управлінням рахунками та аналізом фінансової активності. Основними бізнес-процесами, які реалізує система, є:

Управління користувачем. Процес починається з авторизації або реєстрації користувача. Кожен користувач має ізольоване середовище роботи, що забезпечується через унікальний userId. Всі дані (транзакції, рахунки, категорії) зберігаються у Firestore у відповідних підколекціях користувача.

Управління рахунками. Користувач може створювати, редагувати або видаляти рахунки. Рахунки можуть відображати різні джерела фінансів — наприклад, банківські картки, готівкові кошти, цифрові гаманці. Для кожного рахунку зберігається його баланс, що оновлюється при кожній транзакції.

Управління категоріями. Користувач має змогу створювати власні категорії доходів і витрат (наприклад, «Продукти», «Оренда», «Зарплата» тощо), а також редагувати або видаляти їх. Категорії допомагають групувати та аналізувати транзакції за напрямками.

Облік фінансових операцій. Це основний бізнес-процес системи. Користувач створює транзакції з прив’язкою до категорії та рахунку,

вказуючи тип (дохід або витрата), суму, дату та опис. Система автоматично оновлює баланс відповідного рахунку. Операції можна редагувати або видаляти з історії.

Формування статистики та аналітики. Застосунок генерує статистику витрат і доходів у вигляді числових значень та діаграм. Дані можуть групуватися за категоріями, датами або рахунками. Це дозволяє користувачу аналізувати фінансові звички, виявляти основні напрямки витрат і приймати обґрунтовані рішення.

3.Реалізація мобільного Android-застосунку

3.1. Реалізація архітектури MVVM

У процесі розробки мобільного застосунку для фінансового обліку було обрано архітектурний шаблон MVVM (Model–View–ViewModel), який забезпечує чітке розділення відповідальностей між різними компонентами системи, покращує масштабованість, тестованість та зручність підтримки коду.

Model

Рівень Model відповідає за доступ до даних, логіку зберігання і структуру об’єктів. У проєкті він охоплює:

a) Моделі даних: Account, Category, Transaction, які описують основні сутності.
b) Репозиторії (repository) — проміжний шар між ViewModel та джерелами даних (Firestore, DataStore).
c) Data Sources — з'єднання з Firebase Firestore та локальним сховищем.

Ці компоненти реалізовані в пакеті domain.models та domain.repository.

ViewModel

Рівень ViewModel містить бізнес-логіку, стан інтерфейсу та забезпечує зв'язок між View та Model. Усі ViewModel зберігаються в окремих класах, що відповідають кожному екрану, наприклад:

a) AuthViewModel.kt – логіка авторизації,
b) AccountsViewModel.kt – управління рахунками,
c) TransactionsViewModel.kt – обробка транзакцій,
d) DashboardViewModel.kt – агрегування статистики

ViewModel реалізують реактивний підхід через StateFlow або LiveData, що дозволяє автоматично оновлювати UI при зміні стану.

View (UI)

Рівень View реалізований за допомогою Jetpack Compose, який дозволяє створювати інтерфейс декларативно. Кожен екран є окремим функціональним компонентом:

AuthScreen.kt, DashboardScreen.kt, AccountsScreen.kt, TransactionsScreen.kt.

Ці екрани взаємодіють із відповідними ViewModel, отримують стан, реагують на події та відображають дані користувачу.

Впровадження залежностей (DI)

Для впровадження залежностей у ViewModel, репозиторії та інші компоненти використовується бібліотека Hilt. Це дозволяє інкапсулювати логіку створення об'єктів та дотримуватися принципів Clean Architecture.

Файл AppModule.kt в пакеті di/ описує, які залежності слід інжектити до репозиторіїв, ViewModel та інших служб.

Потік даних

Усі взаємодії відбуваються за принципом однонаправленого потоку:

scss

КопироватьРедактировать

UI (View) → ViewModel → Repository → DataSource (Firebase) ← State (Flow / LiveData)

Це забезпечує контрольований і передбачуваний процес обробки подій, полегшує відлагодження та тестування.

Переваги використання MVVM

(a) Спрощена підтримка коду та рефакторинг.
(b) Тестованість бізнес-логіки (ViewModel можна покривати unit-тестами).
(c) Відсутність залежності між UI та джерелами даних.
(d) Зменшення кількості дубльованого коду.

3.2. Розробка функціоналу обліку транзакцій, рахунків та категорій

Функціонал обліку фінансових операцій є основою розроблюваного мобільного застосунку. У відповідності до архітектури MVVM, логіка обробки даних реалізована у відповідних ViewModel: AccountsViewModel, CategoriesViewModel та TransactionsViewModel. Ці компоненти відповідають за взаємодію між користувацьким інтерфейсом і репозиторіями, що звертаються до бази даних Firebase Firestore.

Облік рахунків

Рахунки користувача — це базова одиниця обліку, яка дозволяє вести фінансовий баланс. Весь функціонал зосереджено у AccountsViewModel. У ViewModel реалізовано:

a) метод loadAccounts(userId) для завантаження всіх рахунків поточного користувача;
b) метод addAccount(account, userId) для створення нового рахунку;
c) метод updateAccount(account, userId) для редагування наявного рахунку;
d) метод deleteAccountWithTransactions(userId, accountId) — видалення рахунку разом із усіма пов'язаними транзакціями через відповідний use-case.

Всі виклики виконуються в корутинному контексті viewModelScope, що дозволяє безпечно виконувати асинхронні операції без блокування UI.

Облік категорій

Категорії транзакцій допомагають структурувати дані, групувати операції та формувати аналітичні звіти. У CategoriesViewModel реалізовано наступну функціональність:

a) addCategory(userId, category) — створення нової категорії;
b) updateCategory(userId, category) — редагування категорії;
c) deleteCategory(userId, categoryId) — видалення категорії;
d) setUserId(userId) — ініціалізація завантаження категорій користувача.

Одержання списку категорій реалізовано у вигляді reactive stream через Flow. Дані оновлюються автоматично при внесенні змін у Firestore.

Облік транзакцій

Транзакції є центральною частиною системи, адже відображають усі дії користувача з доходами та витратами. У TransactionsViewModel реалізовано:

a) setUserId(userId) — ініціалізація підписки на транзакції користувача;
b) addTransaction(transaction) — додавання нової фінансової операції;
c) updateTransaction(transaction) — редагування транзакції;
d) deleteTransaction(transactionId) — видалення транзакції.

Список транзакцій отримується як StateFlow, що автоматично оновлюється в UI при зміні бази даних.

Особливості реалізації

(a) Для кожної ViewModel залежності (репозиторії, use-case) впроваджуються за допомогою Hilt, що спрощує тестування та інкапсуляцію логіки.
(b) Усі операції з Firebase реалізовано асинхронно через корутини, що забезпечує високу продуктивність і стабільність застосунку.
(c) Дані кожного користувача ізольовано відповідно до його userId, що забезпечує безпеку та персоналізацію.

3.3. Синхронізація з Firebase Firestore

Для зберігання даних у розроблюваному мобільному застосунку було обрано Firebase Firestore — масштабовану хмарну базу даних від Google, що забезпечує збереження, читання та оновлення даних у режимі реального часу. Вона ідеально підходить для мобільних застосунків завдяки високій швидкодії, простій інтеграції з Android та підтримці офлайн-доступу.

Причини вибору Firestore:

Підтримка реального часу та live-оновлень.

Вбудована безпечна авторизація через Firebase Authentication.

Ієрархічна структура даних (колекції → документи → підколекції).

Легка масштабованість для майбутнього розширення функціоналу.

Структура колекцій у Firestore

Firestore базується на ієрархічному підході до зберігання документів. У даному проєкті кожен користувач має окрему колекцію, що містить такі основні підрозділи:

```
users (collection)
  │
  ├── {userId} (document)
  │    ├── accounts (subcollection)
  │    ├── categories (subcollection)
  │    └── transactions (subcollection)
```

accounts — містить документи типу Account, які зберігають назву рахунку, баланс та пов’язаного користувача.

categories — зберігає категорії типів INCOME та EXPENSE із відповідними іконками та назвами.

transactions — включає записи про фінансові операції з посиланнями на відповідні рахунки та категорії.

Інтеграція у ViewModel

Синхронізація з Firestore реалізована через відповідні репозиторії, які звертаються до API Firestore за допомогою Kotlin-корутин. Для кожної дії (додавання, оновлення, видалення) реалізовані асинхронні функції, що забезпечують негайну реакцію інтерфейсу.

Цей підхід дозволяє отримувати оновлення з Firestore в реальному часі завдяки Flow API. У випадку зміни будь-якого запису (наприклад, нова транзакція) — UI автоматично оновлюється.

Оновлення інтерфейсу у реальному часі

Використання StateFlow у ViewModel забезпечує реактивність — інтерфейс постійно підписаний на оновлення Firestore. Наприклад, при видаленні рахунку одразу відображається змінений список рахунків.

3.4. Побудова аналітичних звітів та діаграм

Однією з ключових функцій розроблюваного фінансового застосунку є можливість візуалізації даних у вигляді аналітичних звітів та графіків. Такий підхід дозволяє користувачам швидко аналізувати фінансовий стан, динаміку доходів і витрат, а також структуру витрат по категоріях.

Основні функціональні можливості аналітичного модуля:

a) Відображення загального балансу, доходів і витрат.
b) Побудова лінійної діаграми для демонстрації змін доходів і витрат у часі.
c) Побудова кругових діаграм (Pie Chart) для аналізу розподілу доходів та витрат за категоріями.
d) Фільтрація транзакцій за обраним періодом часу.
e) Можливість перегляду останніх транзакцій.

Реалізація ViewModel: DashboardViewModel

У ViewModel відбувається підписка на потік транзакцій користувача з Firebase Firestore. Після отримання даних виконується їх агрегація:

a) Обчислення загальної суми доходів (income) і витрат (expense).
b) Розрахунок балансу як різниці між доходами та витратами.
c) Побудова мапи витрат за категоріями для кругових діаграм.

Стан збережено у StateFlow, що забезпечує реактивність інтерфейсу — будь-які зміни в транзакціях одразу відображаються в UI.

Реалізація інтерфейсу (Jetpack Compose)

Лінійна діаграма

Компонент LineChartIncomeExpense будує дві криві: Зелену — для доходів, Червону — для витрат:

```
// Компонент для відображення лінійного графіка доходів та
витрат
@Composable
fun LineChartIncomeExpense(transactions:
List<com.example.finflow.domain.models.Transaction>, modifier:
Modifier = Modifier) {
    val dateFormat = SimpleDateFormat("dd.MM",
Locale.getDefault())
    val grouped = transactions.groupBy {
dateFormat.format(it.date) }
    val days = grouped.keys.sorted()
    val incomePoints = days.map { day -> grouped[day]?.filter {
it.type == TransactionType.INCOME }?.sumOf { it.amount } ?: 0.0
}
    val expensePoints = days.map { day -> grouped[day]?.filter {
it.type == TransactionType.EXPENSE }?.sumOf { it.amount } ?: 0.0
}
    val maxY = (incomePoints +
expensePoints).maxOrNull()?.coerceAtLeast(1.0) ?: 1.0
    if (days.isEmpty()) {
        Box(modifier, contentAlignment = Alignment.Center) {
            Text(stringResource(R.string.no_data), color =
Color.Gray, textAlign = TextAlign.Center)
        }
        return
    }
    Canvas(modifier = modifier) {
        val w = size.width
        val h = size.height
        val stepX = if (days.size > 1) w / (days.size - 1) else
w
        for (i in 1 until days.size) {
            val x0 = stepX * (i - 1)
            val y0 = h - (incomePoints[i - 1] / maxY *
h).toFloat()
            val x1 = stepX * i
            val y1 = h - (incomePoints[i] / maxY * h).toFloat()
            drawLine(Color(0xFF388E3C), Offset(x0, y0),
Offset(x1, y1), strokeWidth = 4f, cap = StrokeCap.Round)
        }
        for (i in 1 until days.size) {
```

```
            val x0 = stepX * (i - 1)
            val y0 = h - (expensePoints[i - 1] / maxY *
h).toFloat()
            val x1 = stepX * i
            val y1 = h - (expensePoints[i] / maxY * h).toFloat()
            drawLine(Color(0xFFD32F2F), Offset(x0, y0),
Offset(x1, y1), strokeWidth = 4f, cap = StrokeCap.Round)
        }
    }
    Row(Modifier.fillMaxWidth(), horizontalArrangement =
Arrangement.SpaceBetween) {
        days.forEach { day -> Text(day, style =
MaterialTheme.typography.labelSmall) }
    }
}
```

Кожна точка відображає загальну суму транзакцій за певний день. Це дозволяє відслідковувати фінансову динаміку.

Кругові діаграми

Компонент PieChartCategory будує кругову діаграму з підписами категорій і відсотковим співвідношенням. Кольори генеруються автоматично для кожної категорії. Діаграми будуються окремо для: доходів і витрат:

```
// Компонент для відображення кругової діаграми з категоріями
@Composable
fun PieChartCategory(
    data: Map<String, Double>,
    modifier: Modifier = Modifier,
    total: Double = 0.0,
    colorKeyTransform: (String) -> String = { it }
) {
    val sum = data.values.sum()
    if (sum == 0.0) {
        Box(modifier, contentAlignment = Alignment.Center) {
            Text(stringResource(R.string.no_data), color =
Color.Gray, textAlign = TextAlign.Center)
        }
        return
    }
    val chartColors = remember(data) {
generateChartColors(data.size) }
    val colorMap = data.keys.mapIndexed { i, k ->
colorKeyTransform(k) to chartColors[i] }.toMap()
    Box(
        modifier = modifier,
        contentAlignment = Alignment.Center
    ) {
        val size = remember(modifier) { 180.dp }
        Canvas(modifier =
```

```
Modifier.size(size).align(Alignment.Center)) {
            var startAngle = -90f
            data.entries.forEach { (cat, value) ->
                val colorKey = colorKeyTransform(cat)
                val color = colorMap[colorKey] ?:
Color.LightGray
                val sweep = (value / sum * 360f).toFloat()
                drawArc(
                    color = color,
                    startAngle = startAngle,
                    sweepAngle = sweep,
                    useCenter = true
                )
                startAngle += sweep
            }
        }
        Column(modifier =
Modifier.align(Alignment.CenterEnd).padding(start = 16.dp)) {
            data.entries.forEach { (cat, value) ->
                val percent = if (sum > 0) (value / sum *
100).toInt() else 0
                val colorKey = colorKeyTransform(cat)
                val color = colorMap[colorKey] ?:
Color.LightGray
                Row(verticalAlignment =
Alignment.CenterVertically) {
                    Box(Modifier.size(12.dp).background(color))
                    Spacer(Modifier.width(4.dp))
                    Text("$cat: %.2f ₽
($percent%%)".format(value), style =
MaterialTheme.typography.bodySmall)
                }
            }
        }
    }
}
```

Фільтрація

Застосовано DateRangePicker, який дозволяє обирати діапазон дат для фільтрації транзакцій. Це забезпечує гнучкість при аналізі фінансових даних за конкретні періоди.

3.5. Тестування застосунку: функціональне, модульне, UI

Для забезпечення стабільної та коректної роботи розробленого фінансового застосунку було проведено комплексне тестування, яке включало функціональне тестування, модульне тестування та UI-тестування. Метою тестування є виявлення та усунення можливих помилок у логіці

додатку, перевірка правильності обробки даних і стабільності інтерфейсу користувача.

Функціональне тестування

Функціональне тестування охоплює основні сценарії використання застосунку з боку кінцевого користувача. Перевірялися такі функції:

a) Реєстрація та автентифікація користувача через Firebase.
b) Додавання, редагування та видалення транзакцій (доходів і витрат).
c) Робота з рахунками: створення, редагування, видалення.
d) Створення категорій доходів і витрат.
e) Коректне оновлення загального балансу, підрахунок доходів і витрат.
f) Візуалізація даних на діаграмах.
g) Фільтрація транзакцій за обраним рахунком та часовим діапазоном.

Усі функціональні сценарії було протестовано вручну на реальному пристрої під керуванням Android. Система реагує на введення очікуваним чином, усі дані зберігаються та оновлюються в хмарній базі Firebase Firestore.

Модульне тестування

Модульні тести були реалізовані для окремих компонентів логіки, зокрема:

a) ViewModel для транзакцій (TransactionsViewModel).
b) UseCase-класи для обробки дій над категоріями та рахунками.
c) Репозиторії (TransactionRepository, AccountRepository), що взаємодіють з Firebase.

Модульні тести дозволили ізолювати логіку від UI та перевірити її незалежно від стану інтерфейсу. Було перевірено:

a) Правильність обчислення балансу.
b) Обробку помилок при взаємодії з Firebase.
c) Роботу потоків StateFlow при оновленні даних.

Для написання тестів використовувався фреймворк JUnit разом з Mockito для імітації залежностей.

UI-тестування

UI-тестування здійснювалося з використанням інструментів Jetpack Compose UI Test, які дозволяють автоматизовано перевіряти:

a) Відображення елементів інтерфейсу.
b) Реакцію на кліки, введення тексту, прокрутку списків.
c) Правильне відображення діаграм, текстових повідомлень і переходів між екранами.

Було перевірено:

a) Роботу навігації між основними екранами.
b) Відображення повідомлень при відсутності даних.
c) Реакцію елементів UI на зміну мови, теми, стану облікового запису.

Висновок

Усі тести показали, що застосунок працює стабільно, відповідає вимогам до функціональності та має правильну інтеграцію з Firebase. Проведене тестування дозволило підвищити надійність, зменшити кількість помилок і забезпечити позитивний користувацький досвід.

4.Оцінка результатів та перспективи розвитку

4.1. Оцінка відповідності поставленим вимогам

У процесі розробки мобільного застосунку для обліку фінансів у малому бізнесі було визначено низку функціональних та нефункціональних вимог. Після завершення реалізації проведено оцінку відповідності розробленої системи цим вимогам.

Функціональні вимоги

Облік транзакцій (доходів та витрат)
Реалізовано. Користувач може додавати, редагувати й видаляти фінансові операції, вказуючи категорію, рахунок, суму та опис.

Підтримка декількох рахунків
Реалізовано. Система дозволяє створювати та управляти кількома фінансовими рахунками з окремим балансом для кожного.

Підтримка категорій доходів та витрат
Реалізовано. Можна створювати власні категорії, які поділяються на типи (дохід або витрата) та мають відповідну іконку.

Хмарне збереження даних
Реалізовано. Усі дані зберігаються у Firebase Firestore та пов’язані з унікальним користувачем.

Аналітика та звітність
Реалізовано. Застосунок формує діаграми доходів/витрат за категоріями, а також графік динаміки фінансових операцій за датами.

Фільтрація та сортування транзакцій
Реалізовано. Можна відфільтрувати транзакції за рахунком, діапазоном дат і типом.

Нефункціональні вимоги

Зручний інтерфейс користувача
Реалізовано. Інтерфейс побудований з використанням Jetpack Compose та адаптований під мобільні пристрої.

Модульна архітектура (MVVM)
Реалізовано. Архітектура проєкту відповідає шаблону MVVM із розділенням шарів UI, логіки ViewModel і роботи з даними.

Підтримка хмарної авторизації
Реалізовано. Користувач проходить автентифікацію через Firebase Auth.

Можливість розширення функціоналу
Передбачено. Архітектура і реалізація репозиторіїв дозволяють легко додати нові функції, наприклад, повторювані транзакції або нагадування.

Мінімізація затримок і швидкий доступ до даних
Досягнуто. Завдяки використанню StateFlow і локального кешування у ViewModel додаток працює швидко та стабільно.

Висновок

Усі основні вимоги, поставлені на етапі проєктування, були виконані повністю. Застосунок відповідає цілям кваліфікаційної роботи, демонструє стабільність, розширюваність і зручність у користуванні. Це свідчить про досягнення поставленої мети та відповідність функціоналу розробленої системи сучасним вимогам до мобільних застосунків для фінансового обліку.

4.2. Проблеми, що виникли під час реалізації, та шляхи їх вирішення

У ході реалізації мобільного застосунку для обліку фінансів було виявлено низку технічних та організаційних проблем, які вимагали додаткових рішень та адаптацій. Нижче подано опис основних труднощів та підходів, що були застосовані для їх подолання.

Складність роботи з Firestore у реальному часі

Проблема: При організації потокової синхронізації з Firebase Firestore виникали труднощі з відображенням змін у реальному часі без надмірного навантаження на мережу та UI.

Рішення:
Було використано бібліотеку StateFlow у поєднанні з collectLatest у ViewModel. Це дозволило відстежувати оновлення колекцій (транзакцій, рахунків, категорій) та оперативно оновлювати інтерфейс без дублювання запитів і зависань.

Конфлікти між транзакціями в Firestore

Проблема: При одночасному редагуванні даних (наприклад, оновлення рахунку та додавання транзакції) виникали конфлікти, що призводили до втрати або некоректного відображення даних.

Рішення:
Було реалізовано атомарну логіку обробки транзакцій за допомогою окремого UseCase (DeleteAccountWithTransactionsUseCase) для групових змін. Це забезпечило узгодженість даних при видаленні або зміні пов’язаних об'єктів.

Труднощі з відображенням діаграм у Compose

Проблема: На початковому етапі реалізації аналітичних звітів виникали труднощі з відображенням графіків і кругових діаграм у Jetpack Compose, оскільки стандартні компоненти не підтримували таку візуалізацію.

Рішення:
Було розроблено власні компоненти Canvas для побудови лінійного графіка доходів/витрат та кругових діаграм за категоріями. Це дозволило забезпечити гнучкість і кросплатформеність UI без залежності від сторонніх бібліотек.

Обробка порожніх станів

Проблема: У разі відсутності рахунків, транзакцій або категорій інтерфейс виглядав незрозумілим і неінформативним для користувача.

Рішення:
Додано перевірки на порожні списки у ViewModel та відповідні повідомлення в інтерфейсі («Дані відсутні», «Створіть рахунок» тощо), що значно покращило користувацький досвід.

Тестування в умовах обмеженого інтернет-з’єднання

Проблема: В умовах нестабільного з’єднання з мережею частина функцій не працювала коректно або повертала помилки.

Рішення:
Було додано базову обробку помилок на рівні ViewModel, а також реалізовано відображення повідомлень про помилки в UI. У подальшому передбачено реалізацію кешування для офлайн-доступу.

4.3. Перспективи подальшого розвитку системи

Розроблений мобільний застосунок для обліку фінансових операцій є першим кроком до створення повноцінної системи фінансового контролю для малого бізнесу та індивідуальних користувачів. Незважаючи на реалізацію основного функціоналу, існує низка напрямків, у яких можливе подальше розширення та вдосконалення системи.

Одним із ключових векторів розвитку є реалізація багатокористувацького доступу з можливістю обміну рахунками, спільного управління фінансами та ролями користувачів. Це дозволить застосовувати систему не лише для особистих потреб, а й у рамках малого колективу чи родини.

Також перспективним є впровадження функцій аналітики з використанням машинного навчання. Це може включати прогнозування витрат, виявлення аномалій у транзакціях, автоматичне групування витрат за поведінковими шаблонами користувача.

Окремо варто розглянути інтеграцію з банківськими сервісами та API фінансових установ, що дозволить автоматично імпортувати банківські транзакції в систему, зменшуючи потребу у ручному введенні даних.

Також доцільним буде розширення системи звітності з можливістю експорту даних у формати PDF, Excel, а також надсилання звітів електронною поштою.

З точки зору користувацького досвіду, можливо реалізувати віджети для головного екрану смартфона, темну тему, офлайн-режим з подальшою синхронізацією та нагадування про заплановані витрати або платежі.

На завершення, у майбутньому передбачається публікація застосунку в Google Play, що потребує додаткової оптимізації, локалізації, юридичної підготовки та реалізації політики конфіденційності.

Таким чином, система має широкі перспективи подальшого розвитку як у напрямку покращення функціональності, так і в контексті масштабування, розширення аудиторії та інтеграції з іншими цифровими сервісами.

Висновок

У результаті виконання кваліфікаційної роботи було розроблено мобільний застосунок для обліку фінансів, призначений для користувачів малого бізнесу та індивідуального використання. У процесі реалізації було проведено аналіз предметної області, визначено основні вимоги до функціоналу системи, досліджено існуючі аналоги та обґрунтовано вибір сучасних інструментів розробки.

На основі архітектурного підходу MVVM побудовано гнучку й масштабовану структуру проєкту. Реалізовано можливість створення й

обліку рахунків, категорій доходів та витрат, а також ведення транзакцій. Застосунок зберігає дані у хмарній базі Firebase Firestore, забезпечуючи синхронізацію між пристроями.

Користувацький інтерфейс створено з використанням Jetpack Compose, що забезпечує зручність у навігації, гнучку темізацію та підтримку адаптивного дизайну. Окрему увагу приділено аналітиці — реалізовано візуалізацію доходів та витрат у вигляді графіків і діаграм, що підвищує інформативність застосунку.

Проведено функціональне, модульне та UI-тестування, результати якого засвідчили відповідність системи визначеним вимогам. У процесі реалізації було розв’язано ряд технічних проблем, пов’язаних із роботою з Firestore та збереженням стану інтерфейсу.

Розроблений застосунок може бути використаний як основа для створення повноцінної системи фінансового менеджменту. У роботі також окреслено перспективи розвитку системи, зокрема інтеграцію з банківськими сервісами, розширення функціоналу аналітики, впровадження спільного доступу тощо.

Таким чином, поставлені завдання кваліфікаційної роботи виконано в повному обсязі, а розроблене програмне забезпечення демонструє ефективність у вирішенні задачі автоматизованого фінансового обліку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Platform Architecture. URL: https://developer.android.com/guide/platform/

2. Мова програмування Kotlin. URL: https://kotlinlang.org

3. Jetpack Compose UI App Development Toolkit. URL: https://developer.android.com/develop/ui/compose

4. Android Studio IDE. URL: https://developer.android.com/studio

5. Використання шаблону MVVM у розробці мобільних Android-застосунків з Jetpack Compose. URL: https://developer.android.com/topic/libraries/architecture/viewmodel

6. Dependency injection in Android. URL: https://developer.android.com/training/dependency-injection

7. Dependency injection with Hilt. URL: https://developer.android.com/training/dependency-injection/hilt-android

8. Бібліотека Hilt для ін’єкції залежностей у Android-застосунках. URL: https://dagger.dev/hilt/

9.Compose State vs StateFlow: State Management in Jetpack Compose. URL: https://medium.com/@husayn.fakher/compose-state-vs-stateflow-statemanagement-in-jetpack-compose-c99740732023

Заява студента про оригінальність роботи

_____

ПІБ студента

Номер залікової книжки: _____

Засвідчую, що кваліфікаційна робота на тему

__Фінансовий трекер для малого бізнесу: розробка Android-застосунку для обліку доходів і витрат _____

_____

зі спеціальності _____121 – Інженерія програмного забезпечення_____

була підготовлена виключно мною і не порушує авторські права третіх осіб відповідно до закону про авторське право; повністю або частково не була використана як основа для отримання диплома про вищу освіту або науковий ступінь мною або іншою особою. Представлена мною для перевірки електронна версія роботи збігається з друкованим примірником.

Підтверджую, що був(ла) проінформований(на) про права та обов'язки здобувача вищої освіти Університету та правила, що стосуються перевірки оригінальності наукових робіт. Згоден(на) на обробку моєї письмової роботи й архівування цієї роботи в базі даних відповідно антиплагіатних правил і процедур Університету.

З Положенням про академічну доброчесність у Криворізькому національному університеті ознайомлений(на)

_____ _____ _____К. О. Сивожелэзов_____

(дата) (підпис) (ініціали та прізвище автора роботи)

Укладачі: Доценко Ірина Олексіївна
Стрюк Андрій Миколайович
Рибальченко Олена Геннадіївна
Шаповалова Нонна Наілєвна
Реєстрац. № ____

Підписано до друку _____2025 р.
Формат А5
Обсяг 107 сторінок
Тираж ___ примірників
Видавничий центр КНУ, вул. Віталія Матусевича, 11, м. Кривий Ріг

Додоток А

Код застосунку

MainActivity.kt

```
package com.example.finflow

import android.content.res.Configuration
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.isSystemInDarkTheme
import androidx.compose.runtime.*
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.finflow.presentation.MainScreen
import com.example.finflow.presentation.theme.FinFlowTheme
import com.example.finflow.presentation.screens.auth.AuthScreen
import
com.example.finflow.presentation.screens.auth.AuthViewModel
import java.util.Locale
import dagger.hilt.android.AndroidEntryPoint
import androidx.compose.ui.platform.LocalContext
import androidx.compose.runtime.compositionLocalOf

// Головна активність додатку, яка відповідає за ініціалізацію
інтерфейсу користувача
@Suppress("DEPRECATION")
@AndroidEntryPoint
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        val prefs = com.example.finflow.data.UserPreferences
        val savedTheme = prefs.getTheme(this)
        val savedLanguage = prefs.getLanguage(this)
        setContent {
            val systemDarkTheme = isSystemInDarkTheme()
            var isDarkTheme by rememberSaveable {
                mutableStateOf(
                    when (savedTheme) {
                        "dark" -> true
                        "light" -> false
                        else -> systemDarkTheme
                    }
                )
            }
            var selectedLanguage by rememberSaveable {
                mutableStateOf(savedLanguage ?:
Locale.getDefault().language)
            }
            val currentLocale = remember(selectedLanguage) {
```

```
Locale(selectedLanguage) }
            val authViewModel: AuthViewModel = hiltViewModel()
            val isAuthenticated by
authViewModel.isAuthenticated.collectAsState()
            val isLoading by
authViewModel.isLoading.collectAsState()
            val error by authViewModel.error.collectAsState()

            // Функція для оновлення локалі додатку
            fun updateLocale(language: String) {
                val locale = Locale(language)
                Locale.setDefault(locale)
                val config =
Configuration(resources.configuration)
                config.setLocale(locale)
                resources.updateConfiguration(config,
resources.displayMetrics)
            }

            // Оновлюємо локаль при зміні мови
            LaunchedEffect(selectedLanguage) {
                prefs.setLanguage(this@MainActivity,
selectedLanguage)
                updateLocale(selectedLanguage)
            }
            // Зберігаємо тему при її зміні
            LaunchedEffect(isDarkTheme) {
                prefs.setTheme(this@MainActivity, if
(isDarkTheme) "dark" else "light")
            }
            CompositionLocalProvider(LocalAppLocale provides
currentLocale) {
                // Застосовуємо тему до всього додатку
                FinFlowTheme(darkTheme = isDarkTheme) {
                    if (isAuthenticated) {
                        // Головний екран додатку, який
відображає різні функції
                        MainScreen(
                            isDarkTheme = isDarkTheme,
                            onToggleTheme = { isDarkTheme =
!isDarkTheme },
                            selectedLanguage = selectedLanguage,
                            onLanguageChange = {
selectedLanguage = it },
                            onLogout = { authViewModel.logout()
}
                        )
                    } else {
                        // Екран автентифікації, який
відображається, якщо користувач не автентифікований
                        AuthScreen(
                            isLoading = isLoading,
```

```
                            error = error,
                            onSignIn = { email, password ->
authViewModel.signIn(email, password) },
                            onSignUp = { email, password ->
authViewModel.signUp(email, password) }
                        )
                    }
                }
            }
        }
    }
}

val LocalAppLocale = compositionLocalOf { Locale.getDefault() }
```

App.kt

```
package com.example.finflow

import android.app.Application
import dagger.hilt.android.HiltAndroidApp

// Основний клас додатку, який ініціалізує Hilt для впровадження
залежностей
@HiltAndroidApp
class App : Application()
```

MainScreen.kt

```
package com.example.finflow.presentation

import android.annotation.SuppressLint
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.calculateStartPadding
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Settings
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.Scaffold
import androidx.compose.material3.Text
import androidx.compose.material3.TopAppBar
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import androidx.navigation.compose.currentBackStackEntryAsState
import androidx.navigation.compose.rememberNavController
import com.example.finflow.R
import com.example.finflow.ui.components.BottomNavigationBar
import com.example.finflow.presentation.navigation.AppNavHost
import com.example.finflow.presentation.navigation.Screen
```

```
// функція для головного екрана програми
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MainScreen(
    isDarkTheme: Boolean = false,
    onToggleTheme: () -> Unit = {},
    selectedLanguage: String = "uk",
    onLanguageChange: (String) -> Unit = {},
    onLogout: () -> Unit = {}
) {
    val navController = rememberNavController()
    val navBackStackEntry by
navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route

    Scaffold(
        topBar = {
            TopAppBar(
                title = {
                    Text(
                        when (currentRoute) {
                            Screen.Dashboard.route ->
stringResource(id = R.string.dashboard)
                            Screen.Transactions.route ->
stringResource(id = R.string.transactions)
                            Screen.Accounts.route ->
stringResource(id = R.string.accounts)
                            Screen.Categories.route ->
stringResource(id = R.string.categories)
                            else -> stringResource(id =
R.string.app_name)
                        }
                    )
                },
                actions = {
                    IconButton(onClick = {
                        if (currentRoute !=
Screen.Settings.route) {

navController.navigate(Screen.Settings.route) {
                                launchSingleTop = true
                            }
                        }
                    }) {
                        Icon(Icons.Filled.Settings,
contentDescription = stringResource(id = R.string.settings))
                    }
                }
            )
        },
```

```
        bottomBar = {
            BottomNavigationBar(navController, currentRoute)
        }
    ) {
        Box(
            modifier = androidx.compose.ui.Modifier.padding(
                start =
it.calculateStartPadding(androidx.compose.ui.unit.LayoutDirectio
n.Ltr),
                end = 0.dp,
                bottom = it.calculateBottomPadding(),
                top = it.calculateTopPadding()
            )
        ) {
            AppNavHost(
                navController = navController,
                isDarkTheme = isDarkTheme,
                onToggleTheme = onToggleTheme,
                selectedLanguage = selectedLanguage,
                onLanguageChange = onLanguageChange,
                onLogout = onLogout
            )
        }
    }
}
```

TransactionsViewModel.kt

```
package com.example.finflow.presentation.screens.transactions

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.finflow.domain.models.Transaction
import
com.example.finflow.domain.repository.TransactionRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.SharingStarted
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.stateIn
import kotlinx.coroutines.launch
import javax.inject.Inject

// ViewModel для управління транзакціями користувача
@HiltViewModel
class TransactionsViewModel @Inject constructor(
    private val repository: TransactionRepository
) : ViewModel() {
    private var _userId: String = ""
    private var _transactions: StateFlow<List<Transaction>> =
MutableStateFlow(emptyList())
    val transactions: StateFlow<List<Transaction>>
        get() = _transactions
```

```
    // Ініціалізує ViewModel з ідентифікатором користувача
    fun setUserId(userId: String) {
        _userId = userId
        _transactions =
repository.getTransactions(userId).stateIn(
            viewModelScope,
            SharingStarted.WhileSubscribed(5000),
            emptyList()
        )
    }

    // Додає нову транзакцію
    fun addTransaction(transaction: Transaction) {
        viewModelScope.launch {
            repository.addTransaction(_userId, transaction)
        }
    }

    // Оновлює існуючу транзакцію
    fun updateTransaction(transaction: Transaction) {
        viewModelScope.launch {
            repository.updateTransaction(_userId, transaction)
        }
    }

    // Видаляє транзакцію за ідентифікатором
    fun deleteTransaction(transactionId: String) {
        viewModelScope.launch {
            repository.deleteTransaction(_userId, transactionId)
        }
    }
}
```

TransactionsScreen.kt

```
package com.example.finflow.presentation.screens.transactions

import android.annotation.SuppressLint
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Edit
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.MoreVert
import androidx.compose.material3.*
import androidx.compose.runtime.*
```

```
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.KeyboardType
import com.example.finflow.R
import com.example.finflow.domain.models.Transaction
import com.example.finflow.domain.models.TransactionType
import com.example.finflow.domain.models.Category
import androidx.hilt.navigation.compose.hiltViewModel
import androidx.compose.ui.unit.dp
import com.example.finflow.domain.models.Account
import
com.example.finflow.presentation.screens.accounts.AccountsViewMo
del
import
com.example.finflow.presentation.screens.categories.CategoriesVi
ewModel
import com.google.firebase.auth.FirebaseAuth
import java.text.SimpleDateFormat
import java.util.*
import androidx.compose.material3.Icon as Icon1

// Екран для відображення та управління транзакціями користувача
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TransactionsScreen(
    viewModel: TransactionsViewModel = hiltViewModel()
) {
    val accountsViewModel: AccountsViewModel = hiltViewModel()
    val categoriesViewModel: CategoriesViewModel =
hiltViewModel()
    val userId = FirebaseAuth.getInstance().currentUser?.uid
    LaunchedEffect(userId) {
        if (!userId.isNullOrBlank()) {
            viewModel.setUserId(userId)
            accountsViewModel.loadAccounts(userId)
            categoriesViewModel.setUserId(userId)
        }
    }
    if (userId.isNullOrBlank()) {
        Box(modifier = Modifier.fillMaxSize(), contentAlignment
= Alignment.Center) {
            Text(text = "Ошибка: пользователь не авторизован")
        }
        return
    }
    val transactions: List<Transaction> =
        viewModel.transactions.collectAsState(initial =
emptyList()).value
```

```
    val accounts by accountsViewModel.accounts.collectAsState()
    val categories: List<Category> =
categoriesViewModel.categories.collectAsState(emptyList()).value
    var showAddSheet by remember { mutableStateOf(false) }
    var editTx by remember { mutableStateOf<Transaction?>(null)
}
    var searchQuery by remember { mutableStateOf("") }
    var filterType by remember { mutableStateOf("all") }
    var filterAccount by remember { mutableStateOf("") }
    var filterCategory by remember { mutableStateOf("") }
    val filterDateFrom = remember { mutableStateOf<Date?>(null)
}
    val filterDateTo = remember { mutableStateOf<Date?>(null) }
    remember { SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()) }
    val sortBy = remember { mutableStateOf("date_desc") }
    val categoriesList = when (filterType) {
        "all" -> categories.map { it.name }.distinct().sorted()
        else -> categories.filter { it.type.name.lowercase() ==
filterType }.map { it.name }.distinct().sorted()
    }
    val sheetState = rememberModalBottomSheetState()
    var showFilterSheet by remember { mutableStateOf(false) }
    var showDeleteDialog by remember { mutableStateOf(false) }
    var txToDelete by remember {
mutableStateOf<Transaction?>(null) }
    val snackbarHostState = remember { SnackbarHostState() }
    var snackbarMessage by remember {
mutableStateOf<String?>(null) }
    val transactionDeletedMsg = stringResource(id =
R.string.transaction_deleted)
    val transactionAddedMsg = stringResource(id =
R.string.transaction_added)
    val transactionUpdatedMsg = stringResource(id =
R.string.transaction_updated)

    Scaffold(
        snackbarHost = { SnackbarHost(snackbarHostState) },
        floatingActionButton = {
            Row {
                FloatingActionButton(onClick = { showAddSheet =
true }) {
                    Icon1(Icons.Default.Add, contentDescription
= stringResource(id = R.string.add_transaction))
                }
            }
        }
        // topBar повністю прибрано, пошук і фільтр тільки в
Column
    ) { innerPadding ->
        Column(modifier = Modifier.fillMaxSize()) {
            Row(
```

```
                modifier =
Modifier.fillMaxWidth().padding(8.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                OutlinedTextField(
                    value = searchQuery,
                    onValueChange = { searchQuery = it },
                    label = { Text(stringResource(id =
R.string.search)) },
                    modifier = Modifier.weight(1f)
                )
                IconButton(onClick = { showFilterSheet = true })
{
                    Icon1(Icons.Default.MoreVert,
contentDescription = stringResource(id = R.string.filter))
                }
            }
            val filtered = transactions
                .asSequence()
                .filter { filterType == "all" ||
it.type.name.lowercase() == filterType }
                .filter { filterCategory.isBlank() ||
it.categoryName == filterCategory }
                .filter { filterAccount.isBlank() ||
it.accountName == filterAccount }
                .filter { filterDateFrom.value == null ||
!it.date.before(filterDateFrom.value) }
                .filter { filterDateTo.value == null ||
!it.date.after(filterDateTo.value) }
                .filter {
                    searchQuery.isBlank() ||

it.categoryName.contains(searchQuery, true) ||

(it.description.contains(searchQuery, true))
                }
                .toList()
            val sorted = when (sortBy.value) {
                "date_desc" -> filtered.sortedByDescending { t -
> t.date }
                "date_asc" -> filtered.sortedBy { t -> t.date }
                "amount_desc" -> filtered.sortedByDescending { t
-> t.amount }
                "amount_asc" -> filtered.sortedBy { t ->
t.amount }
                else -> filtered
            }
            if (sorted.isEmpty()) {
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) {
```

```
                    Text(
                        text = stringResource(id =
R.string.no_transactions),
                        style =
MaterialTheme.typography.bodyLarge,
                        color =
MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f)
                    )
                }
            } else {
                LazyColumn(modifier = Modifier.fillMaxSize()) {
                    items(sorted) { tx ->
                        TransactionCardModern(
                            tx = tx,
                            categories = categories,
                            onEdit = { editTx = tx },
                            onDelete = {
                                txToDelete = tx
                                showDeleteDialog = true
                            }
                        )
                    }
                    item { Spacer(modifier =
Modifier.Companion.height(200.dp)) }
                }
            }
            if (showDeleteDialog && txToDelete != null) {
                AlertDialog(
                    onDismissRequest = { showDeleteDialog =
false; txToDelete = null },
                    title = { Text(stringResource(id =
R.string.delete_transaction_question)) },
                    text = { Text(stringResource(id =
R.string.delete_transaction_confirm, txToDelete!!.categoryName,
txToDelete!!.amount)) },
                    confirmButton = {
                        TextButton(onClick = {

viewModel.deleteTransaction(txToDelete!!.id)
                            showDeleteDialog = false
                            txToDelete = null
                            snackbarMessage =
transactionDeletedMsg
                        }) { Text(stringResource(id =
R.string.delete)) }
                    },
                    dismissButton = {
                        TextButton(onClick = { showDeleteDialog
= false; txToDelete = null }) { Text(stringResource(id =
R.string.cancel)) }
                    }
                )
```

```
            }
        }
        if (showAddSheet) {
            ModalBottomSheet(
                onDismissRequest = { showAddSheet = false },
                sheetState = sheetState
            ) {
                TransactionSheetForm(
                    onSubmit = { tx ->
                        viewModel.addTransaction(tx)
                        showAddSheet = false
                        snackbarMessage = transactionAddedMsg
                    },
                    onDismiss = { showAddSheet = false },
                    categories = categories,
                    userId = userId
                )
            }
        }
        if (editTx != null) {
            ModalBottomSheet(
                onDismissRequest = { editTx = null },
                sheetState = sheetState
            ) {
                TransactionSheetForm(
                    onSubmit = { tx ->
                        viewModel.updateTransaction(tx)
                        editTx = null
                        snackbarMessage = transactionUpdatedMsg
                    },
                    onDismiss = { editTx = null },
                    categories = categories,
                    initial = editTx,
                    userId = userId
                )
            }
        }
    }
    if (showFilterSheet) {
        val filteredCategoriesList = when (filterType) {
            "all" -> categories.map { it.name
}.distinct().sorted()
            else -> categories.filter { it.type.name.lowercase()
== filterType }.map { it.name }.distinct().sorted()
        }
        ModalBottomSheet(
            onDismissRequest = { showFilterSheet = false },
            sheetState = sheetState
        ) {
            FilterSheet(
                filterType = filterType,
                filterAccount = filterAccount,
```

```
                    filterCategory = filterCategory,
                    filterDateFrom = filterDateFrom.value,
                    filterDateTo = filterDateTo.value,
                    accounts = accounts,
                    categoriesList = filteredCategoriesList,
                    onApply = { type, acc, cat, from, to ->
                        filterType = type
                        filterAccount = acc
                        filterCategory = cat
                        filterDateFrom.value = from
                        filterDateTo.value = to
                        showFilterSheet = false
                    },
                    onReset = {
                        filterType = "all"
                        filterAccount = ""
                        filterCategory = ""
                        filterDateFrom.value = null
                        filterDateTo.value = null
                        showFilterSheet = false
                    },
                    onDismiss = { showFilterSheet = false }
                )
            }
        }
        LaunchedEffect(snackbarMessage) {
            snackbarMessage?.let {
                snackbarHostState.showSnackbar(it)
                snackbarMessage = null
            }
        }
    }

    // Карта транзакції
    @Composable
    fun TransactionCardModern(tx: Transaction, categories:
    List<Category>, onEdit: () -> Unit, onDelete: () -> Unit) {
        Card(
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp),
            elevation = CardDefaults.cardElevation(defaultElevation
    = 4.dp),
            shape = RoundedCornerShape(16.dp),
            colors = CardDefaults.cardColors(
                containerColor = if (tx.type ==
    TransactionType.INCOME)
    MaterialTheme.colorScheme.primaryContainer else
    MaterialTheme.colorScheme.errorContainer
            )
        ) {
            Row(
```

```
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            val category = categories.find { it.id ==
tx.categoryId }
            Box(
                modifier = Modifier
                    .size(48.dp),
                contentAlignment = Alignment.Center
            ) {
                Text(
                    text = category?.icon ?: "",
                    fontSize =
MaterialTheme.typography.headlineMedium.fontSize
                )
            }
            Spacer(modifier = Modifier.width(12.dp))
            Column(modifier = Modifier.weight(1f)) {
                Text(
                    text = tx.categoryName,
                    style =
MaterialTheme.typography.titleMedium,
                    color =
MaterialTheme.colorScheme.onPrimaryContainer
                )
                Spacer(modifier = Modifier.height(2.dp))
                Text(
                    text = tx.accountName,
                    style =
MaterialTheme.typography.labelMedium,
                    color =
MaterialTheme.colorScheme.onPrimaryContainer.copy(alpha = 0.7f)
                )
                Spacer(modifier = Modifier.height(4.dp))
                Text(
                    text = tx.description,
                    style = MaterialTheme.typography.bodySmall,
                    color =
MaterialTheme.colorScheme.onPrimaryContainer.copy(alpha = 0.7f)
                )
                Spacer(modifier = Modifier.height(4.dp))
                Text(
                    text = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()).format(tx.date),
                    style = MaterialTheme.typography.labelSmall,
                    color =
MaterialTheme.colorScheme.onPrimaryContainer.copy(alpha = 0.6f)
                )
            }
            Column(horizontalAlignment = Alignment.End) {
```

```
                Text(
                    text = (if (tx.type ==
TransactionType.INCOME) "+" else "-") +
"%.2f".format(tx.amount),
                    style = MaterialTheme.typography.titleLarge,
                    color = if (tx.type ==
TransactionType.INCOME) MaterialTheme.colorScheme.primary else
MaterialTheme.colorScheme.error
                )
                Row {
                    IconButton(onClick = onEdit) {
                        Icon1(Icons.Default.Edit,
contentDescription = "Edit")
                    }
                    IconButton(onClick = onDelete) {
                        Icon1(Icons.Default.Delete,
contentDescription = "Delete")
                    }
                }
            }
        }
    }
}

// Форма для добавления/редактирования транзакції
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun TransactionSheetForm(
    onSubmit: (Transaction) -> Unit,
    onDismiss: () -> Unit,
    categories: List<Category>,
    initial: Transaction? = null,
    userId: String
) {
    var amount by remember {
mutableStateOf(initial?.amount?.toString() ?: "") }
    var type by remember {
mutableStateOf(initial?.type?.name?.lowercase() ?: "income") }
    var selectedCategory by remember {
mutableStateOf<Category?>(null) }
    var selectedAccount by remember {
mutableStateOf<Account?>(null) }
    var description by remember {
mutableStateOf(initial?.description ?: "") }
    val date = remember { mutableStateOf(initial?.date ?:
Date()) }
    val accountsViewModel: AccountsViewModel = hiltViewModel()
    val allAccounts by
accountsViewModel.accounts.collectAsState()
    val filteredCategories = categories.filter {
        (type == "income" && it.type.name.lowercase() ==
"income") ||
```

```
        (type == "expense" && it.type.name.lowercase() ==
"expense")
    }
    if (selectedCategory != null && filteredCategories.none {
it.id == selectedCategory?.id }) {
        selectedCategory = null
    }
    if (selectedAccount != null && allAccounts.none { it.id ==
selectedAccount?.id }) {
        selectedAccount = null
    }
    LaunchedEffect(allAccounts, filteredCategories, initial) {
        if (initial == null) {
            if (selectedAccount == null &&
allAccounts.isNotEmpty()) {
                selectedAccount = allAccounts.first()
            }
            if (selectedCategory == null &&
filteredCategories.isNotEmpty()) {
                selectedCategory = filteredCategories.first()
            }
        } else {
            if (selectedAccount == null &&
allAccounts.isNotEmpty()) {
                selectedAccount = allAccounts.find { it.id ==
initial.accountId }
            }
            if (selectedCategory == null &&
filteredCategories.isNotEmpty()) {
                selectedCategory = filteredCategories.find {
it.id == initial.categoryId }
            }
        }
    }
    val isEdit = initial != null
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp)
    ) {
        Text(
            text = if (!isEdit) stringResource(id =
R.string.add_transaction) else stringResource(id =
R.string.edit_transaction),
            style = MaterialTheme.typography.titleLarge,
            modifier = Modifier.padding(bottom = 16.dp)
        )
        OutlinedTextField(
            value = amount,
            onValueChange = { newValue ->
                if (newValue.matches(Regex("^\\d*\\.?\\d*"))) {
                    amount = newValue
```

```
                }
            },
            label = { Text(stringResource(id = R.string.amount))
},
            singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
            modifier = Modifier.fillMaxWidth()
        )
        val dateFormat = remember {
SimpleDateFormat("dd.MM.yyyy", Locale.getDefault()) }
        var showDatePicker by remember { mutableStateOf(false) }
        OutlinedTextField(
            value = dateFormat.format(date.value),
            onValueChange = {},
            label = { Text(stringResource(id = R.string.date))
},
            singleLine = true,
            modifier = Modifier.fillMaxWidth(),
            readOnly = true,
            trailingIcon = {
                IconButton(onClick = { showDatePicker = true })
{
                    Icon1(Icons.Default.Edit, contentDescription
= stringResource(id = R.string.select_date))
                }
            }
        )
        OutlinedTextField(
            value = description,
            onValueChange = { description = it },
            label = { Text(stringResource(id =
R.string.description)) },
            singleLine = true,
            modifier = Modifier.fillMaxWidth()
        )
        Row(
            modifier = Modifier.fillMaxWidth().padding(vertical
= 8.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Text("Тип: ", modifier = Modifier.weight(1f))
            RadioButton(
                selected = type == "income",
                onClick = { if (!isEdit) { type = "income";
selectedCategory = null } },
                enabled = !isEdit
            )
            Text(stringResource(id = R.string.income))
            Spacer(modifier = Modifier.width(8.dp))
            RadioButton(
                selected = type == "expense",
```

```
                onClick = { if (!isEdit) { type = "expense";
selectedCategory = null } },
                enabled = !isEdit
            )
            Text(stringResource(id = R.string.expense))
        }
        var accountMenuExpanded by remember {
mutableStateOf(false) }
        ExposedDropdownMenuBox(
            expanded = accountMenuExpanded,
            onExpandedChange = { accountMenuExpanded = it }
        ) {
            OutlinedTextField(
                value = selectedAccount?.name ?: "",
                onValueChange = {},
                label = { Text(stringResource(id =
R.string.account)) },
                modifier = Modifier
                    .fillMaxWidth()
                    .menuAnchor(),
                readOnly = true,
                trailingIcon = {

ExposedDropdownMenuDefaults.TrailingIcon(expanded =
accountMenuExpanded)
                }
            )
            ExposedDropdownMenu(
                expanded = accountMenuExpanded,
                onDismissRequest = { accountMenuExpanded = false
}
            ) {
                allAccounts.forEach { account ->
                    DropdownMenuItem(
                        text = { Text(account.name) },
                        onClick = {
                            selectedAccount = account
                            accountMenuExpanded = false
                        }
                    )
                }
            }
        }
        var categoryMenuExpanded by remember {
mutableStateOf(false) }
        ExposedDropdownMenuBox(
            expanded = categoryMenuExpanded,
            onExpandedChange = { categoryMenuExpanded = it }
        ) {
            OutlinedTextField(
                value = selectedCategory?.name ?: "",
                onValueChange = {},
```

```
                label = { Text(stringResource(id =
R.string.category)) },
                modifier = Modifier
                    .fillMaxWidth()
                    .menuAnchor(),
                readOnly = true,
                trailingIcon = {

ExposedDropdownMenuDefaults.TrailingIcon(expanded =
categoryMenuExpanded)
                }
            )
            ExposedDropdownMenu(
                expanded = categoryMenuExpanded,
                onDismissRequest = { categoryMenuExpanded =
false }
            ) {
                filteredCategories.forEach { category ->
                    DropdownMenuItem(
                        text = { Text(category.name) },
                        onClick = {
                            selectedCategory = category
                            categoryMenuExpanded = false
                        }
                    )
                }
            }
        }
        if (showDatePicker) {
            AndroidDatePickerDialog(
                initialDate = date.value,
                onDateSelected = {
                    date.value = it
                    showDatePicker = false
                },
                onDismiss = { showDatePicker = false }
            )
        }
        Spacer(modifier = Modifier.height(16.dp))
        Button(
            onClick = {
                val userIdToSave = initial?.userId?.takeIf {
it.isNotBlank() } ?: userId
                val tx = Transaction(
                    amount = amount.toDoubleOrNull() ?: 0.0,
                    type = if (type == "income")
TransactionType.INCOME else TransactionType.EXPENSE,
                    categoryId = selectedCategory?.id ?: "",
                    categoryName = selectedCategory?.name ?: "",
                    date = date.value,
                    description = description,
                    id = initial?.id ?:
```

```
UUID.randomUUID().toString(),
                    accountId = selectedAccount?.id ?: "",
                    accountName = selectedAccount?.name ?: "",
                    userId = userIdToSave
                )
                onSubmit(tx)
            },
            modifier = Modifier.fillMaxWidth(),
            enabled = amount.isNotBlank() &&
amount.toDoubleOrNull() != null && amount.toDoubleOrNull()!! > 0
&&
                    selectedCategory != null && selectedAccount
!= null
        ) {
            Text(
                if (initial == null) stringResource(id =
R.string.add) else stringResource(id = R.string.save)
            )
        }
        Spacer(modifier = Modifier.height(8.dp))
        TextButton(
            onClick = onDismiss,
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(stringResource(id = R.string.cancel))
        }
    }
}

// Компонент для выпадающего меню з можливістю вибору
@Composable
fun DropdownMenuBox(label: String, options: List<String>,
selected: String, onSelect: (String) -> Unit) {
    var expanded by remember { mutableStateOf(false) }
    Box {
        OutlinedButton(onClick = { expanded = true }) {
            Text(selected.ifBlank { label })
        }
        DropdownMenu(expanded = expanded, onDismissRequest = {
expanded = false }) {
            options.forEach { option ->
                DropdownMenuItem(
                    text = { Text(option.ifBlank {
stringResource(id = R.string.all) }) },
                    onClick = {
                        onSelect(option)
                        expanded = false
                    }
                )
            }
        }
    }
```

```
}

// Диалог выбора даты
@Composable
fun AndroidDatePickerDialog(
    initialDate: Date,
    onDateSelected: (Date) -> Unit,
    onDismiss: () -> Unit
) {
    val context = LocalContext.current
    val calendar = Calendar.getInstance().apply { time =
initialDate }
    DisposableEffect(Unit) {
        val picker = android.app.DatePickerDialog(
            context,
            { _, year, month, dayOfMonth ->
                val cal = Calendar.getInstance()
                cal.set(year, month, dayOfMonth)
                onDateSelected(cal.time)
            },
            calendar.get(Calendar.YEAR),
            calendar.get(Calendar.MONTH),
            calendar.get(Calendar.DAY_OF_MONTH)
        )
        picker.setOnCancelListener { onDismiss() }
        picker.show()
        onDispose { picker.dismiss() }
    }
}

// Фильтр для транзакцій
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun FilterSheet(
    filterType: String,
    filterAccount: String,
    filterCategory: String,
    filterDateFrom: Date?,
    filterDateTo: Date?,
    accounts: List<Account>,
    categoriesList: List<String>,
    onApply: (String, String, String, Date?, Date?) -> Unit,
    onReset: () -> Unit,
    onDismiss: () -> Unit
) {
    var type by remember { mutableStateOf(filterType) }
    var account by remember { mutableStateOf(filterAccount) }
    var category by remember { mutableStateOf(filterCategory) }
    var dateFrom by remember { mutableStateOf(filterDateFrom) }
    var dateTo by remember { mutableStateOf(filterDateTo) }
    val dateFormat = remember { SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()) }
```

```
    var showDateRangePicker by remember { mutableStateOf(false)
}
    val dateRangePickerState = rememberDateRangePickerState(
        initialSelectedStartDateMillis = dateFrom?.time,
        initialSelectedEndDateMillis = dateTo?.time
    )
    Column(Modifier.padding(16.dp)) {
        Text(
            text = stringResource(id = R.string.filter),
            style = MaterialTheme.typography.titleLarge
        )
        Spacer(Modifier.height(12.dp))
        OutlinedButton(onClick = { showDateRangePicker = true },
modifier = Modifier.fillMaxWidth()) {
            val fromStr = dateFrom?.let { dateFormat.format(it)
} ?: ""
            val toStr = dateTo?.let { dateFormat.format(it) } ?:
""
            val rangeText = if (fromStr.isNotBlank() &&
toStr.isNotBlank()) {
                val text = stringResource(id =
R.string.range_dates, fromStr, toStr)
                if (text.length > 32) text.take(14) + "..." +
text.takeLast(14) else text
            } else {
                stringResource(id = R.string.select_range_dates)
            }
            Text(rangeText, maxLines = 1, overflow =
androidx.compose.ui.text.style.TextOverflow.Ellipsis)
        }
        if (showDateRangePicker) {
            AlertDialog(
                onDismissRequest = { showDateRangePicker = false
},
                text = {
                    DateRangePicker(
                        state = dateRangePickerState
                    )
                },
                confirmButton = {
                    TextButton(onClick = {
                        val start =
dateRangePickerState.selectedStartDateMillis?.let { Date(it) }
                        val end =
dateRangePickerState.selectedEndDateMillis?.let { Date(it) }
                        dateFrom = start
                        dateTo = end
                        showDateRangePicker = false
                    }) { Text(stringResource(id = R.string.ok))
}
                },
                dismissButton = {
```

```
                    TextButton(onClick = { showDateRangePicker =
false }) { Text(stringResource(id = R.string.cancel)) }
                }
            )
        }
        Spacer(Modifier.height(12.dp))
        Row(verticalAlignment = Alignment.CenterVertically) {
            FilterChip(selected = type == "all", onClick = {
type = "all" }, label = { Text(stringResource(id =
R.string.all)) })
            Spacer(Modifier.width(8.dp))
            FilterChip(selected = type == "income", onClick = {
type = "income" }, label = { Text(stringResource(id =
R.string.income_plural)) })
            Spacer(Modifier.width(8.dp))
            FilterChip(selected = type == "expense", onClick = {
type = "expense" }, label = { Text(stringResource(id =
R.string.expense_plural)) })
        }
        Spacer(Modifier.height(12.dp))
        DropdownMenuBox(
            label = stringResource(id = R.string.account),
            options = listOf("") + accounts.map { it.name },
            selected = account,
            onSelect = { account = it }
        )
        Spacer(Modifier.height(8.dp))
        DropdownMenuBox(
            label = stringResource(id = R.string.category),
            options = listOf("") + categoriesList,
            selected = category,
            onSelect = { category = it }
        )
        Spacer(Modifier.height(16.dp))
        Row(Modifier.fillMaxWidth(), horizontalArrangement =
Arrangement.SpaceBetween) {
            TextButton(onClick = onReset) {
Text(stringResource(id = R.string.reset)) }
            Spacer(Modifier.width(8.dp))
            Button(onClick = { onApply(type, account, category,
dateFrom, dateTo) }) { Text(stringResource(id = R.string.apply))
}
        }
        Spacer(Modifier.height(8.dp))
        TextButton(onClick = onDismiss, modifier =
Modifier.fillMaxWidth()) { Text(stringResource(id =
R.string.cancel)) }
    }
}
```

SettingsScreen.kt

```
package com.example.finflow.presentation.screens.settings

import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import com.example.finflow.R
import androidx.compose.ui.unit.dp
import androidx.compose.ui.platform.LocalContext
import com.example.finflow.LocalAppLocale
import java.util.Locale

// Екран налаштувань користувача
@Composable
fun SettingsScreen(
    isDarkTheme: Boolean = false,
    onToggleTheme: () -> Unit = {},
    selectedLanguage: String = "uk",
    onLanguageChange: (String) -> Unit = {},
    userEmail: String? = null,
    onLogout: () -> Unit = {}
) {
    val languages = listOf(
        "en" to "GB English",
        "uk" to "UA Українська",
        "ru" to "RU Русский"
    )
    var expanded by remember { mutableStateOf(false) }
    var showLogoutDialog by remember { mutableStateOf(false) }

    val locale = LocalAppLocale.current
    val context = LocalContext.current
    val configuration = context.resources.configuration
    configuration.setLocale(locale)
    val localizedContext =
context.createConfigurationContext(configuration)

    Column(
        modifier = Modifier
            .fillMaxSize()
            .padding(24.dp),
        verticalArrangement = Arrangement.spacedBy(24.dp,
Alignment.Top),
    ) {
        Text(text =
localizedContext.getString(R.string.go_to_settings), style =
MaterialTheme.typography.headlineSmall)
        userEmail?.let {
            Text(text = it, style =
```

```
MaterialTheme.typography.bodyLarge)
        }
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween,
            modifier = Modifier.fillMaxWidth()
        ) {

Text(localizedContext.getString(R.string.dark_theme))
            Switch(checked = isDarkTheme, onCheckedChange = {
onToggleTheme() })
        }
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween,
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(localizedContext.getString(R.string.language))
            Box {
                Button(onClick = { expanded = true }) {
                    Text(languages.first { it.first ==
selectedLanguage }.second)
                }
                DropdownMenu(expanded = expanded,
onDismissRequest = { expanded = false }) {
                    languages.forEach { (code, label) ->
                        DropdownMenuItem(
                            text = { Text(label) },
                            onClick = {
                                onLanguageChange(code)
                                expanded = false
                            }
                        )
                    }
                }
            }
        }
        Spacer(modifier = Modifier.height(24.dp))
        Button(
            onClick = { showLogoutDialog = true },
            colors = ButtonDefaults.buttonColors(containerColor
= MaterialTheme.colorScheme.error)
        ) {
            Text(localizedContext.getString(R.string.logout))
        }
    }
    if (showLogoutDialog) {
        AlertDialog(
            onDismissRequest = { showLogoutDialog = false },
            title = {
Text(localizedContext.getString(R.string.confirmation)) },
            text = {
```

```
Text(localizedContext.getString(R.string.logout_confirm)) },
            confirmButton = {
                TextButton(onClick = {
                    showLogoutDialog = false
                    onLogout()
                }) {
Text(localizedContext.getString(R.string.yes)) }
            },
            dismissButton = {
                TextButton(onClick = { showLogoutDialog = false
}) { Text(localizedContext.getString(R.string.cancel)) }
            }
        )
    }
}
```

DashboardViewModel.kt

```
package com.example.finflow.presentation.screens.dashboard

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import javax.inject.Inject
import
com.example.finflow.domain.repository.TransactionRepository
import com.example.finflow.domain.models.TransactionType

// ViewModel для управління інформацією на головній панелі
@HiltViewModel
class DashboardViewModel @Inject constructor(
    private val auth: FirebaseAuth,
    private val firestore: FirebaseFirestore,
    private val transactionRepository: TransactionRepository
) : ViewModel() {
    // Стан головної панелі
    data class DashboardState(
        val balance: Double = 0.0,
        val income: Double = 0.0,
        val expense: Double = 0.0,
        val expensesByCategory: Map<String, Double> =
emptyMap(),
        val recentTransactions:
List<com.example.finflow.domain.models.Transaction> =
emptyList(),
        val isLoading: Boolean = true,
        val error: String? = null
    )
```

```
    private val _state = MutableStateFlow(DashboardState())
    val state = _state

    init {
        subscribeToTransactions()
    }

    // Підписка на зміни транзакцій користувача
    private fun subscribeToTransactions() {
        val user = auth.currentUser ?: return
        viewModelScope.launch {

transactionRepository.getTransactions(user.uid).collectLatest {
transactions ->
                var income = 0.0
                var expense = 0.0
                val expensesByCategory = mutableMapOf<String,
Double>()
                for (tx in transactions) {
                    if (tx.type == TransactionType.INCOME) {
                        income += tx.amount
                    } else {
                        expense += tx.amount
                        expensesByCategory[tx.categoryName] =
(expensesByCategory[tx.categoryName] ?: 0.0) + tx.amount
                    }
                }
                val balance = income - expense
                _state.value = _state.value.copy(
                    balance = balance,
                    income = income,
                    expense = expense,
                    expensesByCategory = expensesByCategory,
                    recentTransactions =
transactions.sortedByDescending { it.date }.take(5),
                    isLoading = false,
                    error = null
                )
            }
        }
    }
}
```

DashboardScreen.kt

```
package com.example.finflow.presentation.screens.dashboard

import
androidx.compose.foundation.interaction.MutableInteractionSource
import androidx.compose.foundation.layout.*
import androidx.compose.material3.*
```

```
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.setValue
import androidx.compose.runtime.collectAsState
import androidx.compose.runtime.remember
import androidx.compose.runtime.mutableStateOf
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.graphics.Color
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.dp
import com.example.finflow.R
import androidx.hilt.navigation.compose.hiltViewModel
import
com.example.finflow.presentation.screens.accounts.AccountsViewMo
del
import
com.example.finflow.presentation.screens.categories.CategoriesVi
ewModel
import
com.example.finflow.presentation.screens.transactions.Transactio
nsViewModel
import com.example.finflow.domain.models.TransactionType
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.Canvas
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Info
import androidx.compose.material3.DatePickerDialog
import java.text.SimpleDateFormat
import java.util.Calendar
import java.util.Date
import java.util.Locale
import androidx.compose.runtime.LaunchedEffect
import com.google.firebase.auth.FirebaseAuth
import androidx.compose.foundation.verticalScroll
import androidx.compose.foundation.rememberScrollState
import androidx.compose.ui.graphics.Brush
import androidx.compose.ui.graphics.StrokeCap
import androidx.compose.ui.graphics.drawscope.Stroke
import androidx.compose.ui.graphics.drawscope.Fill
import androidx.compose.ui.graphics.drawscope.DrawScope
import androidx.compose.ui.geometry.Offset
import androidx.compose.foundation.background
import kotlin.math.roundToInt

@OptIn(ExperimentalMaterial3Api::class)
// Єкран для відображення інформації про фінансовий стан
```

```
користувача
@Composable
fun DashboardScreen(
    accountsViewModel: AccountsViewModel = hiltViewModel(),
    categoriesViewModel: CategoriesViewModel = hiltViewModel(),
    transactionsViewModel: TransactionsViewModel =
hiltViewModel(),
    onAddTransactionClick: (accountId: String) -> Unit = {}
) {
    val accounts by accountsViewModel.accounts.collectAsState()
    val transactions by
transactionsViewModel.transactions.collectAsState()
    val categories by
categoriesViewModel.categories.collectAsState()
    val userId = FirebaseAuth.getInstance().currentUser?.uid
    LaunchedEffect(userId) {
        userId?.let {
            accountsViewModel.loadAccounts(it)
            transactionsViewModel.setUserId(it)
        }
    }
    var selectedAccountIndex by remember { mutableStateOf(0) }
    val accountNames = accounts.map { it.name }
    val hasAccounts = accounts.isNotEmpty()
    var expanded by remember { mutableStateOf(false) }
    var showDateRangePicker by remember { mutableStateOf(false)
}
    var startDate by remember { mutableStateOf<Long?>(null) }
    var endDate by remember { mutableStateOf<Long?>(null) }
    val dateFormat = remember { SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault()) }
    val selectedAccount =
accounts.getOrNull(selectedAccountIndex)
    val filteredTransactions = transactions.filter { tx ->
        (selectedAccount == null || tx.accountId ==
selectedAccount.id) &&
        (startDate == null || tx.date.time >= startDate!!) &&
        (endDate == null || tx.date.time <= endDate!!)
    }
    val incomeByCategory = filteredTransactions.filter { it.type
== TransactionType.INCOME }
        .groupBy { it.categoryName ?: "Без категории" }
        .mapValues { it.value.sumOf { t -> t.amount } }
    val expenseByCategory = filteredTransactions.filter {
it.type == TransactionType.EXPENSE }
        .groupBy { it.categoryName ?: "Без категории" }
        .mapValues { it.value.sumOf { t -> t.amount } }
    LazyColumn(modifier = Modifier.fillMaxSize()) {
        item {
            Box(modifier = Modifier.fillMaxWidth(),
contentAlignment = Alignment.Center) {
                ExposedDropdownMenuBox(
```

```
                    expanded = expanded,
                    onExpandedChange = { if (hasAccounts)
expanded = !expanded }
                ) {
                    TextField(
                        value =
accountNames.getOrNull(selectedAccountIndex) ?:
stringResource(R.string.select_account),
                        onValueChange = {},
                        readOnly = true,
                        label = {
Text(stringResource(R.string.account_label)) },
                        trailingIcon = {
ExposedDropdownMenuDefaults.TrailingIcon(expanded = expanded) },
                        modifier =
Modifier.menuAnchor().fillMaxWidth(0.7f),
                        enabled = hasAccounts,
                        interactionSource = remember {
MutableInteractionSource() },
                        singleLine = true
                    )
                    ExposedDropdownMenu(
                        expanded = expanded,
                        onDismissRequest = { expanded = false }
                    ) {
                        accountNames.forEachIndexed { index,
name ->
                            DropdownMenuItem(
                                text = { Text(name) },
                                onClick = {
                                    selectedAccountIndex = index
                                    expanded = false
                                },
                                enabled = hasAccounts
                            )
                        }
                    }
                }
            }
            Spacer(modifier = Modifier.height(8.dp))
        }
        item {
            Button(onClick = { showDateRangePicker = true },
modifier = Modifier.fillMaxWidth()) {
                val from = startDate?.let {
dateFormat.format(it) } ?: stringResource(R.string.from_date)
                val to = endDate?.let { dateFormat.format(it) }
?: stringResource(R.string.to_date)
                Text(stringResource(R.string.period, from, to))
            }
            if (showDateRangePicker) {
                val dateRangePickerState =
```

```
rememberDateRangePickerState(
                    initialSelectedStartDateMillis = startDate,
                    initialSelectedEndDateMillis = endDate
                )
                AlertDialog(
                    onDismissRequest = { showDateRangePicker =
false },
                    text = {
                        DateRangePicker(state =
dateRangePickerState)
                    },
                    confirmButton = {
                        TextButton(onClick = {
                            startDate =
dateRangePickerState.selectedStartDateMillis
                            endDate =
dateRangePickerState.selectedEndDateMillis
                            showDateRangePicker = false
                        }) { Text(stringResource(R.string.ok)) }
                    },
                    dismissButton = {
                        TextButton(onClick = {
showDateRangePicker = false }) {
Text(stringResource(R.string.cancel)) }
                    }
                )
            }
            Spacer(modifier = Modifier.height(16.dp))
        }
        item {

Text(stringResource(R.string.income_expense_dynamics), style =
MaterialTheme.typography.titleMedium)
        }
        item {
            LineChartIncomeExpense(transactions =
filteredTransactions, modifier =
Modifier.height(200.dp).fillMaxWidth())
        }
        item {
            Spacer(modifier = Modifier.height(24.dp))
            Text(stringResource(R.string.income_by_category),
style = MaterialTheme.typography.titleMedium)
        }
        item {
            PieChartCategory(
                data = incomeByCategory,
                modifier =
Modifier.height(180.dp).fillMaxWidth(),
                total = filteredTransactions.filter { it.type ==
TransactionType.INCOME }.sumOf { it.amount },
                colorKeyTransform = { it.trim().lowercase() }
```

```
            )
        }
        item {
            Spacer(modifier = Modifier.height(24.dp))
            Text(stringResource(R.string.piechart_title), style
= MaterialTheme.typography.titleMedium)
        }
        item {
            PieChartCategory(
                data = expenseByCategory,
                modifier =
Modifier.height(180.dp).fillMaxWidth(),
                total = filteredTransactions.filter { it.type ==
TransactionType.EXPENSE }.sumOf { it.amount },
                colorKeyTransform = { it.trim().lowercase() }
            )
        }
    }
}

// Генерує кольори для діаграми на основі кількості категорій
fun generateChartColors(count: Int): List<Color> {
    val palette = listOf(
        Color(0xFFEF5350), Color(0xFFAB47BC), Color(0xFF5C6BC0),
Color(0xFF29B6F6),
        Color(0xFF66BB6A), Color(0xFFFFCA28), Color(0xFFFF7043),
Color(0xFF8D6E63),
        Color(0xFF26A69A), Color(0xFFD4E157), Color(0xFFEC407A),
Color(0xFF7E57C2),
        Color(0xFF42A5F5), Color(0xFF26C6DA), Color(0xFF9CCC65),
Color(0xFFFFEE58)
    )
    return List(count) { palette[it % palette.size] }
}

// Компонент для відображення кругової діаграми з категоріями
@Composable
fun PieChartCategory(
    data: Map<String, Double>,
    modifier: Modifier = Modifier,
    total: Double = 0.0,
    colorKeyTransform: (String) -> String = { it }
) {
    val sum = data.values.sum()
    if (sum == 0.0) {
        Box(modifier, contentAlignment = Alignment.Center) {
            Text(stringResource(R.string.no_data), color =
Color.Gray, textAlign = TextAlign.Center)
        }
        return
    }
    val chartColors = remember(data) {
```

```
generateChartColors(data.size) }
    val colorMap = data.keys.mapIndexed { i, k ->
colorKeyTransform(k) to chartColors[i] }.toMap()
    Box(
        modifier = modifier,
        contentAlignment = Alignment.Center
    ) {
        val size = remember(modifier) { 180.dp }
        Canvas(modifier =
Modifier.size(size).align(Alignment.Center)) {
            var startAngle = -90f
            data.entries.forEach { (cat, value) ->
                val colorKey = colorKeyTransform(cat)
                val color = colorMap[colorKey] ?:
Color.LightGray
                val sweep = (value / sum * 360f).toFloat()
                drawArc(
                    color = color,
                    startAngle = startAngle,
                    sweepAngle = sweep,
                    useCenter = true
                )
                startAngle += sweep
            }
        }
        Column(modifier =
Modifier.align(Alignment.CenterEnd).padding(start = 16.dp)) {
            data.entries.forEach { (cat, value) ->
                val percent = if (sum > 0) (value / sum *
100).toInt() else 0
                val colorKey = colorKeyTransform(cat)
                val color = colorMap[colorKey] ?:
Color.LightGray
                Row(verticalAlignment =
Alignment.CenterVertically) {
                    Box(Modifier.size(12.dp).background(color))
                    Spacer(Modifier.width(4.dp))
                    Text("$cat: %.2f ₽
($percent%%)".format(value), style =
MaterialTheme.typography.bodySmall)
                }
            }
        }
    }
}

// Компонент для відображення лінійного графіка доходів та
витрат
@Composable
fun LineChartIncomeExpense(transactions:
List<com.example.finflow.domain.models.Transaction>, modifier:
Modifier = Modifier) {
```

```
    val dateFormat = SimpleDateFormat("dd.MM",
Locale.getDefault())
    val grouped = transactions.groupBy {
dateFormat.format(it.date) }
    val days = grouped.keys.sorted()
    val incomePoints = days.map { day -> grouped[day]?.filter {
it.type == TransactionType.INCOME }?.sumOf { it.amount } ?: 0.0
}
    val expensePoints = days.map { day -> grouped[day]?.filter {
it.type == TransactionType.EXPENSE }?.sumOf { it.amount } ?: 0.0
}
    val maxY = (incomePoints +
expensePoints).maxOrNull()?.coerceAtLeast(1.0) ?: 1.0
    if (days.isEmpty()) {
        Box(modifier, contentAlignment = Alignment.Center) {
            Text(stringResource(R.string.no_data), color =
Color.Gray, textAlign = TextAlign.Center)
        }
        return
    }
    Canvas(modifier = modifier) {
        val w = size.width
        val h = size.height
        val stepX = if (days.size > 1) w / (days.size - 1) else
w
        for (i in 1 until days.size) {
            val x0 = stepX * (i - 1)
            val y0 = h - (incomePoints[i - 1] / maxY *
h).toFloat()
            val x1 = stepX * i
            val y1 = h - (incomePoints[i] / maxY * h).toFloat()
            drawLine(Color(0xFF388E3C), Offset(x0, y0),
Offset(x1, y1), strokeWidth = 4f, cap = StrokeCap.Round)
        }
        for (i in 1 until days.size) {
            val x0 = stepX * (i - 1)
            val y0 = h - (expensePoints[i - 1] / maxY *
h).toFloat()
            val x1 = stepX * i
            val y1 = h - (expensePoints[i] / maxY * h).toFloat()
            drawLine(Color(0xFFD32F2F), Offset(x0, y0),
Offset(x1, y1), strokeWidth = 4f, cap = StrokeCap.Round)
        }
    }
    Row(Modifier.fillMaxWidth(), horizontalArrangement =
Arrangement.SpaceBetween) {
        days.forEach { day -> Text(day, style =
MaterialTheme.typography.labelSmall) }
    }
}
```

CategoriesViewModel.kt

```
package com.example.finflow.presentation.screens.categories

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.finflow.domain.models.Category
import com.example.finflow.domain.usecases.AddCategoryUseCase
import com.example.finflow.domain.usecases.UpdateCategoryUseCase
import com.example.finflow.domain.usecases.DeleteCategoryUseCase
import com.example.finflow.domain.repository.CategoryRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import javax.inject.Inject

// ViewModel для управління категоріями
@HiltViewModel
class CategoriesViewModel @Inject constructor(
    private val addCategoryUseCase: AddCategoryUseCase,
    private val updateCategoryUseCase: UpdateCategoryUseCase,
    private val deleteCategoryUseCase: DeleteCategoryUseCase,
    private val categoryRepository: CategoryRepository
) : ViewModel() {
    private val _categories =
MutableStateFlow<List<Category>>(emptyList())
    val categories: StateFlow<List<Category>> = _categories

    // Додає нову категорію
    fun addCategory(userId: String, category: Category) {
        viewModelScope.launch {
            addCategoryUseCase(userId, category)
        }
    }

    // Оновлює існуючу категорію
    fun updateCategory(userId: String, category: Category) {
        viewModelScope.launch {
            updateCategoryUseCase(userId, category)
        }
    }

    // Видаляє категорію
    fun deleteCategory(userId: String, categoryId: String) {
        viewModelScope.launch {
            deleteCategoryUseCase(userId, categoryId)
        }
    }

    // Завантажує категорії для користувача
    fun setUserId(userId: String) {
```

```
        viewModelScope.launch {

categoryRepository.getCategories(userId).collectLatest {
                _categories.value = it
            }
        }
    }
}
```

CategoriesScreen.kt

```
package com.example.finflow.presentation.screens.categories

import android.annotation.SuppressLint
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Edit
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.unit.dp
import com.example.finflow.R
import com.example.finflow.domain.models.Category
import com.example.finflow.domain.models.CategoryType
import androidx.hilt.navigation.compose.hiltViewModel
import com.google.firebase.auth.FirebaseAuth

// Екран для управління категоріями
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun CategoriesScreen(
    viewModel: CategoriesViewModel = hiltViewModel()
) {
    val categories by viewModel.categories.collectAsState()
    var showAddSheet by remember { mutableStateOf(false) }
    var editCat by remember { mutableStateOf<Category?>(null) }
    val userId = FirebaseAuth.getInstance().currentUser?.uid
    LaunchedEffect(userId) {
        if (!userId.isNullOrBlank()) {
            viewModel.setUserId(userId)
        }
    }
    if (userId.isNullOrBlank()) {
        Box(modifier = Modifier.fillMaxSize(), contentAlignment
```

```
= Alignment.Center) {
            Text(text = stringResource(id =
R.string.error_user_not_authorized))
        }
        return
    }
    val sheetState = rememberModalBottomSheetState()
    var filterType by remember {
mutableStateOf<CategoryType?>(null) }
    var searchQuery by remember { mutableStateOf("") }
    val sortBy by remember { mutableStateOf("name_asc") }
    val filtered = categories
        .filter { filterType == null || it.type == filterType }
        .filter { searchQuery.isBlank() ||
it.name.contains(searchQuery, true) ||
it.icon.contains(searchQuery, true) }
        .let {
            when (sortBy) {
                "name_asc" -> it.sortedBy { c -> c.name }
                "name_desc" -> it.sortedByDescending { c ->
c.name }
                else -> it
            }
        }
    val snackbarHostState = remember { SnackbarHostState() }
    var snackbarMessage by remember {
mutableStateOf<String?>(null) }
    val categoryDeletedMsg = stringResource(id =
R.string.category_deleted)
    val categoryAddedMsg = stringResource(id =
R.string.category_added)
    val categoryUpdatedMsg = stringResource(id =
R.string.category_updated)
    Scaffold(
        snackbarHost = { SnackbarHost(snackbarHostState) },
        floatingActionButton = {
            FloatingActionButton(onClick = { showAddSheet = true
}) {
                Icon(Icons.Default.Add, contentDescription =
stringResource(id = R.string.add_category))
            }
        }
    ) {
        Column(modifier = Modifier.fillMaxSize()) {
            Row(verticalAlignment = Alignment.CenterVertically,
modifier = Modifier.padding(8.dp)) {
                Text(text = stringResource(id = R.string.type),
modifier = Modifier.padding(end = 8.dp))
                FilterChip(
                    selected = filterType == null,
                    onClick = { filterType = null },
                    label = { Text(stringResource(id =
```

```
R.string.all)) }
                )
                Spacer(modifier = Modifier.width(4.dp))
                FilterChip(
                    selected = filterType ==
CategoryType.INCOME,
                    onClick = { filterType = CategoryType.INCOME
},
                    label = { Text(stringResource(id =
R.string.income)) }
                )
                Spacer(modifier = Modifier.width(4.dp))
                FilterChip(
                    selected = filterType ==
CategoryType.EXPENSE,
                    onClick = { filterType =
CategoryType.EXPENSE },
                    label = { Text(stringResource(id =
R.string.expense)) }
                )
                Spacer(modifier = Modifier.width(16.dp))
                OutlinedTextField(
                    value = searchQuery,
                    onValueChange = { searchQuery = it },
                    label = { Text(stringResource(id =
R.string.search)) },
                    singleLine = true,
                    modifier = Modifier.weight(1f)
                )
            }
            if (filtered.isEmpty()) {
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) {
                    Text(text = stringResource(id =
R.string.no_categories), style =
MaterialTheme.typography.bodyLarge,
                        color =
MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f))
                }
            } else {
                LazyColumn(modifier = Modifier.fillMaxSize()) {
                    items(filtered, key = { cat -> cat.id }) {
cat ->
                        var showDeleteDialog by remember {
mutableStateOf(false) }
                        CategoryCardModern(
                            cat = cat,
                            onEdit = { editCat = cat },
                            onDelete = { showDeleteDialog = true
}
```

```
                        )
                        if (showDeleteDialog) {
                            AlertDialog(
                                onDismissRequest = {
showDeleteDialog = false },
                                title = { Text(stringResource(id
= R.string.delete_category_question)) },
                                text = { Text(stringResource(id
= R.string.delete_category_warning, cat.name)) },
                                confirmButton = {
                                    TextButton(onClick = {

viewModel.deleteCategory(userId, cat.id)
                                            showDeleteDialog = false
                                            snackbarMessage =
categoryDeletedMsg
                                    }) { Text(stringResource(id
= R.string.delete)) }
                                },
                                dismissButton = {
                                    TextButton(onClick = {
showDeleteDialog = false }) { Text(stringResource(id =
R.string.cancel)) }
                                }
                            )
                        }
                    }
                    item { Spacer(modifier =
Modifier.Companion.height(200.dp)) }
                }
            }
        }
        if (showAddSheet) {
            ModalBottomSheet(
                onDismissRequest = { showAddSheet = false },
                sheetState = sheetState
            ) {
                CategorySheetForm(
                    onSubmit = { cat ->
                        viewModel.addCategory(userId,
cat.copy(userId = userId))
                        showAddSheet = false
                        snackbarMessage = categoryAddedMsg
                    },
                    onDismiss = { showAddSheet = false },
                    userId = userId
                )
            }
        }
        if (editCat != null) {
            ModalBottomSheet(
                onDismissRequest = { editCat = null },
```

```
                sheetState = sheetState
            ) {
                CategorySheetForm(
                    initial = editCat,
                    onSubmit = { cat ->
                        viewModel.updateCategory(userId,
cat.copy(userId = userId))
                        editCat = null
                        snackbarMessage = categoryUpdatedMsg
                    },
                    onDismiss = { editCat = null },
                    userId = userId
                )
            }
        }
    }
    // Показываем Snackbar, если есть сообщение
    LaunchedEffect(snackbarMessage) {
        snackbarMessage?.let {
            snackbarHostState.showSnackbar(it)
            snackbarMessage = null
        }
    }
}

// Карточка категорії з кнопками редагування та видалення
@Composable
fun CategoryCardModern(cat: Category, onEdit: () -> Unit,
onDelete: () -> Unit) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
            .padding(8.dp),
        shape = RoundedCornerShape(16.dp),
        colors = CardDefaults.cardColors(
            containerColor =
MaterialTheme.colorScheme.surfaceVariant
        )
    ) {
        Row(
            modifier = Modifier.padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Box(
                modifier = Modifier
                    .size(40.dp)
                    .then(Modifier), // убрано .background и
цвет
                contentAlignment = Alignment.Center
            ) {
                Text(text = cat.icon, fontSize =
MaterialTheme.typography.headlineMedium.fontSize)
```

```
            }
            Spacer(modifier = Modifier.width(16.dp))
            Column(modifier = Modifier.weight(1f)) {
                Text(text = cat.name, style =
MaterialTheme.typography.titleMedium)
                Text(text = stringResource(id = if (cat.type ==
CategoryType.INCOME) R.string.income else R.string.expense),
style = MaterialTheme.typography.bodySmall)
            }
            IconButton(onClick = onEdit) {
                Icon(Icons.Default.Edit, contentDescription =
stringResource(id = R.string.edit))
            }
            IconButton(onClick = onDelete) {
                Icon(Icons.Default.Delete, contentDescription =
stringResource(id = R.string.delete))
            }
        }
    }
}

// Форма для добавления/редактирования категории
@Composable
fun CategorySheetForm(
    onSubmit: (Category) -> Unit,
    onDismiss: () -> Unit,
    initial: Category? = null,
    userId: String
) {
    var name by remember { mutableStateOf(initial?.name ?: "") }
    var icon by remember { mutableStateOf(initial?.icon ?: "🏷️")
}
    fun isSingleEmoji(str: String): Boolean {
        val emojiRegex = Regex("^(?:[\uD83C-\uDBFF\uDC00-
\uDFFF\u2600-\u27BF\uFE0F\u200D]+)$")
        if (str.isBlank() || !emojiRegex.matches(str)) return
false
        return str.codePointCount(0, str.length) == 1
    }
    val isIconValid = isSingleEmoji(icon)
    var type by remember { mutableStateOf(initial?.type ?:
CategoryType.EXPENSE) }
    val isEdit = initial != null
    val isNameValid = name.isNotBlank()
    val id = initial?.id ?: remember {
java.util.UUID.randomUUID().toString() }
    val isValid = isNameValid && isIconValid
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp)
```

```
    ) {
        Text(
            text = if (!isEdit) stringResource(id =
R.string.add_category) else stringResource(id =
R.string.edit_category),
            style = MaterialTheme.typography.titleLarge,
            modifier = Modifier.padding(bottom = 16.dp)
        )
        OutlinedTextField(
            value = name,
            onValueChange = { name = it },
            label = { Text(stringResource(id = R.string.name))
},
            singleLine = true,
            isError = !isNameValid,
            modifier = Modifier.fillMaxWidth()
        )
        OutlinedTextField(
            value = icon,
            onValueChange = { newValue ->
                if (newValue.isBlank() ||
isSingleEmoji(newValue)) {
                    icon = newValue
                }
            },
            label = { Text(stringResource(id = R.string.icon))
},
            singleLine = true,
            isError = !isIconValid,
            modifier = Modifier.fillMaxWidth(),
            supportingText = {
                if (!isIconValid) Text(stringResource(id =
R.string.only_one_emoji))
            }
        )
        if (!isEdit) {
            Row(verticalAlignment = Alignment.CenterVertically)
{
                Text(stringResource(id = R.string.type))
                Spacer(modifier = Modifier.width(8.dp))
                RadioButton(selected = type ==
CategoryType.INCOME, onClick = { type = CategoryType.INCOME })
                Text(stringResource(id = R.string.income))
                Spacer(modifier = Modifier.width(8.dp))
                RadioButton(selected = type ==
CategoryType.EXPENSE, onClick = { type = CategoryType.EXPENSE })
                Text(stringResource(id = R.string.expense))
            }
        } else {
            Row(verticalAlignment = Alignment.CenterVertically)
{
                Text(
```

```
                    text = stringResource(id = R.string.type) +
": " +
                            stringResource(id = if (type ==
CategoryType.INCOME) R.string.income else R.string.expense)
                )
            }
        }
        Spacer(modifier = Modifier.height(16.dp))
        Button(onClick = {
            onSubmit(Category(id = id, name = name, icon = icon,
userId = userId, type = type))
        }, enabled = isValid, modifier =
Modifier.fillMaxWidth()) {
            Text(stringResource(id = if (!isEdit) R.string.add
else R.string.save))
        }
        Spacer(modifier = Modifier.height(8.dp))
        TextButton(onClick = onDismiss, modifier =
Modifier.fillMaxWidth()) { Text(stringResource(id =
R.string.cancel)) }
    }
}
```

AuthViewModel.kt

```
package com.example.finflow.presentation.screens.auth

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.google.firebase.auth.FirebaseAuth
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

// ViewModel для управління аутентифікацією користувача
@HiltViewModel
class AuthViewModel @Inject constructor(
    private val firebaseAuth: FirebaseAuth
) : ViewModel() {
    private val _isLoading = MutableStateFlow(false)
    val isLoading: StateFlow<Boolean> = _isLoading

    private val _error = MutableStateFlow<String?>(null)
    val error: StateFlow<String?> = _error

    private val _isAuthenticated =
MutableStateFlow(firebaseAuth.currentUser != null)
    val isAuthenticated: StateFlow<Boolean> = _isAuthenticated

    fun signIn(email: String, password: String) {
```

```
        _isLoading.value = true
        _error.value = null
        viewModelScope.launch {
            firebaseAuth.signInWithEmailAndPassword(email,
password)
                .addOnCompleteListener { task ->
                    _isLoading.value = false
                    _isAuthenticated.value = task.isSuccessful
                    if (!task.isSuccessful) {
                        _error.value =
task.exception?.localizedMessage
                    }
                }
        }
    }

    fun signUp(email: String, password: String) {
        _isLoading.value = true
        _error.value = null
        viewModelScope.launch {
            firebaseAuth.createUserWithEmailAndPassword(email,
password)
                .addOnCompleteListener { task ->
                    _isLoading.value = false
                    _isAuthenticated.value = task.isSuccessful
                    if (!task.isSuccessful) {
                        _error.value =
task.exception?.localizedMessage
                    }
                }
        }
    }

    fun logout() {
        firebaseAuth.signOut()
        _isAuthenticated.value = false
    }
}
```

AuthScreen.kt

```
package com.example.finflow.presentation.screens.auth

import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
```

```
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.AccountCircle
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Tab
import androidx.compose.material3.TabRow
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableIntStateOf
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import
androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import com.example.finflow.R

// Екран для автентифікації користувача (вхід/реєстрація)
@Composable
fun AuthScreen(
    isLoading: Boolean = false,
    error: String? = null,
    onSignIn: (String, String) -> Unit = { _, _ -> },
    onSignUp: (String, String) -> Unit = { _, _ -> }
) {
    var selectedTab by remember { mutableIntStateOf(0) }
    val tabs = listOf(
        stringResource(id = R.string.sign_in),
        stringResource(id = R.string.sign_up)
    )
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var confirmPassword by remember { mutableStateOf("") }

    Box(modifier = Modifier.fillMaxSize(), contentAlignment =
Alignment.Center) {
        Card(modifier = Modifier.padding(16.dp)) {
            Column(
                modifier = Modifier.padding(24.dp),
                horizontalAlignment =
Alignment.CenterHorizontally
            ) {
                Icon(
                    imageVector = Icons.Default.AccountCircle,
```

```
                    contentDescription = null,
                    modifier = Modifier.size(64.dp)
                )
                Spacer(modifier = Modifier.height(16.dp))
                TabRow(selectedTabIndex = selectedTab) {
                    tabs.forEachIndexed { index: Int, title:
String ->
                        Tab(
                            selected = selectedTab == index,
                            onClick = { selectedTab = index },
                            text = { Text(title) }
                        )
                    }
                }
                Spacer(modifier = Modifier.height(16.dp))
                OutlinedTextField(
                    value = email,
                    onValueChange = { email = it },
                    label = { Text(stringResource(id =
R.string.email)) },
                    singleLine = true,
                    modifier = Modifier.fillMaxWidth()
                )
                Spacer(modifier = Modifier.height(8.dp))
                OutlinedTextField(
                    value = password,
                    onValueChange = { password = it },
                    label = { Text(stringResource(id =
R.string.password)) },
                    singleLine = true,
                    visualTransformation =
PasswordVisualTransformation(),
                    modifier = Modifier.fillMaxWidth()
                )
                if (selectedTab == 1) {
                    Spacer(modifier = Modifier.height(8.dp))
                    OutlinedTextField(
                        value = confirmPassword,
                        onValueChange = { confirmPassword = it
},
                        label = { Text(stringResource(id =
R.string.confirm_password)) },
                        singleLine = true,
                        visualTransformation =
PasswordVisualTransformation(),
                        modifier = Modifier.fillMaxWidth()
                    )
                }
                Spacer(modifier = Modifier.height(16.dp))
                Button(
                    onClick = {
                        if (selectedTab == 0) {
```

```
                            onSignIn(email, password)
                        } else {
                            if (password == confirmPassword) {
                                onSignUp(email, password)
                            }
                        }
                    },
                    modifier = Modifier.fillMaxWidth(),
                    enabled = !isLoading
                ) {
                    if (isLoading) {
                        CircularProgressIndicator(modifier =
Modifier.size(20.dp))
                    } else {
                        Text(if (selectedTab == 0)
stringResource(id = R.string.sign_in) else stringResource(id =
R.string.sign_up))
                    }
                }
                error?.let {
                    Spacer(modifier = Modifier.height(8.dp))
                    Text(it, color =
MaterialTheme.colorScheme.error)
                }
            }
        }
    }
}
```

AccountsViewModel.kt

```
package com.example.finflow.presentation.screens.accounts

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.example.finflow.domain.models.Account
import com.example.finflow.data.repository.AccountRepository
import
com.example.finflow.domain.repository.TransactionRepository
import
com.example.finflow.domain.usecase.DeleteAccountWithTransactions
UseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

// ViewModel для управління акаунтами
@HiltViewModel
class AccountsViewModel @Inject constructor(
    private val repository: AccountRepository,
```

```
    private val deleteAccountWithTransactionsUseCase:
DeleteAccountWithTransactionsUseCase
) : ViewModel() {
    private val _accounts =
MutableStateFlow<List<Account>>(emptyList())
    val accounts: StateFlow<List<Account>> =
_accounts.asStateFlow()

    // Видаляє акаунт разом з усіма транзакціями, пов'язаними з
цим акаунтом
    fun deleteAccountWithTransactions(userId: String, accountId:
String) {
        viewModelScope.launch {
            deleteAccountWithTransactionsUseCase(userId,
accountId)
            loadAccounts(userId)
        }
    }

    // Завантажує акаунти (рахунки) користувача
    fun loadAccounts(userId: String) {
        viewModelScope.launch {
            _accounts.value = repository.getAccounts(userId)
        }
    }

    // Додає новий акаунт (рахунок)
    fun addAccount(account: Account, userId: String) {
        viewModelScope.launch {
            repository.addAccount(account)
            loadAccounts(userId)
        }
    }

    // Оновлює існуючий акаунт (рахунок)
    fun updateAccount(account: Account, userId: String) {
        viewModelScope.launch {
            repository.updateAccount(account)
            loadAccounts(userId)
        }
    }
}
```

AccountsScreen.kt

```
package com.example.finflow.presentation.screens.accounts

import android.annotation.SuppressLint
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.foundation.text.KeyboardOptions
```

```
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Edit
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.stringResource
import androidx.compose.ui.text.input.KeyboardType
import androidx.compose.ui.unit.dp
import androidx.hilt.navigation.compose.hiltViewModel
import com.example.finflow.R
import com.example.finflow.domain.models.Account
import com.google.firebase.auth.FirebaseAuth
import java.util.UUID
import androidx.compose.foundation.shape.RoundedCornerShape

// Компонент для відображення екрана зі списком рахунків
@OptIn(ExperimentalMaterial3Api::class)
@SuppressLint("UnusedMaterial3ScaffoldPaddingParameter")
@Composable
fun AccountsScreen(
    viewModel: AccountsViewModel = hiltViewModel()
) {
    val accounts by viewModel.accounts.collectAsState()
    var showAccountSheet by remember { mutableStateOf(false) }
    var editAccount by remember { mutableStateOf<Account?>(null)
}
    val sheetState = rememberModalBottomSheetState()
    val userId = FirebaseAuth.getInstance().currentUser?.uid
    val snackbarHostState = remember { SnackbarHostState() }
    var snackbarMessage by remember {
mutableStateOf<String?>(null) }
    val accountDeletedMsg = stringResource(id =
R.string.account_deleted)
    val accountAddedMsg = stringResource(id =
R.string.account_added)
    val accountUpdatedMsg = stringResource(id =
R.string.account_updated)

    LaunchedEffect(userId) {
        if (!userId.isNullOrBlank()) {
            viewModel.loadAccounts(userId)
        }
    }

    if (userId.isNullOrBlank()) {
        Box(modifier = Modifier.fillMaxSize(), contentAlignment
= Alignment.Center) {
            Text(text =
stringResource(R.string.error_user_not_authorized))
```

```
        }
        return
    }

    Scaffold(
        snackbarHost = { SnackbarHost(snackbarHostState) },
        floatingActionButton = {
            FloatingActionButton(onClick = { showAccountSheet =
true; editAccount = null }) {
                Icon(Icons.Default.Add, contentDescription =
stringResource(id = R.string.add_account))
            }
        }
    ) {
        var searchQuery by remember { mutableStateOf("") }
        val filtered = accounts
            .filter { searchQuery.isBlank() ||
it.name.contains(searchQuery, true) }
        Column(modifier = Modifier.fillMaxSize()) {
            Row(
                modifier =
Modifier.fillMaxWidth().padding(8.dp),
                verticalAlignment = Alignment.CenterVertically
            ) {
                OutlinedTextField(
                    value = searchQuery,
                    onValueChange = { searchQuery = it },
                    label = { Text(stringResource(id =
R.string.search)) },
                    modifier = Modifier.weight(1f)
                )
            }
            if (filtered.isEmpty()) {
                Box(
                    modifier = Modifier.fillMaxSize(),
                    contentAlignment = Alignment.Center
                ) {
                    Text(text = stringResource(id =
R.string.no_accounts), style =
MaterialTheme.typography.bodyLarge,
                        color =
MaterialTheme.colorScheme.onSurface.copy(alpha = 0.6f))
                }
            } else {
                LazyColumn(modifier = Modifier.fillMaxSize()) {
                    items(filtered, key = { acc -> acc.id }) {
acc ->
                        var showDeleteDialog by remember {
mutableStateOf(false) }
                        AccountCardModern(
                            acc = acc,
                            onEdit = { editAccount = acc;
```

```
showAccountSheet = true },
                                onDelete = { showDeleteDialog = true
}
                            )
                            if (showDeleteDialog) {
                                AlertDialog(
                                    onDismissRequest = {
showDeleteDialog = false },
                                    title = { Text(stringResource(id
= R.string.delete_account_question)) },
                                    text = { Text(stringResource(id
= R.string.delete_account_confirm, acc.name)) },
                                    confirmButton = {
                                        TextButton(onClick = {

viewModel.deleteAccountWithTransactions(userId, acc.id)
                                            showDeleteDialog = false
                                            snackbarMessage =
accountDeletedMsg
                                        }) { Text(stringResource(id
= R.string.delete)) }
                                    },
                                    dismissButton = {
                                        TextButton(onClick = {
showDeleteDialog = false }) { Text(stringResource(id =
R.string.cancel)) }
                                    }
                                )
                            }
                        }
                        item { Spacer(modifier =
Modifier.Companion.height(200.dp)) }
                    }
                }
            }
            if (showAccountSheet) {
                ModalBottomSheet(
                    onDismissRequest = { showAccountSheet = false;
editAccount = null },
                    sheetState = sheetState
                ) {
                    AccountSheetForm(
                        onSubmit = { acc ->
                            if (editAccount == null) {
                                viewModel.addAccount(acc, userId)
                                snackbarMessage = accountAddedMsg
                            } else {
                                viewModel.updateAccount(acc, userId)
                                snackbarMessage = accountUpdatedMsg
                            }
                            showAccountSheet = false
                            editAccount = null
```

```
                    },
                    onDismiss = { showAccountSheet = false;
editAccount = null },
                    initial = editAccount,
                    userId = userId
                )
            }
        }
    }
    // Показываем Snackbar, если есть сообщение
    LaunchedEffect(snackbarMessage) {
        snackbarMessage?.let {
            snackbarHostState.showSnackbar(it)
            snackbarMessage = null
        }
    }
}

// Компонент для форми додавання/редагування рахунку
@Composable
fun AccountSheetForm(
    onSubmit: (Account) -> Unit,
    onDismiss: () -> Unit,
    initial: Account? = null,
    userId: String
) {
    var name by remember { mutableStateOf(initial?.name ?: "") }
    var balance by remember {
mutableStateOf(initial?.balance?.toString() ?: "") }
    val isValid = name.isNotBlank() && balance.isNotBlank() &&
balance.toDoubleOrNull() != null
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(16.dp)
    ) {
        Text(
            text = if (initial == null) stringResource(id =
R.string.add_account) else stringResource(id =
R.string.edit_account),
            style = MaterialTheme.typography.titleLarge,
            modifier = Modifier.padding(bottom = 16.dp)
        )
        OutlinedTextField(
            value = name,
            onValueChange = { name = it },
            label = { Text(stringResource(id = R.string.name))
},
            singleLine = true,
            isError = name.isBlank(),
            modifier = Modifier.fillMaxWidth()
        )
```

```
        OutlinedTextField(
            value = balance,
            onValueChange = { newValue ->
                if (newValue.matches(Regex("^\\d*\\.?\\d*"))) {
                    balance = newValue
                }
            },
            label = { Text(stringResource(id =
R.string.balance)) },
            singleLine = true,
            keyboardOptions = KeyboardOptions(keyboardType =
KeyboardType.Number),
            isError = balance.isBlank() ||
balance.toDoubleOrNull() == null,
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(16.dp))
        Button(
            onClick = {
                val acc = Account(
                    id = initial?.id ?:
UUID.randomUUID().toString(),
                    name = name,
                    balance = balance.toDoubleOrNull() ?: 0.0,
                    userId = userId
                )
                onSubmit(acc)
            },
            enabled = isValid,
            modifier = Modifier.fillMaxWidth()
        ) {
            Text(stringResource(id = if (initial == null)
R.string.add else R.string.save))
        }
        Spacer(modifier = Modifier.height(8.dp))
        TextButton(onClick = onDismiss, modifier =
Modifier.fillMaxWidth()) {
            Text(stringResource(id = R.string.cancel))
        }
    }
}

// Компонент для відображення картки рахунку
@Composable
fun AccountCardModern(
    acc: Account,
    onEdit: () -> Unit,
    onDelete: () -> Unit
) {
    Card(
        modifier = Modifier
            .fillMaxWidth()
```

```
            .padding(8.dp),
        shape = RoundedCornerShape(16.dp),
        colors = CardDefaults.cardColors(
            containerColor =
MaterialTheme.colorScheme.surfaceVariant
        )
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(16.dp),
            verticalAlignment = Alignment.CenterVertically
        ) {
            Column(modifier = Modifier.weight(1f)) {
                Text(
                    text = acc.name,
                    style = MaterialTheme.typography.titleMedium
                )
                Spacer(modifier = Modifier.height(4.dp))
                Text(
                    text = stringResource(R.string.balance) + ":
%.2f".format(acc.balance),
                    style = MaterialTheme.typography.bodySmall
                )
            }
            Row {
                IconButton(onClick = onEdit) {
                    Icon(Icons.Default.Edit, contentDescription
= stringResource(R.string.edit))
                }
                IconButton(onClick = onDelete) {
                    Icon(Icons.Default.Delete,
contentDescription = stringResource(R.string.delete))
                }
            }
        }
    }
}
```

AppNavHost.kt

```
package com.example.finflow.presentation.navigation

import androidx.compose.runtime.Composable
import androidx.navigation.NavHostController
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import
com.example.finflow.presentation.screens.dashboard.DashboardScre
en
import
com.example.finflow.presentation.screens.transactions.Transactio
nsScreen
```

```
import
com.example.finflow.presentation.screens.accounts.AccountsScreen
import
com.example.finflow.presentation.screens.categories.CategoriesSc
reen
import
com.example.finflow.presentation.screens.settings.SettingsScreen

// навігаційні екрани додатку
sealed class Screen(val route: String) {
    data object Dashboard : Screen("dashboard")
    data object Transactions : Screen("transactions")
    data object Accounts : Screen("accounts")
    data object Categories : Screen("categories")
    data object Settings : Screen("settings")
}

// Головний NavHost для навігації між екранами додатку
@Composable
fun AppNavHost(
    navController: NavHostController,
    isDarkTheme: Boolean = false,
    onToggleTheme: () -> Unit = {},
    selectedLanguage: String = "uk",
    onLanguageChange: (String) -> Unit = {},
    userEmail: String? = null,
    onLogout: () -> Unit = {}
) {
    NavHost(navController, startDestination =
Screen.Dashboard.route) {
        composable(Screen.Dashboard.route) { DashboardScreen() }
        composable(Screen.Transactions.route) {
TransactionsScreen() }
        composable(Screen.Accounts.route) { AccountsScreen() }
        composable(Screen.Categories.route) { CategoriesScreen()
}
        composable(Screen.Settings.route) {
            SettingsScreen(
                isDarkTheme = isDarkTheme,
                onToggleTheme = onToggleTheme,
                selectedLanguage = selectedLanguage,
                onLanguageChange = onLanguageChange,
                userEmail = userEmail,
                onLogout = onLogout
            )
        }
    }
}
```

BottomNavigationBar.kt

```
package com.example.finflow.ui.components
```

```
import androidx.annotation.StringRes
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Balance
import androidx.compose.material.icons.filled.Category
import androidx.compose.material.icons.filled.MobiledataOff
import androidx.compose.material.icons.filled.TrendingUp
import androidx.compose.material3.*
import androidx.compose.runtime.Composable
import androidx.compose.ui.graphics.vector.ImageVector
import androidx.compose.ui.res.stringResource
import androidx.navigation.NavController
import com.example.finflow.presentation.navigation.Screen

// Клас для елементів нижньої навігації
sealed class BottomNavItem(val route: String, val icon:
ImageVector?, @StringRes val labelRes: Int) {
    data object Dashboard :
BottomNavItem(Screen.Dashboard.route, Icons.Filled.TrendingUp,
com.example.finflow.R.string.dashboard)
    data object Transactions :
BottomNavItem(Screen.Transactions.route,
Icons.Filled.MobiledataOff,
com.example.finflow.R.string.transactions)
    data object Accounts : BottomNavItem(Screen.Accounts.route,
Icons.Filled.Balance, com.example.finflow.R.string.accounts)
    data object Categories :
BottomNavItem(Screen.Categories.route, Icons.Filled.Category,
com.example.finflow.R.string.categories)
}

// Список елементів нижньої навігації
val bottomNavItems = listOf(
    BottomNavItem.Dashboard,
    BottomNavItem.Transactions,
    BottomNavItem.Accounts,
    BottomNavItem.Categories
)

// Компонент для відображення нижньої навігаційної панелі
@Composable
fun BottomNavigationBar(navController: NavController,
currentRoute: String?) {
    NavigationBar {
        bottomNavItems.forEach { item ->
            NavigationBarItem(
                selected = currentRoute == item.route,
                onClick = {
                    if (currentRoute != item.route) {
                        navController.navigate(item.route) {

popUpTo(navController.graph.startDestinationId) { saveState =
true }
```

```
                                launchSingleTop = true
                                restoreState = true
                            }
                        }
                    },
                    icon = {
                        item.icon?.let { Icon(it, contentDescription
= item.labelRes.toString()) }
                    },
                    label = { Text(stringResource(item.labelRes)) }
                )
            }
        }
    }
```

TransactionUseCases.kt

```
package com.example.finflow.domain.usecases

import com.example.finflow.domain.models.Transaction
import
com.example.finflow.domain.repository.TransactionRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

// Клас для отримання транзакцій
class GetTransactionsUseCase @Inject constructor(private val
repository: TransactionRepository) {
    operator fun invoke(userId: String): Flow<List<Transaction>>
= repository.getTransactions(userId)
}

// Клас для роботи з транзакціями
class AddTransactionUseCase @Inject constructor(private val
repository: TransactionRepository) {
    suspend operator fun invoke(userId: String, transaction:
Transaction) = repository.addTransaction(userId, transaction)
}

// Клас для оновлення транзакції
class UpdateTransactionUseCase @Inject constructor(private val
repository: TransactionRepository) {
    suspend operator fun invoke(userId: String, transaction:
Transaction) = repository.updateTransaction(userId, transaction)
}

// Клас для видалення транзакції
class DeleteTransactionUseCase @Inject constructor(private val
repository: TransactionRepository) {
    suspend operator fun invoke(userId: String, transactionId:
String) = repository.deleteTransaction(userId, transactionId)
}
```

GetCategoriesUseCase.kt

```
package com.example.finflow.domain.usecases

import com.example.finflow.domain.models.Category
import com.example.finflow.domain.repository.CategoryRepository
import kotlinx.coroutines.flow.Flow
import javax.inject.Inject

// Клас для отримання категорій
class GetCategoriesUseCase @Inject constructor(
    private val repository: CategoryRepository
) {
    operator fun invoke(userId: String): Flow<List<Category>> =
repository.getCategories(userId)
}
```

CategoryUseCases.kt

```
package com.example.finflow.domain.usecases

import com.example.finflow.domain.models.Category
import com.example.finflow.domain.repository.CategoryRepository
import
com.example.finflow.domain.repository.TransactionRepository
import javax.inject.Inject

// Клас для отримання категорій
class AddCategoryUseCase @Inject constructor(private val
repository: CategoryRepository) {
    suspend operator fun invoke(userId: String, category:
Category) = repository.addCategory(userId, category)
}

// Клас для оновлення категорії
class UpdateCategoryUseCase @Inject constructor(
    private val repository: CategoryRepository,
    private val transactionRepository: TransactionRepository
) {
    suspend operator fun invoke(userId: String, category:
Category) {
        repository.updateCategory(userId, category)

transactionRepository.updateCategoryNameInTransactions(userId,
category.id, category.name)
    }
}

// Клас для видалення категорії
class DeleteCategoryUseCase @Inject constructor(
```

```
    private val categoryRepository: CategoryRepository,
    private val transactionRepository: TransactionRepository
) {
    suspend operator fun invoke(userId: String, categoryId:
String) {
        transactionRepository.deleteByCategory(userId,
categoryId)
        categoryRepository.deleteCategory(userId, categoryId)
    }
}
```

DeleteAccountWithTransactionsUseCase.kt

```
package com.example.finflow.domain.usecase

import com.example.finflow.data.repository.AccountRepository
import
com.example.finflow.domain.repository.TransactionRepository
import javax.inject.Inject

// Використовується для видалення акаунту разом з усіма
транзакціями, пов'язаними з цим акаунтом
class DeleteAccountWithTransactionsUseCase @Inject constructor(
    private val accountRepository: AccountRepository,
    private val transactionRepository: TransactionRepository
) {
    /**
     * Видаляє акаунт та всі транзакції, пов'язані з цим
акаунтом.
     *
     * @param userId Ідентифікатор користувача.
     * @param accountId Ідентифікатор акаунту, який потрібно
видалити.
     */
    suspend operator fun invoke(userId: String, accountId:
String) {
        transactionRepository.deleteByAccount(userId, accountId)
        accountRepository.deleteAccount(userId, accountId)
    }
}
```

TransactionRepository.kt

```
package com.example.finflow.domain.repository

import com.example.finflow.domain.models.Transaction
import kotlinx.coroutines.flow.Flow

// Інтерфейс для роботи з транзакціями фінансового додатку
interface TransactionRepository {
    // Отримати список транзакцій для користувача
    fun getTransactions(userId: String): Flow<List<Transaction>>
```

```
    // Додати нову транзакцію
    suspend fun addTransaction(userId: String, transaction:
Transaction)
    // Оновити існуючу транзакцію
    suspend fun updateTransaction(userId: String, transaction:
Transaction)
    // Видалити транзакцію за ID
    suspend fun deleteTransaction(userId: String, transactionId:
String)
    // Видалити всі транзакції за категорією
    suspend fun deleteByCategory(userId: String, categoryId:
String)
    // Оновити назву категорії у всіх транзакціях
    suspend fun updateCategoryNameInTransactions(userId: String,
categoryId: String, newCategoryName: String)
    // Видалити всі транзакції за акаунтом
    suspend fun deleteByAccount(userId: String, accountId:
String)
}
```

CategoryRepository.kt

```
package com.example.finflow.domain.repository

import com.example.finflow.domain.models.Category
import com.example.finflow.domain.models.CategoryType
import kotlinx.coroutines.flow.Flow

// Інтерфейс для роботи з категоріями фінансового додатку
interface CategoryRepository {
    // Отримати список категорій для користувача
    fun getCategories(userId: String): Flow<List<Category>>
    // Отримати категорії за типом (витрати/доходи)
    fun getCategoriesByType(userId: String, type: CategoryType):
Flow<List<Category>>
    // Отримати категорію за ID
    suspend fun addCategory(userId: String, category: Category)
    // Додати нову категорію
    suspend fun updateCategory(userId: String, category:
Category)
    // Оновити існуючу категорію
    suspend fun deleteCategory(userId: String, categoryId:
String)
    // Видалити категорію за ID
}
```

Transaction.kt

```
package com.example.finflow.domain.models

import java.util.Date

// Модель транзакції для фінансового додатку
data class Transaction(
```

```
    val id: String = "",
    val amount: Double = 0.0,
    val categoryId: String = "",
    val categoryName: String = "",
    val description: String = "",
    val type: TransactionType = TransactionType.EXPENSE,
    val date: Date = Date(),
    val accountId: String = "",
    val accountName: String = "",
    val userId: String = ""
)

// Типи транзакцій: дохід та витрата
enum class TransactionType {
    INCOME,
    EXPENSE
}
```

Category.kt

```
package com.example.finflow.domain.models

// Модель категорії для фінансового додатку
data class Category(
    val id: String = "",
    val name: String = "",
    val icon: String = "",
    val userId: String = "",
    val type: CategoryType = CategoryType.EXPENSE
)

// Типи категорій: дохід та витрата
enum class CategoryType {
    INCOME,
    EXPENSE
}
```

Account.kt

```
package com.example.finflow.domain.models

// Модель акаунту (рахунку) для фінансового додатку
data class Account(
    val id: String = "",
    val name: String = "",
    val balance: Double = 0.0,
    val userId: String = ""
)
```

AppModule.kt

```
package com.example.finflow.di

import com.google.firebase.auth.FirebaseAuth
```

```
import com.google.firebase.firestore.FirebaseFirestore
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

// об'єднання для надання FirebaseAuth та FirebaseFirestore
@Module
@InstallIn(SingletonComponent::class)
object AppModule {
    // надаємо FirebaseAuth та FirebaseFirestore як сінглтони
    @Provides
    @Singleton
    fun provideFirebaseAuth(): FirebaseAuth =
FirebaseAuth.getInstance()

    // надаємо FirebaseFirestore як сінглтон
    @Provides
    @Singleton
    fun provideFirebaseFirestore(): FirebaseFirestore =
FirebaseFirestore.getInstance()
}
```

UserPreferences.kt

```
package com.example.finflow.data

import android.content.Context
import android.content.SharedPreferences
import androidx.core.content.edit

// об'єкт для зберігання налаштувань користувача
object UserPreferences {
    private const val PREFS_NAME = "user_prefs"
    private const val KEY_THEME = "theme"
    private const val KEY_LANGUAGE = "language"

    // Ключі для зберігання налаштувань користувача
    fun getTheme(context: Context): String? {
        return getPrefs(context).getString(KEY_THEME, null)
    }

    // Зберігає тему користувача
    fun setTheme(context: Context, theme: String) {
        getPrefs(context).edit { putString(KEY_THEME, theme) }
    }

    // Отримує мову користувача
    fun getLanguage(context: Context): String? {
        return getPrefs(context).getString(KEY_LANGUAGE, null)
    }
```

```
    // Зберігає мову користувача
    fun setLanguage(context: Context, language: String) {
        getPrefs(context).edit { putString(KEY_LANGUAGE,
language) }
    }

    // Очищає всі налаштування користувача
    private fun getPrefs(context: Context): SharedPreferences {
        return context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE)
    }
}
```

FirestoreTransactionRepository.kt

```
package com.example.finflow.data.repository

import com.example.finflow.domain.models.Transaction
import
com.example.finflow.domain.repository.TransactionRepository
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.ListenerRegistration
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject
import com.example.finflow.domain.models.TransactionType

// репозиторій для роботи з транзакціями у Firestore
class FirestoreTransactionRepository @Inject constructor(
    private val firestore: FirebaseFirestore,
    private val accountRepository: AccountRepository
) : TransactionRepository {
    // Колекція транзакцій для конкретного користувача
    private fun transactionsCollection(userId: String) =

firestore.collection("users").document(userId).collection("trans
actions")

    // Отримання транзакцій користувача як Flow
    override fun getTransactions(userId: String):
Flow<List<Transaction>> = callbackFlow {
        val listener: ListenerRegistration =
transactionsCollection(userId)
            .addSnapshotListener { snapshot, _ ->
                val transactions =
snapshot?.toObjects(Transaction::class.java) ?: emptyList()
                trySend(transactions)
            }
```

```
        awaitClose { listener.remove() }
    }

    // Отримання транзакцій користувача з певним типом
    override suspend fun addTransaction(userId: String,
transaction: Transaction) {
        val accounts = accountRepository.getAccounts(userId)
        val account = accounts.find { it.id ==
transaction.accountId }
        if (account != null) {
            val newBalance = when (transaction.type) {
                TransactionType.INCOME -> account.balance +
transaction.amount
                TransactionType.EXPENSE -> account.balance -
transaction.amount
            }
            val updatedAccount = account.copy(balance =
newBalance)
            accountRepository.updateAccount(updatedAccount)
        }
        val txId = transaction.id.ifEmpty {
transactionsCollection(userId).document().id }

transactionsCollection(userId).document(txId).set(transaction.co
py(userId = userId, id = txId)).await()
    }

    // Оновлення транзакції
    override suspend fun updateTransaction(userId: String,
transaction: Transaction) {
        if (transaction.id.isNotEmpty()) {
            val doc =
transactionsCollection(userId).document(transaction.id).get().aw
ait()
            val oldTransaction =
doc.toObject(Transaction::class.java)
            if (oldTransaction != null) {
                val accounts =
accountRepository.getAccounts(userId)
                val oldAccount = accounts.find { it.id ==
oldTransaction.accountId }
                val newAccount = accounts.find { it.id ==
transaction.accountId }
                if (oldTransaction.accountId ==
transaction.accountId && oldTransaction.type == transaction.type
&& newAccount != null) {
                    val diff = transaction.amount -
oldTransaction.amount
                    val newBalance = when (transaction.type) {
                        TransactionType.INCOME ->
newAccount.balance + diff
                        TransactionType.EXPENSE ->
```

```
newAccount.balance - diff
                    }

accountRepository.updateAccount(newAccount.copy(balance =
newBalance))
                } else {
                    if (oldAccount != null) {
                        val revertedBalance = when
(oldTransaction.type) {
                            TransactionType.INCOME ->
oldAccount.balance - oldTransaction.amount
                            TransactionType.EXPENSE ->
oldAccount.balance + oldTransaction.amount
                        }

accountRepository.updateAccount(oldAccount.copy(balance =
revertedBalance))
                    }
                    if (newAccount != null) {
                        val appliedBalance = when
(transaction.type) {
                            TransactionType.INCOME ->
newAccount.balance + transaction.amount
                            TransactionType.EXPENSE ->
newAccount.balance - transaction.amount
                        }

accountRepository.updateAccount(newAccount.copy(balance =
appliedBalance))
                    }
                }
            }

transactionsCollection(userId).document(transaction.id).set(tran
saction.copy(userId = userId)).await()
        }
    }

    // Видалення транзакції
    override suspend fun deleteTransaction(userId: String,
transactionId: String) {
        val doc =
transactionsCollection(userId).document(transactionId).get().awa
it()
        val transaction = doc.toObject(Transaction::class.java)
        if (transaction != null) {
            val accounts = accountRepository.getAccounts(userId)
            val account = accounts.find { it.id ==
transaction.accountId }
            if (account != null) {
                val newBalance = when (transaction.type) {
                    TransactionType.INCOME -> account.balance -
```

```
transaction.amount
                    TransactionType.EXPENSE -> account.balance +
transaction.amount
                }
                val updatedAccount = account.copy(balance =
newBalance)
                accountRepository.updateAccount(updatedAccount)
            }
        }

transactionsCollection(userId).document(transactionId).delete().
await()
    }

    // Видалення всіх транзакцій за категорією
    override suspend fun deleteByCategory(userId: String,
categoryId: String) {
        val batch = firestore.batch()
        val querySnapshot = transactionsCollection(userId)
            .whereEqualTo("categoryId", categoryId)
            .get().await()
        for (doc in querySnapshot.documents) {
            batch.delete(doc.reference)
        }
        batch.commit().await()
    }

    // Оновлення імені категорії у всіх транзакціях
    override suspend fun
updateCategoryNameInTransactions(userId: String, categoryId:
String, newCategoryName: String) {
        val batch = firestore.batch()
        val querySnapshot = transactionsCollection(userId)
            .whereEqualTo("categoryId", categoryId)
            .get().await()
        for (doc in querySnapshot.documents) {
            val data = mapOf("categoryName" to newCategoryName)
            batch.update(doc.reference, data)
        }
        batch.commit().await()
    }

    // Видалення всіх транзакцій за рахунком
    override suspend fun deleteByAccount(userId: String,
accountId: String) {
        val batch = firestore.batch()
        val querySnapshot = transactionsCollection(userId)
            .whereEqualTo("accountId", accountId)
            .get().await()
        for (doc in querySnapshot.documents) {
            batch.delete(doc.reference)
        }
```

```
        batch.commit().await()
    }
}
```

FirestoreCategoryRepository.kt

```
package com.example.finflow.data.repository

import com.example.finflow.domain.models.Category
import com.example.finflow.domain.models.CategoryType
import com.example.finflow.domain.repository.CategoryRepository
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.ListenerRegistration
import kotlinx.coroutines.channels.awaitClose
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.callbackFlow
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

// репозиторій для роботи з категоріями у Firestore
class FirestoreCategoryRepository @Inject constructor(
    private val firestore: FirebaseFirestore
) : CategoryRepository {
    // Колекція категорій для конкретного користувача
    private fun categoriesCollection(userId: String) =

firestore.collection("users").document(userId).collection("categ
ories")

    // Отримання категорій користувача як Flow
    override fun getCategories(userId: String):
Flow<List<Category>> = callbackFlow {
        val listener: ListenerRegistration =
categoriesCollection(userId)
            .addSnapshotListener { snapshot, _ ->
                val categories =
snapshot?.toObjects(Category::class.java)
                    ?.distinctBy { it.id } ?: emptyList()
                trySend(categories)
            }
        awaitClose { listener.remove() }
    }

    // Отримання категорій користувача за типом
    override fun getCategoriesByType(userId: String, type:
CategoryType): Flow<List<Category>> = callbackFlow {
        val listener: ListenerRegistration =
categoriesCollection(userId)
            .whereEqualTo("type", type.name)
            .addSnapshotListener { snapshot, _ ->
                val categories =
snapshot?.toObjects(Category::class.java)
```

```
                    ?.distinctBy { it.id } ?: emptyList()
                trySend(categories)
            }
        awaitClose { listener.remove() }
    }

    // Додавання категорії
    override suspend fun addCategory(userId: String, category:
Category) {
        require(category.id.isNotBlank()) { "Category id must
not be blank" }

categoriesCollection(userId).document(category.id).set(category.
copy(userId = userId)).await()
    }

    // Оновлення категорії
    override suspend fun updateCategory(userId: String,
category: Category) {
        if (category.id.isNotEmpty()) {

categoriesCollection(userId).document(category.id).set(category.
copy(userId = userId)).await()
        }
    }

    // Видалення категорії
    override suspend fun deleteCategory(userId: String,
categoryId: String) {

categoriesCollection(userId).document(categoryId).delete().await
()
    }
}
```

FirestoreAccountRepository.kt

```
package com.example.finflow.data.repository

import com.example.finflow.domain.models.Account
import
com.example.finflow.domain.repository.TransactionRepository
import com.google.firebase.firestore.FirebaseFirestore
import kotlinx.coroutines.tasks.await
import javax.inject.Inject

// репозиторій для роботи з обліковими записами в Firestore
class FirestoreAccountRepository @Inject constructor(
    private val firestore: FirebaseFirestore
) : AccountRepository {
    // колекція облікових записів користувача
    private fun accountsCollection(userId: String) :
```

```
com.google.firebase.firestore.CollectionReference {
        require(userId.isNotBlank()) { "userId must not be blank
or empty" }
        return
firestore.collection("users").document(userId).collection("accou
nts")
    }

    // отримання облікових записів користувача
    override suspend fun getAccounts(userId: String):
List<Account> {
        val snapshot = accountsCollection(userId).get().await()
        return snapshot.documents.mapNotNull {
it.toObject(Account::class.java)?.copy(id = it.id) }
    }

    // отримання облікового запису за ID
    override suspend fun addAccount(account: Account) {
        requireNotNull(account.userId) { "Account must have
userId" }
        require(account.userId.isNotBlank()) { "Account userId
must not be blank or empty" }
        accountsCollection(account.userId).add(account).await()
    }

    // видалення облікового запису за ID
    override suspend fun deleteAccount(userId: String,
accountId: String) {
        require(userId.isNotBlank()) { "userId must not be blank
or empty" }
        require(accountId.isNotBlank()) { "accountId must not be
blank or empty" }

accountsCollection(userId).document(accountId).delete().await()
    }

    // оновлення облікового запису
    override suspend fun updateAccount(account: Account) {
        requireNotNull(account.userId) { "Account must have
userId" }
        require(account.userId.isNotBlank()) { "Account userId
must not be blank or empty" }
        requireNotNull(account.id) { "Account must have id" }
        require(account.id.isNotBlank()) { "Account id must not
be blank or empty" }

accountsCollection(account.userId).document(account.id).set(acco
unt).await()
    }
}
```

AccountRepository.kt

```
package com.example.finflow.data.repository

import com.example.finflow.domain.models.Account

// інтерфейс для репозиторія акаунтів (рахунків)
interface AccountRepository {
    suspend fun getAccounts(userId: String): List<Account>
    suspend fun addAccount(account: Account)
    suspend fun deleteAccount(userId: String, accountId: String)
    suspend fun updateAccount(account: Account)
}
```