Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Інститут комп’ютерних систем

Кафедра інформаційних систем

Лисишин Данило Олегович,

студент групи АІ-215

ДИСЦИПЛІНА

Об’єктно-орієнтоване програмування

КУРСОВА РОБОТА

Розробка мобільного додатку для відстежування доходів та витрат

Спеціальність:

122 Комп’ютерні науки

Освітня програма:

Комп’ютерні науки

Керівник:

Годовиченко Микола Анатолійович,

кандидат технічних наук, доцент

Одеса – 2023

Зміст

[**АНОТАЦІЯ** 3](#_Toc137506930)

[**ВСТУП** 4](#_Toc137506931)

[**1 ОГЛЯД СИСТЕМ-АНАЛОГІВ ТА ТЕХНОЛОГІЙ ЇХ РОЗРОБКИ** 5](#_Toc137506932)

[**1.1 Особливості використання технологій для управління своїми фінансами** 5](#_Toc137506933)

[**1.2 Огляд додатків для управління складськими запасами для відстежування доходів та витрат** 6](#_Toc137506934)

[**1.3 Формування вимог до основних функцій мобільного додатку** 8](#_Toc137506935)

[**1.4** **Огляд інформаційних технологій для розробки мобільного додатку** 9](#_Toc137506936)

[**1.5 Висновки до першого розділу** 11](#_Toc137506937)

[**ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ УПРАВЛІННЯ СКЛАДСЬКИМИ ЗАПАСАМИ** 12](#_Toc137506938)

[**2.1 Мета та задачі додатку** 12](#_Toc137506939)

[**2.2 Визначення функціональних вимог до мобільного додатку** 13](#_Toc137506940)

[**2.3 Формування користувацьких історій мобільного додатку** 15](#_Toc137506941)

[**2.4 Визначення нефункціональних вимог до мобільного додатку** 16](#_Toc137506942)

[**2.5 Ідентифікація архетипу мобільного додатку** 17](#_Toc137506943)

[**2.6 Проектування навігаційного графу мобільного додатку** 17](#_Toc137506944)

[**2.8 Висновки до другого розділу** 18](#_Toc137506945)

[**ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ** 19](#_Toc137506946)

[**3.1 Діаграма класів проекту** 19](#_Toc137506947)

[**3.2 Керування вихідним кодом додатку** 22](#_Toc137506948)

[**3.3 Функціональне тестування розробленого додатку** 22](#_Toc137506949)

[**3.4 Інструкція користувача** 25](#_Toc137506950)

[**3.5 Вихідний код веб-ресурсу** 35](#_Toc137506951)

[**3. 6 Висновки до третього розділу** 35](#_Toc137506952)

[**ВИСНОВКИ** 36](#_Toc137506953)

[**ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ** 37](#_Toc137506954)

[**ДОДАТОК А. КОД ПРОГРАМИ** 38](#_Toc137506955)

# **АНОТАЦІЯ**

Курсова робота присвячена розробці мобільного додатку для для відстежування доходів та витрат. Додаток є віртуальним гаманцем, який допомагає користувачам відстежувати свої доходи, витрати та накопичення. Система надає зручні інструменти для контролю над фінансовим станом, що дозволяє користувачам збільшувати свої накопичення та зменшувати витрати. В ході розробки додатку будуть використані сучасні технології та практики розробки мобільних додатків, зокрема мова програмування Kotlin для розробки на платформі Android та архітектура MVVM.

**ABSTRACT**

The course work is devoted to the development of a mobile application for tracking income and expenses. The app is a virtual wallet that helps users track their income, spending and savings. The system provides convenient tools to control the financial situation, which allows users to increase their savings and reduce expenses. During the development of the application, modern technologies and practices of mobile application development will be used, in particular, the Kotlin programming language for development on the Android platform and the MVVM architecture.

# **ВСТУП**

У сучасному світі фінансова свідомість та ефективне управління фінансами стали надзвичайно важливими для багатьох людей. Завдяки швидкому розвитку технологій та поширенню мобільних пристроїв, мобільні додатки стали незамінними помічниками у кожноденному житті. У цьому контексті розробка мобільного додатку для відстежування доходів та витрат має великий потенціал для полегшення управління фінансами користувачів.

Ця курсова робота присвячена розробці мобільного додатку, який виступає в якості віртуального гаманця та надає користувачам зручні інструменти для контролю над їхнім фінансовим станом. Головна мета додатку полягає у допомозі користувачам у відстежуванні їхніх доходів, витрат та накопичень, що дозволяє збільшувати накопичення та зменшувати витрати.

У роботі будуть використані сучасні технології та практики розробки мобільних додатків, зокрема мова програмування Kotlin для розробки на платформі Android. Kotlin є мовою з високою продуктивністю та безпекою, а також надає зручні інструменти для розробки мобільних додатків, забезпечуючи широкий функціонал та гнучкість.

У подальшому розділі роботи будуть розглянуті основні принципи розробки мобільних додатків, описана архітектура додатку та деталі реалізації. Також будуть розглянуті основні функції додатку, його інтерфейс та можливості для подальшого розширення.

# **1 ОГЛЯД СИСТЕМ-АНАЛОГІВ ТА ТЕХНОЛОГІЙ ЇХ РОЗРОБКИ**

## **1.1 Особливості використання технологій для управління своїми фінансами**

У сучасному світі зростаюча кількість людей орієнтується на використання мобільних додатків для управління своїми фінансами. Технології мобільних пристроїв та їх доступність зробили це популярним і зручним рішенням. Велика кількість мобільних додатків для відстежування доходів та витрат доступна на ринку, і вони пропонують різноманітні функції та можливості.

Основна перевага використання мобільних додатків для відстежування доходів та витрат полягає в їхній мобільності. Користувачі можуть вести облік своїх фінансів у будь-який час та в будь-якому місці, використовуючи свої смартфони або планшети. Це дає їм можливість миттєво отримувати доступ до інформації про свої доходи та витрати, а також контролювати своє фінансове становище.

Крім того, мобільні додатки для відстежування доходів та витрат зазвичай пропонують різноманітні інструменти та функції, які допомагають користувачам ефективно управляти своїми фінансами. Наприклад, такі додатки можуть надавати можливість встановлення бюджетів, створення категорій доходів та витрат, аналізувати фінансові звіти та графіки, нагадувати про платежі та багато іншого.

Проте, при розробці мобільного додатку для відстежування доходів та витрат необхідно враховувати деякі особливості. Перш за все, потрібно забезпечити зручний та інтуїтивно зрозумілий інтерфейс, щоб користувачі могли швидко та легко використовувати додаток. Також важливо забезпечити безпеку фінансової інформації користувачів, наприклад, шляхом застосування шифрування даних та ідентифікації користувача.

У цій курсовій роботі будуть використані сучасні технології та практики розробки мобільних додатків, зокрема мова програмування Kotlin для розробки на платформі Android. Kotlin є популярною мовою програмування для розробки Android-додатків, оскільки вона поєднує в собі простоту, ефективність та безпеку. Використання Kotlin дозволить забезпечити швидку та надійну розробку мобільного додатку для відстежування доходів та витрат.

## **1.2 Огляд додатків для управління складськими запасами для відстежування доходів та витрат**

На сьогоднішній день існує велика кількість мобільних додатків, які призначені для відстежування доходів та витрат. Деякі з них набули великої популярності та визнання серед користувачів. Давайте розглянемо кілька найбільш популярних додатків у цій категорії:

1. Mint: Цей додаток від Intuit є одним з найпопулярніших інструментів для фінансового управління. Він дозволяє користувачам збирати інформацію про свої доходи та витрати, автоматично категоризувати транзакції, створювати бюджети та отримувати попередження про перевищення лімітів. Mint також надає зручний аналітичний звіт, який допомагає користувачам аналізувати свої фінанси.

2. PocketGuard: Цей додаток пропонує простий та зручний спосіб відстежування доходів та витрат. Він автоматично імпортує транзакції з банківських рахунків користувача і категоризує їх. PocketGuard також допомагає створювати бюджети, визначає поточний фінансовий стан та надає поради щодо економії грошей.

3. Wally: Цей додаток пропонує просте та елегантне рішення для відстежування доходів та витрат. Він дозволяє користувачам вручну вводити транзакції або автоматично імпортувати їх з банківських рахунків. Wally надає деталізовану статистику, графіки та аналітику для аналізу фінансового стану.

4. YNAB (You Need a Budget): Цей додаток базується на концепції активного бюджетування. Він допомагає користувачам призначати кожному долару певну роль у їхньому бюджеті. YNAB надає можливість відстежувати доходи та витрати в режимі реального часу, створювати бюджетні категорії та отримувати практичні поради щодо збереження грошей.

Ці додатки є лише кількома прикладами існуючих рішень для відстежування доходів та витрат. Кожен з них має свої унікальні особливості та функції, які варто враховувати при розробці власного мобільного додатку. Огляд цих додатків дозволяє виявити сильні сторони та потенційні покращення для нашого додатку і забезпечити його конкурентоспроможність на ринку.

## **1.3 Формування вимог до основних функцій мобільного додатку**

Формування вимог до основних функцій мобільного додатку для відстежування доходів та витрат є ключовим етапом в розробці. Основні функції додатку мають відповідати потребам користувачів та забезпечувати зручність та ефективність в управлінні фінансами. Далі наведено деякі загальні вимоги до основних функцій мобільного додатку:

1. Реєстрація та аутентифікація користувача: Додаток має надати можливість реєстрації нових користувачів та аутентифікації існуючих користувачів. Це може бути реалізовано шляхом використання електронної пошти, пароля або інших методів ідентифікації.

2. Додавання доходів та витрат: Користувачі повинні мати можливість додавати нові записи про свої доходи та витрати. Це може включати вибір категорії, дати, суми та опису транзакцій.

3. Категоризація транзакцій: Додаток має дозволяти користувачам категоризувати свої транзакції для зручної організації та аналізу фінансових даних. Наприклад, категорії можуть включати харчування, транспорт, житло, розваги та інші.

4. Створення та відстежування бюджетів: Користувачі повинні мати можливість створювати бюджети на основі своїх доходів та витрат. Додаток повинен надати засоби для моніторингу та відстежування використання бюджету та надавати попередження у разі його перевищення.

5. Аналітика та звіти: Додаток повинен надавати зручні інструменти для аналізу фінансових даних, такі як графіки, діаграми та звіти. Це дозволить користувачам оцінити свої фінансові показники, виявити тенденції та приймати обґрунтовані рішення щодо управління грошима.

6. Нагадування та сповіщення: Додаток може надавати функцію нагадування про сплату рахунків, відкладення коштів або інші фінансові події. Це допомагає користувачам залишатися в курсі своїх фінансових зобов'язань та досягати своїх фінансових цілей.

7. Синхронізація та резервне копіювання: Для забезпечення зручного використання додатку на кількох пристроях користувачі мають мати можливість синхронізувати свої дані та зробити резервну копію важливої інформації.

Це лише загальний огляд основних функцій, які можуть бути включені в мобільний додаток для відстежування доходів та витрат. Конкретні вимоги до функціоналу додатку можуть варіюватися в залежності від потреб та цілей користувачів.

## **Огляд інформаційних технологій для розробки мобільного додатку**

Архітектурні компоненти Android - це набір компонентів, які допомагають розробникам побудувати стабільні, модульні та легко обслуговувані додатки для платформи Android. Нижче наведено огляд деяких ключових архітектурних компонентів Android:

1. Activity (Діяльність): Activity є основним компонентом взаємодії з користувачем у додатках Android. Вона представляє окремий екран з інтерфейсом користувача, з яким користувач може взаємодіяти. Кожна активність має свої методи життєвого циклу, які дозволяють реагувати на зміни стану та події, такі як створення, пауза, відновлення та знищення активності.

2. Fragment (Фрагмент): Фрагменти дозволяють розділити користувацький інтерфейс та поведінку на більш малий та переиспользуемый компонент. Вони можуть бути включені у складі активності або розміщені біля один одного. Фрагменти дозволяють більш гнучко управляти користувацьким інтерфейсом та забезпечують кращу підтримку для різних розмірів екрану та орієнтацій.

3. ViewModel: ViewModel допомагає зберігати та управляти даними, пов'язаними з UI, такими як стан екрану та дані користувача. Він забезпечує відокремлення логіки бізнесу від UI та зберігає дані під час зміни конфігурації, наприклад, при повороті екрану. ViewModel рекомендується використовувати разом з фрагментами або активностями.

4. LiveData: LiveData є спеціальним об'єктом, який надає спостереження за змінами даних. Він автоматично оновлює спостерігачі (наприклад, фрагменти або активності), коли дані змінюються. LiveData допомагає підтримувати консистентність даних між різними компонентами та уникнути некоректного стану.

5. Room: Room є бібліотекою, яка надає високорівневий доступ до бази даних SQLite на платформі Android. Вона спрощує роботу з базою даних, надаючи абстракцію над SQL-запитами та забезпечуючи перевірку на етапі компіляції.

6. Retrofit: Retrofit є бібліотекою для роботи з мережевими запитами у додатках Android. Вона дозволяє легко виконувати HTTP-запити до віддалених серверів та обробляти отримані відповіді.

Ці архітектурні компоненти допомагають розробникам створювати добре структуровані та легко розширювані додатки Android. Вони пропонують підхід, що сприяє використанню найкращих практик розробки та полегшує управління функціональністю додатка.

## **1.5 Висновки до першого розділу**

У цьому розділі було проведено огляд систем-аналогів та технологій, пов'язаних з розробкою мобільних додатків для відстежування доходів та витрат. Було виявлено, що мобільні технології мають великий потенціал у полі фінансового управління, надаючи зручні інструменти для контролю над фінансовим станом та плануванням.

Огляд додатків показав, що на ринку вже існують різноманітні рішення для відстежування доходів та витрат, але кожен з них має свої переваги та недоліки. Розробка власного мобільного додатку дозволить створити інструмент, який відповідатиме конкретним потребам та вимогам користувачів.

Основні вимоги до функціоналу мобільного додатку були сформульовані, включаючи можливість додавання доходів та витрат, категоризацію транзакцій, створення бюджетів, аналітику та звіти, нагадування та сповіщення, а також синхронізацію та резервне копіювання.

Крім того, було розглянуто архітектурні компоненти Android, такі як Activity, Fragment, ViewModel, LiveData, Room та Retrofit, які можуть бути використані при розробці мобільного додатку. Ці компоненти надають зручні засоби для організації структури додатку, управління даними та взаємодії з мережею.

В цілому, перший розділ дав загальне уявлення про предмет дослідження та поклав основу для подальшої розробки мобільного додатку для відстежування доходів та витрат. У наступних розділах будуть детальніше розглянуті процес розробки, вибір мови програмування Kotlin та інші аспекти реалізації проекту.

# **ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ УПРАВЛІННЯ СКЛАДСЬКИМИ ЗАПАСАМИ**

## **2.1 Мета та задачі додатку**

Метою мобільного додатку для відстежування доходів та витрат є надання користувачам зручного та ефективного інструменту для контролю над їхнім фінансовим станом. Головною метою є допомогти користувачам зберігати, аналізувати та планувати свої доходи та витрати, щоб здійснювати розумні фінансові рішення та досягати своїх фінансових цілей.

Для досягнення цієї мети мобільний додаток буде ставити перед собою наступні задачі:

1. Мати змогу обирати періоду час, для перегляду всіх транзакцій.
2. Мати змогу обирати категорію, куди саме були витрачені гроші.
3. Мати змогу очищати всі транзакцї.
4. Мати можливість переглядати свій баланс, щоб бути в курсі свого фінансового стану.
5. Видаляти помилкові транзакції.
6. Мати змогу відстежувати свої витрати.
7. Мати можливість описувати транзакцію
8. Мати можливість додавати витрати до системи.
9. Мати можливість додавати дохід до системи, щоб відстежувати свій фінансовий стан.

## **2.2 Визначення функціональних вимог до мобільного додатку**

Визначення функціональних вимог є важливим етапом у процесі створення мобільного додатку. Функціональні вимоги визначають, які конкретні функції та можливості повинен мати додаток. Вони встановлюють чітку спрямованість розробки, допомагають уникнути неоднозначностей та непорозумінь.

Крім того, визначення функціональних вимог дозволяє зосередитися на потребах та вимогах користувачів. Вони допомагають врахувати, які функції та можливості будуть найбільш корисними для користувачів додатку.

Також, функціональні вимоги слугують основою для комунікації між розробниками, дизайнерами та іншими учасниками проекту. Вони допомагають зрозуміти, що саме потрібно реалізувати та які очікувані результати.

Встановлення функціональних вимог дозволяє визначити обсяг роботи та потребу в ресурсах для реалізації додатку. Це допомагає планувати час, бюджет та ресурси проекту ефективно.

Єдиним актором мобільного додатку є актор «користувач»:

* користувач – користувач мобільного додатку. Має доступ до всіх функцій додатку та несе всю відповідальність за його роботу.

З метою визначення користувацьких історій та нефункціональних вимог до веб-ресурсу, було розроблено діаграму сценаріїв використання мобільного додатку (рис. 2.1). Діаграма сценаріїв UML (Unified Modeling Language) - це графічний інструмент для опису функціональної взаємодії між користувачами та системою. Вона складається з акторів, сценаріїв та взаємодії між ними.

Ця діаграма містить основних акторів системи та описує сценарії їх взаємодії з системою. Вона допомагає проаналізувати залежності між акторами та можливими варіантами використання системи. Також ця діаграма швидко демонструє основні функції системи для розробників.

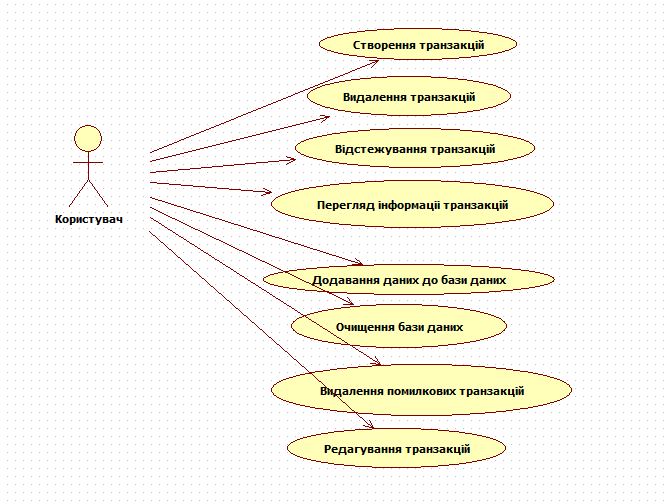


Рисунок 2.1 – Діаграма варіантів використання мобільного додатку

## **2.3** **Формування користувацьких історій мобільного додатку**

Проектування діаграми прецедентів дозволяє визначити такі користувацькі історії до мобільного додатку.

US1 Як користувач, я хочу створити нову транзакцію, щоб зафіксувати важливу інформацію.

Сценарій користувацької історії виглядає таким чином:

* користувач відкриває додаток;
* він натискає кнопку «+»;
* користувач вводить дані;
* після завершення введення, користувач натискає кнопку «Зберегти»;
* нова транзакція зберігається в додатку і з'являється у списку;

US2 Як користувач, я хочу редагувати вміст транзації, щоб вносити зміни або доповнення до інформації.

Сценарій користувацької історії виглядає таким чином:

* користувач відкриває конкретну транзакцію;
* він натискає на кнопку «Редагувати» або вибирає опцію редагування;
* користувач вносить необхідні зміни або доповнення до тексту;
* після завершення редагування, користувач натискає кнопку «Зберегти» або подібний елемент, щоб зберегти оновлену версію замітки.

US3 Як користувач, я хочу видаляти транзакції, які більше не потрібні, для підтримки чистоти та організованості списку.

Сценарій користувацької історії виглядає таким чином:

* користувач відкриває конкретну транзакцію;
* він натискає кнопку «Видалити» або вибирає опцію видалення;
* користувач підтверджує видалення;
* транзакції видаляються з додатку та зникають зі списку.

US4 Як користувач, я хочу переглянути транзакції за певний період часу.

Сценарій користувацької історії виглядає таким чином:

* користувач натискає на кнопку фільтру;
* користувач вибирає період часу та натискає на кнопку «Ок»;
* після збереження список та баланс зміняться.

US4 Як користувач, я хочу очистити дані.

Сценарій користувацької історії виглядає таким чином:

* користувач переходить до налаштувань;
* користувач натискає на кнопку «Видалити»
* користувач підтверджує видалення;
* транзакції видаляються з додатку та зникають зі списку.

## **2.4 Визначення нефункціональних вимог до мобільного додатку**

Після визначення функціональних вимог, необхідно сформувати нефункціональні вимоги, які висуваються до мобільного додатку для ведення заміток.

NFR1 Версія операційної системи – додаток може підтримувати певний діапазон версій, наприклад, Android 8.0 або новіше.

NFR2 Процесор – мінімальний процесор з відповідною архітектурою (наприклад, ARM або x86), який підтримується операційною системою.

NFR3 Оперативна пам'ять (RAM) – мінімальний обсяг оперативної пам'яті, необхідний для плавної роботи додатку. Зазвичай, це може бути від 1 ГБ і більше.

NFR4 Внутрішня пам'ять – додаток може вимагати певний обсяг внутрішньої пам'яті для збереження даних заміток і налаштувань. Це може бути від кількох мегабайт до кількох гігабайт.

NFR5 Роздільна здатність екрану – мінімальна роздільна здатність екрану, яка забезпечує належний візуальний досвід користувача.

NFR6 Інтернет-підключення – додаток може вимагати наявності активного Інтернет-підключення для синхронізації заміток, резервного копіювання, відправки сповіщень тощо.

## **2.5 Ідентифікація архетипу мобільного додатку**

Інформаційна система відноситься до архетипу Mobile Application (MA) –класичний додаток, який запускається на мобільному пристрою під управлінням операційної системи Android.

## **2.6 Проектування навігаційного графу мобільного додатку**

Навігаційний граф (Navigation Graph) – це компонент фреймворку Android Jetpack, який використовується для візуалізації і управління навігацією між екранами (фрагментами) в додатку Android. Він визначає структуру навігації та взаємозв'язки між різними екранами(рис. 2.2).

За допомогою навігаційного графа розробник може легко організувати навігацію між різними екранами в додатку. Він дозволяє встановлювати точки входу (destination) для кожного екрану, визначати перехідні анімації, задавати параметри переходів (наприклад, передачу даних), а також встановлювати специфічні правила навігації, такі як повернення до попереднього екрану (back navigation) або виконання визначених дій перед показом нового екрану.

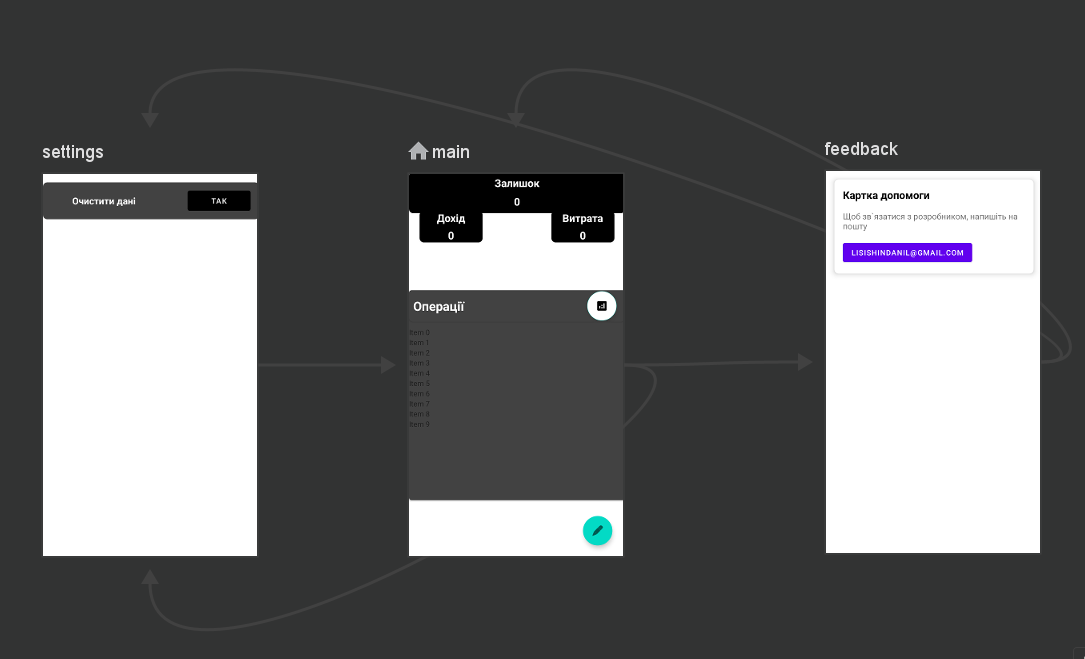


Рисунок 2.2 – Навігаційний граф мобільного додатку

## **2.8 Висновки до другого розділу**

У другому розділі були визначені мета та задачі мобільного додатку для відстежування доходів та витрат. Основна мета додатку полягає в наданні користувачам зручного та ефективного інструменту для контролю над їхнім фінансовим станом. Для досягнення цієї мети були сформульовані наступні задачі:

1. Обирання періоду часу для перегляду всіх транзакцій.

2. Обирання категорії для витрат.

3. Очищення всіх транзакцій.

4. Перегляд балансу для контролю фінансового стану.

5. Видалення помилкових транзакцій.

6. Відстеження витрат.

7. Опис транзакцій.

8. Додавання витрат до системи.

9. Додавання доходу до системи.

Також було проведено формування користувацьких історій, визначення функціональних і нефункціональних вимог до додатку, ідентифікацію архетипу мобільного додатку (Mobile Application - MA), а також проектування навігаційного графу.

Висновки до другого розділу включають у собі зазначення мети та задачі додатку, а також вимоги до його функціональності та нефункціональні характеристики. Крім того, було ідентифіковано архетип мобільного додатку та розроблено навігаційний граф. Загалом, цей розділ визначає основні принципи та цілі, що використовуються для подальшого проектування та розробки мобільного додатку.

# **ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ**

## **3.1 Діаграма класів проекту**

Діаграма класів є одним із типів UML-діаграм, що використовується для візуалізації структури системи або додатку шляхом відображення класів, інтерфейсів та зв'язків між ними. Вона допомагає отримати зовнішній огляд компонентів системи та розуміння їх взаємозв'язків та взаємодії.

На рисунку 3.1 наведена діаграма класів, що відображає структуру додатку для відстежування доходів та витрат. На діаграмі представлені класи з їх атрибутами та методами, а також зв'язки між ними. Ця діаграма дозволяє краще розібратися у структурі системи та виявити класи з їх взаємозв'язками та функціональними можливостями.

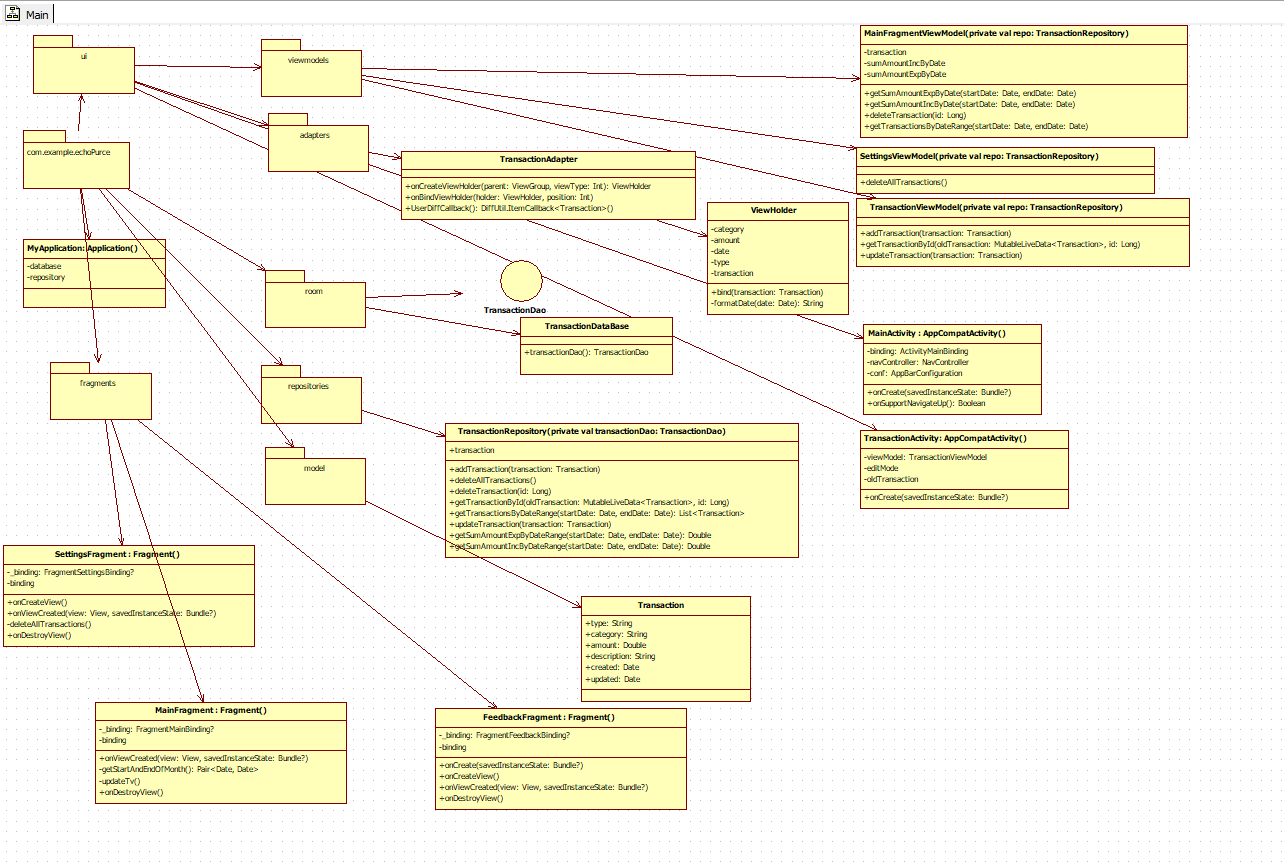


Рисунок 3.1 – Діаграма класів проєкту

## **3.2 Керування вихідним кодом додатку**

Використання системи контролю версій є дуже важливим для будь-якого програмного проекту, в тому числі і для додатку. Використання системи контролю версій дозволяє зберігати історію всіх змін, які були внесені до веб-ресурсу.

Це дозволяє в разі потреби повернутися до попередніх версій додатку та відновити попередній стан, який був на момент попередньої версії.

Також, система контролю версій дозволяє контролювати версії веб-ресурсу, що значно полегшує роботу з проектом та дозволяє підтримувати його в актуальному стані.

Основні метрики репозиторію веб-ресурсу наведені в таблиці 3.1.

Таблиця 3.1 – Метрики керування програмним кодом додатку

|  |  |  |  |
| --- | --- | --- | --- |
| Веб-ресурс | Кількість комітів | Кількість pull-реквестів | Кількість гілок у репозиторії |
| Клієнтська частина | 3 | 3 | 2 |

## **3.3 Функціональне тестування розробленого додатку**

Функціональне тестування розробленого додатку для відстежування своїх доходів, витрат та накопичень є важливим етапом для переконанняся, що система працює належним чином і відповідає вимогам користувачів. Основною метою функціонального тестування є перевірка функціональності окремих модулів та функцій додатку.

Для функціонального тестування веб-ресурсу продуктової крамниці був розроблений наступний протокол тестування.

TC1 Тест-кейс для додавання нової транзакції:

* відкрити додаток;
* натиснути на кнопку «+»;
* ввести інформацію про транзакцію;
* натиснути кнопку «Зберегти»;
* переконатись, що нова транзакція з'явився в списку у програмі.

TC2 Тест-кейс для редагування існуючого транзакції:

* відкрити додаток;
* обрати транзакцію, яку потрібно змінити;
* натиснути кнопку «Редагувати»;
* змінити потрібні поля;
* натиснути кнопку «Зберегти»;
* переконатись, що зміни збереглися і відображаються правильно.

TC3 Тест-кейс для видалення транзакції:

* відкрити додаток;
* обрати транзакцію, яку потрібно видалити;
* натиснути кнопку «Видалити»;
* підтвердити, що ви дійсно хочете видалити;
* переконатись, що транзакція більше не відображається в програмі.

TC4 Тест-кейс для відобрабраження транзакцій за певний період часу:

* відкрити додаток;
* натиснути на кнопку фільтру;
* обрати період часу на натиснути кнопку «Зберегти»;
* переконатись, що усі транзакції, що виведені за певний період часу.

TC5 Тест-кейс для відображення інформації про транзакцію:

* відкрити додаток;
* обрати транзакцію, про яку потрібно отримати інформацію;
* переконатись, що на екрані відображається інформація про транзакцію.

TC6 Тест-кейс для очищення даних:

* відкрити додаток;
* перейти до налаштувань;
* натиснути на кнопку «Очистити дані».
* підтвердити, що ви дійсно хочете видалити.
* переконатись, що дані були очищені в програмі
* переконатись, що інформація про замовлення відображається на екрані.
* переконатись, що нова категорія з'явилась в списку категорій у програмі.

Це лише деякі приклади тестових сценаріїв, і функціональне тестування може бути розширене залежно від специфікацій та вимог до додатку. Важливо також враховувати різні можливі ситуації, такі як обробка помилок, відповідність безпековим стандартам тощо.

Таблиця 3.2 – Протокол функціонального тестування веб-ресурсу

|  |  |  |  |
| --- | --- | --- | --- |
| Номер тест-кейсу | Очікуваний результат | Фактичний результат | Результат тестування |
| ТС1 | Успішне додавання та відстеження доходів | Нова транзакція успішно додається до бази даних | Успішно |
| TC2 | Успішне редагування існуючого транзакції | Поля запису в базі даних змінюються успішно | Успішно |
| TC3 | Успішне видалення транзакції | Запис в базі даних успішно видаляється | Успішно |
| TC4 | Успішне відобрабраження транзакцій за певний період часу | Транзакція знаходиться успішно | Успішно |
| TC5 | Вірне відображення інформації про транзакцію | Відображена інформація відповідає дійсності | Успішно |
| ТС6 | Успішне очищення даних | База даних успішно очищена | Успішно |

## **3.4 Інструкція користувача**

Для забезпечення успішного користувацького досвіду використання розробленого веб-ресурсу необхідно скласти інструкцію користувача, яка включає в себе знімки екранів та пояснювальний текст до кожного знімка. Це допоможе користувачам легко зорієнтуватись у функціоналі веб-ресурсу та ефективно використовувати його.

При відкритті додатку “EchoPurce”, автоматично відкривається графічний інтерфейс головного меню, де користувач може виконати всі основні операції(рис.3.3).

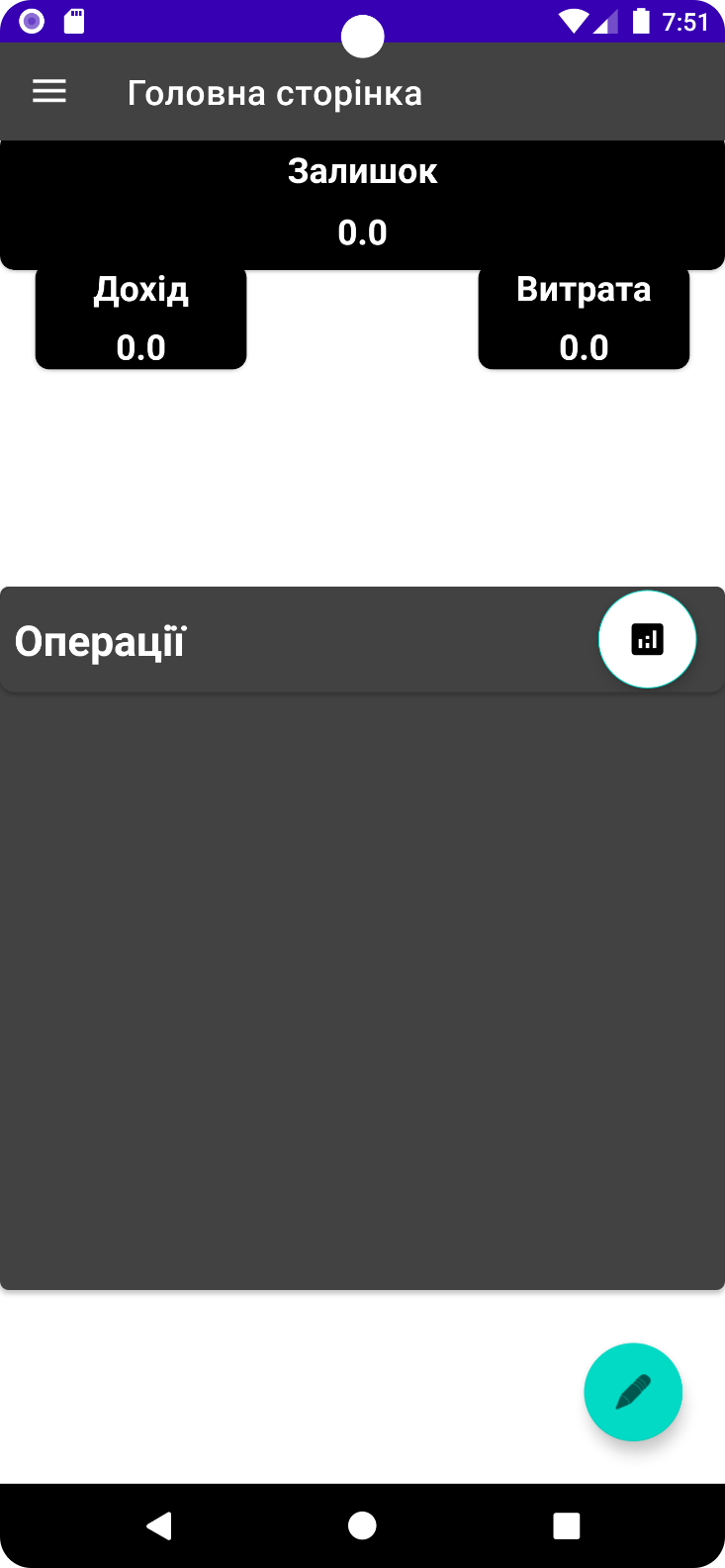


Рис. 3.3 – Головна сторінка додатку

Якщо коритстувач хоче додати нову транзакцію, тоді йому потрібно натиснути на Floating Action Button з малюнком ручки, після чого відкриється екран на якому потрібно буде запонити дані транзакції (рис.3.4). Після заповнення форми, потрібно натиснути кнопку “Зберегти”, для збереження данних до бази даних, або “Відмінити”, для виходу з екрану без збереження даних. Якщо користувач натиснув зберегти, то його поверне на головну сторінку, де баланс та інші елементи будуть оновлені(рис.3.5).

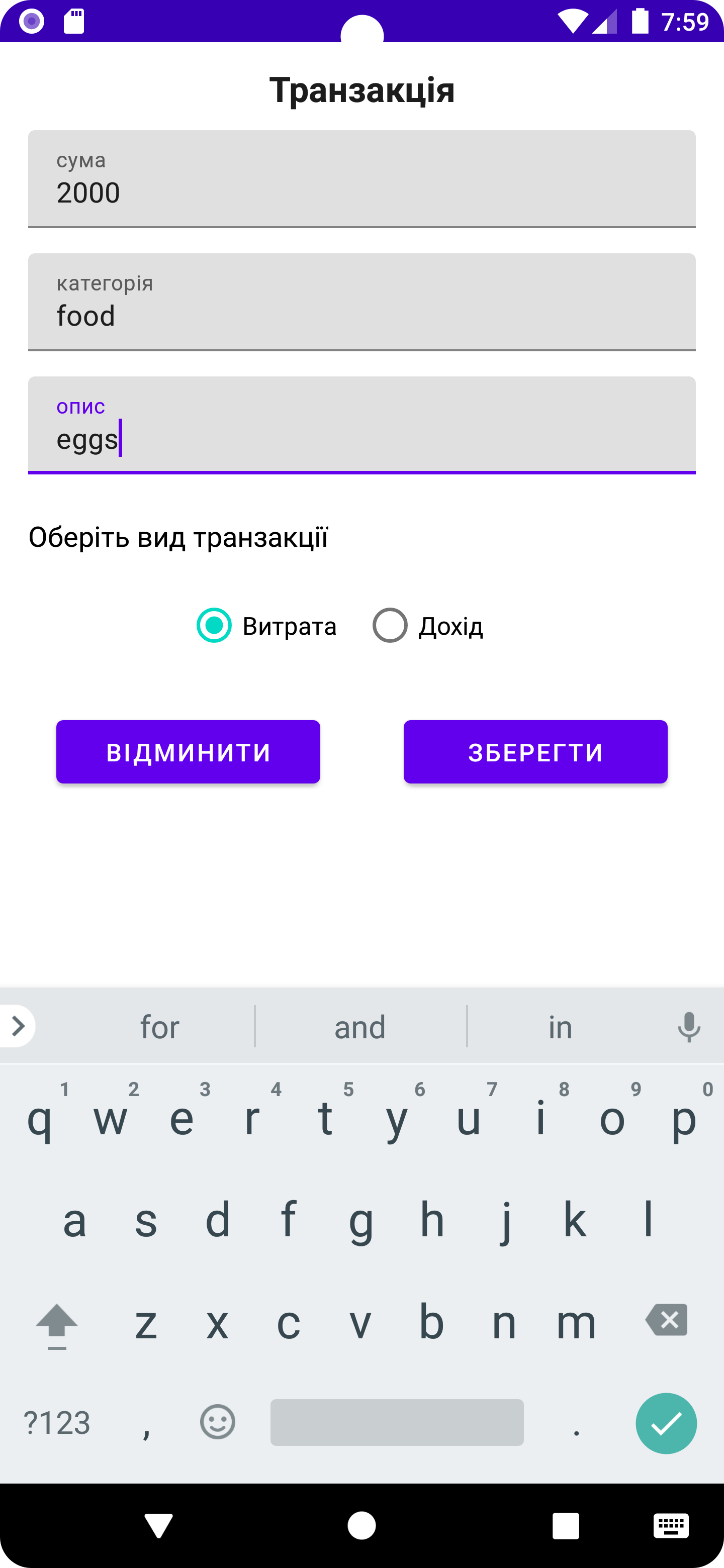


Рис.3.4 – Екран для заповнення форми транзакціїї



Рис.3.5 – Оновлене головне меню при додавані транзакції до бази даних

Якщо користувач хоче переглянути дані про певну транзакцію, йому потрібно натиснути на потрібну транзакцю зі списку, після чого відкриється вікно, де буде описана інформація про транзакцію (рис.3.6). Також в даному вікні користувач зможе повернутися назад, якщо натисне на кнопку “Повернутися”, видалити обрану транзакцію, при натисканні на кнопку “Видалити” вам потрібно буде підтвердити дію, якщо ви натисните на кнопку “Так”, то транзакція буде видалена з бази даних, тобто елементи на екрані автоматично оновляться, якщо натиснути кнопку “Ні”, то користувача поверне до головного меню. Та редагувати дані транзакції, при натисканні на кнопку “Редагувати” користувача буде перекинуто у вікно з редагуванням даних, яке виглядає ідентично з вікном створення нової транзакції.

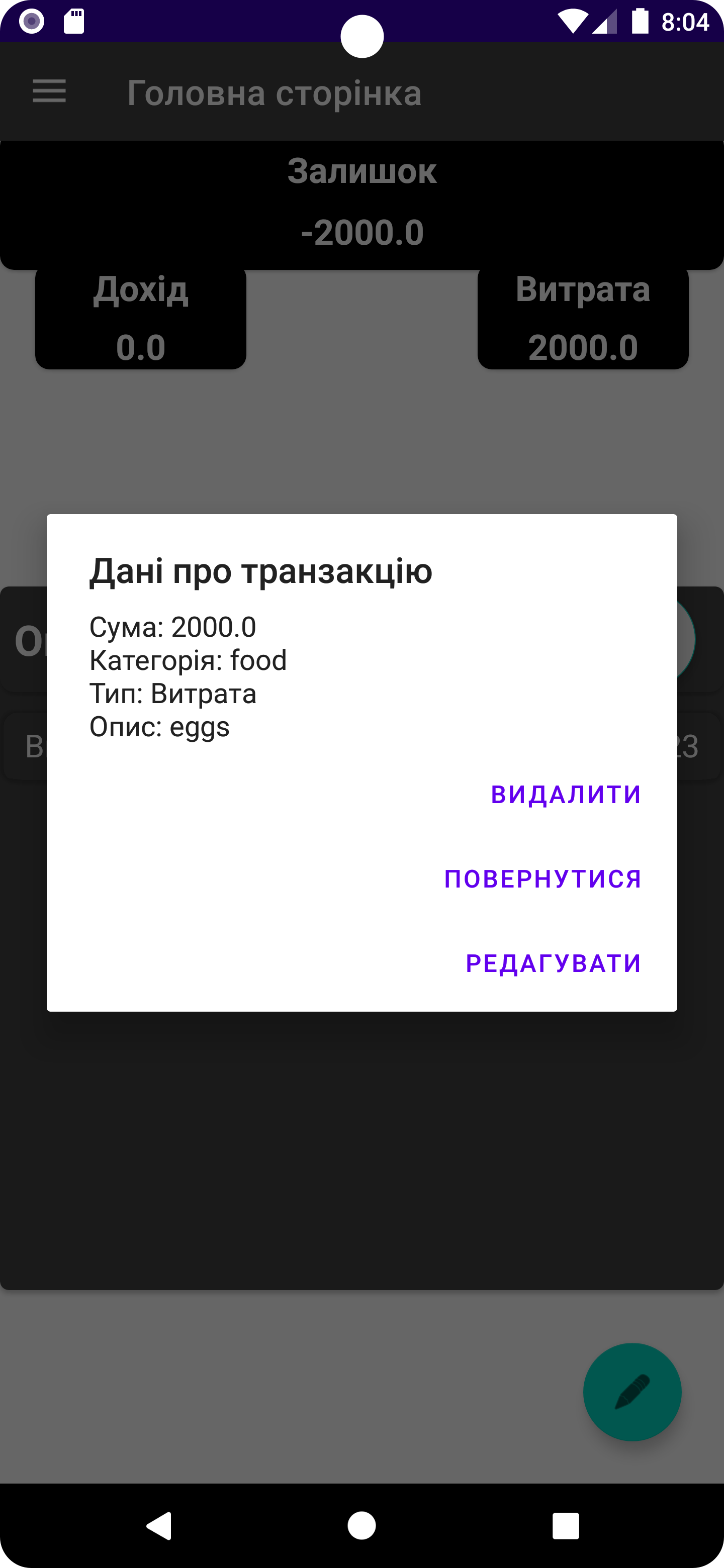


Рис.3.6 – Вікно з даними транзакції

Якщо користувач захоче переглянути транзакції за певний період часу, тоді йому потрібно натиснути на Floating Action Button з малюнком фільтру, яка знаходиться над списком транзакцій, після натискання вікриється вікно (рис.3.7), в якому потрібно обрати період часу та натиснути кнопку «SAVE», після чого головна сторінка оновиться та виведе актуальну інформацію.



Рис.3.7 – Вікно для оберання періоду часу

Якщо користувач захоче очистити дані додатку, то йому потрібно перейти до налаштувань (рис.3.8). Натиснути на кнопку “Так”, та підтвердити операцію (рис.3.9). Після чого база даних буде очищена.



Рис.3.8 – Екран налаштувань

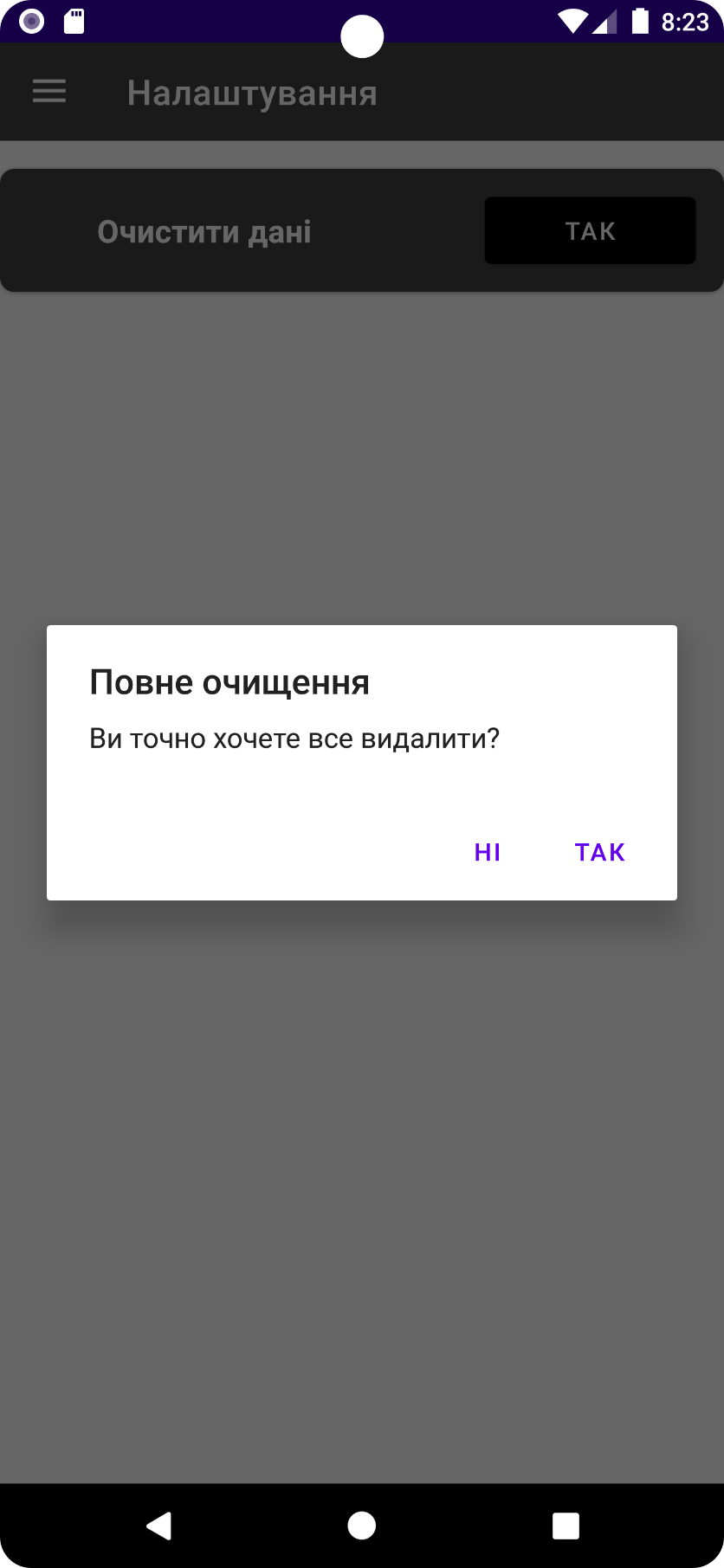


Рис.3.9 – Вікно пітдвердження операціїї очищення

Також користувач може отримати інформацію для зв`язку з розробником, для цього потрібно перейти на екран “Зв`язок з розробником” (рис.3.10).

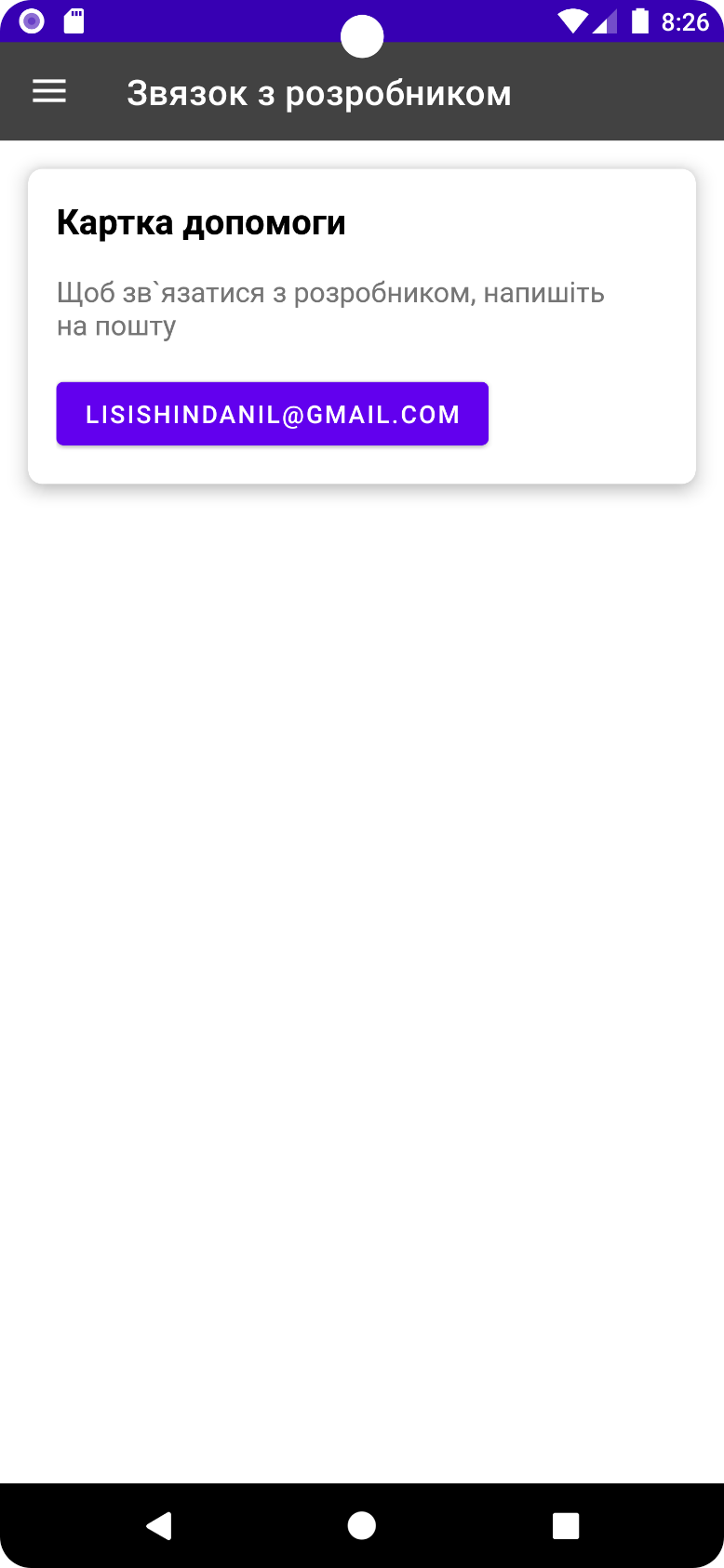


Рис.3.10 - Екран “Зв`язок з розробником”

## **3.5 Вихідний код веб-ресурсу**

Вихідний код програмних класів розробленого веб-ресурсу наведений в Додатку А даної роботи.

## **3. 6 Висновки до третього розділу**

На етапі програмної реалізації додатку для управління складськими запасами було проведено детальне структурування програмного проєкту, визначено ключові класи та їх взаємозв'язок. Розроблено діаграму класів проєкту, яка допомагає візуалізувати основні компоненти програми та їх взаємодію.

Управління вихідним кодом додатку було виконане з використанням системи контролю версій, що забезпечує безпеку коду, можливість відстежування змін і співпрацю між розробниками.

Функціональне тестування розробленого додатку було виконане для перевірки відповідності реалізації встановленим функціональним вимогам. Результати тестування показали, що додаток виконує свої основні функції коректно.

Інструкція користувача додатку була розроблена для надання користувачам інформації про основні функції та способи використання додатку. Вона включає в себе рекомендації з використання і налаштування додатку.

Вихідний код додатку було написано з дотриманням найкращих практик програмування, забезпечуючи його читабельність, масштабованість та легкість подальшого супроводу.

Таким чином, програмна реалізація додатку для управління складськими запасами була успішно здійснена, що відкриває шлях до його впровадження в реальному середовищі.

# **ВИСНОВКИ**

У даній курсовій роботі було розроблено віртуальний гаманець, який допомагає користувачам відстежувати свої доходи, витрати та накопичення. Система надає зручні інструменти для контролю над фінансовим станом, що дозволяє користувачам збільшувати свої накопичення та зменшувати витрати.

У першому розділі досліджено особливості використання технологій, а також проведено аналіз сучасних додатків для управління складськими запасами. Вивчено платформи Android Studio та її функціонал. Виокремлено вимоги до основних функцій додатку. Також було розглянуто інформаційні технології для розробки додатк.

У другому розділі роботи було визначено мету та задачі додатку, виокремлено функціональні та нефункціональні вимоги. Виконано формування користувацьких історій додатку, ідентифіковано архетип додатку. На цій основі було розроблено навігаційний граф та користувацький інтерфейс додатку.

В третьому розділі роботи було здійснено програмну реалізацію додатку. Побудовано структуру програмного проєкту, розроблено діаграму класів. Виконано керування вихідним кодом додатку, проведено функціональне тестування. Також було розроблено інструкцію користувача додатку та представлено вихідний код.

Загалом, мета даної роботи досягнута в повному обсязі, було вирішено всі поставлені завдання, створено фундамент для подальшого розроблення та впровадження мобільного додатку.

# **ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. MySQL Home. URL: https://www.mysql.com/ (дата звернення 01.03.2023).
2. SQLite Home Page. URL: https://sqlite.org/index.html (дата звернення 01.03.2023).
3. Angular is a platform for building mobile and desktop web applications. URL: https://angular.io/ (дата звернення 01.03.2023).
4. Database | Oracle. URL: https://www.oracle.com/database/ (дата звернення 01.03.2023).
5. Firebase Home. URL: https://firebase.google.com/ (дата звернення 01.03.2023).

# **ДОДАТОК А. КОД ПРОГРАМИ**

Код класу MyApplication:

**package** com.example.echoPurse  
  
**import** android.app.Application  
**import** com.example.echoPurse.repositories.TransactionRepository  
**import** com.example.echoPurse.room.TransactionDataBase  
  
  
**class** MyApplication: Application() {  
  
 **private val** database **by** *lazy* **{** TransactionDataBase.getDatabase(**this**)  
 **}  
  
 val** repository **by** *lazy* **{** TransactionRepository(database.transactionDao())  
 **}**}

Код класу MainActivity:

**package** com.example.echoPurse.ui  
  
**import** android.os.Bundle  
**import** androidx.appcompat.app.AppCompatActivity  
**import** androidx.navigation.NavController  
**import** androidx.navigation.findNavController  
**import** androidx.navigation.ui.AppBarConfiguration  
**import** androidx.navigation.ui.navigateUp  
**import** androidx.navigation.ui.setupActionBarWithNavController  
**import** androidx.navigation.ui.setupWithNavController  
**import** com.example.echoPurse.R  
**import** com.example.echoPurse.databinding.ActivityMainBinding  
  
**class** MainActivity : AppCompatActivity() {  
  
 **private lateinit var** binding: ActivityMainBinding  
 **private lateinit var** navController: NavController  
 **private lateinit var** conf: AppBarConfiguration  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 binding = ActivityMainBinding.inflate(*layoutInflater*)  
 setContentView(binding.*root*)  
 setSupportActionBar(binding.actBar.toolbar)  
 navController = *findNavController*(R.id.*fragmentContainerView*)  
 conf = *AppBarConfiguration*(  
 *setOf*(  
 R.id.*main*,  
 R.id.*settings*,  
 R.id.*feedback* ), binding.drawer  
 )  
 *setupActionBarWithNavController*(navController, conf)  
 binding.navView.*setupWithNavController*(navController)  
 }  
  
 **override fun** onSupportNavigateUp(): Boolean {  
 **return** navController.*navigateUp*(conf) || **super**.onSupportNavigateUp()  
 }  
  
}

Код класу TransactionActivity:

**package** com.example.echoPurse.ui  
  
**import** android.os.Bundle  
**import** android.widget.Toast  
**import** androidx.activity.viewModels  
**import** androidx.appcompat.app.AppCompatActivity  
**import** androidx.lifecycle.MutableLiveData  
**import** com.example.echoPurse.MyApplication  
**import** com.example.echoPurse.R  
**import** com.example.echoPurse.databinding.TransactionActivityBinding  
**import** com.example.echoPurse.model.Transaction  
**import** com.example.echoPurse.ui.viewmodels.TransactionViewModel  
**import** java.util.\*  
  
**class** TransactionActivity: AppCompatActivity() {  
  
 **private val** viewModel: TransactionViewModel **by** viewModels() **{** TransactionViewModel.TransactionViewModelFactory((application **as** MyApplication).repository)  
 **}  
  
 private var** editMode = **false  
 private val** oldTransaction = MutableLiveData<Transaction>()  
  
 **override fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
  
 **val** binding = TransactionActivityBinding.inflate(layoutInflater)  
 setContentView(binding.root)  
  
 **val** id = intent.getLongExtra("transaction\_id", -1L)  
 editMode = id != -1L  
  
 **if** (editMode)  
 viewModel.getTransactionById(oldTransaction, id)  
  
 oldTransaction.observe(**this**) **{** it?.let **{** binding.amountEd.editText?.setText(it.amount.toString())  
 binding.categoryEd.editText?.setText(it.category)  
 **}** ?: **throw** java.lang.IllegalArgumentException("transaction\_id is incorrect")  
 **}** binding.cancelBtn.setOnClickListener **{** finish()  
 **}** binding.save.setOnClickListener **{  
 val** transactionType = **when** (binding.radioGroup.checkedRadioButtonId) {  
 R.id.radioButtonExp -> "Витрата"  
 R.id.radioButtonInc -> "Дохід"  
 **else** -> **return**@setOnClickListener  
 }  
  
 **val** date = Date()  
  
 **val** category = binding.categoryEd.editText?.text.toString()  
 **val** amount = binding.amountEd.editText?.text.toString()  
 **val** description = binding.descriptionEd.editText?.text.toString()  
  
 **if** (category.isEmpty() || amount.isEmpty() || transactionType.isEmpty() || description.isEmpty()) {  
 Toast.makeText(**this**, "Ви залишили поле пустим!", Toast.LENGTH\_SHORT).show()  
 **return**@setOnClickListener  
 }  
  
 **if** (editMode) {  
 oldTransaction.value?.let **{** it.amount = binding.amountEd.editText?.text.toString().toDouble()  
 it.category = binding.categoryEd.editText?.text.toString()  
 it.type = transactionType  
 it.description = binding.descriptionEd.editText?.text.toString()  
 it.updated = date  
 viewModel.updateTransaction(it)  
 **}** } **else** {  
 **val** transaction = Transaction(  
 category = binding.categoryEd.editText?.text.toString(),  
 amount = binding.amountEd.editText?.text.toString().toDouble(),  
 type = transactionType,  
 description = binding.descriptionEd.editText?.text.toString(),  
 created = date,  
 updated = date  
 )  
 viewModel.addTransaction(transaction)  
 }  
 finish()  
 **}** }  
}

Код класу TransactionAdapter:

**package** com.example.echoPurse.ui.adapters  
  
  
**import** android.view.LayoutInflater  
**import** android.view.View  
**import** android.view.ViewGroup  
**import** android.widget.TextView  
**import** androidx.recyclerview.widget.DiffUtil  
**import** androidx.recyclerview.widget.ListAdapter  
**import** androidx.recyclerview.widget.RecyclerView  
**import** com.example.echoPurse.R  
**import** com.example.echoPurse.model.Transaction  
**import** java.text.SimpleDateFormat  
**import** java.util.\*  
  
**class** TransactionAdapter(  
 **private val** inflater: LayoutInflater,  
 **private val** onClick: (Transaction) -> Unit  
) :  
 ListAdapter<Transaction, TransactionAdapter.ViewHolder>(UserDiffCallback) {  
  
 **class** ViewHolder(  
 view: View,  
 **val** onClick: (Transaction) -> Unit  
 ) : RecyclerView.ViewHolder(view) {  
 **private val** category = view.findViewById<TextView>(R.id.*categoryTv*)  
 **private val** amount = view.findViewById<TextView>(R.id.*amountTV*)  
 **private val** date = view.findViewById<TextView>(R.id.*dateTv*)  
 **private val** type = view.findViewById<TextView>(R.id.*typeTV*)  
 **private var** transaction: Transaction? = **null  
  
 init** {  
 view.setOnClickListener **{** transaction?.*let* **{** onClick(**it**)  
 **}  
 }** }  
  
 **fun** bind(transaction: Transaction) {  
 **this**.category.*text* = transaction.category  
 **this**.amount.*text* = transaction.amount.toString()  
 **this**.type.*text* = transaction.type  
 **this**.date.*text* = formatDate(transaction.created)  
 **this**.transaction = transaction  
 }  
  
 **private fun** formatDate(date: Date): String {  
 **val** dateFormat = SimpleDateFormat("HH:mm dd.MM.yyyy", Locale.getDefault())  
 **return** dateFormat.format(date)  
 }  
 }  
  
 **override fun** onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {  
 **val** view = inflater.inflate(R.layout.*transaction\_item*, parent, **false**)  
 **return** ViewHolder(view, onClick)  
 }  
  
 **override fun** onBindViewHolder(holder: ViewHolder, position: Int) {  
 **val** transaction = getItem(position)  
 holder.bind(transaction)  
 }  
  
 **object** UserDiffCallback : DiffUtil.ItemCallback<Transaction>() {  
 **override fun** areItemsTheSame(  
 oldItem: Transaction,  
 newItem: Transaction  
 ) = oldItem == newItem  
  
 **override fun** areContentsTheSame(  
 oldItem: Transaction,  
 newItem: Transaction  
 ) = oldItem.category == newItem.category &&  
 oldItem.amount == newItem.amount &&  
 oldItem.created == newItem.created &&  
 oldItem.type == newItem.type  
 }  
  
}

Код класу MainFragmentViewModel:

**package** com.example.echoPurse.ui.viewmodels  
  
**import** androidx.lifecycle.MutableLiveData  
**import** androidx.lifecycle.ViewModel  
**import** androidx.lifecycle.ViewModelProvider  
**import** androidx.lifecycle.*viewModelScope***import** com.example.echoPurse.repositories.TransactionRepository  
**import** kotlinx.coroutines.Dispatchers  
**import** kotlinx.coroutines.launch  
**import** kotlinx.coroutines.withContext  
**import** java.util.Date  
  
**class** MainFragmentViewModel(**private val** repo: TransactionRepository) : ViewModel() {  
  
 **val** transaction = repo.transaction  
 **val** sumAmountIncByDate = MutableLiveData(0.0)  
 **val** sumAmountExpByDate = MutableLiveData(0.0)  
  
 **fun** getSumAmountExpByDate(startDate: Date, endDate: Date) = *viewModelScope*.*launch* **{  
 val** sumExpByDate = repo.getSumAmountExpByDateRange(startDate, endDate)  
 sumAmountExpByDate.*value* = sumExpByDate  
 **}  
  
 fun** getSumAmountIncByDate(startDate: Date, endDate: Date) = *viewModelScope*.*launch* **{  
 val** sumIncByDate = repo.getSumAmountIncByDateRange(startDate, endDate)  
 sumAmountIncByDate.*value* = sumIncByDate  
 **}  
  
 fun** deleteTransaction(id: Long) = repo.deleteTransaction(id)  
  
 **fun** getTransactionsByDateRange(startDate: Date, endDate: Date) = repo.getTransactionsByDateRange(startDate, endDate)  
  
 **class** MainFragmentViewModelFactory(**private val** repo: TransactionRepository) : ViewModelProvider.Factory {  
 **override fun** <T : ViewModel> create(modelClass: Class<T>): T {  
 **return** MainFragmentViewModel(repo) **as** T  
 }  
 }  
}

Код класу SettingsViewModel:

**package** com.example.echoPurse.ui.viewmodels  
  
**import** androidx.lifecycle.ViewModel  
**import** androidx.lifecycle.ViewModelProvider  
**import** com.example.echoPurse.repositories.TransactionRepository  
  
**class** SettingsViewModel(**private val** repo: TransactionRepository) : ViewModel() {  
  
 **fun** deleteAllTransactions() = repo.deleteAllTransactions()  
  
 **class** SettingsViewModelFactory(**private val** repo: TransactionRepository) : ViewModelProvider.Factory {  
 **override fun** <T : ViewModel> create(modelClass: Class<T>): T {  
 **return** SettingsViewModel(repo) **as** T  
 }  
 }  
  
}

Код класу TransactionViewModel:

**package** com.example.echoPurse.ui.viewmodels  
  
**import** androidx.lifecycle.LiveData  
**import** androidx.lifecycle.MutableLiveData  
**import** androidx.lifecycle.ViewModel  
**import** androidx.lifecycle.ViewModelProvider  
**import** com.example.echoPurse.model.Transaction  
**import** com.example.echoPurse.repositories.TransactionRepository  
  
**class** TransactionViewModel(**private val** repo: TransactionRepository) : ViewModel() {  
  
 **fun** addTransaction(transaction: Transaction) = repo.addTransaction(transaction)  
 **fun** getTransactionById(oldTransaction: MutableLiveData<Transaction>, id: Long) = repo.getTransactionById(oldTransaction, id)  
 **fun** updateTransaction(transaction: Transaction) = repo.updateTransaction(transaction)  
  
 **class** TransactionViewModelFactory(**private val** repo: TransactionRepository) : ViewModelProvider.Factory {  
 **override fun** <T : ViewModel> create(modelClass: Class<T>): T {  
 **return** TransactionViewModel(repo) **as** T  
 }  
 }  
  
}

Код абстракного класу TransactionDataBase:

**package** com.example.echoPurse.room  
  
**import** android.content.Context  
**import** androidx.room.Database  
**import** androidx.room.Room  
**import** androidx.room.RoomDatabase  
**import** androidx.room.TypeConverters  
**import** com.example.echoPurse.model.Transaction  
**import** com.example.echoPurse.room.converters.DateConverter  
  
@Database(entities = [Transaction::**class**], version = 3)  
@TypeConverters(DateConverter::**class**)  
**abstract class** TransactionDataBase : RoomDatabase(){  
  
 **abstract fun** transactionDao() : TransactionDao  
  
 **companion object** {  
  
 @Volatile  
 **private var** INSTANCE : TransactionDataBase? = **null  
  
 fun** getDatabase(context: Context): TransactionDataBase {  
 **return** INSTANCE ?: *synchronized*(**this**) **{  
 val** instance = Room.databaseBuilder(context.*applicationContext*,  
 TransactionDataBase::**class**.*java*, "transaction\_db")  
 .fallbackToDestructiveMigration()  
 .build()  
  
 INSTANCE = instance  
 **return** instance  
 **}** }  
 }  
}

Код інтрефейсу TransactionDao:

**package** com.example.echoPurse.room  
  
**import** androidx.lifecycle.LiveData  
**import** com.example.echoPurse.model.Transaction  
**import** androidx.room.\*  
**import** java.util.\*  
  
@Dao  
**interface** TransactionDao {  
  
 @Query("SELECT \* FROM transactions ORDER BY updated DESC")  
 **fun** getAllTransactions(): LiveData<List<Transaction>>  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 **fun** insert(transaction: Transaction)  
  
 @Query("SELECT \* FROM transactions WHERE id = :id")  
 **fun** getTransactionById(id: Long): Transaction  
  
 @Update  
 **fun** updateTransaction(transaction: Transaction)  
  
 @Query("DELETE FROM transactions")  
 **fun** deleteAllTransaction()  
  
 @Query("DELETE FROM transactions WHERE id = :id")  
 **fun** deleteTransaction(id: Long)  
  
 @Query("SELECT \* FROM transactions WHERE created BETWEEN :startDate AND :endDate")  
 **fun** getTransactionsByDateRange(startDate: Date, endDate: Date): List<Transaction>  
  
 @Query("SELECT SUM(amount) FROM transactions WHERE type = 'Дохід' AND created >= :startDate AND created <= :endDate")  
 **fun** getSumAmountIncByDateRange(startDate: Date, endDate: Date): Double  
  
 @Query("SELECT SUM(amount) FROM transactions WHERE type = 'Витрата' AND created >= :startDate AND created <= :endDate")  
 **fun** getSumAmountExpByDateRange(startDate: Date, endDate: Date): Double  
  
}

Код класу DateConverter:

**package** com.example.echoPurse.room.converters  
  
**import** androidx.room.TypeConverter  
**import** java.util.\*  
  
**class** DateConverter {  
  
 @TypeConverter  
 **fun** fromTimestamp(value: Long?) = value?.*let* **{** Date(**it**) **}** @TypeConverter  
 **fun** dateToTimestamp(date: Date?) = date?.*time*}

Код класу TransactionRepository:

**package** com.example.echoPurse.repositories  
  
**import** androidx.lifecycle.LiveData  
**import** androidx.lifecycle.MutableLiveData  
**import** com.example.echoPurse.model.Transaction  
**import** com.example.echoPurse.room.TransactionDao  
**import** kotlinx.coroutines.Dispatchers  
**import** kotlinx.coroutines.withContext  
**import** java.util.\*  
  
**class** TransactionRepository(**private val** transactionDao: TransactionDao) {  
  
 **val** transaction: LiveData<List<Transaction>> = transactionDao.getAllTransactions()  
  
 **fun** addTransaction(transaction: Transaction) = Thread **{** transactionDao.insert(transaction) **}**.start()  
  
 **fun** deleteAllTransactions() = Thread **{** transactionDao.deleteAllTransaction() **}**.start()  
  
 **fun** deleteTransaction(id: Long) = Thread **{** transactionDao.deleteTransaction(id) **}**.start()  
  
 **fun** getTransactionById(oldTransaction: MutableLiveData<Transaction>, id: Long) {  
 Thread **{** transactionDao.getTransactionById(id).*let* **{** oldTransaction.postValue(**it**)  
 **}  
 }**.start()  
 }  
  
 **fun** getTransactionsByDateRange(startDate: Date, endDate: Date): List<Transaction> = transactionDao.getTransactionsByDateRange(startDate, endDate)  
 **fun** updateTransaction(transaction: Transaction) = Thread **{** transactionDao.updateTransaction(transaction) **}**.start()  
  
 **suspend fun** getSumAmountExpByDateRange(startDate: Date, endDate: Date): Double {  
 **return** withContext(Dispatchers.IO) **{** transactionDao.getSumAmountExpByDateRange(startDate, endDate)  
 **}** }  
  
 **suspend fun** getSumAmountIncByDateRange(startDate: Date, endDate: Date): Double {  
 **return** withContext(Dispatchers.IO) **{** transactionDao.getSumAmountIncByDateRange(startDate, endDate)  
 **}** }  
}

Код дата-класу Transaction:

**package** com.example.echoPurse.model  
  
**import** androidx.room.Entity  
**import** androidx.room.PrimaryKey  
**import** java.util.\*  
  
@Entity(tableName = "transactions")  
**data class** Transaction(  
 @PrimaryKey(autoGenerate = **true**) **var** id: Long = 0L,  
 **var** type: String = "",  
 **var** category: String = "",  
 **var** amount: Double,  
 **var** description: String = "",  
 **var** created: Date = Date(),  
 **var** updated: Date = Date()  
)

Код класу FeedbackFragment:

**package** com.example.echoPurse.fragments  
  
**import** android.content.ClipData  
**import** android.content.ClipboardManager  
**import** android.content.Context  
**import** android.os.Bundle  
**import** androidx.fragment.app.Fragment  
**import** android.view.LayoutInflater  
**import** android.view.View  
**import** android.view.ViewGroup  
**import** android.widget.Toast  
**import** androidx.core.content.ContextCompat.getSystemService  
**import** com.example.echoPurse.R  
**import** com.example.echoPurse.databinding.FragmentFeedbackBinding  
**import** com.example.echoPurse.databinding.FragmentSettingsBinding  
  
**class** FeedbackFragment : Fragment() {  
  
 **private var** \_binding: FragmentFeedbackBinding? = **null  
 private val** binding **get**() = \_binding!!  
  
 override **fun** onCreate(savedInstanceState: Bundle?) {  
 **super**.onCreate(savedInstanceState)  
 }  
  
 **override fun** onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 \_binding = FragmentFeedbackBinding.inflate(inflater, container, **false**)  
 **return** binding.*root* }  
  
 **override fun** onViewCreated(view: View, savedInstanceState: Bundle?) {  
 **super**.onViewCreated(view, savedInstanceState)  
  
 binding.feedbackBtn.setOnClickListener **{  
 val** textToCopy = "lisishindanil@gmail.com"  
 **val** clipboardManager = *activity*?.getSystemService(Context.*CLIPBOARD\_SERVICE*) **as** ClipboardManager?  
 clipboardManager?.*let* **{  
 val** clipData = ClipData.newPlainText("email", textToCopy)  
 **it**.setPrimaryClip(clipData)  
 **}** Toast.makeText(requireContext(), "Ви скопіювали пошту", Toast.*LENGTH\_SHORT*).show()  
 **}** }  
  
 **override fun** onDestroyView() {  
 **super**.onDestroyView()  
 \_binding = **null** }  
}

Код класу MainFragment:

**package** com.example.echoPurse.fragments  
  
**import** android.app.AlertDialog  
**import** android.content.Intent  
**import** android.os.Bundle  
**import** android.view.\*  
**import** android.widget.Toast  
**import** androidx.fragment.app.Fragment  
**import** androidx.fragment.app.viewModels  
**import** androidx.recyclerview.widget.LinearLayoutManager  
**import** com.example.echoPurse.MyApplication  
**import** com.example.echoPurse.R  
**import** com.example.echoPurse.databinding.FragmentMainBinding  
**import** com.example.echoPurse.ui.TransactionActivity  
**import** com.example.echoPurse.ui.adapters.TransactionAdapter  
**import** com.example.echoPurse.ui.viewmodels.MainFragmentViewModel  
**import** com.google.android.material.datepicker.MaterialDatePicker  
**import** kotlinx.coroutines.CoroutineScope  
**import** kotlinx.coroutines.Dispatchers  
**import** kotlinx.coroutines.launch  
**import** kotlinx.coroutines.withContext  
**import** java.util.\*  
  
**class** MainFragment : Fragment() {  
  
 **private var** \_binding: FragmentMainBinding? = **null  
 private val** binding **get**() = \_binding!!  
  
 **private val** viewModel: MainFragmentViewModel **by** *viewModels* **{** MainFragmentViewModel.MainFragmentViewModelFactory((requireActivity().*application* **as** MyApplication).repository)  
 **}  
  
 override fun** onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 \_binding = FragmentMainBinding.inflate(inflater, container, **false**)  
 **return** binding.*root* }  
  
 **override fun** onViewCreated(view: View, savedInstanceState: Bundle?) {  
 **super**.onViewCreated(view, savedInstanceState)  
  
 binding.fab.setOnClickListener **{  
 val** i = Intent(requireContext(), TransactionActivity::**class**.*java*)  
 startActivity(i)  
 **}** binding.list.*layoutManager* = LinearLayoutManager(requireContext())  
 **val** adapter = TransactionAdapter(*layoutInflater*) **{  
 val** builder = AlertDialog.Builder(requireContext())  
 builder.setTitle("Дані про транзакцію")  
 builder.setMessage(  
 "Cумa: ${**it**.amount}\n"  
 + "Категорія: ${**it**.category}\n"  
 + "Тип: ${**it**.type}\n"  
 + "Опис: ${**it**.description}"  
 )  
 builder.setNeutralButton("Редагувати") **{**\_, \_ **->  
 val** i = Intent(requireContext(), TransactionActivity::**class**.*java*)  
 i.putExtra("transaction\_id", **it**.id)  
 startActivity(i)  
 **}** builder.setPositiveButton("Видалити") **{** \_, \_ **->  
 val** builder2 = AlertDialog.Builder(requireContext())  
 builder2.setTitle("Видалення транзакції")  
 .setMessage("Ви впевнені, що хочете видалити дану транзакцію")  
 .setPositiveButton("Так") **{** \_, \_ **->** viewModel.deleteTransaction(id = **it**.id)  
 Toast.makeText(requireContext(), "Транзакція успішно видалена", Toast.*LENGTH\_SHORT*).show()  
 **}** .setNegativeButton("Ні") **{** \_, \_ **-> }** .create().show()  
 **}** builder.setNegativeButton("Повернутися") **{** \_, \_ **-> }** builder.create().show()  
 **}** binding.list.*adapter* = adapter  
  
 **fun** showDatePicker() {  
 **val** datePicker = MaterialDatePicker.Builder.dateRangePicker().build()  
 datePicker.addOnPositiveButtonClickListener **{** selection **->  
 val** startDate = Date(selection.first ?: 0)  
 **val** endDate = Date(selection.second ?: 0)  
  
 *CoroutineScope*(Dispatchers.IO).*launch* **{  
 val** transactions = viewModel.getTransactionsByDateRange(startDate, endDate)  
  
 withContext(Dispatchers.Main) **{** adapter.submitList(transactions)  
 viewModel.getSumAmountExpByDate(startDate, endDate)  
 viewModel.getSumAmountIncByDate(startDate, endDate)  
 updateTv()  
 **}  
 }  
 }** datePicker.show(*parentFragmentManager*, "datePicker")  
 }  
  
 binding.analyticBtn.setOnClickListener **{** showDatePicker()  
 **}** viewModel.transaction.observe(*viewLifecycleOwner*) **{** adapter.submitList(**it**)  
 **val** (startDate, endDate) = getStartAndEndOfMonth()  
 viewModel.getSumAmountIncByDate(startDate, endDate)  
 viewModel.getSumAmountExpByDate(startDate, endDate)  
 updateTv()  
 **}** }  
  
 **private fun** getStartAndEndOfMonth(): Pair<Date, Date> {  
 **val** calendar = Calendar.getInstance()  
  
 calendar.*time* = Date()  
  
 calendar.set(Calendar.*DAY\_OF\_MONTH*, 1)  
 calendar.set(Calendar.*HOUR\_OF\_DAY*, 0)  
 calendar.set(Calendar.*MINUTE*, 0)  
 calendar.set(Calendar.*SECOND*, 0)  
 **val** startOfMonth = calendar.*time* calendar.add(Calendar.*MONTH*, 1)  
 calendar.add(Calendar.*SECOND*, -1)  
 **val** endOfMonth = calendar.*time* **return** Pair(startOfMonth, endOfMonth)  
 }  
  
 **private fun** updateTv() {  
 viewModel.sumAmountExpByDate.observe(*viewLifecycleOwner*) **{** expAmount **->** viewModel.sumAmountIncByDate.observe(*viewLifecycleOwner*) **{** incAmount **->  
 val** balance = (incAmount ?: 0.0) - (expAmount ?: 0.0)  
 binding.expTV.*text* = expAmount.toString()  
 binding.incTV.*text* = incAmount.toString()  
 binding.balTV.*text* = balance.toString()  
 **}  
 }** }  
  
  
 **override fun** onDestroyView() {  
 **super**.onDestroyView()  
 \_binding = **null** }  
}

Код класу SettingsFragment:

**package** com.example.echoPurse.fragments  
  
**import** android.app.AlertDialog  
**import** android.os.Bundle  
**import** android.view.LayoutInflater  
**import** android.view.View  
**import** android.view.ViewGroup  
**import** android.widget.Toast  
**import** androidx.fragment.app.Fragment  
**import** androidx.fragment.app.viewModels  
**import** com.example.echoPurse.MyApplication  
**import** com.example.echoPurse.R  
**import** com.example.echoPurse.databinding.FragmentSettingsBinding  
**import** com.example.echoPurse.ui.viewmodels.SettingsViewModel  
  
**class** SettingsFragment : Fragment() {  
  
 **private var** \_binding: FragmentSettingsBinding? = **null  
 private val** binding **get**() = \_binding!!  
  
 **private val** viewModel: SettingsViewModel **by** *viewModels* **{** SettingsViewModel.SettingsViewModelFactory((requireActivity().*application* **as** MyApplication).repository)  
 **}  
  
 override fun** onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 \_binding = FragmentSettingsBinding.inflate(inflater, container, **false**)  
 **return** binding.*root* }  
  
 **override fun** onViewCreated(view: View, savedInstanceState: Bundle?) {  
 **super**.onViewCreated(view, savedInstanceState)  
  
 binding.deleteAllBtn.setOnClickListener **{** deleteAllTransactions()  
 **}** }  
  
 **private fun** deleteAllTransactions() {  
 **val** builder = AlertDialog.Builder(requireContext())  
 builder.setTitle("Повне очищення")  
 builder.setMessage("Ви точно хочете все видалити?")  
 builder.setPositiveButton("Так") **{** \_, \_ **->** viewModel.deleteAllTransactions()  
 Toast.makeText(requireContext(), "Транзакції успішно видалені", Toast.*LENGTH\_SHORT*).show()  
 **}** builder.setNegativeButton("Ні") **{** \_, \_ **-> }** builder.create().show()  
 }  
  
 **override fun** onDestroyView() {  
 **super**.onDestroyView()  
 \_binding = **null** }  
}