

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки,
Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВІДСЛІДКОВУВАННЯ
ФІНАНСОВИХ ВИТРАТ ЗА ГЕОЛОКАЦІЄЮ
SOFTWARE FOR GEOLOCATION-BASED TRACKING OF FINANCIAL
EXPENSES

Виконав: студент(ка) 4 курсу, групи КНТ-217

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

Руснак Н.А.

(прізвище та ініціали)

Керівник Льовкін В.М.

(прізвище та ініціали)

Рецензент Скрупський С.Ю

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ІРЕ, ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерні науки
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 20 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Руснака Назара Андрійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення відслідковування фінансових витрат за геолокацією. Software for Geolocation-Based Tracking of Financial Expenses

керівник проєкту (роботи) Льовкін Валерій Миколайович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “30” березня 2021 року № 103

2. Строк подання студентом проєкту (роботи) травня 2021 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Основні рішення щодо реалізації компонентів системи. 4. Експлуатація, тестування та експериментальне дослідження програми. Додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Льовкін В.М., доцент		
Нормоконтроль	Камінська Ж.К., асистент		

7. Дата видачі завдання “ 10 ” березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2-3 тижні	Розділ 1
3	Розробка архітектури програми	4 тиждень	Розділ 2
4	Розробка програми	5-6 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	7 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	8 тиждень	Додатки
7	Захист роботи	9 тиждень	

Студент(ка)

(підпис) Руснак Н.А.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Льовкін В.М.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра: 124 с., 4 табл., 67 рис., 3 дод., 26 джерел.

ФІНАНСОВИЙ МЕНЕДЖМЕНТ, ГЕОЛОКАЦІЯ, ФІНАНСОВІ ВИТРАТИ, ANDROID, KOTLIN, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ.

Об'єкт дослідження – процес відслідковування фінансових витрат.

Предмет дослідження – програмні засоби відслідковування фінансових витрат за геолокацією.

Мета роботи – розробка спеціалізованого програмного забезпечення відслідковування фінансових витрат для підвищення ефективності подальшого аналізу даних грошових операцій людиною та зменшення витрати часу на процес їх документування з використанням геоінформації як одного з критеріїв класифікації даних.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Kotlin, мобільний пристрій на базі операційної системи Android 11 з підтримкою геолокації та підключенням до мережі інтернет.

Результати. Розроблено програмний застосунок для операційної системи Android для відслідковування фінансових витрат з можливостями перегляду статистики, синхронізації з банком, додання геоінформації до записів про витрати, що дозволяє ефективно проводити документування та аналіз грошових операцій.

Висновки. В ході роботи було проаналізовано існуючі рішення. Розроблено діаграму прецедентів, структурну схему, часткову діаграму класів програмного забезпечення, концептуальну та фізичну схеми бази даних.

Галузь використання – фінансовий менеджмент.

ABSTRACT

Explanatory note to the final qualifying work of the bachelor: 124 pages, 67 figures, 4 tables, 3 appendixes, 26 sources.

FINANCIAL MANAGEMENT, GEOLOCATION, FINANCIAL EXPENSES, ANDROID, KOTLIN, OBJECT-ORIENTED PROGRAMMING.

Object of study is the process of tracking financial expenses.

Subject of study are software for tracking financial expenses by geolocation.

The purpose of the work is to develop specialized software for tracking financial expenses to increase efficiency of further analysis of monetary operations by human and reduce amount of time needed for documenting them using geoinformation as one of criteria for data classification.

Materials, methods and technical means: object-oriented programming, Kotlin programming language, mobile device based on the Android 11 operating system with geolocation support and Internet connection.

Results. Software application for the Android operating system for tracking financial expenses with the ability to view statistics, synchronize monetary operations with the bank, add geographic information to records of expenses, which allows to effectively document and analyze monetary transactions.

Conclusions. In the course of work existing software solutions were analyzed. A use case diagram, structural diagram, partial classes diagram of developed software, conceptual and physical diagrams of the database were developed.

The field of use is financial management.

ЗМІСТ

Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	10
1.1 Постановка проблеми.....	10
1.2 Огляд існуючих програмних засобів.....	10
1.3 Вимоги до програмного забезпечення	14
1.4 Висновки за розділом 1	17
2 Проєктування програмного забезпечення	19
2.1 Вибір мови програмування	19
2.2 Проєктування архітектури програмного забезпечення.....	23
2.3 Проєктування структури бази даних.....	26
2.4 Структурна схема програмного проєкту.....	30
2.5 Висновки за розділом 2	31
3 Основні рішення щодо реалізації компонентів системи.....	32
3.1 Опис використаних бібліотек	32
3.2 Огляд віддаленого API банку	35
3.3 Алгоритм функціонування програмного забезпечення	39
3.4 Опис модулів реалізованої програми	46
3.5 Висновки за розділом 3	60
4 Експлуатація, тестування та експериментальне дослідження програми.....	61
4.1 Призначення програми.....	61
4.2 Умови виконання програми.....	62
4.3 Тестування сценаріїв роботи з програмою	62
4.4 Висновки за розділом 4	83
Висновки.....	84
Перелік джерел посилання	85
Додаток А Технічне завдання.....	88

Додаток Б Текст програми.....	92
Додаток В Слайди презентації	117

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
ART	– Android Runtime;
ISO	– International Organization for Standardization;
JRE	– Java Runtime Environment;
JSON	– JavaScript Object Notation;
JVM	– Java Virtual Machine;
MCC	– Merchant Category Code;
MVC	– Model-View-Controller;
MVP	– Model-View-Presenter;
MVVM	– Model-View-Viewmodel;
XML	– eXtensible Markup Language;
ОС	– операційна система;
СКБД	– система керування базами даних.

ВСТУП

Відслідковування фінансових витрат та надходжень є важливою складовою повсякденного життя сучасної людини – добре задокументований грошовий потік може впливати на велику кількість прийнятих рішень щодо фінансових можливостей та тенденцій.

Об'єктом дослідження є процес відслідковування фінансових витрат, що забезпечують документування грошових операцій з доданням геоінформації, їх класифікацію та представлення цих даних у вигляді, що дозволяє користувачу цих засобів визначити основні вузькі місця бюджету.

Метою роботи є розробка спеціалізованого програмного забезпечення відслідковування фінансових витрат для підвищення ефективності подальшого аналізу даних грошових операцій людиною та зменшення витрати часу на процес їх документування з використанням геоінформації як одного з критеріїв класифікації даних.

Для досягнення поставленої мети у роботі вирішувалися такі завдання:

- аналіз існуючих рішень, виділення їх недоліків та переваг;
- постановка вимог до розроблюваного програмного засобу;
- проєктування архітектури програмного засобу;
- розробка програмного застосунку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка проблеми

Існує декілька способів відслідковування фінансових витрат: записи на папері, використання текстових редакторів та інших неспеціалізованих програм, спеціалізоване програмне забезпечення.

Документування таких даних на папері є дуже неефективним методом через великі часові витрати на ручне написання та класифікацію. Більш того такі записи займають велику кількість місця, схильні до втрати та пошкодження, а також не дозволяють швидко оглянути дані за деякий період часу.

В свою чергу неспеціалізоване програмне забезпечення (таке як текстові та табличні процесори офісних пакетів) краще справляються з задачею документування та огляду, але мають свої обмеження такі як, наприклад, ефективний обсяг даних, перед тим як ці програми будуть втрачати швидкодію. Крім того, такий спосіб документування витрат не є прийнятним на мобільних пристроях з обмеженим дисковим простором та малим розміром екрану з сенсорним введенням даних, що потребує адаптованого інтерфейсу, щоб бути ефективним та зручним, через це виникає потреба у спеціалізованих програмних застосунках.

1.2 Огляд існуючих програмних засобів

Monefy [1] – програмне забезпечення, розроблене компанією Aimbity AS. Має версії для операційних систем Android, IOS та Microsoft Windows.

Застосунок включає наступний перелік функціональних можливостей:

- додання грошових рахунків;
- додання відомостей про грошові операції, що включають дату, грошовий рахунок, категорію та суму;
- додання власних категорій витрат;

- відображення історії операцій у вигляді списку.

Серед переваг можна виділити версії для різних операційних систем, можливість швидко додавати відомості про грошові операції.

В ході експлуатації були знайдені наступні недоліки: використання нетипових для операційної системи Android елементів користувацького інтерфейсу, що збільшує складність використання, присутність реклами та базові функціональні можливості.

Головний екран версії програмного застосунку для операційної системи (OC) Android наведено на рис. 1.1.

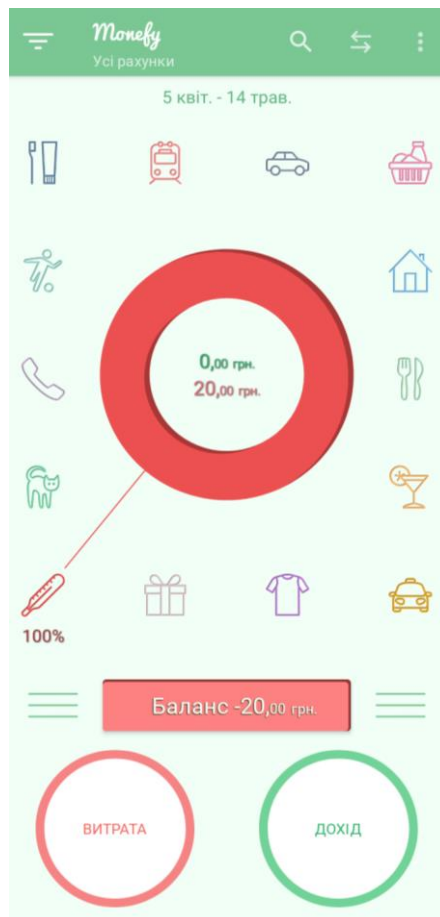


Рисунок 1.1 – Головний екран Monefy [1]

Money manager [2] – програмний застосунок для ОС Android та IOS, який розроблений компанією Innim Mobile Exp.

Серед функцій програмного забезпечення можна виділити наступне:

- додання грошових рахунків;
- додання відомостей про грошові операції, що включають дату, грошовий рахунок, категорію та суму;
- додання власних категорій витрат;
- відображення історії операцій у вигляді списку;
- відображення кругових діаграм витрат та надходжень за категоріями;
- можливість створення нагадувань;
- підтримка декількох видів валют.

До переваг належить простий та зрозумілий користувацький інтерфейс.

Недоліки:

- відсутність підтримки української мови;
- базові функціональні можливості.

Головний екран програмного застосунку зображено на рис. 1.2.

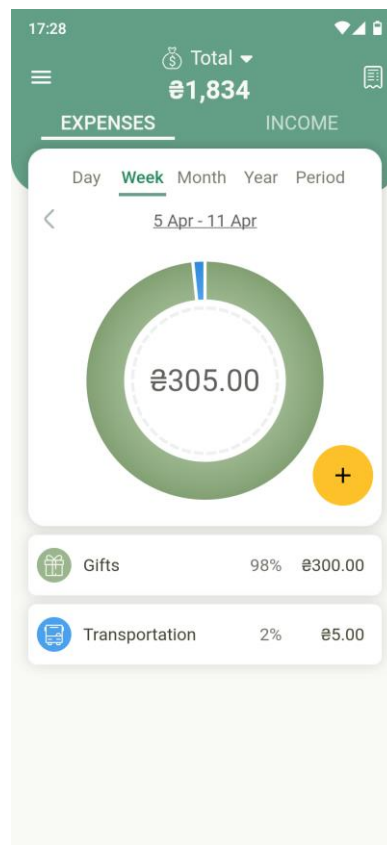


Рисунок 1.2 – Головний екран Money manager [2]

Money Lover [3] – програмне забезпечення відслідковування фінансів від компанії Finsify. Має багату функціональність, включаючи перелічені вище функції аналогів, можливість перегляду звітів за історією операцій, підтримку інструментів планування. До недоліків відноситься велика кількість реклами та часті рекламні повідомлення у безкоштовній версії, що заважають роботі з застосунком. На рис. 1.3 зображено екран огляду операцій.

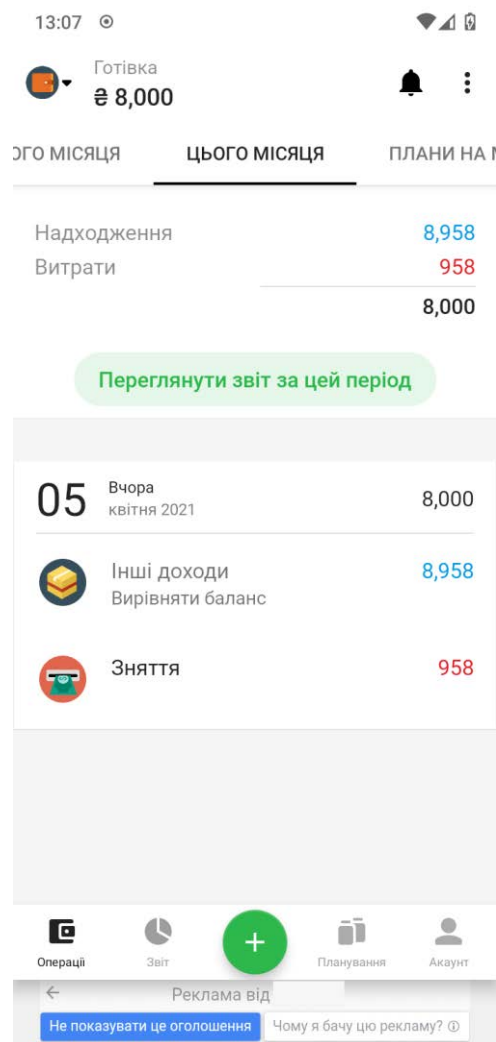


Рисунок 1.3 – Екран огляду операцій Money Lover [3]

У табл. 1.1 наведено порівняння описаних вище програмних застосунків.

Таблиця 1.1 – Порівняння програмних аналогів

Критерій порівняння	Monefy	Money manager	Money Lover
Підтримка операційних систем	Android, IOS, Microsoft Windows	Android, IOS	Android, IOS
Підтримка української мови	Є	Немає	Є
Функціональні можливості	Базові	Базові	Розширені
Наявність реклами	Немає	Немає	Є
Використання типових елементів інтерфейсу	Ні	Так	Так
Підтримка збереження геоінформації	Ні	Ні	Ні

1.3 Вимоги до програмного забезпечення

Для розробки програмного забезпечення була обрана платформа Android через широку розповсюдженість мобільних пристроїв на її основі та їх апаратних можливостей, які включають підключення до мережі інтернет, датчики геолокації та інші. Згідно з джерелом [4], станом на 2019-й рік на базі ОС Android працюють майже два з половиною мільярда пристроїв.

Android – операційна система з відкритим кодом для мобільних приладів, що розробляється корпорацією Google. Як проект з відкритим

кодом, метою Android є уникнення ситуації, коли один гравець галузі може обмежити або контролювати нововведення будь-якого іншого гравця [5]. Тому Android є повноцінною якісною операційною системою для повсякденних пристроїв, яку можна перенести майже на будь-який пристрій [5].

Проаналізувавши існуючі програмні рішення, можна побачити, що більшість з них не використовує переваги платформи, на якій працюють, зокрема геолокаційні можливості.

Виходячи з вищесказаного, програмний застосунок повинен:

- мати простий та сучасний інтерфейс;
- працювати на сучасних версіях ОС Android;
- використовувати переваги та можливості пристроїв на базі

Android.

Використання геоінформації в програмному застосунку дозволить вести статистику для більш глибокого аналізу проведених грошових операцій, зручно відображати таку інформацію на мапі міста для її більш зручного огляду користувачем. Проте необхідно брати до уваги, що геоінформація користувача – це конфіденційна інформація, він має знати, що його місцезнаходження використовується для виконання деяких функцій програмного забезпечення та давати на це згоду. Через це у платформи Android є ряд обмежень для захищення користувачів від додатків, які зловживають відстеженням місцезнаходження. Починаючи з Android 8.0, були введені обмеження на отримання місцезнаходження користувача, коли програмний застосунок не є активним та працює як фоновий процес [6]. Також з Android 10 була введена індикація використання місцезнаходження, як частина стандартного користувацького інтерфейсу та новий рівень дозволу на використання місцезнаходження фоновими процесами [7].

Майже постійна підключеність мобільних пристроїв до мережі інтернет та розповсюдженість відкритих Application Programming Interface

(API) зовнішніх систем дозволяє реалізувати тісну взаємодію між програмним застосунком та такими сервісами.

На рис. 1.4 зображена use case діаграма для роботи з розроблюваним програмним застосунком.

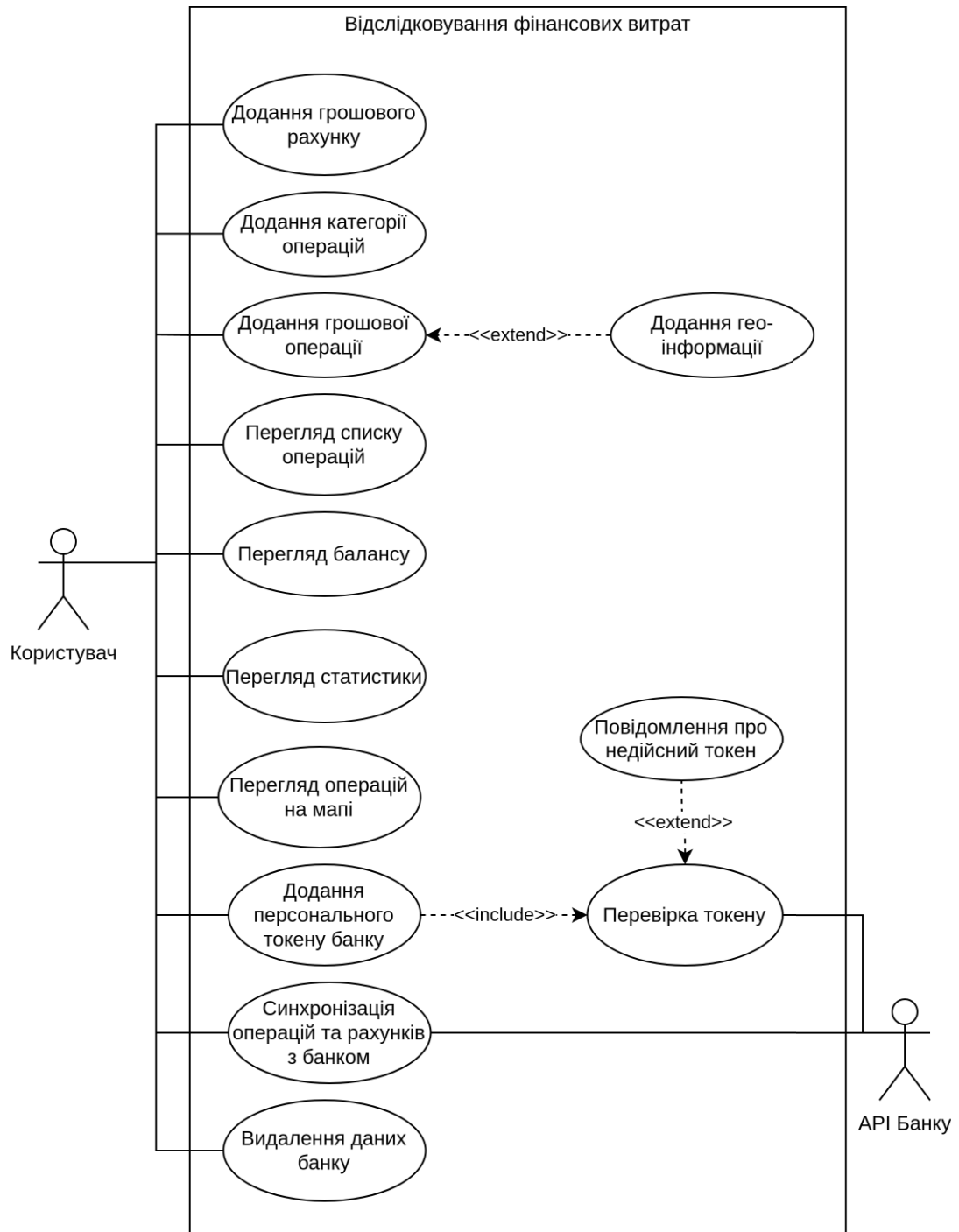


Рисунок 1.4 – Use case діаграма

Виходячи з діаграми на рис. 1.4, вимоги до функціональності програмного забезпечення включають:

- додання грошового рахунку з визначенням назви, балансу та типу;
- додання категорій грошових операцій для їх класифікації;
- додання грошової операції, що має включати грошовий рахунок, опис, суму операції, категорію, час та дату та, необов'язково, геодані де проходила ця операція (координати та назва місця). При цьому, додання грошової операції оновлює баланс відповідного рахунку;
- користувач має мати змогу переглядати список грошових операцій та застосовувати фільтр за рахунком до нього;
- перегляд сумарного балансу за всіма грошовими рахунками;
- перегляд статистики грошових операцій та станів балансу рахунку;
- перегляд операцій з геоданими за останній на мапі;
- додання персонального токена банку, для подальшої синхронізації рахунків та грошових операцій користувача. Програмний засіб повинен перевірити цей токен на коректність та вивести повідомлення для користувача при помилці;
- синхронізація рахунків та грошових операцій користувача з банком, якщо персональний токен є збереженим у програмному застосунку;
- при необхідності користувач має мати можливість видалити персональний токен банку та усі синхронізовані рахунки та операції з програмного застосунку.

1.4 Висновки за розділом 1

У розділі було виконано аналіз предметної області роботи. Були розглянуті існуючі програмні застосунки, які вирішують поставлене

завдання, виділенні їх переваги та недоліки. За результатами огляду було висунуто ряд вимог до розроблюваного програмного забезпечення.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір мови програмування

Починаючи з першої версії, ОС Android мала підтримку мови програмування Java.

Java – це одночасно платформа та мова програмування. Як платформа, це Java runtime environment (JRE), що включає до себе Java virtual machine (JVM), яка є програмним середовищем, де виконуються програмні засоби [8]. Як мова програмування Java є високорівневою, об'єктно орієнтованою мовою [8].

Для виконання програмний текст мовою Java має бути скомпільовано у Java байткод.

Java байт код – це високо оптимізований набір інструкцій, які спроектовано для виконання за допомогою JVM [9]. Оригінальна JVM була розроблена як інтерпретатор для байт коду [9]. На противагу цьому підходу багато сучасних мов програмування спроектовані бути скомпільованими до коду для конкретної архітектури процесору через стурбованість про швидкодію результуючого програмного застосунку [9]. Коли програма інтерпретується, вона, у загальному випадку, виконується повільніше, ніж та сама програма, яку було скомпільовано до виконуваного коду, але у випадку з Java відмінність у швидкодії є дуже малою через використання оптимізованого байт коду, що виконується у JVM швидше, ніж очікується [9]. Крім того, такий підхід дозволяє спростити виконання програм у різноманітних оточеннях через те, що тільки JRE повинно бути портовано на обрану платформу [9]. Якщо JRE існує на обраній системі, будь-яка Java програма може бути виконана на ній [9].

Для виконання програмних застосунків, написаних мовою Java, в ОС Android використовується Android Runtime (ART).

При компілюванні Android застосунку Java байткод за допомогою програми «dx» перетворюється до файлу Dalvik Executable (DEX), після чого може бути виконаний за допомогою віртуального процесору Dalvik [10].

Процесор Dalvik є дещо схожим на JVM: вони двоє є віртуальними процесорами [10]. Але робота JVM заснована на роботі зі стеком, а Dalvik засновано на операціях з регістрами, код з якими можливо оптимізувати для прискорення роботи та зменшення об'єму [10].

Оскільки в 2017 році платформа Android отримала офіційну підтримку мови програмування Kotlin, а у 2019 році ця мова програмування була об'явлена головною мовою для розробки програмних застосунків для ОС Android [11], [12], то саме Kotlin розглядається в якості основного вибору для реалізації програмного застосунку.

Kotlin розроблюється компанією JetBrains з 2011-го року [13]. Подібно до того, як C++ спочатку задумувався як «кращий C», Kotlin спочатку був орієнтований на те, щоб бути «кращою Java» [13]. Kotlin прагматично вибирає лише найуспішніші та корисні функції з інших мов програмування – після того, як ці функції були перевірені і було доведено, що вони є особливо цінними [13].

Код мовою програмування Kotlin може бути скомпільований для підтримки наступних платформ:

- JVM: код компілюється до байт коду JVM, що може працювати на будь-якій Java Virtual Machine;
- Android: код мовою програмування Kotlin може бути скомпільовано для роботи у ART;
- JavaScript для роботи у браузері;
- машинний код для конкретних процесорів та операційних систем [13].

Крім того, код Kotlin, що буде скомпільовано для роботи у JVM або ART, є повністю сумісним з кодом Java, їх можна використовувати пліч-о-

пліч в одному програмному проєкті та використовувати бібліотеки, які написано на Java, з Kotlin програмними застосунками [13].

Головні відмінності мов програмування Java та Kotlin наведено у табл. 2.1.

Таблиця 2.1 – Вибір мови програмування

Предмет порівняння	Мови програмування	
	Java	Kotlin
1	2	3
Null безпека	Немає	Розділення типів на ті, що можуть бути null, і ті, що не можуть.
Функції розширення	Додання функцій до існуючих класів можливо лише через спадкування	Є
Класи даних	Тільки створення власноруч	Вбудований клас за ключовим словом «data»
Розумні перетворення типів	Необхідно проводити перевірку елементів до перетворення	Вбудована підтримка перевірки (ключове слово «is»)
Типи	Статична типізація. Необхідно завжди вказувати тип змінної	Статична типізація. Автоматичне визначення типу на основі присвоюваного значення.

Продовження таблиці 2.1

1	2	3
Функціональне програмування	Підтримка лямбда функцій.	Підтримка лямбда функцій, перевантаження операторів, операцій над списками та інше.
Статичні члени класу	Має	Немає, але є альтернативи
Наявність підтримки Android	Є	Є, рекомендовано корпорацією Google

Крім того, відповідно до [12] мова Kotlin має наступні переваги:

- дозволяє робити більше за допомогою меншої кількості коду порівняно з Java, що збільшує продуктивність та зменшує кількість програмних помилок;
- код мовою Kotlin є більш безпечним на виняткові ситуації посилення на null;
- дозволяє використовувати код та бібліотеки Java в одному проєкті, якщо на те є потреба;
- підтримка співпроцедур (coroutines), які суттєво спрощують управління фоновими завданнями для всього, від мережових запитів до доступу до локальних даних [12].

Виходячи з вищесказаного, для розроблення програмного застосунку було обрано мову програмування Kotlin.

2.2 Проектування архітектури програмного забезпечення

При розробці Android застосунків розпоширеними шаблонами архітектури є: Model View Controller (MVC), Model View Presenter (MVP), Model View ViewModel (MVVM).

На рис. 2.1 зображена схема роботи архітектурного шаблону MVC.

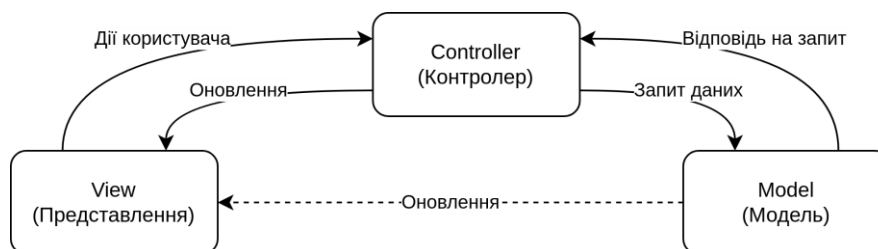


Рисунок 2.1 – Схема роботи MVC

У шаблоні MVC зазвичай використовуються три компоненти:

- view (представлення) – користувацький інтерфейс, відображає дані на екрані та передає події, здійснені користувачем до контролеру: у випадку Android додатку це користувацький інтерфейс у форматі extensible markup language (XML) розмітки;
- controller (контролер) – працює з view та model, отримує події користувача від view, відправляє запит даних до model, оброблює отримані від model дані та оновлює view, у випадку Android це класи activity або fragment;
- model (модель) – містить логіку щодо даних та працює з базою даних, інколи може оновлювати view [14].

До переваг MVC можна віднести:

- відділення логіки в моделі;
- підтримка асинхронних технік;
- модифікації окремих частин не потребують змін у всьому застосунку [14].

Недоліки:

- великий розмір контролеру, який, через це, важко підтримувати;
- ускладнене тестування;
- підвищена складність застосунку [14].

До складу MVP входять: model (модель), view (представлення) та presenter (презентер).

До моделі входить логіка додатку та робота з даними (запис до диску, робота з базою даних або віддаленим сервером). Модель спілкується з презентером, але на відміну від MVC не може працювати напряму з view [14].

View відображає та контролює інтерфейс користувача, не містить логіки, повністю контролюється презентером. Включає до себе розмітку та частину коду відображення даних у ній (на прикладі Android застосунку це розмітка XML у зв'язку з класами activity або fragment) [14].

Презентер представляє дані, отримані від моделі, керує поведінкою додатку та рівнем представлення. Взаємодія представлення та моделі проходить через презентер [14].

На рис. 2.2 зображена схема роботи MVP.

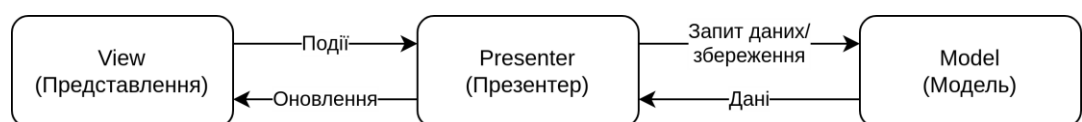


Рисунок 2.2 – Схема роботи MVP

Переваги MVP:

- прибрана вся логіка з представлення, що полегшує внесення змін;
- програмний код більш зручно підтримувати та оновлювати;
- краще тестується, ніж MVC;
- можливе повторне використання view та presenter [14].

До недоліків включають:

- тісне об'єднання view та presenter ;

- велика кількість інтерфейсів для взаємодії між шарами архітектурного шаблону;
- підвищення кількості програмного коду [14].

Архітектурний шаблон MVVM вирішує проблеми тісного об'єднання шарів та великої кількості інтерфейсів для взаємодії.

MVVM включає в себе:

- model (модель), що включає в себе логіку даних, локальні та віддалені джерела даних та репозиторій, якій працює з цими джерелами та надає доступ до них, відповідно запитів від ViewModel;
- view (представлення) – передає події до ViewModel, але не отримує відповіді на них напряду, не містить логіки роботи з даними (тільки відображає їх), для отримання відповіді на події користувацького інтерфейсу відслідковує дані, які надає ViewModel;
- ViewModel – містить логіку користувацького інтерфейсу, є мостом між view та model, не має прямого посилання на view, що дає змогу повторно використовувати його з іншими view, сповіщає про зміну в даних підписаних до них view [14].

На рис. 2.3 зображена схема MVVM.

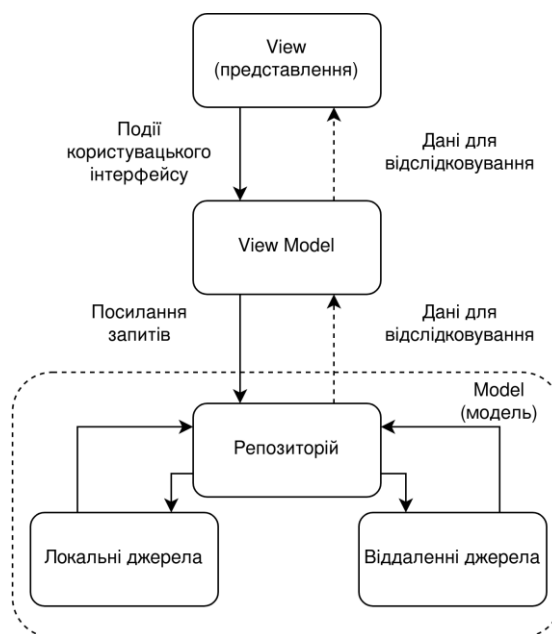


Рисунок 2.3 – Схема MVVM

До переваг MVVM можна віднести:

- відсутність тісного об'єднання між view та ViewModel;
- відсутність необхідності в інтерфейсі між view та ViewModel;
- легке тестування та керований подіями програмний код [14].

Недоліки :

- необхідність створювати спостережувані компоненти до кожного елемента користувацького інтерфейсу;
- досить велика кількість програмного коду [14].

Зважаючи на інформацію наведену вище та офіційні рекомендації [15] щодо проєктування архітектури, для розробки програмного застосунку було обрано використання шаблону проєктування MVVM.

2.3 Проєктування структури бази даних

ОС Android надає інструменти для роботи з системою керування реляційних баз даних SQLite для збереження даних програмних застосунків.

СКБД SQLite використовується через наступний ряд переваг

- мінімальні затрати ресурсів – не потребує серверу, представлення у вигляді файлу, коли база даних не використовується, вона не займає ресурси мобільного пристрою;
- оптимізація для одного користувача (у загальному випадку тільки один програмний застосунок користується відповідною базою даних);
- надійність та швидкість – підтримка транзакцій та використання оптимізованого коду мовою програмування C [10].

На рисунку 2.4 зображена концептуальна схема бази даних.

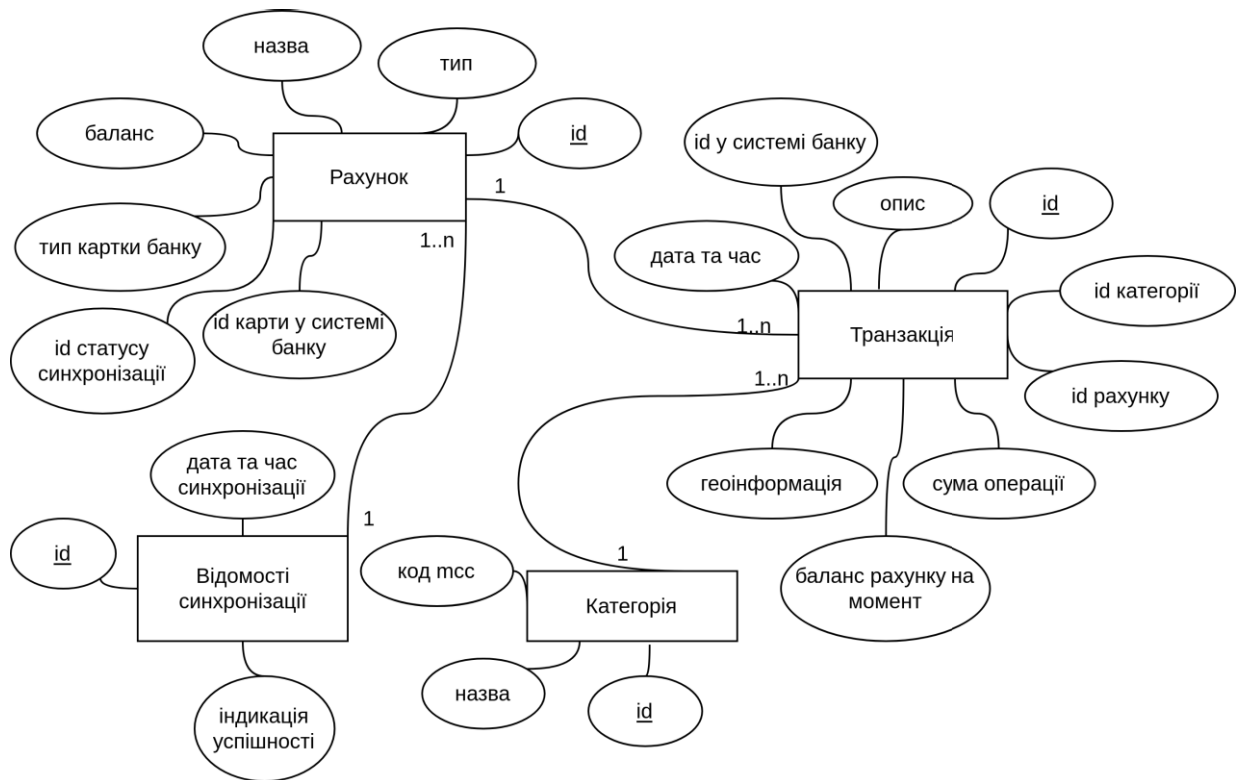


Рисунок 2.4 – Концептуальна схема бази даних

Розроблюваний програмний застосунок має наступні сутності бази даних:

- рахунок – містить інформацію про стан рахунку;
- транзакція – містить інформацію про грошову операцію;
- категорія – категорія транзакції;
- відомості синхронізації – інформація про спробу синхронізації з банком.

СКБД SQLite підтримує роботу з наступними типами даних:

- NULL – значення NULL;
- INTEGER – число зі знаком, яке зберігається в 1, 2, 3, 4, 5, 6 або 8 байтах залежно від величини значення;
- REAL – число з плаваючою крапкою, зберігається у 8-ми байтах;
- TEXT – рядок тексту, зберігається використовуючи кодування бази даних (UTF-8, UTF-16BE або UTF-16LE);
- BLOB – бінарні дані [16].

На рис. 2.5 зображена фізична схема бази даних програмного застосунку.

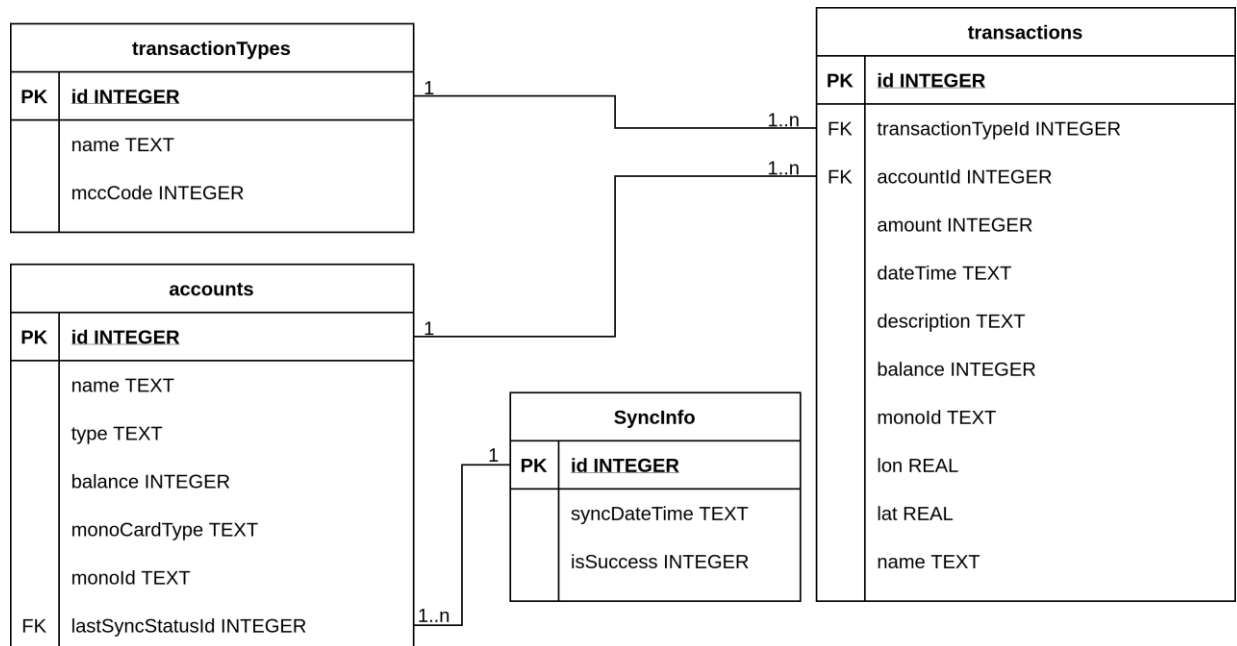


Рисунок 2.5 – Фізична схема бази даних

Таблиця «transactionTypes» містить дані про категорії грошових операцій та включає наступні поля:

- «id» типу integer – ідентифікатор категорії грошової операції, використовується як первинний ключ;
- «name» типу text – назва категорії;
- «mccCode» типу integer – код категорії продавця merchant category code (MCC) використовується для класифікації грошових операцій при синхронізації з банком. Для категорій які було додано користувачем дорівнює -1.

Таблиця «accounts» містить відомості про грошові рахунки користувача та має наступні поля:

- «id» типу integer – ідентифікатор грошового рахунку, використовується як первинний ключ;
- «name» типу text – назва грошового рахунку;
- «type» типу text – тип грошового рахунку;

- «balance» типу integer – баланс рахунку у копійках;
- «monoCardType» типу text – тип картки банку для синхронізації;
- «monoId» типу text – ідентифікатор картки у системі банку, з яким проходить синхронізація.

Таблиця «transactions» містить інформацію про грошові операції та має наступні поля:

- «id» типу integer – ідентифікатор грошової операції, використовується як первинний ключ;
- «transactionTypeId» типу integer – ідентифікатор категорії грошової операції, зовнішній ключ;
- «accountId» типу integer – ідентифікатор грошового рахунку, зовнішній ключ;
- «amount» типу integer – сума грошової операції у копійках, негативна для витрат та позитивна для надходжень;
- «dateTime» типу text – дата та час проведення операції у вигляді рядку;
- «description» типу text – опис грошової операції;
- «balance» типу integer – баланс грошового рахунку у копійках на момент здійснення грошової операції;
- «monoId» типу text – ідентифікатор грошової операції у системі банку, з яким проводиться синхронізація;
- «lon» типу real – довгота місця проведення операції;
- «lat» типу real – широта місця проведення операції;
- «name» типу text – назва місця проведення грошової операції.

Таблиця «SyncInfo» містить дані про спроби синхронізації та має наступні поля:

- «id» типу integer – ідентифікатор спроби синхронізації, використовується як первинний ключ;
- «syncDateTime» типу text – дата та час спроби синхронізації у вигляді рядку;

– «isSuccess» типу `integer` – індикатор успішності спроби синхронізації.

2.4 Структурна схема програмного проєкту

На рис. 2.6 зображена структурна схема програмного проєкту.

Розроблюваний програмний проєкт складається з наступних структурних складових:

- «res» – містить ресурси програмного застосунку, такі як XML розмітку користувацького інтерфейсу, рядки локалізації, зображення тощо;
- «com.harukeyua.fintrack» – пакет, який містить у собі весь програмний код застосунку;
- «ui» – містить класи керування користувацьким інтерфейсом та пакет «adapters» який, в свою чергу, містить у собі класи представлення списків у інтерфейсі користувача;
- «viewmodels» – містить класи `ViewModel`, що забезпечують логіку поведінки програмного застосунку;
- «geros» – класи репозиторієв роботи з даними;
- «data» – класи роботи з базою даних, також містить пакети «model» та «api» які мають класи представлення моделей даних застосунку та віддаленого API банку відповідно;
- «api» – містить класи роботи з віддаленими ресурсами мережі інтернет;
- «di» – класи-утиліти ін'єкції залежностей;
- «workers» – класи фонових процесів програмного застосунку;
- «utils» – класи-утиліти для повторного використання.

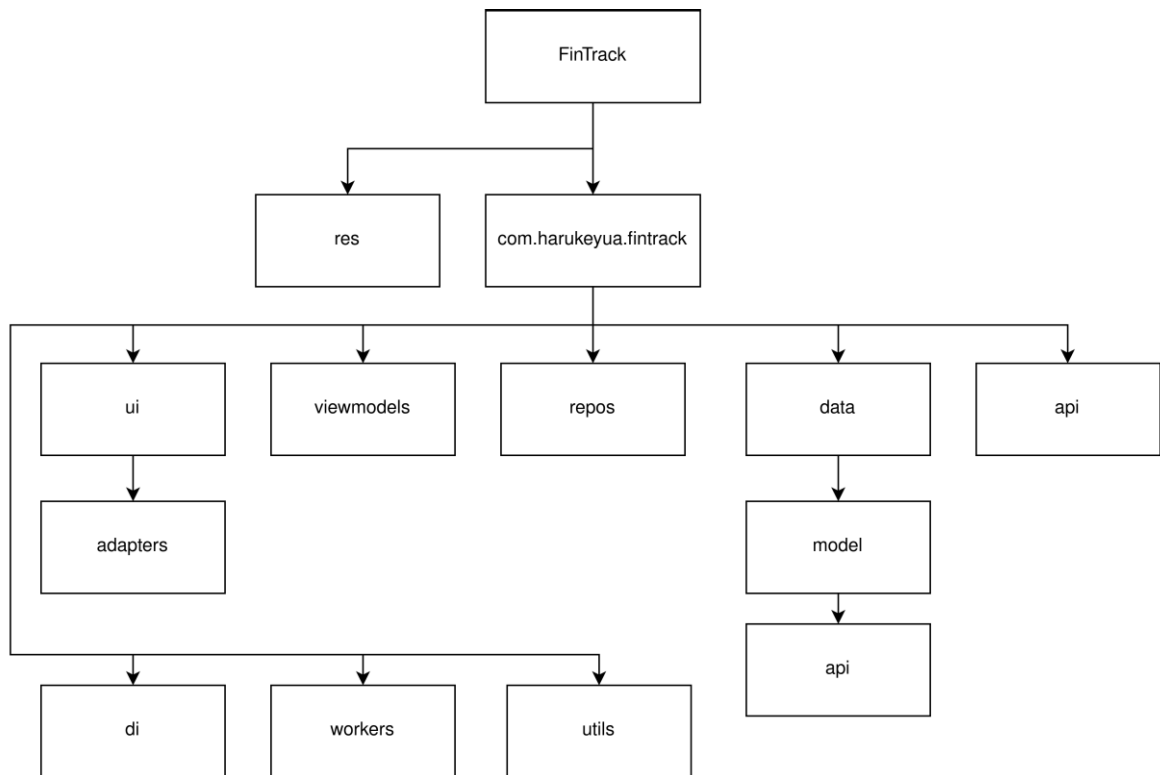


Рисунок 2.6 – Структурна схема програмного проєкту

2.5 Висновки за розділом 2

У розділі було розглянуто та обрано мову для розробки програмного застосунку. Також було вирішене питання щодо використання архітектурного шаблону програмного забезпечення. Розроблено структуру бази даних та розглянуто структуру програмного проєкту.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

3.1 Опис використаних бібліотек

Для прискорення темпів розробки програмного забезпечення, підвищення його якості та зменшення помилок в роботі при розробці було використано ряд широко поширених бібліотек для розробки Android застосунків.

Підтримка інструментів мови програмування Kotlin: стандартна бібліотека Kotlin, «Core KTX» та бібліотека для підтримки співпроцедур Kotlin. Стандартна бібліотека Kotlin («kotlin-stdlib») включає до себе:

- функції вищого порядку, що реалізують ідіоматичні шаблони;
- функції розширення, що забезпечують операції запиту для колекцій та послідовності;
- утиліти для роботи з рядками та послідовностями символів;
- розширення для класів JDK, що роблять зручною роботу з файлами, введенням-виведенням та потоками [17].

В якості інструменту обробки формату даних JavaScript Object Notation (Json) використовується бібліотека «Moshi» від компанії Square. «Moshi» – це сучасна бібліотека JSON для Android та Java. Вона полегшує синтаксичний аналіз JSON, дозволяючи перетворювати його об'єкти на об'єкти Java. Більш того, бібліотека підтримує null типи та стандартні параметри Kotlin [18].

Для роботи з протоколом HTTP застосовані інструменти HTTP клієнту «Retrofit» компанії Square, що дозволяє перетворювати HTTP API віддалених сервісів на інтерфейси Java. Має підтримку багатьох рішень з синтаксичного аналізу результатів, в тому числі і «Moshi», та співпроцедур Kotlin для забезпечення асинхронної роботи.

Отримання геолокаційних даних з датчиків мобільного пристрою відбувається за допомогою бібліотеки «Google Play location services», яка

надає інструменти отримання останнього місця знаходження мобільного пристрою з необхідною точністю, об'єднуючі інформацію базових технологій визначення місцезнаходження, таких як GPS та Wi-Fi [19].

«Maps SDK for Android » та «Places SDK for Android» від корпорації Google дозволяють відображати геоінформаційні дані на мапі «Google Maps» у мобільному Android застосунку та дізнаватися інформацію про місця на ній. Базові функціональні можливості цих бібліотек (відображення мапи, без панорамних знімків, та отримання базової інформації про місця) доступні безкоштовно для мобільних додатків відповідно до [20], [21], що дозволило застосувати їх при розробці програми.

Відображення статистичної інформації у вигляді графіків використовує бібліотеку «MPAndroid Chart», що підтримує лінійні графіки, кругові діаграми, тощо, а також забезпечує велику кількість налаштувань для них.

Платформа Android надає доступ до набору бібліотек «Android Jetpack», що офіційно підтримуються корпорацією Google та рекомендовані до використання для створення сучасних програмних застосунків. Цей набір бібліотек допомагає слідувати найкращим практикам, зменшує кількість типового програмного коду та гарантує одну й ту саму поведінку на різних версіях Android [22]. Нижче наведений перелік використаних бібліотек, які входять до набору «Android Jetpack».

Для забезпечення зворотної сумісності з застарілими версіями ОС Android застосовується офіційна бібліотека «Appcompat», що забезпечує однакову поведінку API та зовнішній вигляд на різноманітних версіях.

В свою чергу бібліотека «Core KTX» надає доступ до функцій розширення Kotlin для класів Android API та офіційних бібліотек.

Використання бібліотеки компонентів графічного інтерфейсу «Material Components» надає змогу використовувати сучасні типові елементи інтерфейсу користувача та забезпечує зовнішній вигляд програмного забезпечення, який відповідає зовнішньому вигляду операційної системи.

Для створення комплексних та респонсивних графічних інтерфейсів користувача для різних типів та розмірів екранів мобільних пристроїв використовується бібліотека «Constraint layout».

Бібліотеки «Recyclerview» та «Paging 3» надають можливості імплементації швидких списків відображення даних. В першому випадку це досягається за рахунок повторного використання вже відображених елементів списку, що економить ресурси мобільного пристрою, а в другому – за рахунок завантаження великого обсягу даних поступово «сторінками» повної кількості елементів з мережі інтернет чи локального сховища за потреби. Для досягнення високої швидкодії ці бібліотеки використовуються разом.

Для забезпечення шаблону проектування MVVM використовується бібліотека «lifecycle» куди входять тип даних з можливістю відслідковування в ньому змін «LiveData» та клас «ViewModel» для створення на його основі власних класів ViewModel.

Для взаємодії з локальною базою даних використовується «Room». Бібліотека «Room» забезпечує рівень абстракції над SQLite для зручного доступу до бази даних зі збереженням всіх можливостей СКБД SQLite [23]. Зокрема, «Room» має наступні переваги:

- перевірка запитів SQL під час компіляції програмного проекту;
- анотації, які мінімізують повторюваний та схильний до помилок шаблонний код;
- зручна міграція баз даних [23].

Відповідно до [23] рекомендується використовувати «Room» замість безпосередньої роботи з API SQLite.

Ін'єкція залежностей класів відбувається за допомогою «Hilt». «Hilt» забезпечує стандартний спосіб використання ін'єкції залежностей у додатку, забезпечуючи контейнери для Android класів та автоматично керуючи їх життєвим циклом [24].

Створення фонових процесів синхронізації даних з банком використовує «WorkManager», що дозволяє планувати коли, з якою періодичністю, вони будуть виконуватися та відстежувати їх стан.

Для реалізації навігації між екранами програмного застосунку використовується бібліотека «Navigation Component», що надає інструменти для легкого створення графів переходів та передачі даних між точками навігації.

Захист персональних даних користувача (персональний токен банку) забезпечується зберіганням у зашифрованому вигляді алгоритмом AES-256 GCM інструментами бібліотеки «security-crypto».

3.2 Огляд віддаленого API банку

Одним з перших українських банків, який надає доступ до свого API, є сервіс банку Monobank «Monobank open API». У цього API існує дві версії: публічна та корпоративна. Публічна версія дозволяє отримувати стан рахунків користувача та виписку грошових операцій за останню 31 добу, але має обмеження на кількість запитів за короткий проміжок часу та необхідність ручного отримання персонального токена авторизації банку користувачем. В свою чергу корпоративна версія API має всі можливості публічної, а також можливість автоматичної авторизації та перевірки доступу до користувацьких даних та немає обмежень на кількість запитів. Відповідно до [25] використання публічного API дозволяється для розробки функціональних можливостей бібліотек та програм, якщо дані користувачів не передаються на сервер розробника програмних проєктів, що є прийнятним для використання розроблюваним програмним застосунком відслідковування фінансових витрат через відсутність функціональних вимог, що порушують умови використання API.

Виходячи з вищевказаного та досить тривалого процесу отримання доступу до корпоративного API, було прийнято рішення про використання

його публічної версії, функціональні можливості якого задовольняють функціональним вимогам програмного застосунку.

Вхідною точкою публічної версії «Monobank open API» є [26], що одночасно служить місцем авторизації та отримання персонального токена користувачем.

Для отримання інформації про стан рахунків користувача необхідно зробити GET запит за шляхом «/personal/client-info» з додаванням персонального токена до заголовку запиту з назвою параметра «X-Token». Приклад відповіді API у форматі Json наведено на рис. 3.1, проте треба зазначити, що конфіденційні дані (такі як номери, ідентифікатори рахунків, баланси) було замасковано.

```

1  {
2      "clientId": "6X*****uxc",
3      "name": "Руснак Назар",
4      "webHookUrl": "",
5      "permissions": "psf",
6      "accounts": [
7          {
8              "id": "jec1*****n0Hw",
9              "currencyCode": 980,
10             "cashbackType": "UAH",
11             "balance": 10000000,
12             "creditLimit": 10000000,
13             "maskedPan": [
14                 "537541*****4755"
15             ],
16             "type": "black",
17             "iban": "UA543*****565"
18         }
19     ]
20 }

```

Рисунок 3.1 – Приклад відповіді на запит рахунків користувача

Поля відповіді представляють наступні лані:

- «clientId» типу string – ідентифікатор клієнта;
- «name» типу string – ім'я клієнта;
- «webHookUrl» типу string – URL для отримання інформації про нову транзакцію;
- «permissions» типу string – незадокументоване поле;
- масив «accounts» – перелік доступних рахунків.

В свою чергу, кожний об'єкт рахунку містить [25]:

- «id» типу string – ідентифікатор рахунку;
- «currencyCode» типу number – код валюти рахунку відповідно до ISO 4217;
- «cashbackType» типу string – тип кешбеку який нараховується на рахунок, може мати наступні значення: «None», «UAH», «Miles»;
- «balance» типу number – баланс рахунку в мінімальних одиницях валюти (копійках, центах);
- «creditLimit» типу number – кредитний ліміт;
- «maskedPan» – масив, що містить замасковані номери карток за рахунком типу string;
- «type» типу string – тип рахунку, може мати наступні значення: «black», «white», «platinum», «iron», «for», «yellow»;
- «iban» типу string – міжнародний номер банківського рахунку [25].

Отримання виписки грошових операцій за деякий час виконується за допомогою GET запиту за шляхом «/personal/statement/{account}/{from}/{to}», де «account» – ідентифікатор рахунку чи «0» для рахунку за замовчанням, «from» – початковий час виписки у секундах, «to» – необов'язковий параметр кінця часу виписки у секундах. Необхідним є, як і з отриманням даних користувача, додання персонального токена до заголовку запиту з назвою параметра «X-Token». Приклад відповіді API у форматі Json наведено на рис. 3.2, проте треба зазначити, що конфіденційні дані (такі як номери, ідентифікатори рахунків, баланси) було замасковано.

Результатом запиту є масив елементів транзакцій з наступними полями:

- «id» типу string – унікальний id транзакції;
- «time» типу number – час транзакції в секундах;
- «description» типу string – опис грошової операції;

- «mcc» типу number – код типу транзакції (Merchant Category Code), відповідно до ISO 18245, який було скориговано банком;
- «originalMcc» типу number – оригінальний код типу транзакції Merchant Category Code (MCC), відповідно ISO 18245;
- «amount» типу number – сума у валюті рахунку в мінімальних одиницях валюти (копійках, центах);
- «operationAmount» типу number – сума у валюті транзакції в мінімальних одиницях валюти (копійках, центах);
- «currencyCode» типу number – код валюти рахунку відповідно до ISO 4217;
- «commissionRate» типу number – розмір комісії в мінімальних одиницях валюти (копійках, центах);
- «cashbackAmount» типу number – розмір кешбеку в мінімальних одиницях валюти (копійках, центах);
- «balance» типу number – баланс рахунку в мінімальних одиницях валюти (копійках, центах);
- «hold» тип Boolean – статус блокування суми;
- «receiptId»: типу string – номер квитанції для check.gov.ua;
- «comment» типу string – коментар до переказу, уведений користувачем: якщо не вказаний, поле буде відсутнім [25].

```

1  [
2    {
3      "id": "dP*****fn",
4      "time": 1618909845,
5      "description": "Продукти",
6      "mcc": 5815,
7      "originalMcc": 5815,
8      "amount": -5900,
9      "operationAmount": -5900,
10     "currencyCode": 980,
11     "commissionRate": 0,
12     "cashbackAmount": 0,
13     "balance": 1000000,
14     "hold": true,
15     "receiptId": "*****_*****_*****_*****"
16   }
17 ]

```

Рисунок 3.2 – Приклад відповіді на запит виписки

3.3 Алгоритм функціонування програмного забезпечення

На рис. 3.3 наведено схему функціонування програмного забезпечення.

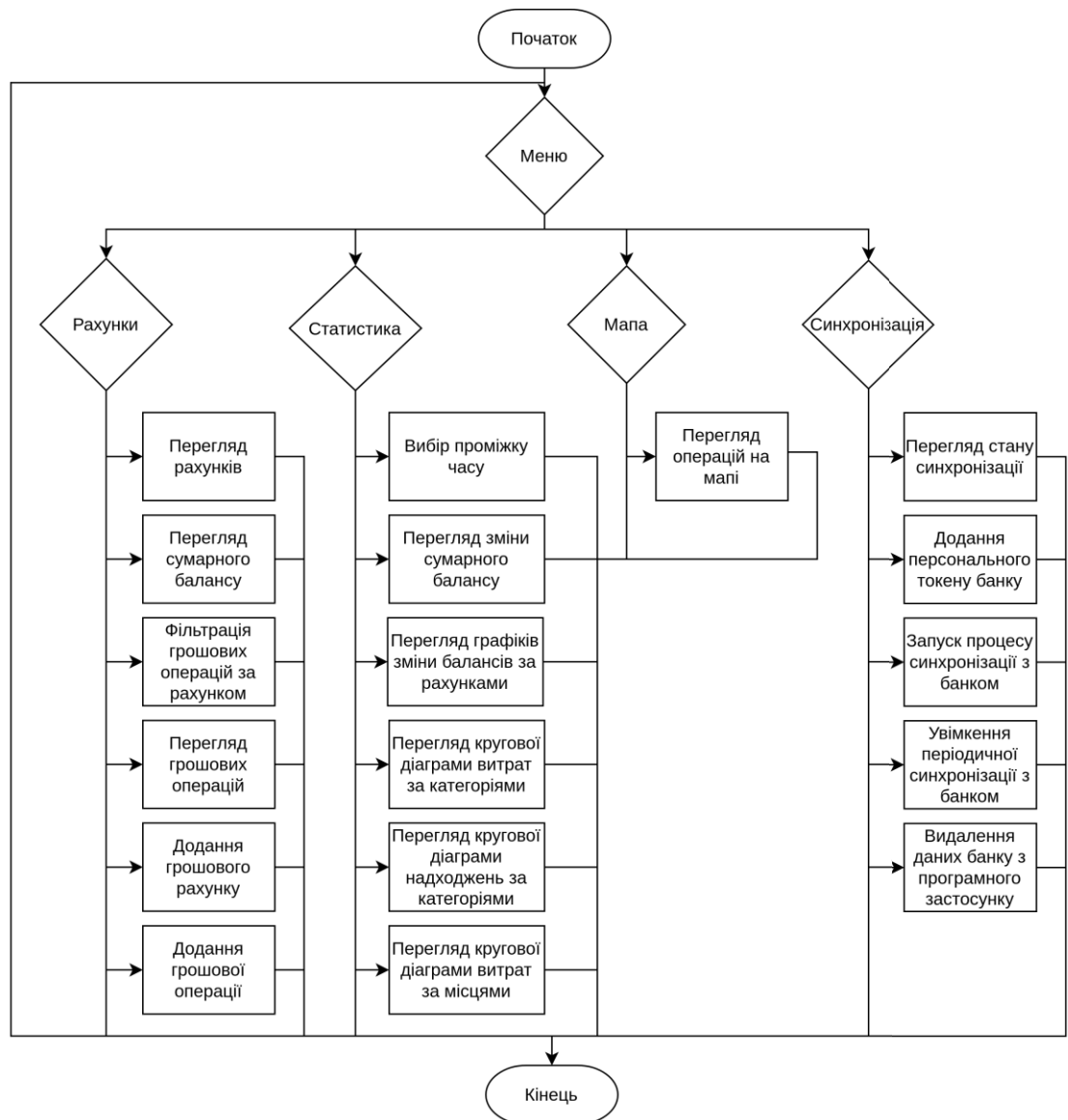


Рисунок 3.3 – Схема функціонування програмного забезпечення

Функціональні можливості програмного застосунку поділені на модулі, які користувач може обрати у будь-який час роботи з застосунком.

За замовчанням при запуску програмного засобу запускається модуль «рахунки», як один з головних, що включає до себе наступні функціональні можливості:

- перегляд грошових рахунків;
- перегляд сумарного балансу;
- фільтрація грошових операцій за рахунками;
- перегляд грошових операцій;
- додання грошового рахунку;
- додання грошової операції.

Наступним по частоті використання є «статистика», яка надає доступ до:

- вибору проміжку часу статистики;
- перегляду зміни сумарного балансу за обраний проміжок часу;
- перегляду графіків зміни балансів грошових рахунків за обраний проміжок часу;
- перегляду кругової діаграми витрат за категоріями за обраний проміжок часу;
- перегляду кругової діаграми нарахувань за категоріями за обраний проміжок часу;
- перегляду кругової діаграми витрат за місцями за обраний проміжок часу.

«Мапа» надає можливість перегляду грошових операцій на карті міста.

В свою чергу «синхронізація» містить наступні можливості:

- перегляд стану останньої спроби синхронізації – час та успішність;
- додання персонального токена банку, для можливості синхронізації даних грошових рахунків з ним;
- запуск процесу синхронізації власноруч;
- увімкнення періодичної синхронізації з банком: якщо увімкнено, програмний застосунок, навіть якщо є закритим, буде кожні 12 годин запускати процес синхронізації, якщо мобільний пристрій підключено до мережі інтернет;

– видалення даних банку з програмного застосунку, що включає до себе: синхронізовані рахунки, грошові операції та персональний токену користувача.

Нижче наведено алгоритм роботи синхронізації грошових операцій та рахунків з банку. Він складається з наступних частин: синхронізація станів грошових рахунків, синхронізація грошових операцій, керування порядком синхронізації зі сповіщення користувача про помилки.

На рис. 3.4 наведено схему алгоритму синхронізації станів грошових рахунків з банком.

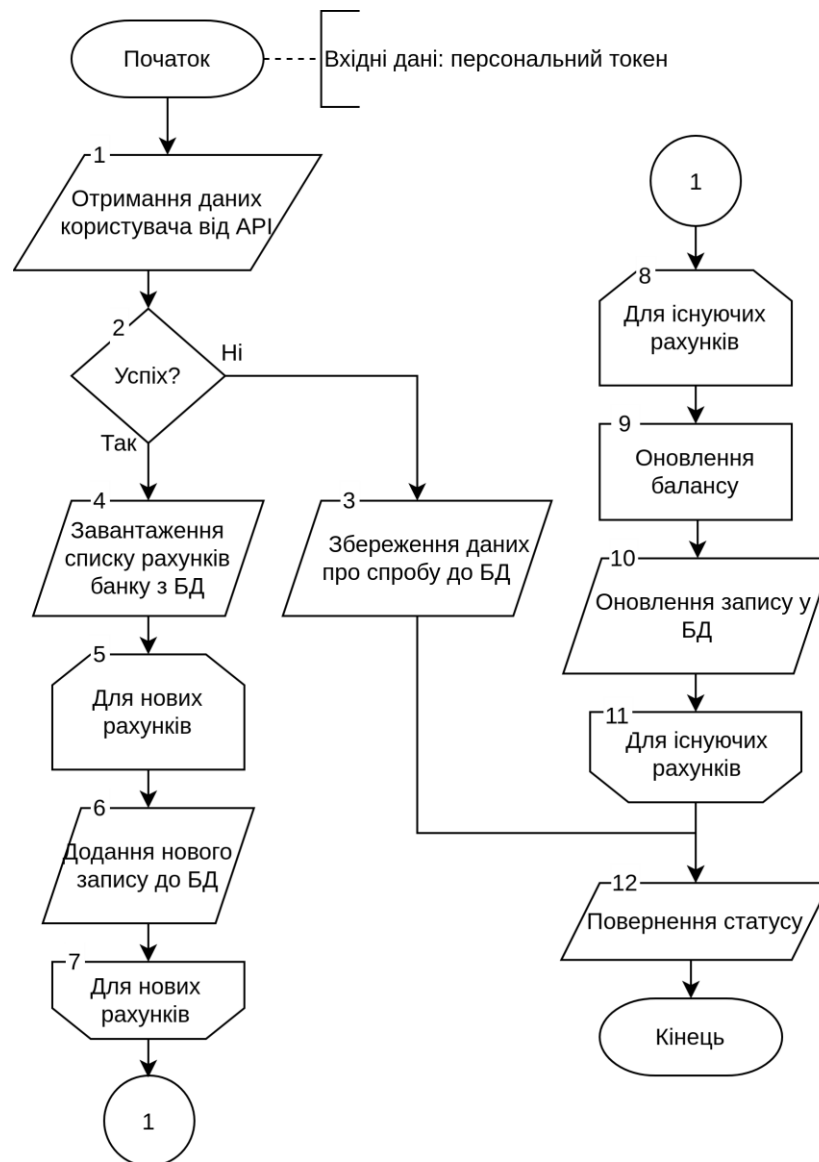


Рисунок 3.4 – Схема алгоритму синхронізації грошових рахунків

Вхідними даними до алгоритму є персональний токен для авторизації у віддаленій системі банку.

На першому кроці відбувається отримання відомостей про стан грошових рахунків користувача. При виникненні помилки (наприклад, при відсутності інтернет з'єднання, чи недійсному персональному токені) алгоритм переходить до кроку 3, де зберігає дані (такі як час та стан) спроби синхронізації до бази даних, та повертає статус запиту.

При успішному отриманні даних про рахунки користувача, на кроці 4 програмний засіб завантажує список рахунків банку, вже збережених в базі даних, після чого, на кроках 5-7 відбувається створення нових записів для всіх нових грошових рахунків банку.

На кроках 8-11 програмний застосунок оновлює стан балансу вже існуючих до початку синхронізації рахунків з подальшим їх оновленням у базі даних. Після завершення оновлення, відбувається повернення статусу запиту, після чого алгоритм завершує свою роботу.

На рис. 3.5 наведено схему алгоритму синхронізації грошових операцій з банком.

Вхідними даними до алгоритму є персональний токен користувача, для авторизації у віддаленому сервісі банку та список тсс кодів з описами українською мовою, для класифікації категорій операцій, отримуваних від банку.

Алгоритм починає свою роботу з кроку 1, на якому відбувається завантаження інформації про останню успішну операцію, що включає її час. Після чого, на кроці 2, з бази даних завантажується список рахунків банку, для яких потрібно провести синхронізацію грошових операцій.

При наявності успішної спроби синхронізації значення змінної «from» встановлюється у значення часу останньої успішної спроби, інакше – у значення поточного часу мінус 31 день (кроки 3-5).

На наступних кроках (6-15) у циклі, для всіх рахунків банку користувача, відбувається процес синхронізації. Для окремого рахунку на

кроці 7 програмний застосунок отримує виписку від часу «from» через віддалений сервіс банку. Якщо запит був успішний, переходимо до кроку 9, інакше – до кроку 16, де відбувається збереження даних про час і стан спроби до БД та повернення стану запити, після чого алгоритм завершує роботу.

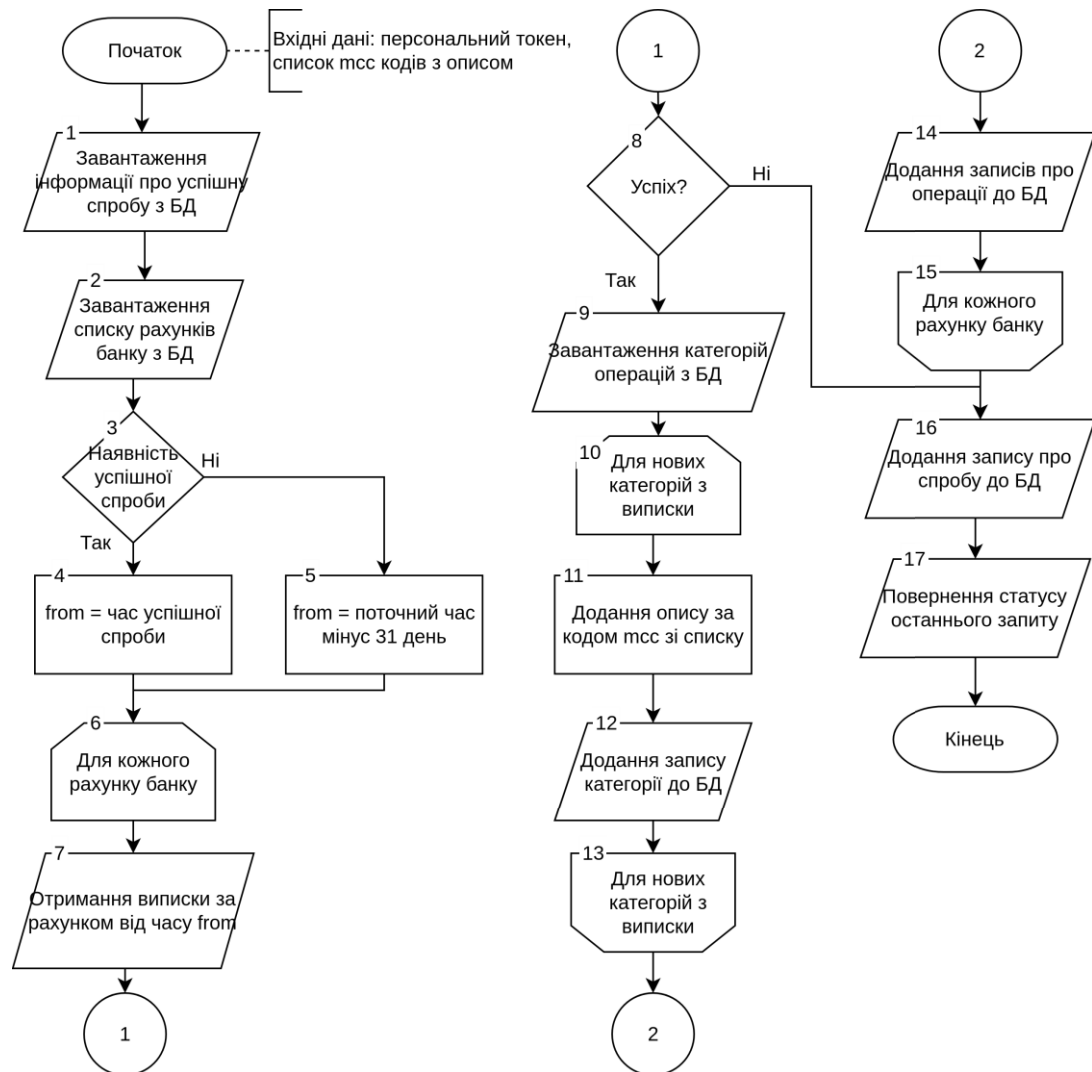


Рисунок 3.5 – Схема алгоритму синхронізації операцій

Після успішного отримання виписки за окремим рахунком на кроці 9 відбувається завантаження списку категорій операцій з бази даних застосунку. Для нових кодів тсс з виписки (яких не має в базі даних) на кроці 11 додається опис з списку тсс кодів, після чого створюється запис у базі даних про нову категорій (крок 12).

Після додання нових категорій (якщо такі були) до бази даних додаються записи про грошові операції з виписки банку.

Після завершення циклу з синхронізації операцій для всіх рахунків, дані про спробу синхронізації додаються до бази даних на кроці 16 та на кроці 17 повертається стан останнього запиту, після чого алгоритм завершує роботу.

На рис. 3.6 зображена схема алгоритму сервісу синхронізації.

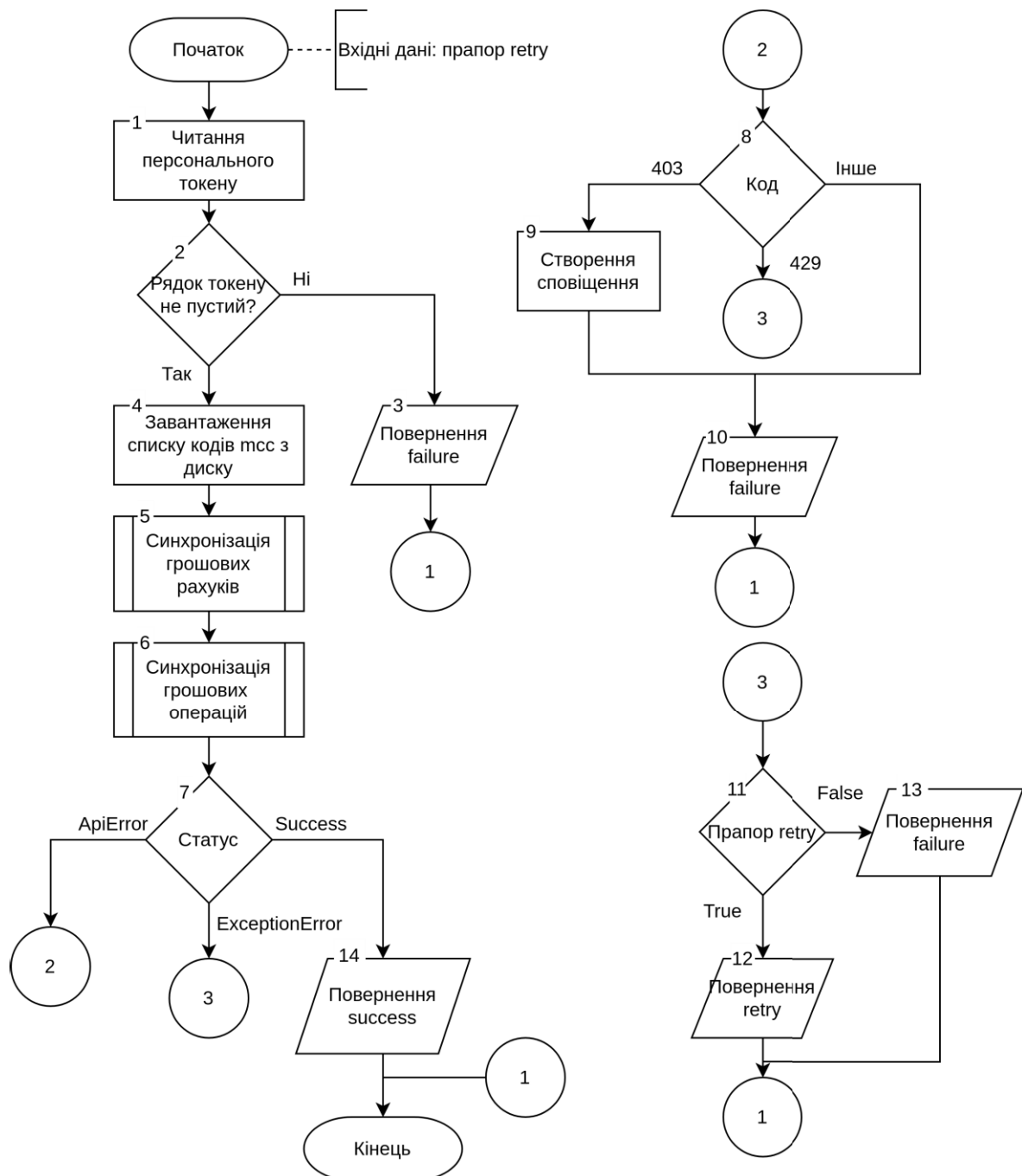


Рисунок 3.6 – Схема алгоритму сервісу синхронізації

Вхідними даними до алгоритму є прапор «retry».

Алгоритм фонового сервісу синхронізації виконує наступні кроки.

Крок 1. Зчитування персонального токenu користувача.

Крок 2. Якщо рядок токenu не порожній, перейти до кроку 4, інакше до кроку 3.

Крок 3. Повернення значення «failure», завершення алгоритму.

Крок 4. Завантаження списку кодів МСС з диску.

Крок 5. Запуск процесу синхронізації грошових рахунків банку.

Крок 6. Запуск процесу синхронізації грошових операцій для рахунків банку.

Крок 7. Якщо повернений статус синхронізації «ApiError» (помилка, яку було повернено віддаленим сервісом) перейти до кроку 8, «ExceptionError» (помилка введення або виведення) – перейти до кроку 11, «Success» (дані було успішно синхронізовано) – перейти до кроку 14.

Крок 8. Якщо код помилки віддаленого сервісу дорівнює «403», перейти до кроку 9, «429» – перейти до кроку 11, інший код – до кроку 10.

Крок 9. Створення сповіщення про недійсність персонального токenu банку для звернення уваги користувача інструментами системи сповіщень Android.

Крок 10. Повернення значення «failure», завершення алгоритму.

Крок 11. Якщо прапор «retry» дорівнює «true», перейти до кроку 12, інакше до кроку 13.

Крок 12. Повернення значення «retry», що є індикацією для бібліотеки «WorkManager», про те, що фоновий процес слід перезапустити через деякий час. Завершення алгоритму.

Крок 13. Повернення значення «failure», завершення алгоритму.

Крок 14. Повернення значення «success», завершення алгоритму.

3.4 Опис модулів реалізованої програми

На рис. 3.7 наведено файли реалізації програмного застосунку.

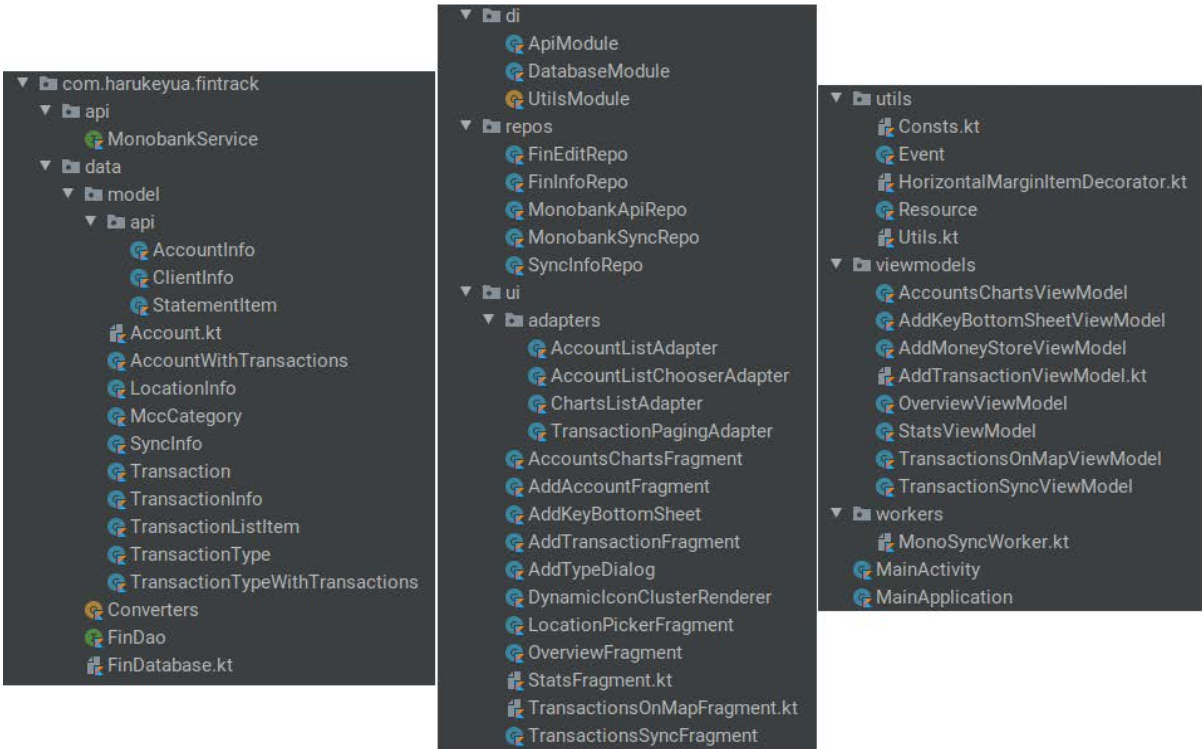


Рисунок 3.7 – Файли реалізації програмного застосунку

У таблиці 3.1 наведено опис застосування реалізованих модулів програмного застосунку.

Таблиця 3.1 – Опис реалізованих модулів

Назва пакету	Назва модулю	Опис
1	2	3
api	MonobankService	Інтерфейс опису запитів до віддаленого сервісу банку для бібліотеки «Retrofit»
data.model.api	AccountInfo	Клас представлення відомостей про рахунок у Monobank Open Api

Продовження таблиці 3.1

1	2	3
data.model.api	ClientInfo	Клас представлення відомостей про клієнта у Monobank Open Api
data.model.api	StatementItem	Клас представлення грошової операції у Monobank Open Api
data.model	Account	Клас представлення грошового рахунку користувача у базі даних
data.model	AccountType	Клас-перелічення (enum class) можливих типів грошових рахунків (картка, гаманець або банк)
data.model	AccountWithTransactions	Клас, що представляє рахунок з його грошовими операціями
data.model	LocationInfo	Клас для збереження геоданих
data.model	MccCategory	Клас представлення для збереження mcc кодів з описом
data.model	SyncInfo	Клас представлення інформації про спробу синхронізації
data.model	Transaction	Клас представлення грошової операції
data.model	TransactionInfo	Клас представлення базової інформації про грошову операцію
data.model	TransactionListItem	«sealed» клас елементу списку грошових операцій користувацького інтерфейсу

Продовження таблиці 3.1

1	2	3
data.model	TransactionListItem.Item	Клас спадок «sealed» класу TransactionListItem, що являє собою елемент грошової операції списку користувачького інтерфейсу
data.model	TransactionListItem.Separator	Клас спадок «sealed» класу TransactionListItem, що являє собою роздільний елемент списку користувачького інтерфейсу, містить дату операцій
data.model	TransactionType	Клас представлення категорії транзакції
data	Converters	Клас, що містить інструменти конвертації дати у рядок тексту для збереження у БД
data	FinDao	Інтерфейс для використання бібліотекою «Room», містить опис запитів до БД
data	FinDatabase	Абстрактний клас опису бази даних для бібліотеки «Room»
di	ApiModule	Клас надання залежності MonobankService
di	DatabseModule	Клас надання залежності FinDao

Продовження таблиці 3.1

1	2	3
di	UtilsModule	Клас надання залежностей: майстер-ключа додатку, захищеного сховища
repos	FinEditRepo	Клас репозиторію внесення змін до БД
repos	FinInfoRepo	Клас репозиторію отримання даних з БД
repos	MonobankApiRepo	Клас репозиторію перевірки персонального токена банку
repos	MonobankSyncRepo	Клас репозиторію синхронізації рахунків та грошових операцій з банком
repos	SyncInfoRepo	Клас репозиторія надання інформації про стан спроб синхронізації
ui.adapters	AccountListAdapter	Клас адаптеру списку карток грошових рахунків користувача
ui.adapters	AccountListChooserAdapter	Клас адаптеру списку карток грошових рахунків користувача з можливістю обробки вибору окремого рахунку
ui.adapters	ChartsListAdapter	Клас адаптеру списку графіків зміни балансу окремих грошових рахунків

Продовження таблиці 3.1

1	2	3
ui.adapters	TransactionPagingAdapter	Клас адаптеру списку грошових операцій з підтримкою поступового завантаження елементів та роздільників списку
ui	AccountChartsFragment	Клас екрану користувацького інтерфейсу, що містить список графіків зміни балансу рахунків
ui	AddAccountFragment	Клас екрану користувацького інтерфейсу, що містить функцію додавання грошового рахунку
ui	AddKeyBottomSheet	Клас елемента користувацького інтерфейсу у вигляді нижнього діалогу з можливістю додавання персонального токена банку
ui	AddTransactionFragment	Клас екрану користувацького інтерфейсу, що надає можливість додавання грошової операції
ui	AddTypeDialog	Клас діалогу користувацького інтерфейсу додавання категорії грошової операції
ui	DynamicIconClusterRenderer	Клас керування відображенням маркерів на мапі грошових операцій

Продовження таблиці 3.1

1	2	3
ui	LocationPickerFragment	Клас екрану користувацького інтерфейсу ручного вибору позиції на мапі
ui	OverviewFragment	Клас екрану користувацького інтерфейсу, що надає доступ до функцій перегляду та додавання грошових рахунків, операцій
ui	StatsFragment	Клас екрану користувацького інтерфейсу, що надає доступ до перегляду статистики
ui	TransactionsOnMapFragment	Клас екрану користувацького інтерфейсу, що надає доступ до перегляду грошових операцій на мапі
ui	TransactionClusterItem	Клас кластеру грошових операцій на мапі
ui	TransactionsSyncFragment	Клас екрану користувацького інтерфейсу, що надає доступ до налаштувань синхронізації з банком
utils	Event	Клас, для спрощення обробки відстеження подій користувацького інтерфейсу
utils	HorizontalMarginItemDecorator	Клас для налаштування горизонтального відступу у списках

Продовження таблиці 3.1

1	2	3
utils	Resource	«sealed» клас статусу запиту до віддаленого ресурсу
utils	Resource.Loading	Клас статусу запиту до віддаленого ресурсу – завантаження
utils	Resource.Success	Клас статусу запиту до віддаленого ресурсу – успіх
utils	Resource.ApiError	Клас статусу запиту до віддаленого ресурсу – помилка API
utils	Resource.ExceptionError	Клас статусу запиту до віддаленого ресурсу – помилка API
viewmodels	AccountsChartsViewModel	Клас ViewModel, що містить логіку для екрану статистики
viewmodels	AddKeyBottomSheetViewModel	Клас ViewModel для підтримки діалогу додання персонального токenu банку
viewmodels	AddMoneyStoreViewModel	Клас ViewModel, що містить логіку для екрану додання грошових рахунків
viewmodels	AddTransactionViewModel	Клас ViewModel, що містить логіку для екрану додання грошових операцій

Продовження таблиці 3.1

1	2	3
viewmodels	Operation	Клас перелічення, що містить типи операцій: «ADD» – нарахування, «REMOVE» – відрахування
viewmodels	AmountErrorTypes	Клас перелічення, що містить тип помилки введення суми: «EMPTY» – пусте поле, «ZERO» – нуль, «FORMAT» – помилка формату
viewmodels	OverviewViewModel	Клас ViewModel, що містить логіку екрану перегляду операцій та рахунків
viewmodels	StatsViewModel	Клас ViewModel, що забезпечує обробку статистики для екрану перегляду статистики
viewmodels	TransactionOnMapViewModel	Клас ViewModel, що забезпечує логіку екрану перегляду операцій на мапі
viewmodels	TransactionSyncViewModel	Клас ViewModel, що забезпечує логіку екрану налаштувань синхронізації з банком
workers	MonoSyncWorker	Клас, що виконує фонову синхронізацію з банком

Продовження таблиці 3.1

1	2	3
Глобальний пакет	MainActivity	Клас головного екрану користувачького інтерфейсу, який керує всіма іншими екранами та має елементи навігації
Глобальний пакет	MainApplication	Клас ініціалізації програмного застосунку

На рис. 3.8 зображені файли ресурсів програмного застосунку. Треба зазначити, що у директоріях «drawable» та «mipmap» зберігаються ресурси зображень, які використовуються у застосунку, та варіанти зображень для головного екрану ОС Android відповідно.

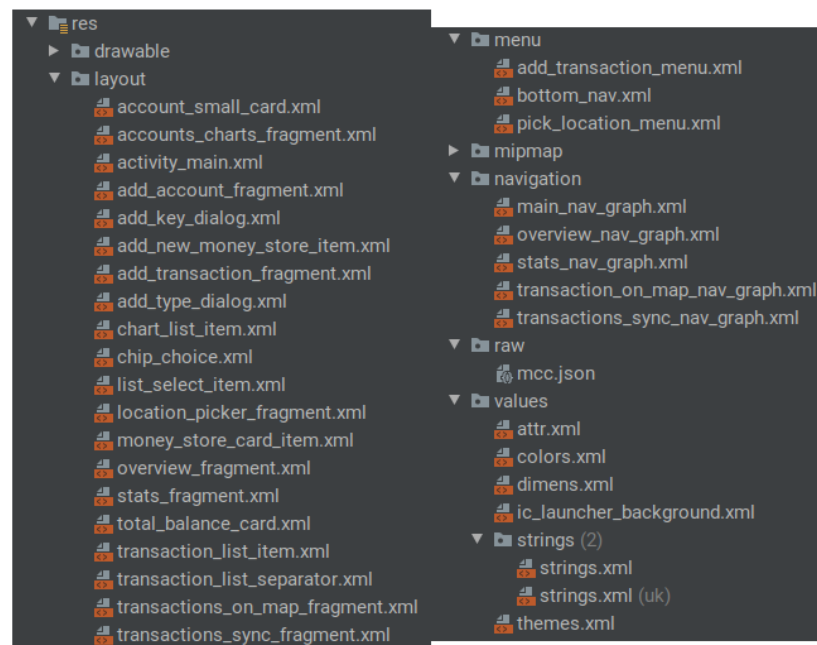


Рисунок 3.8 – Файли ресурсів програмного застосунку

У табл. 3.2 виконано опис використання ресурсів програмного застосунку.

Таблиця 3.2 – Опис ресурсів програмного застосунку

Шлях	Назва	Опис
1	2	3
res/layout	account_small_card.xml	Розмітка зовнішнього вигляду елемента вибору грошового рахунку
res/layout	accounts_charts_fragment.xml	Розмітка екрану перегляду графіків зміни балансу окремих рахунків
res/layout	activity_main.xml	Розмітка головного екрану, що містить нижнє меню навігації та місце для дочірніх екранів
res/layout	add_account_fragment.xml	Розмітка екрану додавання грошових рахунків
res/layout	add_key_dialog.xml	Розмітка діалогу додавання персонального токена банку
res/layout	add_new_money_store_item.xml	Розмітку елемента списку, при натисканні на який відбувається перехід д екрану додавання рахунку
res/layout	add_transaction_fragment.xml	Розмітка екрану додавання грошової операції
res/layout	add_type_dialog.xml	Розмітка діалогу додавання нової категорії грошової операції

Продовження таблиці 3.2

1	2	3
res/layout	chart_list_item.xml	Розмітка елементу списку графіків зміни балансу рахунків
res/layout	chip_coice.xml	Розмітка елементу надання вибору категорія для динамічного додання до екрану створення грошової операції
res/layout	list_select_item.xml	Розмітка елементу випадального списку
res/layout	location_picker_fragment.xml	Розмітка екрану вибору позиції на мапі
res/layout	money_store_card_item.xml	Розмітка елементу списку грошових рахунків
res/layout	overview_fragment.xml	Розмітка екрану перегляду грошових рахунків та операцій
res/layout	stats_fragment.xml	Розмітка екрану перегляду статистики
res/layout	total_balance_card.xml	Розмітка елементу списку грошових рахунків, що містить сумарний баланс
res/layout	transaction_list_item.xml	Розмітка елементу списку грошових операцій
res/layout	transaction_list_separator.xml	Розмітка роздільного елементу списку грошових операцій

Продовження таблиці 3.2

1	2	3
res/layout	transactions_on_map_fragment.xml	Розмітка екрану перегляду грошових операцій на мапі
res/layout	transactions_sync_fragment.xml	Розмітка екрану керування функцією синхронізації з банком
res/menu	add_transaction_menu.xml	Опис елементів меню додавання грошової операції
res/menu	bottom_nav.xml	Опис елементів нижнього навігаційного меню
res/menu	pick_location_menu.xml	Опис елементів меню вибору позиції на мапі
res/navigation	main_nav_grap.xml	Головний навігаційний граф, що містить усі інші графи
res/navigation	overview_nav_graph.xml	Навігаційний граф екрану перегляду рахунків та операцій
res/navigation	stats_nav_graph.xml	Навігаційний граф екрану перегляду статистики
res/navigation	transaction_on_map_nav_graph.xml	Навігаційний граф екрану перегляду операцій на мапі

Продовження таблиці 3.2

1	2	3
res/navigation	transaction_sync_nav_graph.xml	Навігаційний граф екрану керування синхронізацією з банком
res/raw	mcc.json	Файл формату Json, що містить опис mcc кодів
res/values	attr.xml	Файл, що містить перелік створених атрибутів стилів програмного застосунку
res/values	colors.xml	Перелік кодів кольорів, що використовуються у програмному застосунку
res/values	dimens.xml	Перелік розмірів
res/values	ic_launcher_background.xml	Колір іконки програмного застосунку
res/values	strings.xml	Перелік рядків користувацького інтерфейсу англійською мовою
res/values-uk	strings.xml	Перелік рядків користувацького інтерфейсу українською мовою
res/values	themes.xml	Перелік стилів програмного застосунку

На рис. 3.9 зображена діаграма класів, що забезпечують роботу перегляду грошових рахунків та операцій. Треба зазначити, що для компактності та ясності представлення деталі реалізації списків користувацького інтерфейсу було спрощено.

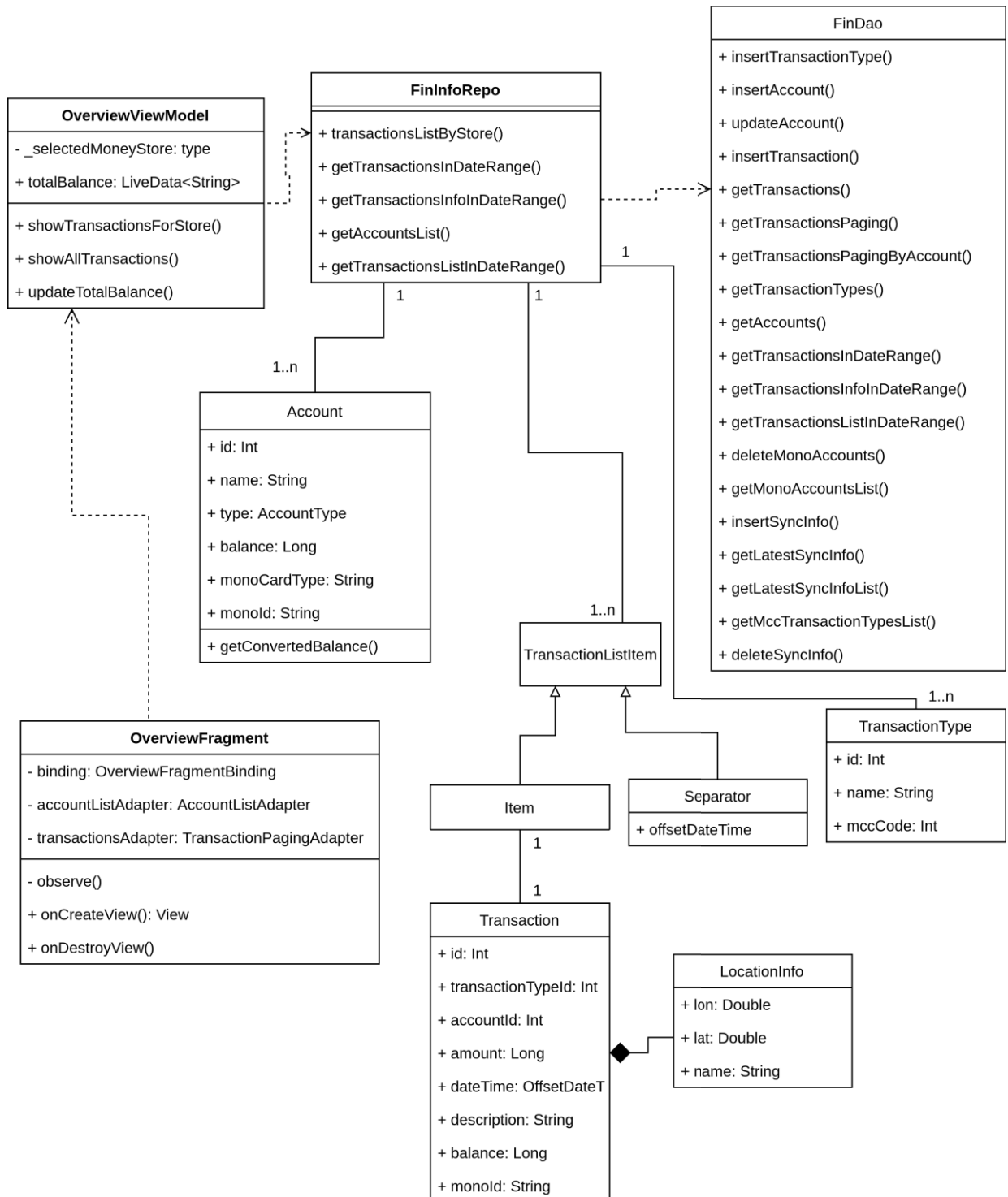


Рисунок 3.9 – Діаграма класів, що забезпечують відображення грошових рахунків та операцій

3.5 Висновки за розділом 3

У розділі 3 було описано питання розробки програмного застосунку, а саме: висвітлено використовувані бібліотеки, наведено та описано схеми алгоритмів, що забезпечують роботу програмних застосунків, оглянуто можливості віддаленого сервісу банку та описано програмні модулі застосунку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення програми

Призначенням програмного застосунку є відстеження грошових операцій з доданням геоінформації до них.

Програмне забезпечення надає наступні функціональні можливості:

- перегляд грошових рахунків;
- перегляд сумарного балансу;
- фільтрація грошових операцій за рахунками;
- перегляд грошових операцій з можливістю додати геоінформацію (позицію на мапі, назву місця);
- відображення підказок назви місця знаходження користувача при доданні грошової операції із геоінформацією;
- додання грошового рахунку;
- додання грошової операції;
- вибір проміжку часу статистики;
- перегляд зміни сумарного балансу за обраний проміжок часу;
- перегляд графіків зміни балансів грошових рахунків за обраний проміжок часу;
- перегляд кругової діаграми витрат за категоріями за обраний проміжок часу;
- перегляд кругової діаграми нарахувань за категоріями за обраний проміжок часу;
- перегляд кругової діаграми витрат за місцями за обраний проміжок часу;
- перегляд грошових операцій за останній місяць на мапі;
- перегляд стану останньої спроби синхронізації з банком – час та успішність;

- додання персонального токену банку, для можливості синхронізації даних грошових рахунків з ним;
- запуск процесу синхронізації власноруч;
- увімкнення періодичної синхронізації з банком: якщо увімкнено, програмний застосунок, навіть якщо є закритим, буде кожні 12 годин запускати процес синхронізації, якщо мобільний пристрій підключено до мережі інтернет;
- видалення даних банку з програмного застосунку, що включає до себе: синхронізовані рахунки, грошові операції та персональний токену користувача.

4.2 Умови виконання програми

Для використання програмного забезпечення мобільний пристрій користувача має працювати на базі ОС Android версії 5 та новіше та мати встановлені сервіси Google Play, що є встановленими на всіх сертифікованих Android пристроях, для коректної роботи геофункцій (таких як: додання поточного місцезнаходження до грошових операцій, відображення підказок назви поточного місцезнаходження, відображення грошових операцій на мапі).

Також, для функціонування можливостей відображення мапи, підказок назви місця та синхронізації з банком пристрій має мати підключення до мережі інтернет.

4.3 Тестування сценаріїв роботи з програмою

Запуск програмного застосунку виконується з екрану сітки програмних засобів ОС Android натисканням на зображення (рис. 4.1).



Рисунок 4.1 – Зображення програмного застосунку

Якщо програмний застосунок відкрито у перший раз, то на головному екрані буде доступно тільки додавання нового грошового рахунку по натисканню на картку (рис. 4.2).



Рисунок 4.2 – Вигляд програмного застосунку при першому відкритті

Після натискання на карту додавання грошового рахунку відкриється новий екран, що зображено на рис. 4.3.

Для додавання рахунку необхідно ввести його назву, яка максимально може складати 15 символів та натиснути на кнопку «Додати», яка знаходиться у верхньому меню. При необхідності користувач може вказати

тип рахунку, обравши необхідний варіант з випадаючого списку, доступними є типи: рахунок, гаманець. Також, за необхідності можливе вказання балансу рахунку. Треба звернути увагу, що при введенні балансу застосовується фільтр, що забороняє введення неприйнятного формату.

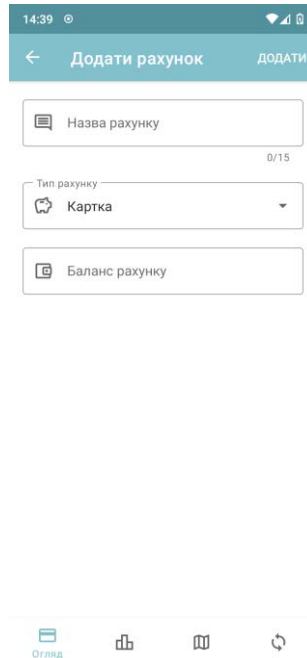


Рисунок 4.3 – Екран додання рахунку

Після додання принаймні одного грошового рахунку на головному екрані стають доступні функції перегляду сумарного балансу, перегляду балансів окремих рахунків та додання грошових операцій. На рис. 4.4 зображено верхню частину головного екрану після додання грошового рахунку.

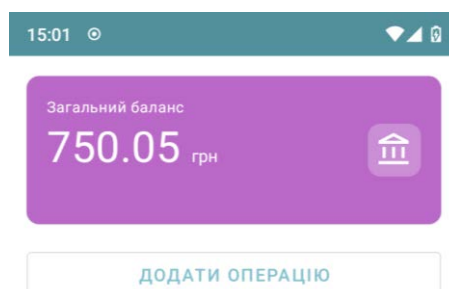


Рисунок 4.4 – Верхня частина головного екрану після додання рахунку

Перехід до екрану додання операції здійснюється натисканням кнопки «Додати операцію».

На рис. 4.5 зображено екран додання грошової операції.

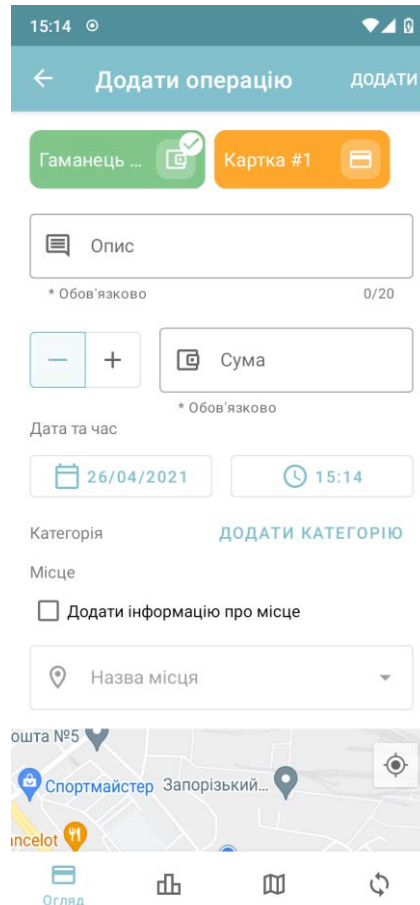


Рисунок 4.5 – Екран додання грошової операції

Вибір грошового рахунку виконується натисканням на потрібний елемент. Обраний рахунок до додання операції помічено знаком у правому верхньому куті елемента. За замовчанням обрано перший рахунок. Якщо список рахунків не вміщується на екрані, є можливість горизонтальної прокрутки цього списку, при цьому прокручується лише список рахунків. Для перегляду довгих назв грошових рахунків необхідно зробити довге натискання по елементу, після чого поруч з ним з'явиться повна назва рахунку (рис. 4.6).

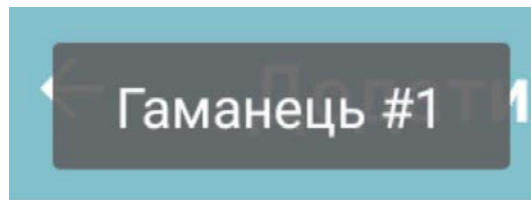


Рисунок 4.6 – Спливаюче повідомлення з назвою рахунку

Далі необхідно ввести короткий опис грошової операції в поле «Опис».

Тип операції (витрата або нарахування) визначається за допомогою відповідного перемикача з лівої сторони від поля суми. За замовчанням його положення встановлено у позицію витрати.

Поле суми операції має той же самий фільтр перевірки форматування грошової суми, що й в доданні рахунку. Це поле є обов'язковим до заповнення.

Дата та час операції визначаються натисканням відповідних кнопок у розділі «Дата та час», після чого буде відкрито діалогові вікна, що зображено на рис. 4.7 та 4.8 відповідно. За замовчанням дата та час операції дорівнює даті та часу відкриття екрану додання грошової операції.

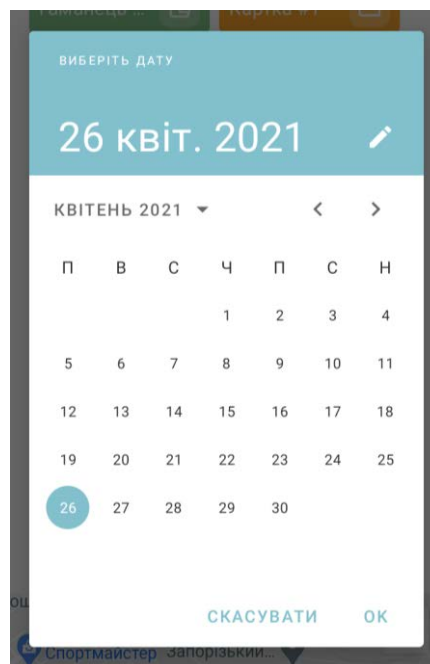


Рисунок 4.7 – Діалог вибору дати грошової операції

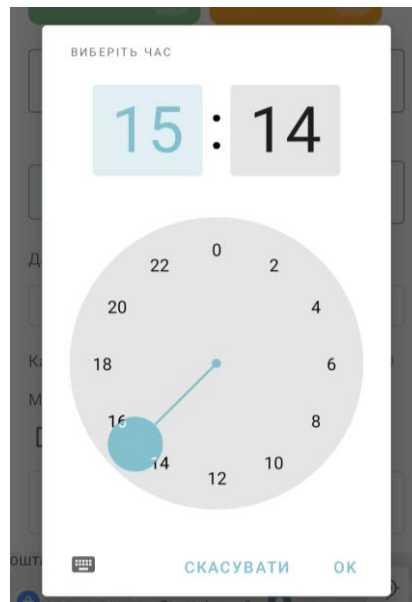


Рисунок 4.8 – Діалог вибору часу грошової операції

Для додання нової категорії грошових операцій необхідно натиснути кнопку «Додати категорію», після чого у діалоговому вікні ввести назву та підтвердити збереження. Діалог додання категорії зображено на рис. 4.9.

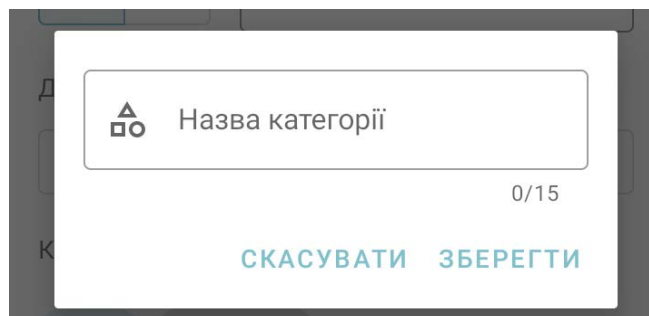


Рисунок 4.9 – Діалог додання категорії грошових операцій

Додані категорії відображаються у вигляді малих кнопок у розділі «Категорія». Можливо обрати лише одну категорію, за замовчанням обрана перша за списком. Зміна обраної категорії виконується натисканням на кнопку відповідної категорії, при цьому попередньо обрана категорія змінить свій колір на неактивний, а вибрана – на активний. На рис. 4.10 зображено розділ «Категорія» з обраною категорією грошової операції.

Категорія

ДОДАТИ КАТЕГОРІЮ

Одяг

Продукти

Рисунок 4.10 – Зовнішній вигляд обраної категорії

Для додання геоінформації до грошової операції необхідно увімкнути опцію «Додати інформацію про місце», після чого стане доступне поле «Назва місця». При натисканні на стрілку поля «Назва місця» програмний застосунок відобразить підказки назв місць, що знаходяться неподалік від місцезнаходження користувача (рис. 4.11). При натисканні на будь який варіант назва буде автоматично підставлена до поля, а на мапі з'явиться маркер місцезнаходження місця (рис. 4.12).

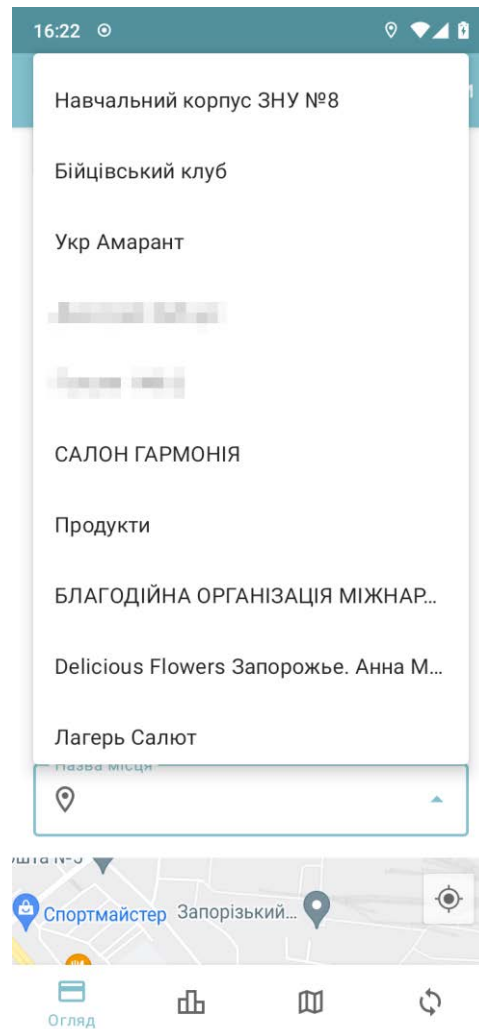


Рисунок 4.11 – Підказки назви поточного місця користувача

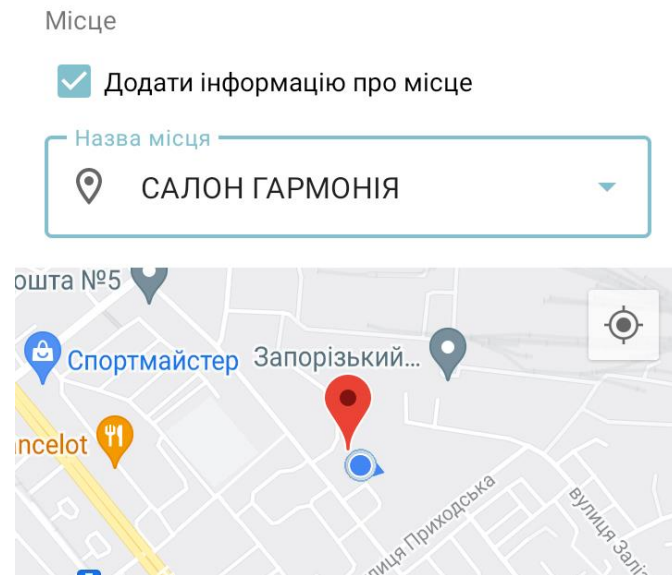


Рисунок 4.12 – Вигляд обраного місця

Окрім вибору місця за підказками програмного застосунку є можливість вибору місця власноруч на мапі місця. Для цього необхідно натиснути на зображення мапи, після чого відкриється екран вибору місця (рис. 4.13), де необхідно перетягнути мапу відповідно до маркера в центрі на необхідне місце, та підтвердити натиском кнопки «Обрати».



Рисунок 4.13 – Екран вибору місцезнаходження

Після вибору позиції на мапі необхідно вручну вказати назву у полі «Назва місця».

Для збереження грошової операції необхідно натиснути «Додати» на екрані додання грошової операції. Після збереження грошової операції користувача буде повернено на екран огляду, а баланс відповідного грошового рахунку буде оновлено відповідно до типу (нарахування або витрата) та суми грошової операції.

Треба звернути увагу, що іноді, наприклад, при першому запуску екрану додання грошової операції або у випадку тривалого невикористання програмного застосунку система Android може вивести діалог з запитом програми на доступ до місцезнаходження користувача, що зображено на рис. 4.14. Для доступу до усіх функціональних можливостей програмного забезпечення необхідно, в такому випадку, натиснути на «Під час використання додатка» або «Лише цього разу». При відмові надання місцезнаходження будуть недоступні наступні функціональні можливості: автоматичне позиціонування мапи, підказки назви поточного місцезнаходження користувача.

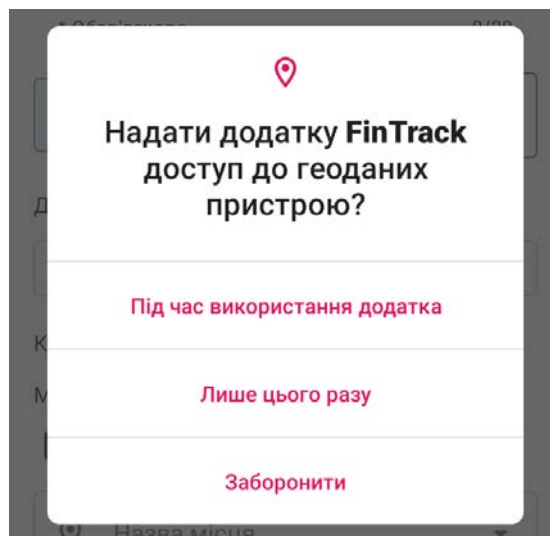


Рисунок 4.14 – Запит на надання поточного місцезнаходження

На рис. 4.15 зображено зовнішній вигляд елементів списку грошових операцій.

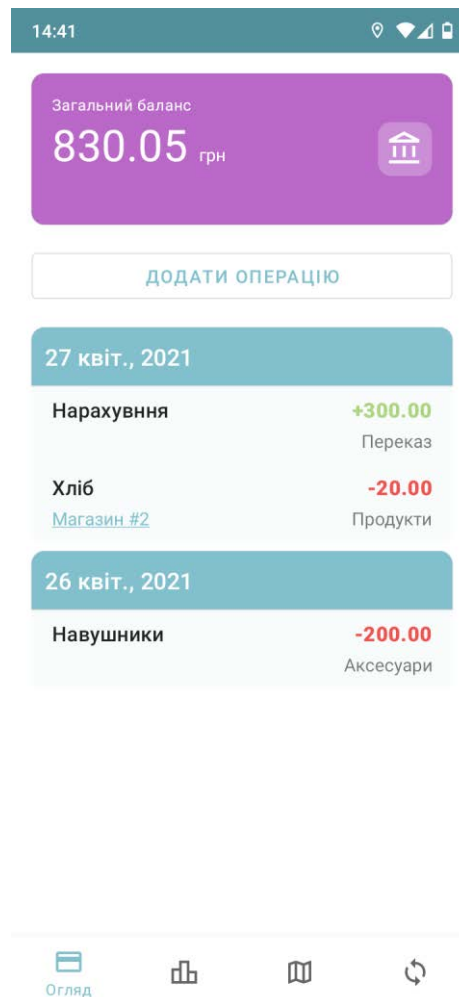


Рисунок 4.15 – Екран огляду з доданими грошовими операціями

Грошові операції у списку відображаються відсортованими по часу в порядку убутання та є згрупованими за датою.

Дата групи операцій відображена у роздільних елементах.

Окремий елемент грошової операції складається з наступних частин: опису у верхньому лівому куті, суми та типу операції, що позначаються відповідним кольором та знаком, у правому верхньому куті, назви категорії грошової операції у правому нижньому куті та, за наявності, назви місця у лівому нижньому куті. При наявності назви місця у елементі, натискання на

нього відкриє програмне забезпечення «Google Maps» (якщо таке встановлене на пристрої користувача) з обраним місцем на мапі.

Перехід до окремого грошового рахунку та його операцій здійснюється перегортанням верхньої картки. Тип рахунку позначено іконкою та кольором картки, так жовта картка – рахунок типу «картка», зелена – рахунок типу «гаманець», сіра – синхронізований рахунок банку. Вигляд екрану огляду з обраним грошовим рахунком зображено на рис. 4.16.

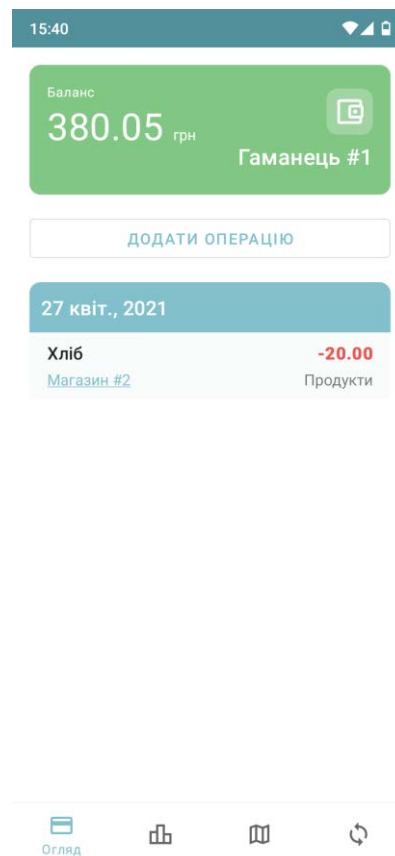


Рисунок 4.16 – Перегляд окремого рахунку та його грошових операцій

На рис. 4.17 зображено екран статистики, що надає можливість перегляду зміни балансу, кругових діаграм витрат за категоріями, доходів за категоріями, витрат за місцями та переходу до екрану перегляду графіків історії балансів окремих грошових рахунків.

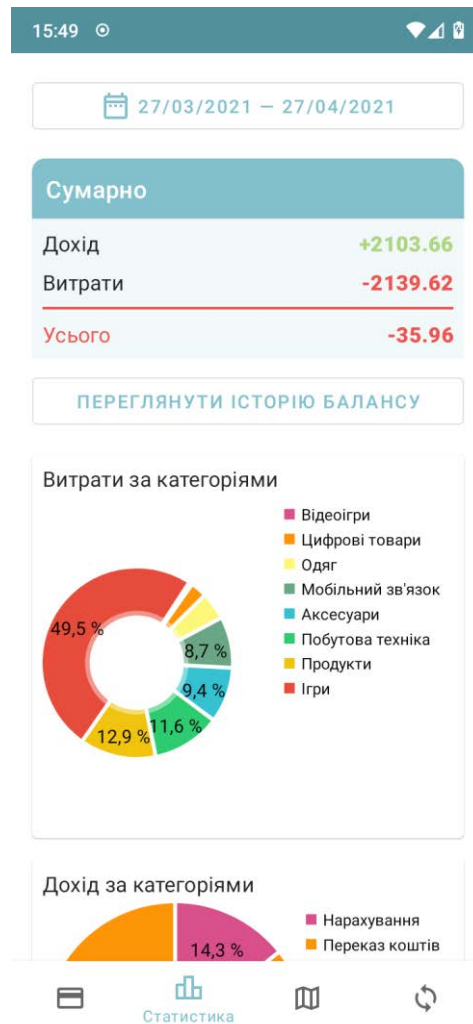


Рисунок 4.17 – Екран статистики

За замовчанням відображається статистика за останній місяць. Для перегляду статистики за інший період часу необхідно натиснути на кнопку проміжку дат у верхній частині екрану та у діалозі (рис. 4.18), обрати діапазон дат, після чого натиснути «зберегти» що призведе до закриття діалогового вікна та оновлення статистики.

У верхній частині екрану, під кнопкою діапазону дат розташовано елемент відображення зміни стану балансів рахунків, де відображено сумарний дохід та сумарні витрати рахунків за визначений діапазон дат.

Кругова діаграма витрат за категоріями відображає витрати у процентному співвідношенні. Треба зазначити, що надписи процентів менше 5 скриваються.



Рисунок 4.18 – Діалог вибору діапазону дат статистики

На рис. 4.19 зображено приклад кругових діаграм доходу за категоріями та витрат за місцями, які функціонують так само, як і діаграма витрат за категоріями.

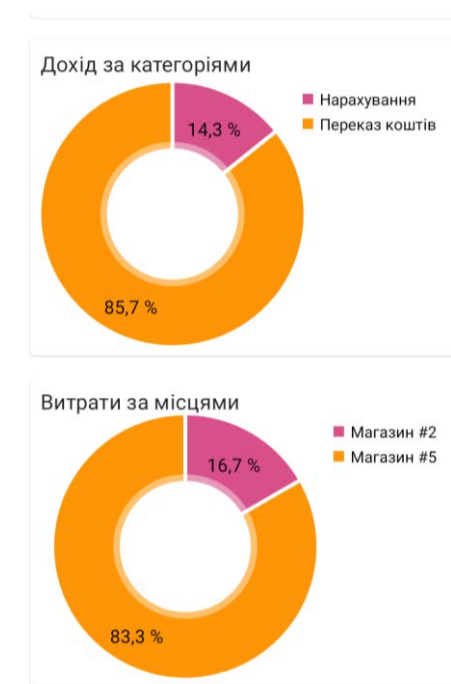


Рисунок 4.19 – Приклад кругових діаграм

При натисканні на кнопку «Переглянути історію балансу» буде здійснено перехід до екрану, що містить графіки балансу окремих рахунків за обраний період часу (рис. 4.20).

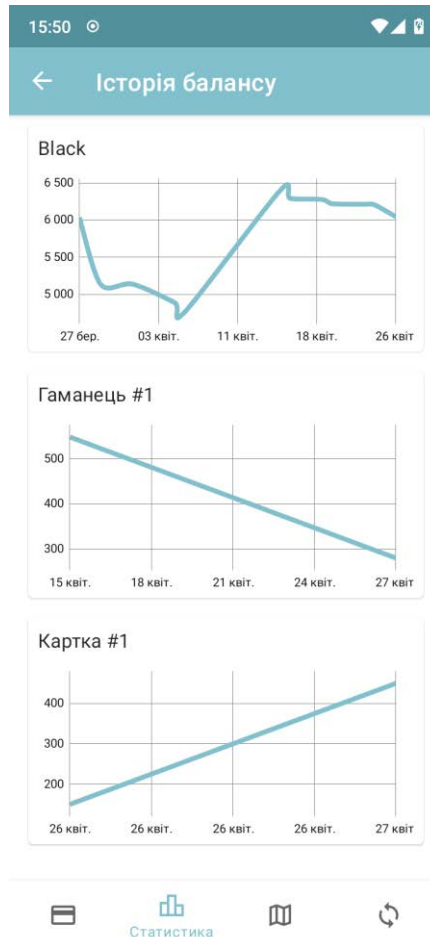


Рисунок 4.20 – Екран історії балансів

На рис. 4.21 зображено екран перегляду грошових операцій на мапі. Усі операції, здійснені останнього місяця та які мають геоінформацію, відображені у вигляді позначок різного кольору: червоні у випадку витрати, та зелені у випадку нарахування. Задля отримання інформації про окрему операцію необхідно натиснути на позначку, після чого з'явиться спливаюче повідомлення з зазначенням опису та суми операції (рис. 4.22).

При зменшенні масштабу мапи, для запобігання великої кількості позначок на малій кількості місця, позначки грошових операцій

об'єднуються до кластеру з зазначенням кількості (рис. 4.23). При збільшенні масштабу кластер позначок розбивається на окремі позначки.

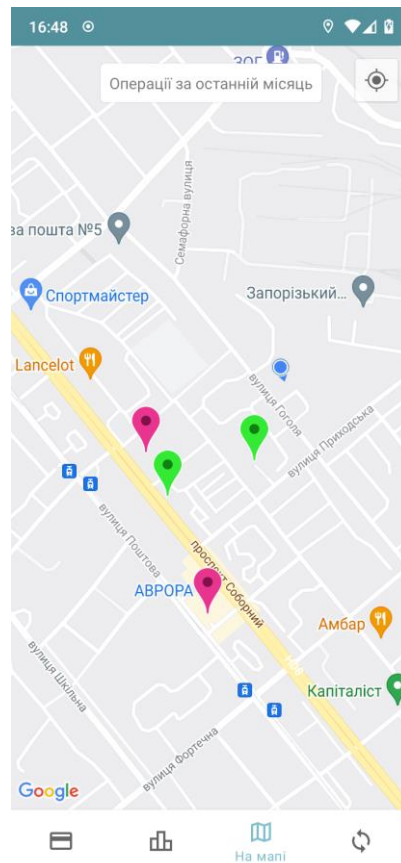


Рисунок 4.21 – Перегляд грошових операцій на мапі

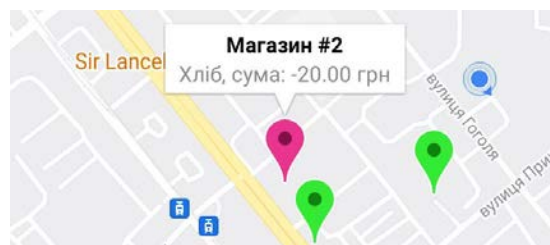


Рисунок 4.22 – Перегляд інформації про грошову операцію

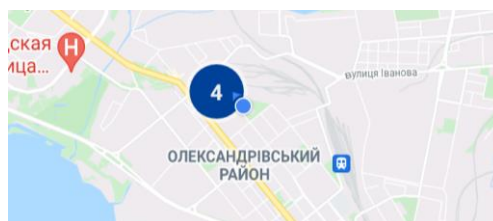


Рисунок 4.23 – Кластер позначок

На рис. 4.24 зображено екран налаштувань синхронізації грошових рахунків та операцій з банком при першому запуску.

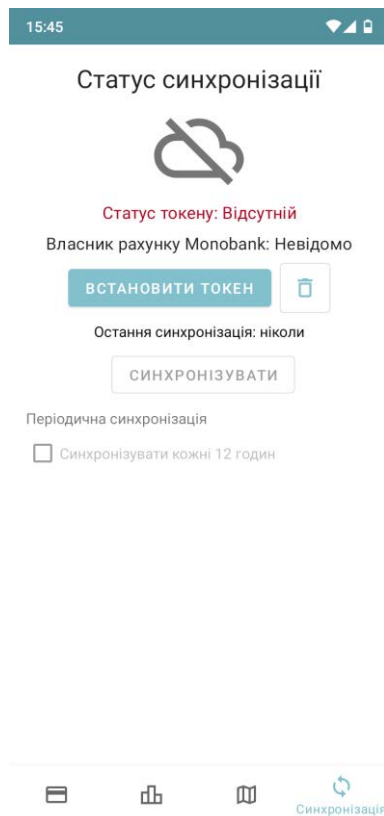


Рисунок 4.24 – Екран налаштувань синхронізації з банком

Для початку роботи з функцією синхронізації необхідно додати персональний токен банку. Для цього слід натиснути кнопку «встановити токен», після чого відкриється діалог з інструкцією та полем для вставки токenu (рис. 4.25).

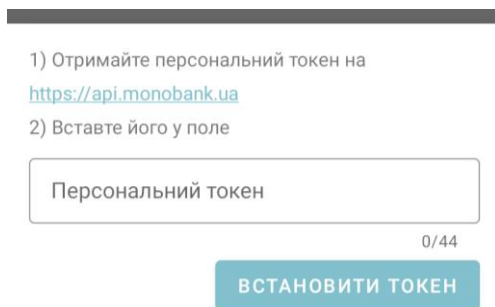


Рисунок 4.25 – Діалог додання персонального токenu

Для отримання персонального токenu банку необхідно перейти до сторінки, яку вказано у відкритому діалозі, та пройти авторизацію, після чого скопіювати персональний токен, вставити його у поле «Персональний токен» та натиснути кнопку «Встановити токен». Після натиснення кнопки «Встановити токен» програмний застосунок перевіряє токен та у разі успіху закриває діалогове вікно.

Після встановлення персонального токenu на екрані налаштування синхронізації з банком змінюється статус токenu та відображається власник рахунку «Монобанк» та стан останньої спроби синхронізації (рис. 4.26). Для початку синхронізації необхідно натиснути кнопку «Синхронізувати». Приклад результату синхронізації наведено на рис. 4.27. Для ввімкнення періодичної синхронізації з банком, навіть коли програмний застосунок є «закритим», необхідно поставити прапорець на «Синхронізувати кожні 12 годин».

Для видалення всіх даних банку з програмного застосунку необхідно натиснути кнопку з іконкою корзини, після чого вся персональна інформація (токену, власник рахунку) та рахунки і операції банку будуть видалені.

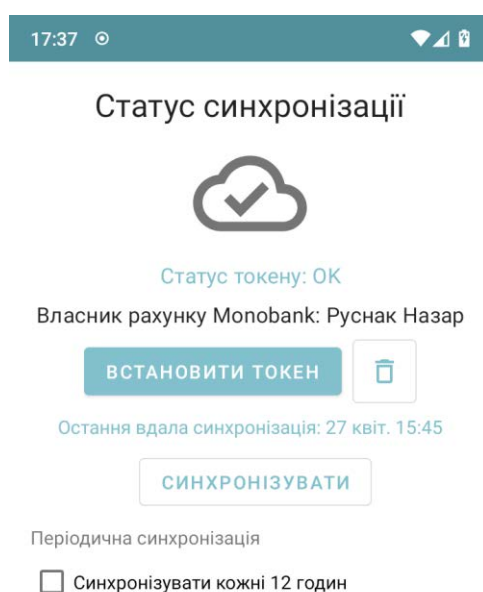


Рисунок 4.26 – Інформація про статус синхронізації



Рисунок 4.27 – Приклад синхронізованого рахунку

Треба зазначити, що не є можливим власноручне додання грошових операцій до синхронізованих грошових рахунків.

Далі наведено опис повідомлень про помилки, що можуть виникнути при використанні програмного застосунку.

На рис. 4.28 наведено помилку при спробі додання грошового рахунку з пустим полем назви.

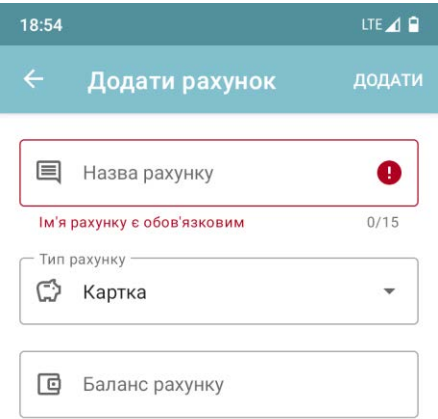


Рисунок 4.28 – Помилка пустої назви рахунку

На рис 4.29 наведені приклади оброблення програмою помилок спроби додання грошової операції з незаповненими полями опису, суми та не обраної позиції на мапі, коли поставлено прапорець додання геоінформації.

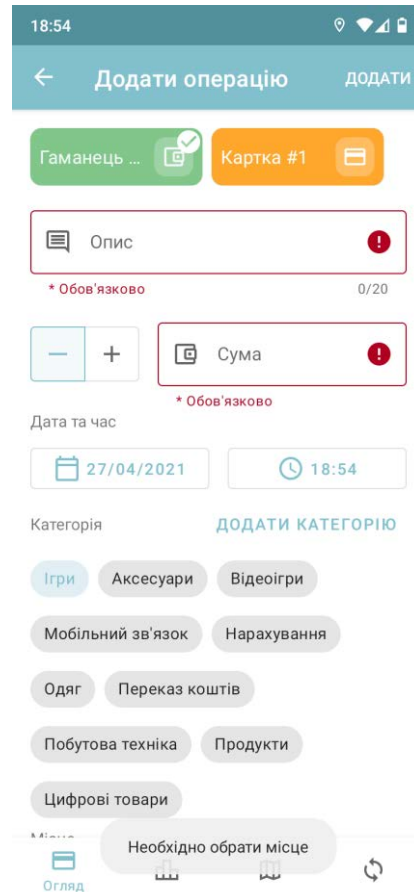


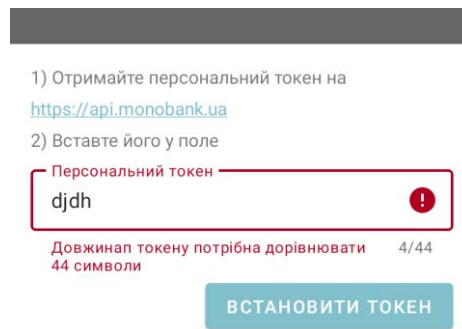
Рисунок 4.29 – Помилки додання грошової операції

На рис. 4.30 наведена помилка спроби додання грошової операції витрати з сумою, що перевищує суму на балансі рахунку.



Рисунок 4.30 – Помилка низького балансу

На рис. 4.31 наведено приклад оброблення програмою спроби додати занадто короткий токену користувача.



1) Отримайте персональний токен на <https://api.monobank.ua>

2) Вставте його у поле

Персональний токен

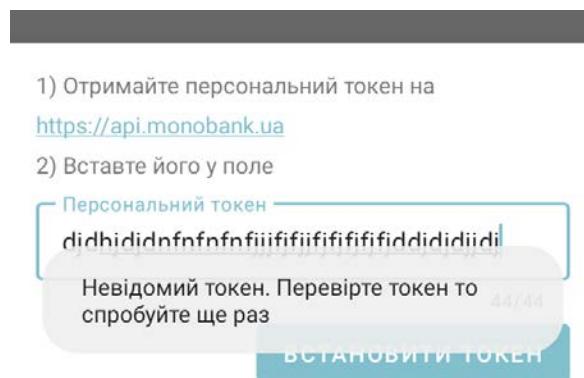
djdh

Довжина токена повинна дорівнювати 44 символи 4/44

ВСТАНОВИТИ ТОКЕН

Рисунок 4.31 – Помилка занадто короткого персонального токена

На рис. 4.32 наведено приклад оброблення програмою помилки неіснуючого персонального токена в системі «Monobank Open API».



1) Отримайте персональний токен на <https://api.monobank.ua>

2) Вставте його у поле

Персональний токен

didhidiidnfnfnfnfiiififiififiiddidididi

Невідомий токен. Перевірте токен та спробуйте ще раз 44/44

ВСТАНОВИТИ ТОКЕН

Рисунок 4.32 – Помилка неіснуючого персонального токена

На рис. 4.33 наведено приклад оброблення програмою помилки з'єднання з сервером для синхронізації.

На рис. 4.34 зображено повідомлення системи сповіщень ОС Android про невдалу спробу синхронізації, якщо персональний токен було деактивовано у системі банку.

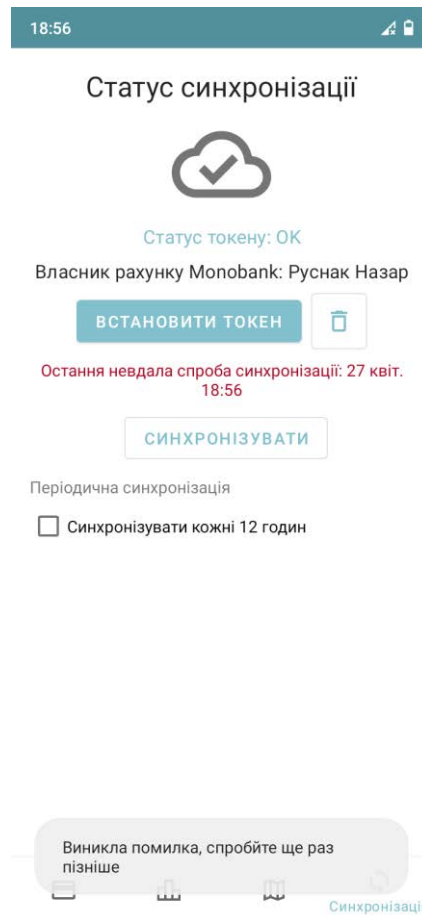


Рисунок 4.33 – Помилка з'єднання

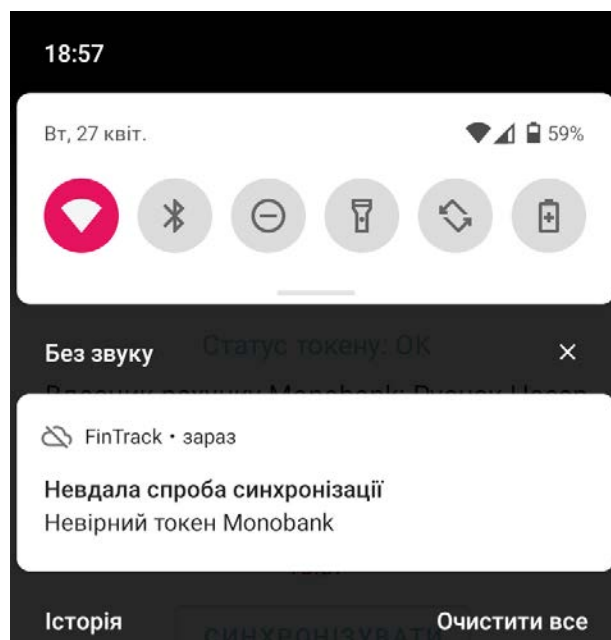


Рисунок 4.34 – Сповіщення про невдалу спробу синхронізації у випадку деактивованого токена

4.4 Висновки за розділом 4

У розділі було представлено функціональні можливості та вимоги розробленого програмного забезпечення. Описано процес роботи з програмою та наведено відомості про можливі помилки при роботі з програмним застосунком.

ВИСНОВКИ

У роботі було досліджено процес відслідковування фінансових витрат, у результаті чого було розроблено програмний застосунок для операційної системи Android.

Було проаналізовано існуючі програмні засоби вирішення проблеми відслідковування фінансових витрат, виділено їх переваги, недоліки та запропоновано додання додаткових функціональних можливостей, таких як використання геоінформації та синхронізації з банком до розроблюваного програмного застосунку та визначено функціональні вимоги.

Було вирішено питання розробки архітектури програмного застосунку: проаналізовано та обрано мову програмування Kotlin, проаналізовано архітектурні шаблони для розробки програмних застосунків, що дало підстави для вибору MVVM в якості шаблону архітектури, розроблено і представлено структуру бази даних та програмного забезпечення.

В процесі розробки програмного забезпечення було оглянуто допоміжні інструменти у вигляді бібліотек для пришвидшення темпів розробки та зменшення програмних помилок, проаналізовано можливості та обмеження віддалених сервісів «Monobank open API» та бібліотек отримання та відображення геоінформаційних даних «Maps SDK for Android» та «Places SDK for Android».

Розроблено програмний застосунок відслідковування фінансових витрат з використанням геоінформаційних можливостей мобільних пристроїв, що задовольняє поставленим функціональним вимогам. Програма дозволяє виконувати документування фінансових витрат, наводити статистику, відображати збережені дані у вигляді списку та на мапі, має можливості з синхронізації грошових операцій та рахунків із банком.

Усі завдання, які було визначено у дипломній кваліфікаційній роботі бакалавра, виконано у повному обсязі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Monefy - трекер бюджету та журнал витрат – Додатки в Google Play [Електрон. ресурс]. – Режим доступу : <https://play.google.com/store/apps/details?id=com.monefy.app.lite>.
2. Трекер бюджету, облік витрат та фінансів, гроші – Додатки в Google Play [Електрон. ресурс]. – Режим доступу : https://play.google.com/store/apps/details?id=ru.innim.my_finance.
3. Money Lover - рахунок менеджер – Додатки в Google Play [Електрон. ресурс]. – Режим доступу : <https://play.google.com/store/apps/details?id=com.bookmark.money>.
4. There are now more than 2.5 billion active Android devices [Electronic resource]. – Access mode : <https://www.androidpolice.com/2019/05/07/there-are-now-more-than-2-5-billion-active-android-devices/>.
5. Android Open Source Project [Electronic resource]. – Access mode : <https://source.android.com/>.
6. Background Location Limits | Android Developers [Electronic resource]. – Access mode : <https://developer.android.com/about/versions/oreo/background-location-limits>.
7. Privacy changes in Android 10 | Android Developers [Electronic resource]. – Access mode : <https://developer.android.com/about/versions/10/privacy/changes>.
8. Bell, D. Java: A Step-by-Step Guide for Absolute Beginners [Text] / D. Bell. – Berkeley : Guzzler Media LLC, 2019. – 187 p.
9. Schildt, H. Java : A Beginner's Guide, Eighth Edition [Text] / H. Schildt. – New York : McGraw-Hill Education, 2018. – 720 p.
10. Griffiths, D. Head First Android Development : A Brain-Friendly Guide [Text] / D. Griffiths, D. Griffiths. – Sebastopol : O'Reilly Media, 2017. – 887 p.

11. Android Developers Blog: Android Announces Support for Kotlin [Electronic resource]. – Access mode : <https://android-developers.googleblog.com/2017/05/android-announces-support-for-kotlin.html>.
12. Android's Kotlin-first approach | Android Developers [Electronic resource] – Access mode : <https://developer.android.com/kotlin/first>.
13. Eckel, B. Atomic Kotlin [Text] / B. Eckel, S. Isakova. – Crested Butte : Mindview LLC, 2021. – 636 p.
14. MVC vs MVP vs MVVM architecture in Android [Electronic resource]. – Access mode : <https://blog.mindorks.com/mvc-mvp-mvvm-architecture-in-android>.
15. Guide to app architecture | Android Developers [Electronic resource]. – Access mode : <https://developer.android.com/jetpack/guide>.
16. Datatypes In SQLite Version 3 [Electronic resource]. – Access mode : <https://sqlite.org/datatype3.html>.
17. kotlin-stdlib - Kotlin Programming Language 3 [Electronic resource]. – Access mode : <https://kotlinlang.org/api/latest/jvm/stdlib/>.
18. square/moshi: A modern JSON library for Kotlin and Java [Electronic resource]. – Access mode : <https://github.com/square/moshi>.
19. Fused Location Provider API | Google Developers [Electronic resource]. – Access mode : <https://developers.google.com/location-context/fused-location-provider>.
20. Maps SDK for Android Usage and Billing | Google Developers [Electronic resource]. – Access mode : <https://developers.google.com/maps/documentation/android-sdk/usage-and-billing>.
21. Places SDK for Android Usage and Billing | Google Developers [Electronic resource]. – Access mode : <https://developers.google.com/maps/documentation/places/android-sdk/usage-and-billing>.
22. Android Jetpack | Android Developers [Electronic resource]. – Access mode : <https://developer.android.com/jetpack>.

23. Save data in a local database using Room | Android Developers [Electronic resource]. – Access mode : <https://developer.android.com/training/data-storage/room>.

24. Dependency injection with Hilt | Android Developers [Electronic resource]. – Access mode : <https://developer.android.com/training/dependency-injection/hilt-android>.

25. Monobank open API | API Reference [Електрон. ресурс]. – Режим доступу : <https://api.monobank.ua/docs/>.

26. Налаштування доступу до вашого рахунку [Електрон. ресурс]. – Режим доступу : <https://api.monobank.ua/>.

ДОДАТОК А
Технічне завдання

A.1 Вступ

Програмне забезпечення відслідковування фінансових витрат за геолокацією.

A.2 Призначення розробки

Призначення програмного забезпечення – документування фінансових операцій з доданням геоінформації, розрахунків статистики.

A.3 Вимоги до програми або програмного виробу

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення повинно забезпечувати наступні функціональні можливості:

- додання грошового рахунку з визначенням назви, балансу та типу;
- додання категорій грошових операцій для їх класифікації;
- додання грошової операції, що має включати грошовий рахунок, опис, суму операції, категорію, час та дату та, необов'язково, геодані де проходила ця операція (координати та назва місця). При цьому, додання грошової операції оновлює баланс відповідного рахунку;
- користувач має мати змогу переглядати список грошових операцій та застосовувати фільтр за рахунком до нього;
- перегляд сумарного балансу за всіма грошовими рахунками;
- перегляд статистики грошових операцій та станів балансу рахунку;
- перегляд операцій з геоданими за останній на мапі;
- додання персонального токена банку, для подальшої синхронізації рахунків та грошових операцій користувача. Програмний засіб

повинен перевірити цей токен на коректність та вивести повідомлення для користувача при помилці;

- синхронізація рахунків та грошових операцій користувача з банком, якщо персональний токен є збереженим у програмному застосунку;
- при необхідності користувач має мати можливість видалити персональний токен банку та усі синхронізовані рахунки та операції з програмного застосунку.

А.3.2 Вимоги до надійності

Програмне забезпечення має коректно виконувати свої функції, описані у «Вимогах до функціональних характеристик».

А.3.3 Умови експлуатації

Програмне забезпечення повинно працювати з ОС Android версії 5 або новіше.

А.3.4 Вимоги до складу та параметрів технічних засобів

Мобільний пристрій під керування ОС Android версії 5 або новіше з підтримкою геолокаційних можливостей та доступом до мережі інтернет.

А.3.5 Вимоги до інформаційної та програмної сумісності

До програми вводяться дані рахунків та грошових операцій. Вихідною інформацією є історія грошових операцій у вигляді списку, статистика за введеними даними грошових операцій.

A.4 Техніко-економічні показники

Програмне забезпечення підвищує швидкість та продуктивність документування фінансових витрат, представляє статистичні дані у зручному вигляді для подальшого їх аналізу людиною.

A.5 Стадії та етапи розробки

Далі наведено етапи розробки програмного забезпечення:

- аналіз предметної області;
- розробка архітектури програми;
- реалізація програмного забезпечення;
- тестування програмного забезпечення.

A.6 Порядок контролю та приймання

Програмний застосунок повинно бути протестовано на пристрої, що задовольняє вимогам експлуатації. Програмний застосунок не повинен мати програмних помилок та збоїв у роботі.

Принципи контролю виконання роботи визначаються календарним планом, розташованим у завданні на дипломну кваліфікаційну роботу, де вказано назви етапів, строки їх виконання та результат.

ДОДАТОК Б
Текст програми

Б.1 Текст файла «MonobankService.kt»

```
package com.harukeyua.fintrack.api

import com.harukeyua.fintrack.data.model.api.ClientInfo
import com.harukeyua.fintrack.data.model.api.StatementItem
import okhttp3.OkHttpClient
import okhttp3.logging.HttpLoggingInterceptor
import retrofit2.Retrofit
import retrofit2.converter.moshi.MoshiConverterFactory
import retrofit2.http.GET
import retrofit2.http.Header
import retrofit2.http.Path

interface MonobankService {

    @GET("/personal/client-info")
    suspend fun getAccountInfo(@Header("X-Token") key: String): ClientInfo

    @GET("/personal/statement/{account}/{from}")
    suspend fun getStatements(
        @Header("X-Token") key: String,
        @Path("account") account: String,
        @Path("from") from: Long
    ): List<StatementItem>

    companion object {
        private const val BASE_URL = "https://api.monobank.ua"

        fun create(): MonobankService {
            val interceptor = HttpLoggingInterceptor()
            interceptor.level = HttpLoggingInterceptor.Level.BASIC
            val client =
                OkHttpClient.Builder().addInterceptor(interceptor).build()
            return Retrofit.Builder()
                .baseUrl(BASE_URL)
                .client(client)
                .addConverterFactory(MoshiConverterFactory.create())
                .build()
                .create(MonobankService::class.java)
        }
    }
}
```

Б.2 Текст файла «AccountInfo.kt»

```
package com.harukeyua.fintrack.data.model.api

import com.squareup.moshi.JsonClass

@JsonClass(generateAdapter = true)
data class AccountInfo(
```

```

        val id: String,
        val balance: Long,
        val creditLimit: Long,
        val type: String,
        val currencyCode: Int,
        val cashbackType: String,
        val maskedPan: List<String>
    )

```

Б.3 Текст файла «ClientInfo.kt»

```

package com.harukyua.fintrack.data.model.api

import com.squareup.moshi.JsonClass

@JsonClass(generateAdapter = true)
data class ClientInfo(
    val clientId: String,
    val name: String,
    val accounts: List<AccountInfo>
)

```

Б.4 Текст файла «StatementItem.kt»

```

package com.harukyua.fintrack.data.model.api

import com.squareup.moshi.JsonClass

@JsonClass(generateAdapter = true)
data class StatementItem(
    val id: String,
    val time: Long,
    val description: String,
    val mcc: Int,
    val hold: Boolean,
    val amount: Long,
    val operationAmount: Long,
    val currencyCode: Int,
    val commissionRate: Long,
    val cashbackAmount: Long,
    val balance: Long
)

```

Б.5 Текст файла «Account.kt»

```

package com.harukyua.fintrack.data.model

import androidx.room.Entity

```

```

import androidx.room.PrimaryKey

@Entity(tableName = "accounts")
data class Account(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val type: AccountType,
    val balance: Long,
    val monoCardType: String? = null,
    val monoId: String? = null
) {
    fun getConvertedBalance(): String {
        val balanceString = balance.toString()
        return if (balance == 0L)
            "0.00"
        else
            StringBuilder(balanceString).insert(balanceString.length - 2,
            ".").toString()
    }
}

enum class AccountType {
    CARD, CASH, MONO
}

```

Б.6 Текст файла «AccountWithTransactions.kt»

```

package com.harukeyua.fintrack.data.model

import androidx.room.Embedded
import androidx.room.Relation

data class AccountWithTransactions(
    @Embedded val account: Account,
    @Relation(
        parentColumn = "id",
        entityColumn = "accountId"
    )
    val transactions: List<Transaction>
)

```

Б.7 Текст файла «LocationInfo.kt»

```

package com.harukeyua.fintrack.data.model

data class LocationInfo(
    val lon: Double,
    val lat: Double,
    val name: String = ""
)

```

Б.8 Текст файла «MccCategory.kt»

```
package com.harukyua.fintrack.data.model

import com.squareup.moshi.JsonClass

@JsonClass(generateAdapter = true)
data class MccCategory(
    val mcc: Int,
    val shortDescription: String,
    val fullDescription: String
)
```

Б.9 Текст файла «SyncInfo.kt»

```
package com.harukyua.fintrack.data.model

import androidx.room.Entity
import androidx.room.PrimaryKey
import java.time.OffsetDateTime

@Entity
data class SyncInfo(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val syncDateTime: OffsetDateTime,
    val isSuccess: Boolean
)
```

Б.10 Текст файла «Transaction.kt»

```
package com.harukyua.fintrack.data.model

import androidx.room.*
import java.time.OffsetDateTime

@Entity(
    tableName = "transactions",
    foreignKeys = [
        ForeignKey(
            entity = Account::class,
            childColumns = ["accountId"],
            parentColumns = ["id"],
            onDelete = ForeignKey.CASCADE
        ),
        ForeignKey(
            entity = TransactionType::class,
            childColumns = ["transactionTypeId"],
            parentColumns = ["id"],
            onDelete = ForeignKey.CASCADE
        )
    ]
)
```

```

    )
]
)

data class Transaction(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    @ColumnInfo(index = true) val transactionTypeId: Int,
    @ColumnInfo(index = true) val accountId: Int,
    val amount: Long,
    @Embedded val location: LocationInfo? = null,
    val dateTime: OffsetDateTime,
    val description: String = "",
    val balance: Long,
    val monoId: String = ""
)

```

Б.11 Текст файла «TransactionInfo.kt»

```

package com.harukyua.fintrack.data.model

import androidx.room.DatabaseView
import java.time.OffsetDateTime

@DatabaseView(
    "SELECT transactions.id, transactions.amount," +
        "      transactions.dateTime,      transactionTypes.name      AS
transactionTypeName" +
        " FROM transactions INNER JOIN transactionTypes" +
        " ON transactions.transactionTypeId = transactionTypes.id"
)
data class TransactionInfo(
    val id: Int,
    val transactionTypeName: String,
    val amount: Long,
    val dateTime: OffsetDateTime
)

```

Б.12 Текст файла «TransactionListItem.kt»

```

package com.harukyua.fintrack.data.model

import java.time.OffsetDateTime

sealed class TransactionListItem {
    data class Item(val transaction: Transaction): TransactionListItem()
    data class Separator(val offsetDateTime: OffsetDateTime):
TransactionListItem()
}

```

Б.13 Текст файлу «TransactionType.kt»

```
package com.harukeyua.fintrack.data.model

import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "transactionTypes")
data class TransactionType(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val name: String,
    val mccCode: Int = -1
)
```

Б.14 Текст файлу «TransactionTypeWithTransactions.kt»

```
package com.harukeyua.fintrack.data.model

import androidx.room.Embedded
import androidx.room.Relation

data class TransactionTypeWithTransactions(
    @Embedded val transactionType: TransactionType,
    @Relation(
        parentColumn = "id",
        entityColumn = "transactionTypeId"
    )
    val transactions: List<Transaction>
)
```

Б.15 Текст файлу «Converters.kt»

```
package com.harukeyua.fintrack.data

import androidx.room.TypeConverter
import java.time.OffsetDateTime
import java.time.format.DateTimeFormatter

object Converters {
    private val formatter = DateTimeFormatter.ISO_OFFSET_DATE_TIME

    @TypeConverter
    @JvmStatic
    fun toOffsetDateTime(value: String): OffsetDateTime {
        return formatter.parse(value, OffsetDateTime::from)
    }

    @TypeConverter
    @JvmStatic
```

```

    fun fromOffsetDateTime(date: OffsetDateTime): String {
        return date.format(formatter)
    }
}

```

Б.16 Текст файла «FinDao.kt»

```

package com.harukeyua.fintrack.data

import androidx.lifecycle.LiveData
import androidx.paging.PagingSource
import androidx.room.*
import com.harukeyua.fintrack.data.model.*
import com.harukeyua.fintrack.data.model.Transaction
import java.time.OffsetDateTime

@Dao
interface FinDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTransactionType(transactionType: TransactionType)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertAccount(account: Account)

    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insertAccounts(account: List<Account>)

    @Update(onConflict = OnConflictStrategy.ABORT)
    suspend fun updateAccount(account: Account)

    @Update(onConflict = OnConflictStrategy.IGNORE)
    suspend fun updateAccounts(accounts: List<Account>)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTransaction(transaction: Transaction)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertTransaction(transaction: List<Transaction>)

    @Query("SELECT * FROM transactions ORDER BY datetime(dateTime) DESC")
    fun getTransactions(): LiveData<List<Transaction>>

    @Query("SELECT * FROM transactions ORDER BY datetime(dateTime) DESC")
    fun getTransactionsPaging(): PagingSource<Int, Transaction>

    @Query("SELECT * FROM transactions WHERE accountId = :accountId ORDER BY datetime(dateTime) DESC")
    fun getTransactionsPagingByAccount(accountId: Int): PagingSource<Int, Transaction>

    @Query("SELECT * FROM transactionTypes ORDER BY name ASC")
    fun getTransactionTypes(): LiveData<List<TransactionType>>
}

```

```

@Query("SELECT * FROM accounts ORDER BY name ASC")
fun getAccounts(): LiveData<List<Account>>

@Query("SELECT * FROM accounts ORDER BY name ASC")
suspend fun getAccountsList(): List<Account>

@Query("SELECT * FROM transactions WHERE dateTime(dateTime) BETWEEN
dateTime(:startDate) AND dateTime(:endDate) ORDER By dateTime(dateTime)")
fun getTransactionsInDateRange(
    startDate: OffsetDateTime,
    endDate: OffsetDateTime
): LiveData<List<Transaction>>

@Query("SELECT * FROM TransactionInfo WHERE dateTime(dateTime) BETWEEN
dateTime(:startDate) AND dateTime(:endDate) ORDER By dateTime(dateTime)")
fun getTransactionsInfoInDateRange(
    startDate: OffsetDateTime,
    endDate: OffsetDateTime
): LiveData<List<TransactionInfo>>

@Query("SELECT * FROM transactions WHERE dateTime(dateTime) BETWEEN
dateTime(:startDate) AND dateTime(:endDate) ORDER By dateTime(dateTime)")
suspend fun getTransactionsListInDateRange(
    startDate: OffsetDateTime,
    endDate: OffsetDateTime
): List<Transaction>

@Query("DELETE FROM accounts WHERE type = \"MONO\"")
suspend fun deleteMonoAccounts()

@Query("SELECT * FROM accounts WHERE type = \"MONO\" ORDER BY name ASC")
suspend fun getMonoAccountsList(): List<Account>

@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insertSyncInfo(item: SyncInfo)

@Query("SELECT * FROM SyncInfo ORDER BY id DESC")
fun getLatestSyncInfo(): LiveData<List<SyncInfo>>

@Query("SELECT * FROM SyncInfo ORDER BY id DESC")
suspend fun getLatestSyncInfoList(): List<SyncInfo>

@Query("SELECT * FROM transactionTypes WHERE NOT mccCode = -1")
suspend fun getMccTransactionTypesList(): List<TransactionType>

@Query("DELETE FROM SyncInfo")
suspend fun deleteSyncInfo()
}

```

Б.17 Текст файла «FinDatabase.kt»

```
package com.harukyua.fintrack.data
```

```

import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.room.TypeConverters
import com.harukyua.fintrack.data.model.*

const val DATABASE_NAME = "fin.db"

@Database(
    entities = [TransactionType::class, Account::class, Transaction::class,
SyncInfo::class],
    views = [TransactionInfo::class],
    version = 1,
    exportSchema = false
)
@TypeConverters(Converters::class)
abstract class FinDatabase : RoomDatabase() {
    abstract fun finDao(): FinDao

    companion object {
        @Volatile
        private var instance: FinDatabase? = null

        fun getInstance(context: Context): FinDatabase {
            return instance ?: synchronized(this) {
                instance ?: buildDatabase(context).also { instance = it }
            }

            private fun buildDatabase(context: Context): FinDatabase {
                return Room.databaseBuilder(context, FinDatabase::class.java,
DATABASE_NAME).build()
            }
        }
    }
}

```

Б.18 Текст файла «ApiModule.kt»

```

package com.harukyua.fintrack.di

import com.harukyua.fintrack.api.MonobankService
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

@InstallIn(SingletonComponent::class)
@Module
class ApiModule {

    @Singleton

```

```

    @Provides
    fun provideMonobankService(): MonobankService = MonobankService.create()
}

```

Б.19 Текст файла «DatabaseModule.kt»

```

package com.harukyua.fintrack.di

import android.content.Context
import com.harukyua.fintrack.data.FinDatabase
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent
import javax.inject.Singleton

@InstallIn(SingletonComponent::class)
@Module
class DatabaseModule {

    @Singleton
    @Provides
    fun provideFinDatabase(@ApplicationContext context: Context) =
        FinDatabase.getInstance(context)

    @Provides
    fun provideFinDao(finDatabase: FinDatabase) = finDatabase.finDao()
}

```

Б.20 Текст файла «UtilsModule.kt»

```

package com.harukyua.fintrack.di

import android.content.Context
import android.content.SharedPreferences
import androidx.security.crypto.EncryptedSharedPreferences
import androidx.security.crypto.MasterKey
import com.harukyua.fintrack.utils.encryptedPrefFile
import dagger.Module
import dagger.Provides
import dagger.hilt.InstallIn
import dagger.hilt.android.qualifiers.ApplicationContext
import dagger.hilt.components.SingletonComponent

@Module
@InstallIn(SingletonComponent::class)
object UtilsModule {

    @Provides
    fun provideSecureEncryptedSharedPreferences(

```

```

        @ApplicationContext context: Context,
        masterKey: MasterKey
    ): SharedPreferences {
        return EncryptedSharedPreferences.create(
            context,
            ENCRYPTED_PREF_FILE,
            masterKey,
            EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
            EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
        )
    }

    @Provides
    fun provideMasterKey(@ApplicationContext context: Context): MasterKey {
        return MasterKey.Builder(context)
            .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
            .build()
    }
}

```

Б.21 Текст файлу «FinEditRepo.kt»

```

package com.harukyua.fintrack.repos

import com.harukyua.fintrack.data.FinDao
import com.harukyua.fintrack.data.model.Account
import com.harukyua.fintrack.data.model.Transaction
import com.harukyua.fintrack.data.model.TransactionType
import javax.inject.Inject

class FinEditRepo @Inject constructor(private val dao: FinDao) {

    suspend fun insertAccount(item: Account) {
        dao.insertAccount(item)
    }

    suspend fun updateAccount(item: Account) {
        dao.updateAccount(item)
    }

    suspend fun insertType(type: TransactionType) {
        dao.insertTransactionType(type)
    }

    suspend fun insertTransaction(transaction: Transaction) {
        dao.insertTransaction(transaction)
    }

    suspend fun deleteMonoAccounts() {
        dao.deleteMonoAccounts()
        dao.deleteSyncInfo()
    }
}

```

Б.22 Текст файла «FinInfoRepo.kt»

```
package com.harukyua.fintrack.repos

import androidx.lifecycle.LiveData
import androidx.paging.Pager
import androidx.paging.PagingConfig
import androidx.paging.insertSeparators
import androidx.paging.map
import com.harukyua.fintrack.data.FinDao
import com.harukyua.fintrack.data.model.Account
import com.harukyua.fintrack.data.model.Transaction
import com.harukyua.fintrack.data.model.TransactionInfo
import com.harukyua.fintrack.data.model.TransactionListItem
import kotlinx.coroutines.flow.map
import java.time.OffsetDateTime
import javax.inject.Inject

class FinInfoRepo @Inject constructor(private val dao: FinDao) {

    val accountsList = dao.getAccounts()

    val typesList = dao.getTransactionTypes()

    fun transactionsListByStore(storeId: Int?) = Pager(
        config = PagingConfig(
            pageSize = 60,
            enablePlaceholders = true,
            maxSize = 200
        )
    ) {
        if (storeId == null) {
            dao.getTransactionsPaging()
        } else {
            dao.getTransactionsPagingByAccount(storeId)
        }
    }.flow.map { pagingData ->
        pagingData.map { item -> TransactionListItem.Item(item) }
            .insertSeparators { before: TransactionListItem.Item?, after:
TransactionListItem.Item? ->
                if (before == null && after == null) {
                    // List is empty after fully loaded; return null to skip
adding separator.
                    null
                } else if (after == null) {
                    // Footer; return null here to skip adding a footer.
                    null
                } else if (before == null) {
                    TransactionListItem.Separator(after.transaction.dateTime)
                } else if (after.transaction.dateTime.dayOfMonth !=
before.transaction.dateTime.dayOfMonth) {
                    TransactionListItem.Separator(after.transaction.dateTime)
                } else {
                    null
                }
            }
    }
```

```

        }
    }
}

fun getTransactionsInDateRange(range: Pair<OffsetDateTime,
OffsetDateTime>): LiveData<List<Transaction>> {
    return dao.getTransactionsInDateRange(
        range.first,
        range.second.plusDays(1).withHour(0).withMinute(0)
    )
}

fun getTransactionsInfoInDateRange(range: Pair<OffsetDateTime,
OffsetDateTime>): LiveData<List<TransactionInfo>> {
    return dao.getTransactionsInfoInDateRange(
        range.first,
        range.second.plusDays(1).withHour(0).withMinute(0)
    )
}

suspend fun getAccountsList(): List<Account> {
    return dao.getAccountsList()
}

suspend fun getTransactionsListInDateRange(range: Pair<OffsetDateTime,
OffsetDateTime>): List<Transaction> {
    return dao.getTransactionsListInDateRange(range.first, range.second)
}
}

```

Б.23 Текст файлу «MonobankApiRepo.kt»

```

package com.harukyua.fintrack.repos

import android.util.Log
import androidx.lifecycle.LiveData
import androidx.room.withTransaction
import com.harukyua.fintrack.api.MonobankService
import com.harukyua.fintrack.data.FinDatabase
import com.harukyua.fintrack.data.model.Account
import com.harukyua.fintrack.data.model.AccountType
import com.harukyua.fintrack.data.model.api.AccountInfo
import com.harukyua.fintrack.utils.Resource
import retrofit2.HttpException
import java.io.IOException
import javax.inject.Inject

class MonobankApiRepo @Inject constructor(
    private val service: MonobankService,
    private val db: FinDatabase
) {

    fun initialMonoClientInfoSync(key: String) =

```

```

        LiveData {
            emit(Resource.Loading)
            try {
                val clientInfo = service.getAccountInfo(key)
                replaceMonoAccounts(clientInfo.accounts)
                emit(Resource.Success(clientInfo))
            } catch (e: HttpException) {
                Log.e(TAG, "Error requesting client info", e)
                emit(Resource.ApiError(e.code()))
            } catch (e: IOException) {
                Log.e(TAG, "Error requesting client info", e)
                emit(Resource.ExceptionError(e.message ?: ""))
            } catch (e: Exception) {
                Log.e(TAG, "Error requesting client info", e)
                emit(Resource.ExceptionError(e.message ?: ""))
            }
        }
    }

    fun updateMonoClientInfoSync(key: String) =
        LiveData {
            emit(Resource.Loading)
            try {
                val clientInfo = service.getAccountInfo(key)
                updateMonoAccounts(clientInfo.accounts)
                emit(Resource.Success(clientInfo))
            } catch (e: HttpException) {
                Log.e(TAG, "Error requesting client info", e)
                emit(Resource.ApiError(e.code()))
            } catch (e: IOException) {
                Log.e(TAG, "Error requesting client info", e)
                emit(Resource.ExceptionError(e.message ?: ""))
            } catch (e: Exception) {
                Log.e(TAG, "Error requesting client info", e)
                emit(Resource.ExceptionError(e.message ?: ""))
            }
        }
    }

    private suspend fun replaceMonoAccounts(monoAccounts: List<AccountInfo>)
    {
        db.withTransaction {
            db.finDao().deleteMonoAccounts()
            val accountsToInsert =
                monoAccounts.filter { account -> account.currencyCode ==
UAH_CODE }
            db.finDao().insertAccounts(accountsToInsert.map {
                Account(
                    name = it.type,
                    type = AccountType.MONO,
                    balance = it.balance,
                    monoCardType = it.type,
                    monoId = it.id
                )
            })
        }
    }
}

```

```

private suspend fun updateMonoAccounts(monoAccounts: List<AccountInfo>) {
    db.withTransaction {
        val monoAccountsInDb = db.finDao().getMonoAccountsList()

        // Insert new accounts if any
        val monoAccountsToInsert =
            monoAccounts.filter { account -> monoAccountsInDb.find {
it.monoCardType == account.type } == null && account.currencyCode == UAH_CODE
}

        db.finDao().insertAccounts(monoAccountsToInsert.map {
            Account(
                name = it.type,
                type = AccountType.MONO,
                balance = it.balance,
                monoCardType = it.type,
                monoId = it.id
            )
        })

        // Update existing accounts
        val monoAccountsToUpdate =
            monoAccountsInDb.mapNotNull { account ->
                val monoAccount = monoAccounts.find { it.type ==
account.monoCardType }
                if (monoAccount != null)
                    account.copy(balance = monoAccount.balance)
                else
                    null
            }
        db.finDao().updateAccounts(monoAccountsToUpdate)
    }
}

companion object {
    const val TAG = "MonobankApiRepo"
    const val UAH_CODE = 980
}
}

```

Б.24 Текст файлу «MonobankSyncRepo.kt»

```

package com.harukyua.fintrack.repos

import android.util.Log
import androidx.room.withTransaction
import com.harukyua.fintrack.api.MonobankService
import com.harukyua.fintrack.data.FinDatabase
import com.harukyua.fintrack.data.model.*
import com.harukyua.fintrack.data.model.api.AccountInfo
import com.harukyua.fintrack.data.model.api.ClientInfo
import com.harukyua.fintrack.data.model.api.StatementItem
import com.harukyua.fintrack.utils.Resource
import retrofit2.HttpException

```

```

import java.io.IOException
import java.time.Instant
import java.time.OffsetDateTime
import java.time.ZoneOffset
import javax.inject.Inject

class MonobankSyncRepo @Inject constructor(
    private val service: MonobankService,
    private val db: FinDatabase
) {

    suspend fun monoClientInfoSync(
        key: String,
        mccList: List<MccCategory>
    ): Resource<ClientInfo> {
        return try {
            val clientInfo = service.getAccountInfo(key)
            updateMonoAccounts(clientInfo.accounts)
            syncTransactions(key, mccList)
            Resource.Success(clientInfo)
        } catch (e: HttpException) {
            Log.e(TAG, "Error requesting client info", e)
            db.finDao().insertSyncInfo(SyncInfo(1, OffsetDateTime.now(),
false))
            Resource.ApiError(e.code())
        } catch (e: IOException) {
            Log.e(TAG, "Error requesting client info, no connection?", e)
            db.finDao().insertSyncInfo(SyncInfo(1, OffsetDateTime.now(),
false))
            Resource.ExceptionError(e.message ?: "")
        } catch (e: Exception) {
            Log.e(TAG, "Error requesting client info, something borked", e)
            db.finDao().insertSyncInfo(SyncInfo(1, OffsetDateTime.now(),
false))
            Resource.ExceptionError(e.message ?: "")
        }
    }

    private suspend fun updateMonoAccounts(monoAccounts: List<AccountInfo>) {
        db.withTransaction {
            val monoAccountsInDb = db.finDao().getMonoAccountsList()

            // Insert new accounts if any
            val monoAccountsToInsert =
                monoAccounts.filter { account -> monoAccountsInDb.find {
it.monoCardType == account.type } == null && account.currencyCode ==
MonobankApiRepo.UAH_CODE }
            db.finDao().insertAccounts(monoAccountsToInsert.map {
                Account(
                    name = it.type,
                    type = AccountType.MONO,
                    balance = it.balance,
                    monoCardType = it.type,
                    monoId = it.id
                )
            })
        }
    }
}

```

```

        // Update existing accounts
        val monoAccountsToUpdate =
            monoAccountsInDb.mapNotNull { account ->
                val monoAccount = monoAccounts.find { it.type ==
account.monoCardType }
                if (monoAccount != null)
                    account.copy(balance = monoAccount.balance)
                else
                    null
            }
        db.finDao().updateAccounts(monoAccountsToUpdate)
    }
}

private suspend fun syncTransactions(key: String, mcclList:
List<MccCategory>) {
    val syncInfo = db.finDao().getLatestSyncInfoList()
    val monoAccounts = db.finDao().getMonoAccountsList()
    val from = if (syncInfo.isEmpty()) OffsetDateTime.now()
        .minusMonths(1) else syncInfo.first().syncDateTime

    monoAccounts.forEach { account ->
        val statements =
            service.getStatements(key, account.monoId ?: "0",
from.toEpochSecond())

        val mccCategories = insertMissingTypes(statements, mcclList)

        val toInsert = statements.map { statement ->
            val transactionType = mccCategories.find { it.mccCode ==
statement.mcc }
            val dateTime = OffsetDateTime.ofInstant(
                Instant.ofEpochSecond(statement.time),
                ZoneOffset.systemDefault()
            )
            Transaction(
                transactionTypeId = transactionType!!.id,
                accountId = account.id,
                amount = statement.amount,
                description = statement.description,
                balance = statement.balance,
                monoId = statement.id,
                dateTime = dateTime
            )
        }

        db.withTransaction {
            db.finDao().insertSyncInfo(SyncInfo(1, OffsetDateTime.now(),
true))
            db.finDao().insertTransaction(toInsert)
            Log.i(
                TAG,
                "Successfully inserted ${toInsert.size} transaction
entries for account ${account.monoCardType}"
            )
        }
    }
}

```

```

    }

    }

    private suspend fun insertMissingTypes(
        statementsList: List<StatementItem>,
        mccList: List<MccCategory>
    ): List<TransactionType> {
        val mccCategories =
db.finDao().getMccTransactionTypesList().toMutableList()
        db.withTransaction {
            // Check categories and add missing ones to db
            statementsList.forEach { statement ->
                if (mccCategories.find { it.mccCode == statement.mcc } ==
null) {
                    db.finDao().insertTransactionType(
                        TransactionType(
                            name = mccList.find { it.mcc == statement.mcc
}?.shortDescription
                                ?: "Unknown",
                                mccCode = statement.mcc
                            ).also { mccCategories.add(it) }
                        )
                    }
                }
            }
        return db.finDao().getMccTransactionTypesList()
    }
    companion object {
        const val TAG = "MonobankSyncRepo"
    }
}

```

Б.25 Текст файла «SyncInfoRepo.kt»

```

package com.harukyua.fintrack.repos

import com.harukyua.fintrack.data.FinDao
import javax.inject.Inject

class SyncInfoRepo @Inject constructor(private val dao: FinDao) {

    val lastSyncInfo = dao.getLatestSyncInfo()
}

```

Б.26 Текст файла «AccountListAdapter.kt»

```

package com.harukyua.fintrack.ui.adapters

import android.text.SpannableStringBuilder

```

```

import android.text.Spanned
import android.text.style.RelativeSizeSpan
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.core.text.toSpannable
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.harukyuua.fintrack.R
import com.harukyuua.fintrack.data.model.Account
import com.harukyuua.fintrack.data.model.AccountType
import com.harukyuua.fintrack.databinding.AddNewMoneyStoreItemBinding
import com.harukyuua.fintrack.databinding.MoneyStoreCardItemBinding
import com.harukyuua.fintrack.databinding.TotalBalanceCardBinding
import com.harukyuua.fintrack.utils.getThemedColor
import java.util.*

class AccountListAdapter(val onAddClick: () -> Unit) :
    ListAdapter<Account, RecyclerView.ViewHolder>(AccountDiffCallback()) {

    private enum class ViewTypes { TOTAL, ADD, ACCOUNT }

    var totalBalance: String? = null
        set(value) {
            field = value
            this.notifyItemChanged(0)
        }

    fun getCurrentAccount(position: Int): Account? {
        val account = getItem(position)
        return if (account.id != -1 && account.id != -2) account else null
    }

    override fun submitList(list: List<Account>?) {
        // Additional element serves as last "add" card and first "total"
card
        val listWithFooter = list?.toMutableList().also {
            // "Total" card
            if (list?.isEmpty() == true) {
                it?.add(
                    0,
                    Account(
                        id = -1,
                        name = "",
                        type = AccountType.CASH,
                        balance = 0
                    )
                )
            }
            // "Add" card
            it?.add(
                Account(
                    id = -2,
                    name = "",
                    type = AccountType.CASH,
                    balance = 0
                )
            )
        }
    }

```

```

        )
    )
}
super.submitList(listWithFooter)
}

inner class TotalBalanceViewHolder(private val binding:
TotalBalanceCardBinding) :
    RecyclerView.ViewHolder(binding.root) {

    fun bind(balance: String?) {
        balance?.let {
            val spanned = SpannableStringBuilder(balance)
            spanned.append(" ")
            spanned.append(
                binding.root.context.getString(R.string.uah_label),
                RelativeSizeSpan(0.4f),
                Spanned.SPAN_EXCLUSIVE_EXCLUSIVE
            )
            binding.balanceAmount.text = spanned.toSpannable()
        }
    }
}

inner class AddAccountViewHolder(private val binding:
AddNewMoneyStoreItemBinding) :
    RecyclerView.ViewHolder(binding.root) {

    fun bind() {
        binding.root.setOnClickListener {
            onAddClick()
        }
    }
}

class AccountViewHolder(private val binding: MoneyStoreCardItemBinding) :
    RecyclerView.ViewHolder(binding.root) {

    fun bind(item: Account) {
        with(binding) {
            val cardColor = when (item.type) {
                AccountType.CARD ->
root.context.getThemedColor(R.attr.colorMoneyStoreCard)
                AccountType.CASH ->
root.context.getThemedColor(R.attr.colorMoneyStoreCash)
                AccountType.MONO ->
root.context.getThemedColor(R.attr.colorMoneyStoreMonoBlack)
            }
            moneyStoreCard.setCardBackgroundColor(cardColor)

            val iconDrawable = when (item.type) {
                AccountType.CARD -> R.drawable.ic_credit_card
                AccountType.CASH -> R.drawable.ic_wallet
                AccountType.MONO -> R.drawable.ic_credit_card_sync
            }

```

```

        moneyStoreIcon.setImageResource(iconDrawable)
        balanceDescription.setText(R.string.store_balance)
        val spanned = SpannableStringBuilder(item.getConvertedBalance())
        spanned.append(" ")
        spanned.append(
            binding.root.context.getString(R.string.uah_label),
            RelativeSizeSpan(0.4f),
            Spanned.SPAN_EXCLUSIVE_EXCLUSIVE
        )
        balanceAmount.text = spanned.toSpannable()
        storeName.text = item.name.trim().capitalize(Locale.getDefault())
    }
}

override fun getItemViewType(position: Int): Int {
    return when (getItem(position).id) {
        -1 -> ViewTypes.TOTAL.ordinal
        -2 -> ViewTypes.ADD.ordinal
        else -> ViewTypes.ACCOUNT.ordinal
    }
}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
RecyclerView.ViewHolder {
    return when (viewType) {
        ViewTypes.TOTAL.ordinal -> TotalBalanceViewHolder(
            TotalBalanceCardBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )
        ViewTypes.ADD.ordinal -> AddAccountViewHolder(
            AddNewMoneyStoreItemBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )
        else -> AccountViewHolder(
            MoneyStoreCardItemBinding.inflate(
                LayoutInflater.from(parent.context),
                parent,
                false
            )
        )
    }
}

override fun onBindViewHolder(holder: RecyclerView.ViewHolder, position:
Int) {
    if (holder is AccountViewHolder)
        holder.bind(getItem(position))
}

```

```

        else if (holder is AddAccountViewHolder)
            holder.bind()
        else if (holder is TotalBalanceViewHolder)
            holder.bind(totalBalance)
    }

    private class AccountDiffCallback : DiffUtil.ItemCallback<Account>() {
        override fun areItemsTheSame(oldItem: Account, newItem: Account):
Boolean {
            return oldItem.id == newItem.id
        }

        override fun areContentsTheSame(oldItem: Account, newItem: Account):
Boolean {
            return oldItem == newItem
        }
    }
}

```

Б.27 Текст файла «AccountListChooserAdapter.kt»

```

package com.harukyua.fintrack.ui.adapters

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.harukyua.fintrack.R
import com.harukyua.fintrack.data.model.Account
import com.harukyua.fintrack.data.model.AccountType
import com.harukyua.fintrack.databinding.AccountSmallCardBinding
import com.harukyua.fintrack.utils.getThemedColor
import java.util.*

class AccountListChooserAdapter :
    ListAdapter<Account,
AccountListChooserAdapter.AccountViewHolder>(AccountDiffCallback()) {

    private var selectedCard = 0
    private set(value) {
        val oldValue = selectedCard
        field = value
        notifyItemChanged(oldValue)
        notifyItemChanged(value)
    }

    fun getSelectedAccount(): Account? = if (itemCount != 0)
getItem(selectedCard) else null

    inner class AccountViewHolder(private val binding:
AccountSmallCardBinding) :

```

```

RecyclerView.ViewHolder(binding.root) {

    init {
        binding.root.setOnClickListener {
            selectedCard = layoutPosition
        }
    }

    fun bind(item: Account, position: Int) {
        with(binding) {
            val cardColor = when (item.type) {
                AccountType.CARD ->
root.context.getThemedColor(R.attr.colorMoneyStoreCard)
                AccountType.CASH ->
root.context.getThemedColor(R.attr.colorMoneyStoreCash)
                AccountType.MONO ->
root.context.getThemedColor(R.attr.colorMoneyStoreMonoBlack)
            }
            accountCard.setCardBackgroundColor(cardColor)

            val iconDrawable = when (item.type) {
                AccountType.CARD -> R.drawable.ic_credit_card
                AccountType.CASH -> R.drawable.ic_wallet
                AccountType.MONO -> R.drawable.ic_credit_card
            }
            accountIcon.setImageResource(iconDrawable)
            accountName.text =
item.name.trim().capitalize(Locale.getDefault())

            root.isChecked = selectedCard == position

            if (android.os.Build.VERSION.SDK_INT >=
android.os.Build.VERSION_CODES.O) {
                root.tooltipText = item.name
            }
        }
    }
}

override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
AccountViewHolder {
    return AccountViewHolder(
        AccountSmallCardBinding.inflate(
            LayoutInflater.from(parent.context),
            parent,
            false
        )
    )
}

override fun onBindViewHolder(holder: AccountViewHolder, position: Int) {
    holder.bind(getItem(position), position)
}

private class AccountDiffCallback : DiffUtil.ItemCallback<Account>() {

```

```
        override fun areItemsTheSame(oldItem: Account, newItem: Account):  
Boolean {  
            return oldItem.id == newItem.id  
        }  
  
        override fun areContentsTheSame(oldItem: Account, newItem: Account):  
Boolean {  
            return oldItem == newItem  
        }  
    }  
}
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ВІДСЛІДКУВАННЯ
ФІНАНСОВИХ ВИТРАТ ЗА ГЕОЛОКАЦІЄЮ
SOFTWARE FOR GEOLOCATION-BASED TRACKING OF
FINANCIAL EXPENSES

Виконав
Студент групи кнт-217

Н. А. Руснак

Керівник
к.т.н, доцент

В. М. Льовкін

2021

Рисунок В.1 – Слайд 1

2

- Об'єкт дослідження – процес відслідковування фінансових витрат.
- Предмет дослідження – програмні засоби відслідковування фінансових витрат за геолокацією.
- Мета роботи – розробка спеціалізованого програмного забезпечення відслідковування фінансових витрат для підвищення ефективності подальшого аналізу даних грошових операцій людиною та зменшення витрати часу на процес їх документування з використанням геоінформації як одного з критеріїв класифікації даних

Рисунок В.2 – Слайд 2

Завдання роботи

3

- аналіз існуючих рішень, виділення їх недоліків та переваг;
- постановка вимог до розроблюваного програмного засобу;
- проєктування архітектури програмного засобу;
- розробка програмного застосунку.

Рисунок В.3 – Слайд 3

Порівняння існуючих рішень

4

Критерій порівняння	Monefy	Money manager	Money Lover
Підтримка операційних систем	Android, IOS, Microsoft Windows	Android, IOS	Android, IOS
Підтримка української мови	Є	Немає	Є
Функціональні можливості	Базові	Базові	Розширені
Наявність реклами	Немає	Немає	Є
Використання типових елементів інтерфейсу	Ні	Так	Так
Підтримка збереження геоінформації	Ні	Ні	Ні

Рисунок В.4 – Слайд 4

Діаграма прецедентів

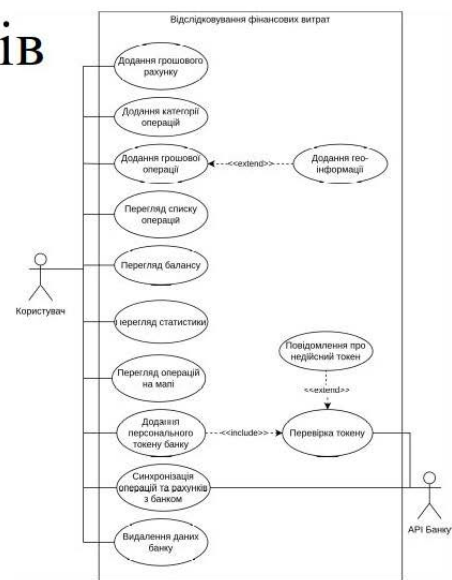


Рисунок В.5 – Слайд 5

Вибір мови програмування

Предмет порівняння	Java	Kotlin
Null безпека	Немає	Розділення типів на ті, що можуть бути null і ті, що не можуть.
Функції розширення	Додання функцій до існуючих класів можливо лише через спадкування	Є
Класи даних	Тільки створення власноруч	Вбудований клас за ключовим словом «data»
Розумні перетворення типів	Необхідно проводити перевірку елементів до перетворення	Вбудована підтримка перевірки (ключове слово «is»)
Типи	Статична типізація. Необхідно завжди вказувати тип змінної	Статична типізація. Автоматичне визначення типу на основі присвоюваного значення.
Функціональне програмування	Підтримка лямбда функцій	Підтримка лямбда функцій, переваження операторів, операцій над списками та інше.
Статичні члени класу	Мас	Немає, але є альтернативи
Наявність підтримки Android	Є	Є, рекомендовано корпорацією Google

Рисунок В.6 – Слайд 6

Архітектурний шаблон

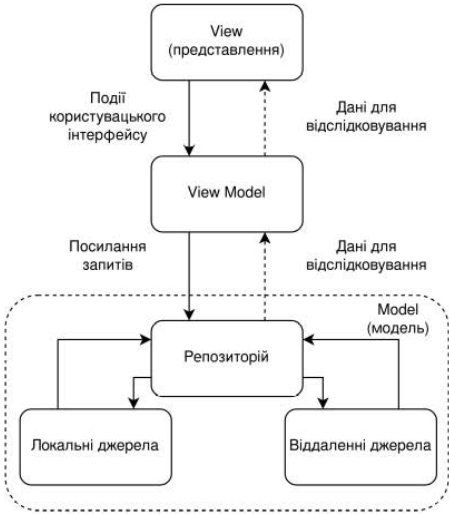


Рисунок В.7 – Слайд 7

Структура бази даних

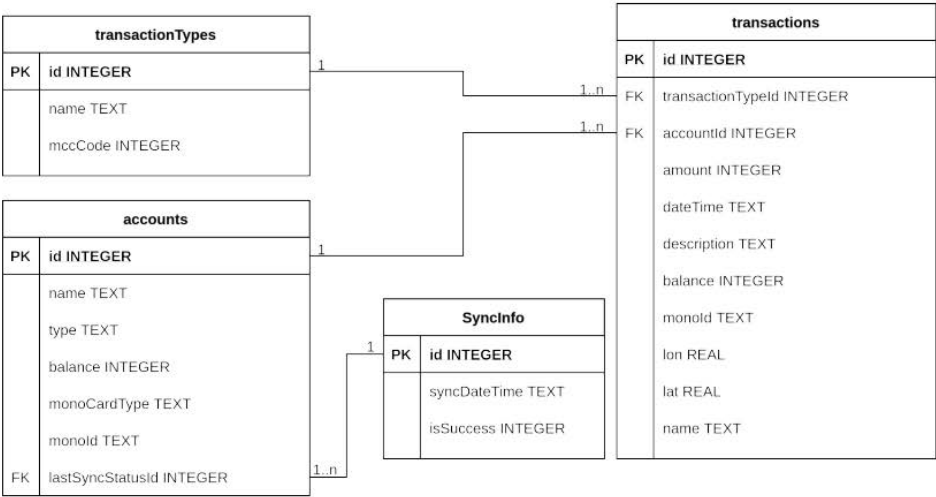
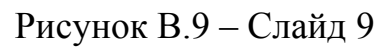
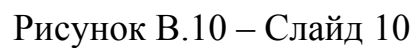


Рисунок В.8 – Слайд 8

9



10



Алгоритм синхронізації грошових рахунків

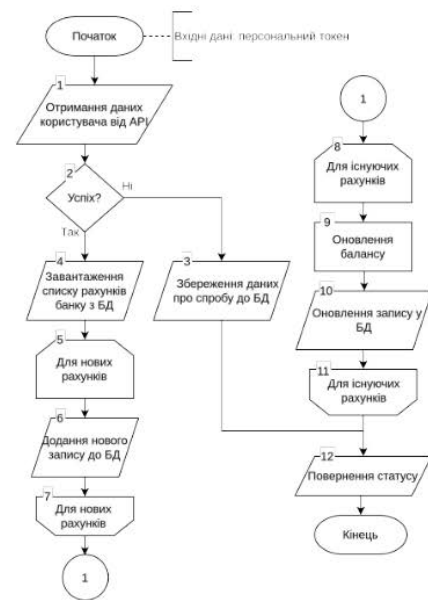


Рисунок В.11 – Слайд 11

Екранні форми

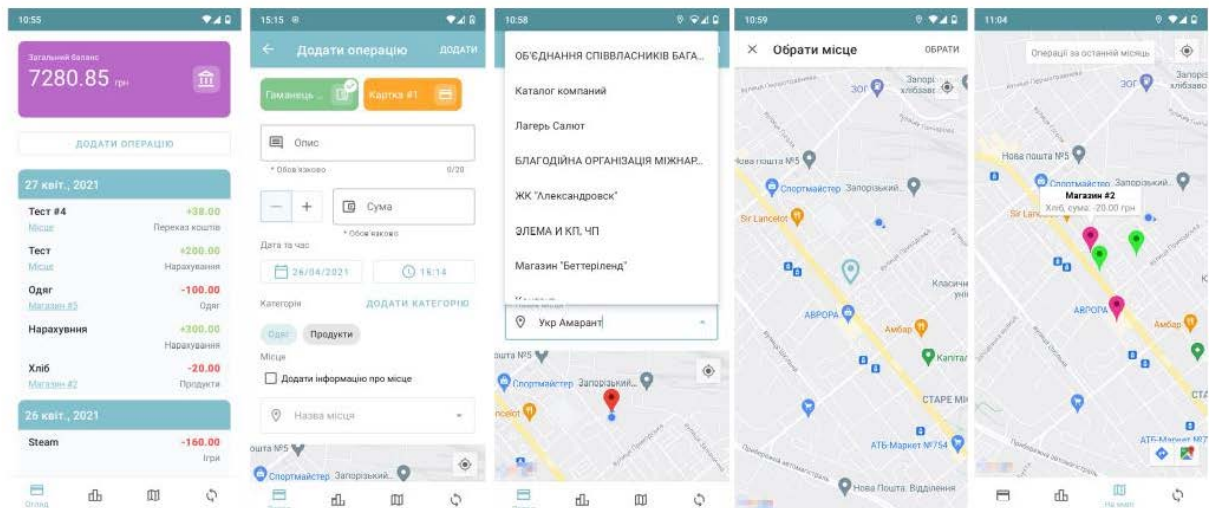


Рисунок В.12 – Слайд 12

Екранні форми

13

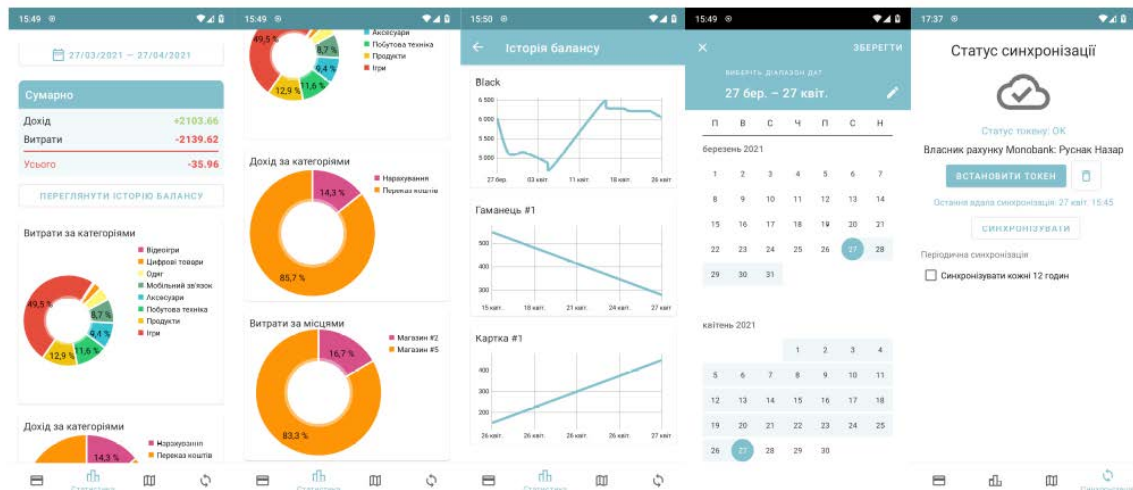


Рисунок В.13 – Слайд 13

Висновки

14

У роботі було досліджено процес відслідковування фінансових витрат результатом чого було розроблено програмний застосунок для операційної системи Android.

Було проаналізовано існуючі програмні засоби вирішення проблеми відслідковування фінансових витрат, виділено їх переваги, недоліки та запропоновано додання додаткових функціональних можливостей, таких як використання геоінформації та синхронізації з банком до розроблюваного програмного застосунку та висунуто функціональні вимоги.

Було вирішено питання розробки архітектури програмного застосунку: проаналізовано та обрано мову програмування Kotlin, проаналізовано архітектурні шаблони для розробки програмних застосунків, що дало підстави для вибору MVVM шаблону архітектури, розроблено структуру бази даних та програмного забезпечення.

Розроблено програмний застосунок відслідковування фінансових витрат з використанням геоінформаційних можливостей мобільних пристроїв, що задовольняє поставленим функціональним вимогам

Рисунок В.14 – Слайд 14