Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту»

Виконав: студент II курсу
групи ЗПІ-22м спеціальності
121 – Інженерія програмного забезпечення
Ярмола В.С.

Керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.
«13» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ Кожем'яко А.В.
«13» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
к.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«13» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Ярмолі Віталію Сергійовичу

1. Тема роботи – Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту.

Керівник роботи: Майданюк Володимир Павлович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи : 5 грудня 2023 р.

3. Вихідні дані: модель розробки – каскадна; шаблон проектування – MVVM; операційна система – Android; середовище розробки – Android Studio; мова програмування – Kotlin.

4.Зміст текстової частини: вступ; обґрунтування вибору методу розробки та постановка задач; розробка методів та алгоритмів реалізації програмних засобів; розробка програмних засобів для контролю витрат; тестування програмного засобу; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): мета, предмет та об'єкт дослідження; актуальність розробки; аналіз аналогів; розробка загальної архітектури додатку; програмування бази даних; розробка блок-схеми алгоритму роботи програми; аналіз та обґрунтування вибору програмних засобів для розробки; вибір середовища розробки; програмна реалізація.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–4	Майданюк В.П. к.т.н., доцент кафедри ПЗ	19.09.23	05.12.23
5	Причепа І.В. к.е.н. доцент кафедри ЕПВМ	19.11.23	26.11.23

7. Дата видачі завдання 19 вересня 2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітк
1	Обгрунтування вибору методу розробки та постановка задач	20.09.2023 – 30.09.2023	вик
2	Розробка методів та алгоритмів реалізації програмних засобів	01.10.2023 – 10.10.2023	вик
3	Аналіз і вибір мови програмування та середовища розробки	11.10.2023 – 14.10.2023	вик
4	Розробка програмних засобів для контролю витрат	15.10.2023 – 14.11.2023	вик
5	Тестування програмного засобу	15.11.2023 – 18.11.2023	вик
6	Економічна частина	19.11.2023 – 26.11.2023	вик
7	Оформлення матеріалів до захисту МКР	27.11.2023 – 01.12.2023	вик

Студент (підпис) **Ярмола В. С.** (прізвище та ініціали)

Керівник магістерської дипломної роботи (підпис) **Майданюк В. П.** (прізвище та ініціали)

АНОТАЦІЯ

УДК. 004.031.42

Ярмола В.С. Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту. Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. – 136.

На укр. мові. Бібліогр.: 40 назв; рис.: 29; табл. 11.

Метою магістерської кваліфікаційної роботи є оптимізація структури витрат за допомогою розробки нових методів та програмних засобів, що базуються на принципах штучного інтелекту і забезпечують геолокацію місць витрат.

У рамках дослідження розроблено новітній метод оптимізації структури витрат, який вирізняється інтеграцією штучного інтелекту, зокрема використанням сучасних сервісів машинного та глибокого навчання. Таке поєднання технологій дозволяє збільшити точність прогнозування майбутніх витрат, а також запровадити динамічне управління бюджетом з метою оптимального розподілу фінансових ресурсів. Додатково, алгоритми будуть налаштовані на виявлення неефективних витрат і пропонувати шляхи їх зменшення чи ліквідації.

Також в процесі роботи розроблено метод обліку витрат, який не лише здійснює облік витрат, але й точно фіксує місцезнаходження здійснених операцій. Визначення географічної позиції витрат дає змогу маркувати їх на інтерактивній карті, забезпечуючи більш глобальний і деталізований аналіз. Це нововведення сприяє кращій візуалізації фінансових потоків та допомагає виявити шаблони в поведінці споживачів, що в подальшому може слугувати базою для розробки індивідуальних стратегій оптимізації витрат.

Ключові слова: Android, Android Studio, Kotlin, геолокація, витрати, бюджетування, мобільний додаток, штучний інтелект, програмне забезпечення, облік витрат, аналіз даних.

ANNOTATION

UDC 004.031.42

Yarmola V.S. Methods and software tools for cost structure optimization using artificial intelligence. Master's qualification work on specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. – 136.

In Ukrainian language. Bibliographer: 40 titles; images.: 29; tables: 11.

The goal of the master's qualification work is to optimize the cost structure by developing new methods and software based on the principles of artificial intelligence and providing geolocation of the cost locations.

As part of the research, the latest method of optimizing the cost structure was developed, which is characterized by integration with artificial intelligence, in particular, modern machine learning and deep learning services. This combination of technologies makes it possible to increase the accuracy of forecasting future costs, as well as to introduce dynamic budget management with the aim of optimal distribution of financial resources. In addition, algorithms will be configured to identify inefficient costs and suggest ways to reduce or eliminate them.

Also included in the work is the cost accounting method, which not only considers cost items, but also accurately records the location of cost transactions. Determining the geographic position of costs allows you to mark them on an interactive map, providing a more global and detailed analysis. This innovation contributes to a better visualization of financial flows and helps to identify patterns in consumer behavior, which can later serve as a basis for developing individual strategies for cost optimization.

Keywords: Android, Android Studio, Kotlin, geolocation, costs, budgeting, mobile application, artificial intelligence, software, cost accounting, data analysis.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі, де багато людей борються з фінансовим балансуванням, облік витрат стає не просто критично важливим, а й недостатньо простим. Багато користувачів шукають унікальний підхід, який би легко інтегрувався в їхнє повсякденне життя, забезпечуючи практичність, підтримку та персоналізацію. Вводячи в розгляд новий мобільний додаток, ми розглядаємо інноваційний інструмент, що об'єднує геолокацію для відстеження витрат та штучний інтелект для аналізу фінансових звичок користувачів. Це передбачає, що користувачі отримують від програми не лише інформацію про те, де та як вони витрачають гроші, а й цінні поради щодо ефективного бюджетування [1].

Область управління витратами та контролю вимагає значного обсягу даних та аналізу. Це пояснює, чому вона стала сферою застосування програмного забезпечення для автоматизації. Цей процес може бути оптимізований за допомогою програмного забезпечення, що виконує складні обчислення та інші завдання, забезпечуючи їхню точність та захист від помилок. Використовуючи інформаційні панелі, що прості у використанні та мають зручний інтерфейс користувача, дані можуть бути легше аналізовані [1].

Мета обліку витрат збігається з цілями окремих осіб, груп людей або компаній: обидві мають на увазі зниження витрат і підвищення прибутковості та конкурентоспроможності.

Проте, багато популярних програм обліку витрат не звертають достатньої уваги на питання геолокації витрат, тобто визначення розташування об’єктів, де витрачались кошти. Геолокація є надзвичайно важливою для споживачів, що хочуть контролювати свої витрати, з урахуванням того, що вони часто здійснюють нерегулярні витрати в різних місцях. Крім того, розробка мобільного додатку для оптимізації контролю витрат стає важливою, оскільки це дозволяє контролювати витрати в реальному часі і стане доступним рішенням для великої кількості людей.

Таким чином, тема роботи "Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту" є вкрай актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення [2].

Мета та завдання дослідження. Метою роботи є оптимізація структури витрат за допомогою розробки нових методів та програмних засобів, що базуються на принципах штучного інтелекту і забезпечують геолокацію місць витрат.

Основними задачами дослідження є:

- аналіз відомих мобільних додатків для контролю витрат;
- розробка алгоритму роботи для програмного засобу оптимізації структури витрат;
- вибір програмних засобів для вирішення поставлених завдань;
- розробка коду програмного продукту;
- тестування роботи програмного продукту.

Об’єкт дослідження – процес оптимізації структури витрат з використанням штучного інтелекту та геолокації місць витрат.

Предмет дослідження – методи та програмні засоби оптимізації структури витрат.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія імовірності та математична статистика, дискретна математика для розробки моделей та методів обліку витрат; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Вперше запропоновано метод оптимізації структури витрат, особливість якого полягає у використанні сервісів штучного інтелекту, що дозволяє підвищити точність прогнозування та збільшити ефективність процесу оптимального розподілу ресурсів.

2. Подальшого розвитку отримав метод обліку витрат, у який на відміну від існуючих, інтегровано функції геолокації, що розширює можливості для аналізу даних щодо витрат і полегшує користувачам процес вибору способів оптимізації витрат.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено мовою програмування Kotlin мобільні програмні засоби для оптимізації структури витрат з інтеграцією сервісів штучного інтелекту та геолокації місць витрат.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованій праці, опублікованих у співавторстві, автору належать такі результати: розробка моделі та інтерфейсу мобільного додатку обліку витрат [3], інтеграція модулю геолокації в мобільний додаток [4], розробка методів оптимізації витрат за рахунок використання сервісів штучного інтелекту [5].

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних та всеукраїнських конференціях: Науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.); Міжнародна науково-практична конференція «Інформаційні технології і автоматизація» (Одеса, 2023); Міжнародна науково-практична конференція «Електронні інформаційні ресурси: створення, використання, доступ» (Вінниця, 2023).

Публікації. Основні результати досліджень опубліковано в 3 наукових працях у матеріалах конференцій.

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Аналіз предметної області і сучасного стану питання

Контроль витрат стає важливим елементом фінансового впорядкування для більшості людей. Це не лише стосується власників бізнесу або людей на високооплачуваних посадах, але й пересічних людей, які бажають вести раціональне та ефективне управління своїми фінансами. Витрати стають складнішими для контролю, коли з'являється багато різних рахунків, що вимагають оплати, і коли витрати становлять значну частину доходів. Це може бути викликом для людей, особливо для тих, хто веде активний спосіб життя, працює на повну ставку або веде бізнес [1].

З цих причин програми контролю фінансів стали популярними серед користувачів інтернету. Через ефективне відстеження грошових потоків та балансування бюджету, користувачі можуть легко контролювати свої витрати і доходи. Усе, що потрібно, це використати програмне забезпечення для контролю фінансів, котре надає можливість користувачу помістити свої доходи і витрати в контексті місячного, квартального або річного бюджету та багато інших можливостей.

З розвитком технологій, смартфони стали не лише розважальними пристроями, але й потужними інструментами для управління різними аспектами буденного життя, включаючи особисті фінанси. Вони дозволяють користувачам вести облік, аналізувати, прогнозувати і покращувати свій фінансовий стан, використовуючи мобільні додатки [1].

Цілком природно, що використання цих додатків повинно включати оптимізацію витрат, оскільки це один із найважливіших аспектів управління особистими фінансами. Оптимізація витрат може включати в себе такі речі, як спрощення процесу планування бюджету, покращення відслідковування витрат, зменшення необґрунтованих витрат та покращення загальної фінансової стабільності.

З іншої сторони, використання штучного інтелекту в таких системах може внести значний внесок у посилення їх ефективності. Штучний інтелект може виступати в ролі потужного інструменту аналізу, адаптації та прогнозування, що дає можливість користувачам отримувати персоналізовані рекомендації, виходячи з їхніх конкретних витрат і фінансових цілей.

Вивчення існуючих програм для контролю витрат дало змогу виявити їхні переваги та недоліки. Це надало напрямок для адекватного планування та розробки власного додатка для контролю витрат. Оскільки створення прототипу і тестування є важливими етапами цього процесу, були виявлені найважливіші функції, які мають бути виконані на цих стадіях для подальшої реалізації в кінцевому продукті.

Ці методи та програмні засоби є націлені на створення простого, зручного та ефективного середовища для управління витратами користувачів. Вони дозволяють користувачам краще розуміти, куди витрачаються їх гроші, а також, як можна знизити витрати і покращити свій загальний фінансовий стан.

1.2 Порівняльний аналіз аналогів

Аналіз конкуруючих продуктів або послуг на ринку є важливим кроком у процесі проектування та розробки нового продукту. Порівняльна аналітика допомагає зрозуміти, які тренди і стандарти вже є на ринку, виявити сильні та слабкі сторони конкурентів, а також визначити власні конкурентні переваги.

У контексті розробки мобільного додатку для оптимізації витрат, аналіз аналогічних додатків дозволяє виявити, які функції та сервіси вже пропонуються користувачам, які з них є найбільш популярними і ефективними. Окрім того, можна виявити проблеми або недоліки, які користувачі можуть мати із схожими додатками, та намагатись їх вирішити у власному продукті [6].

Це допомагає приймати обґрунтовані рішення про дизайн, юзабіліті (зручність використання), набір характеристик та інші аспекти розробки додатку. Наприклад, якщо аналогічні додатки мають складні або неінтуїтивні

взаємодії, це може бути шансом для покращення та розробки юзабіліті власного додатку.

Таким чином, порівняння аналогів допомагає не лише розуміти, що вже працює і що ні, але і знаходити нові можливості для інновацій та забезпечувати більшу вартість для користувачів, ніж існуючі аналоги.

Аналогом мобільного додатку, що розробляється в процесі магістерської роботи є мобільний додаток «Гаманець – облік витрат і доходів, бюджет» (рисунок 1.1).

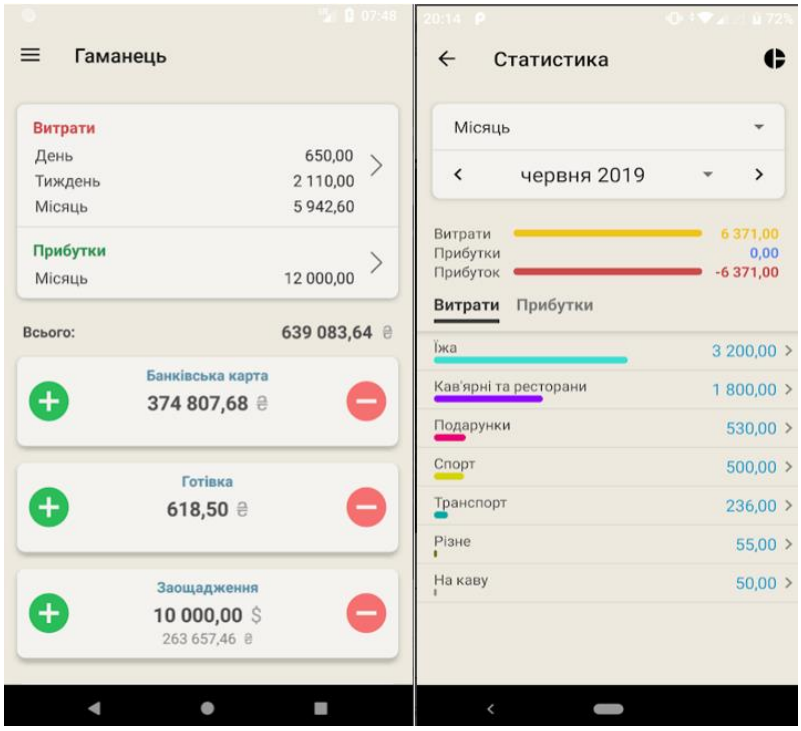

Рисунок 1.1 – Додаток «Гаманець – облік витрат і доходів, бюджет»

Цей мобільний додаток є ефективним інструментом для швидкого та зручного обліку власних доходів та витрат. Незалежно від того, чи отримує користувач дохід, чи здійснює покупку, додаток миттєво дозволяє записати цю інформацію, що спрощує процес управління фінансами [7].

Одним із ключових переваг цього додатка є його здатність візуалізувати ваші витрати. За допомогою зрозумілої діаграми він демонструє, на які категорії витрачаються ваші гроші. Це не тільки демонструє, куди "зникають" гроші, але й сприяє плануванню та накопиченню коштів.

Додаток також оснащений функцією статистики, що дозволяє користувачу відстежувати свої доходи та витрати за визначений час: день, тиждень, місяць, рік або інший період. Це робить управління фінансами максимально простим та зручним, незалежно від вашого бюджету чи фінансових цілей.

"1Money – облік витрат, бюджет" є ще одним прикладом мобільного додатку, що допомагає користувачам стежити за своїми фінансами. На рисунку 1.2 зображено приклад інтерфейсу додатку. Даний додаток допомагає користувачам відслідковувати, куди витрачаються їхні кошти. Фінансовий контроль – виклик, який вимагає певних навичок та умінь, але "1Money" може полегшувати цей процес. Його функціональність дає змогу легко та швидко вносити інформацію про витрати, що призводить до ефективного та комфортного контролю над особистими фінансами прямо на ходу [8].

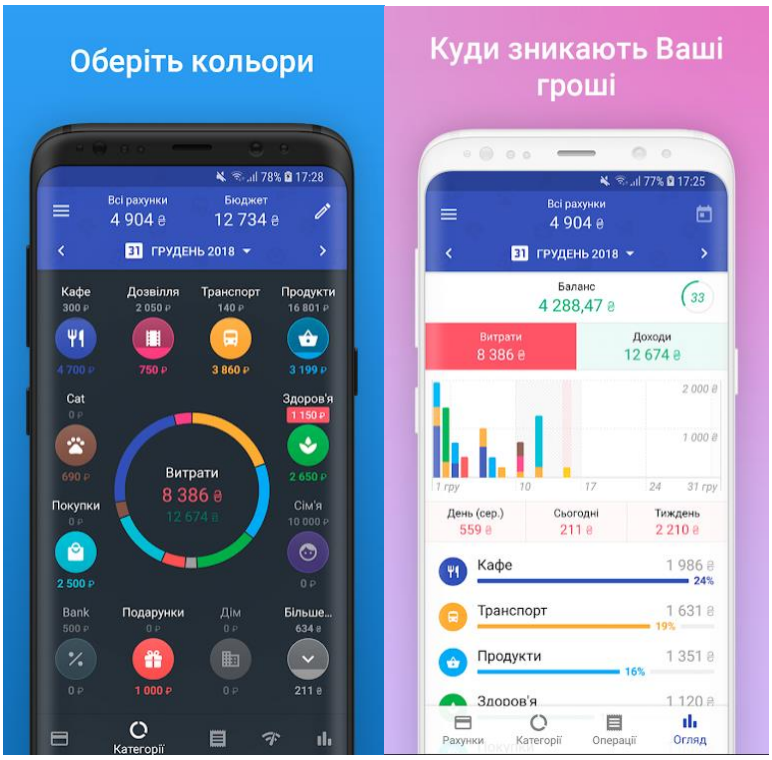

Рисунок 1.2 – Додаток «1Money – облік витрат, бюджет»

Виходячи з аналізу існуючих додатків для управління витратами, більшість з них доступні для загального користування, пропонують

привабливий та функціональний дизайн. Однак кожна така програма має свої недоліки, а відмітка витрат на карті – це функція, якої зазвичай бракує.

Тому розробка мобільного додатку, який б не лише простежував за витратами, але й дозволяв користувачам відзначати їх геолокацію, є актуальною. Розроблення такого додатку заповнює цю прогалину на ринку та забезпечує додаткову цінність для користувачів, допомагаючи їм більш повно розуміти контекст своїх витрат.

Завданням роботи є створення додатку, який буде не лише функціональним, але й легким у використанні. Він буде містити зручний та інтуїтивний інтерфейс, що дозволить користувачам легко додавати, переглядати, редагувати та видаляти свої доходи та витрати в будь-який час. Крім того, користувачі матимуть можливість керувати категоріями витрат, що подальше оптимізує процес планування та контролю бюджету.

1.3 Вибір моделі життєвого циклу для розробки додатку

В ході розробки будь-якого програмного додатку, вибір певної моделі життєвого циклу відіграє критичну роль. Це стосується не лише процесу розробки, а й експлуатації та супроводу продукту, включаючи всі етапи – від визначення вимог до припинення використання додатку.

Моделі розробки програмного забезпечення - це сукупність конкретних інструментів, підходів і методологій, що застосовуються протягом усього життєвого циклу проекту. Вони спрямовані на максимізацію продуктивності при розробці, в той же час, мінімізуючи операційні витрати. Це означає, що обраний дизайн програмного забезпечення може мати значний вплив на успішність проекту, а в деяких випадках - його провал [9].

Щоб знизити ризик несподіваних проблем у рамках проекту, важливо ретельно проаналізувати складові життєвого циклу розробки програмного забезпечення (SDLC) та врахувати, як вони можуть вплинути на вибір моделей розробки програмного забезпечення.

Чітке дотримання процесу SDLC є однією з ключових складових

успішної реалізації проекту з розробки програмного забезпечення. Такий підхід допомагає не лише запобігти затримкам і перевищенням бюджету, але й гарантує, що кінцевий продукт відповідає всім заздалегідь встановленим вимогам.

Ітераційна модель. Ітераційний підхід до розробки програмного забезпечення є одним з ключових у методологіях SDLC. Ця модель пропонує розірвати процес розробки на менші, легко вимірювані "ітерації" або фази. Кожна ітерація представляє собою замкнутий цикл розробки, що містить власне проектування, розробку, тестування та внесення змін [10]. На рисунку 1.3 зображено принцип ітераційної моделі.

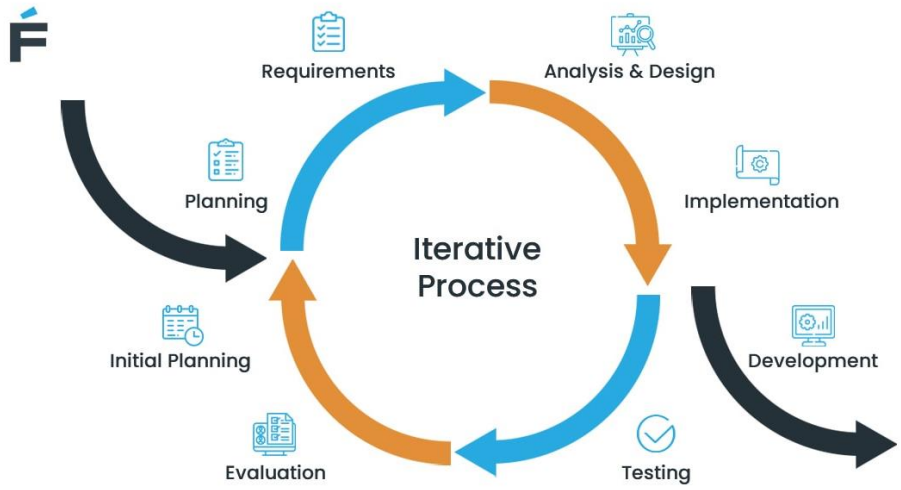

Рисунок 1.3 – Принцип ітераційної моделі

Переваги використання моделі:

1. Гнучкість: Зміни можна легко внести після кожної ітерації, що робить процес розробки адаптивним до вимог замовника.

2. Постійний зворотній зв'язок: На кожній стадії отримується звратний зв'язок від клієнта або конечного користувача.

3. Рання ідентифікація проблем: Помилки або проблеми можна виявляти та виправляти під час кожної ітерації, а не після завершення всього проекту.

4. Ітеративна модель забезпечує швидше внесення змін та поліпшень.

Недоліки:

1. Потреба в детальному плануванні кожної ітерації.

2. Витрати ресурсів: Часті зворотній зв'язок і внесення змін можуть призвести до високих витрат на ресурси.

3. Помилка в одній ітерації може вплинути на наступні ітерації.

Ітераційна модель ідеально підходить для проектів, де вимоги клієнтів можуть змінюватися або коли проект є великим та складним.

AGILE модель. Agile (гнучка) модель – це тип ітеративного та інкрементного підходу до розробки програмного забезпечення. Вона є колективним викликом для ряду методологій розробки, таких як Scrum, Kanban, Lean, Extreme Programming (XP) та інші. Agile-модель включає часті інспекції та адаптацію розробки [11].

В процесі розробки Agile розглядається відповідність змінюваним вимогам замовників та гнучкість в системі. Використовуючи цю модель, команда може продовжувати роботу, приймаючи зміни у вимогах і удосконалюючи продукт крок за кроком, ітерацію за ітерацією. На рисунку 1.4 схематично зображено Agile модель.

Переваги Agile моделі:

1. Гнучкість: Agile сприймає зміни як частину процесу розробки. Якщо вимоги змінюються, команда розробників готова вносити відповідні зміни.

2. Задоволення користувача: Постійний видача версій та швидка відповідь на відгуки клієнтів допомагає підвищити рівень задоволення користувача.

3. Висока продуктивність: Висока взаємодія серед членів команди та регулярні зустрічі щодо процесу розробки допомагають підвищити продуктивність.

Недоліки Agile моделі:

1. Недостатня документація: Через фокус на розробку продукту, Agile може залишити слабким усними докментацією.

2. Важкість оцінки: Відсутність детального плану від початку проекту може зробити важким складеним достовірних оцінок.

3. Потреба в залученості конечних користувачів: Agile вимагає активного участі конечних користувачів протягом усього циклу розробки.

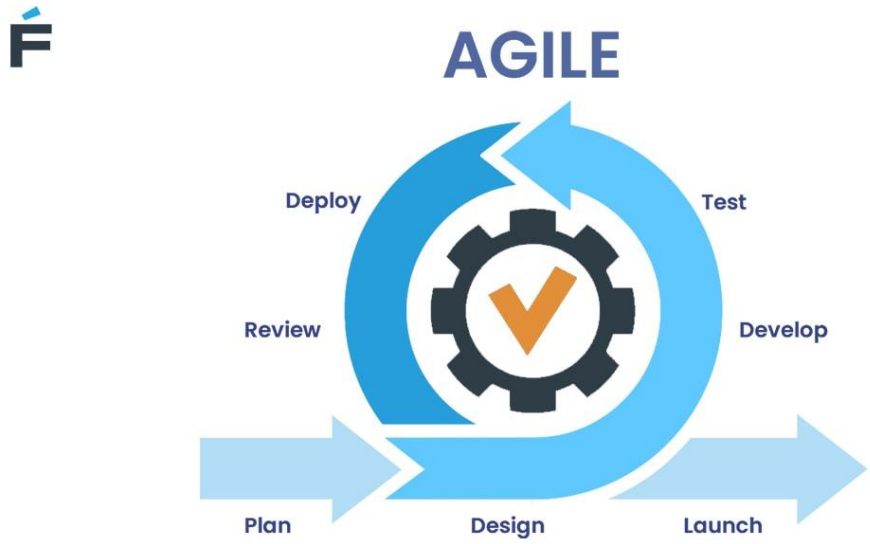

Рисунок 1.4 – Agile модель

Agile достатньо гнучкий, щоб його можна було використовувати в різних проектах, але він виявляється особливо корисним в проектах, де вимоги можуть змінюватися протягом часу, і коли цінується швидка розробка та доставка продукту.

Водоспадна модель. Модель водоспаду або водоспадна модель є однією з найбільш традиційних та широко використовуваних моделей в програмній інженерії. За свою назву вона зобов'язана графічному зображенню процесу розробки, що нагадує водоспад, з водою, яка струмить зверху вниз, подібно до того, як проекти переходять від одного етапу до наступного в цій моделі.

Водоспадна модель передбачає строго послідовний процес розробки, який починається з ідеї та закінчується реалізацією проекту [12]. Такий процес може бути поділений на п'ять основних етапів:

1. Вимоги: на цьому етапі відбувається вивчення проблеми й визначення вимог до програмного продукту.

2. Проектування: після визначення вимог визначається архітектура системи і планується її розробка.

3. Реалізація: це етап кодування, коли пишуть програмний код.

4. Тестування: тут відбувається перевірка продукту на наявність помилок і відповідність вимогам.

5. Супроводу й експлуатація: включає в себе установку програми, вирішення будь-яких виниклих проблем і оновлення.

Водоспадна модель відзначається своїм строгим і послідовним підходом, де кожна наступна стадія починається лише після завершення попередньої. На рисунку 1.5 зображено принцип водоспадної моделі.

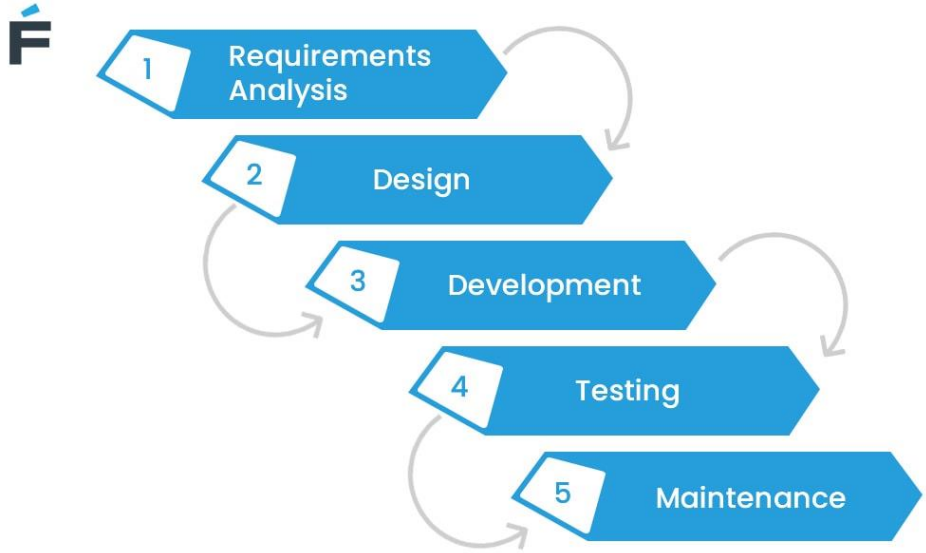

Рисунок 1.5 – Принцип водоспадної моделі

Використання водоспадної моделі для розробки мобільних додатків має такі переваги:

1. Чітка структура і послідовність: водоспадна модель дуже структурована і вимагає всебічного планування з самого початку, що дозволяє розробникам і клієнтам бути в курсі всіх етапів розробки.

2. Детальна документація: розробка мобільних додатків на водоспадній моделі має ретельно документований процес, який допомагає в майбутньому при внесені змін або модифікацій.

3. Контроль якості: на кожній стадії проекту передбачені чіткі метрики тестування та контроль якості, що гарантує створення якісного продукту.

Незважаючи на численні переваги моделі водоспаду для розробки мобільних додатків, є деякі недоліки, які потрібно враховувати.

Серед найбільш очевидних недоліків водоспадної моделі є нееластичність щодо змін. Зміни в вимогах або планах проекту можуть бути дорогою та складною справою, бо це може вимагати повернення до попередніх етапів.

У водоспадній моделі важко недооцінити час та ресурси, що необхідні для завершення кожного етапу. Це може призвести до затримок у графіку та збільшення вартості проекту.

Однак, незважаючи на ці недоліки, водоспадна модель відіграє важливу роль у розробці мобільних додатків. Ця модель ідеально підходить для проектів з відомими та стабільними вимогами, а також коли важливо дотримуватися строгого плану та бюджету. Водоспадна модель також відома своєю стабільністю, передбачуваністю і простотою управління.

1.4 Постановка задачі розробки

Основним завданням цієї кваліфікаційної роботи є розробка мобільного додатку, що включає геолокацію для обліку витрат. В цей процес входить розробка методів та програмних засобів для оптимізації структури витрат. Серед основних задач, що стоять перед командою розробників, можна виділити наступні:

1. Аналіз існуючих інструментів для управління та розрахунку витрат, з дослідженням їх переваг та недоліків.

2. Визначення ключових модулів та понять, які будуть впроваджені в додаток для покращення його функціональності та корисності для користувача.

3. Розвиток структури бази даних, яка буде ефективною, стабільною та безпечною для зберігання і обробки інформації про витрати користувача.

4. Аналіз доступних хмарних середовищ для розміщення бази даних з метою забезпечення надійності, доступності та масштабованості розробленого рішення.

5. Розробка серверної частини додатку, що включає проектування архітектури, реалізацію необхідних API інтерфейсів, розробку алгоритмів обробки та аналізу даних.

1.5 Висновки

1. Здійснено дослідження методів та програмних засобів, що використовують штучний інтелект для оптимізації структури витрат. Використання машинного навчання та інтелектуального аналізу даних дозволяє глибше розуміти зв'язки і взаємозв'язки між різними типами витрат, що надає можливість знайти шляхи оптимізації витрат, які виходять за межі традиційних методів.

2. Здійснено детальне порівняння наявних на ринку аналогів, та виявлено, що більшість з них має обмежений функціонал і не задовольняє потреб користувачів у повному контексті. У відповідь на це, прийнято рішення розробити додаток, який не лише втілює всі елементи, які мають додатки-аналоги, але й включає в себе використання штучного інтелекту та функцію геолокації.

3. Обрано модель життєвого циклу для розробки програмного забезпечення. Перевагу отримала водоспадна модель, оскільки вона ідеально підходить для проектів невеликого масштабу. Ця модель забезпечує чітку структуру розробки, і кожен етап слідує за попереднім в строго визначеному порядку, що робить процес управління проектом значно простішим та ефективнішим.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ РЕАЛІЗАЦІЇ ПРОГРАМНИХ ЗАСОБІВ

2.1 Розробка загальної архітектури додатку

Архітектура програмного додатку представляє собою керівну структурну карту, продумано організовану для збору різних програмних модулів у сукупність, що в результаті виходить згуртованим цілим [13]. При цьому, окремі програмні компоненти неперервно взаємодіють одна з одною, щоб ефективно задовольняти комерційні вимоги бізнесу або спеціальні потреби користувачів. Постійна аналітична робота по розвитку архітектури додатку сприяє забезпеченню масштабованості та стабілізації надійності програм, що в свою чергу допомагає адаптуватися підприємству до сучасних ринкових тенденцій, успішно виявляючи недоліки у функціональності та швидко поправляючи їх. Мета архітектури програми зосереджена на визначенні ефективного способу взаємодії програм з ключовими елементами системи, включаючи проміжне програмне забезпечення, бази даних та пов'язані програми. Напрямок розробки архітектури додатків підпорядкований принципам розробки програмного забезпечення, які є широко визнаними серед фахівців своєї галузі, проте характеризуються відсутністю офіційно впроваджених галузевих стандартів.

Фактором успіху стає ефективна архітектура додатків, яка робить можливою продуктивну співпрацю IT-спеціалістів та організаторів бізнес-планування, створюючи високоякісне технічне середовище для досягнення бізнес-цілей. Переваги використання чітко визначеної та сформованої архітектури програми:

– Основний вклад виразний у зменшенні фінансових витрат через виявлення можливих повторення та подвійності, особливо у випадках радикального використання двох незалежних баз даних, які можна спокійно замінити на одну економну модель;

– Суттєве підвищення ефективності, сформоване на підставі виявлення та усунення недоліків, зокрема, негативних ситуацій, коли користувачі не мають

можливості отримати доступ до основних послуг через мобільні додатки; Впровадження корпоративної платформи для швидкого доступу до додатків та ефективної сторонньої інтеграції;

– Методична підтримка, яка допомагає архітектору "побачити детальну картину" та узгодити стратегію розробки програмного забезпечення з фундаментальними бізнес-цілями організації.

Чітко структурована діаграма варіантів використання. Цей тип діаграм UML виокремлюється, як один з найбільш визнаних у категорії діаграм поведінкових типів. Їх головна цінність виявляється у графічному представленні акторів, які задіяні в системі, їх конкретних функціональних обов'язках, необхідних для цих акторів, а також в візуалізації взаємодії між цими різноманітними функціями. Така укомплектована складністю діаграма може стати чудовим стартовим пунктом для різноманітних обговорень проєкту, адже створює сприятливі умови для оперативного визначення головних учасників та оцінки важливості основних процесів системи [14]. На рисунку 2.1 зображено діаграму використання для додатку з геолокацією обліку витрат.

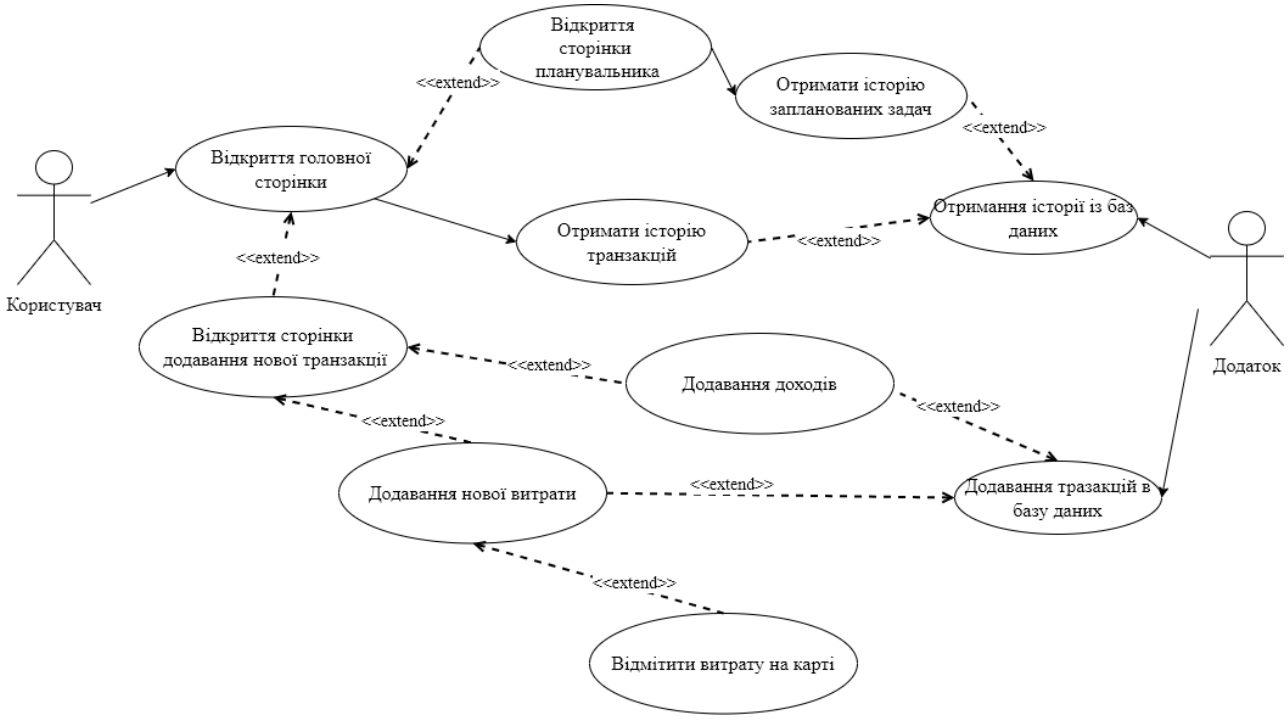

Рисунок 2.1 – Діаграма варіантів використання

Діаграма послідовності – це важливий інструмент в області об'єктно-орієнтованого проектування, передбачений для представлення динаміки і взаємодії між об'єктами в системі. В основному, це діаграми взаємодії, які відображають, як об'єкти та компоненти системи співпрацюють для досягнення певного результату. Вони використовуються розробниками програмного забезпечення та спеціалістами з бізнес-аналізу для збагачення розуміння вимог до нової системи та документації щодо існуючих процесів [15]. На рисунку 2.2 зображено діаграму послідовності додатку.

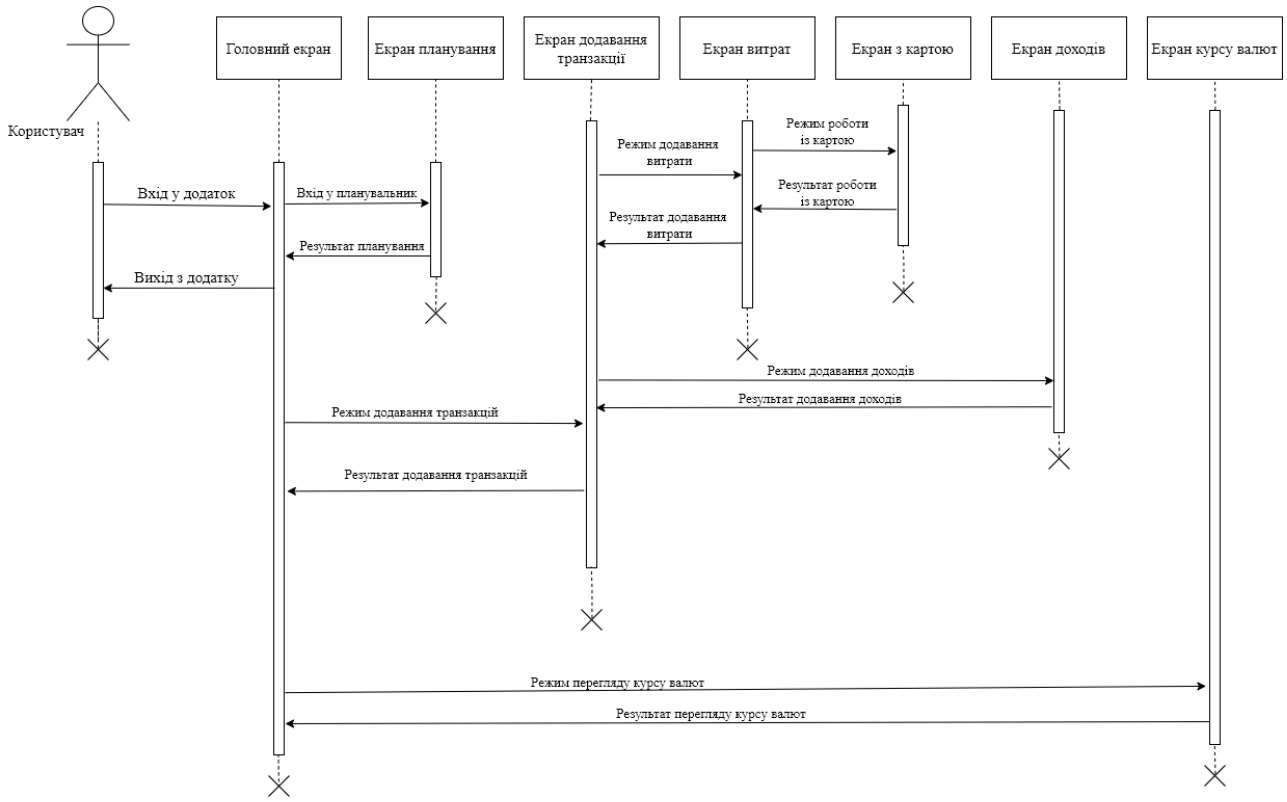

Рисунок 2.2 – Діаграма послідовності мобільного додатку

Переваги використання діаграм послідовності:

1. Представлення детальної структури UML сценарію використання.

2. Детальне планування і зрозуміння функціональних аспектів поточних або майбутніх процесів.

3. Візуалізація взаємодії об'єктів та компонентів для досягнення завершення процесу.

4. Моделювання логіки складних процедур, функцій або операцій для точного відтворення процесу.

Варто вирізнити два основних типи діаграм послідовності: UML діаграми та код-орієнтовані діаграми. UML діаграми використовуються для візуальної репрезентації взаємодій об'єктів, тоді як код-орієнтовані діаграми зорієнтовані на представлення специфіки взаємодії на більш технічному, програмістському рівні.

Діаграма послідовності розглядає наступні об'єкти: "Користувач", "Головний екран", "Екран планування", "Екран додавання транзакції", "Екран затрат", "Екран з картою", "Екран доходів" та "Екран курсу валют". Ця діаграма відображає можливі сценарії користувача додатком.

З головного екрану, користувач може переглянути деталі своїх витрат та доходів, перейти до секції планування, додати нові транзакції, а також відкрити екран з актуальним курсом валют.

Ці діаграми послідовностей слугують як основний путівник по архітектурі програмного забезпечення для оптимізації структури витрат. Вони дають чітке уявлення про те, які конкретні функціональні можливості та характеристики необхідно реалізувати в процесі розробки додатку.

2.2 Розробка методу оптимізації витрат за допомогою штучного інтелекту

Завдяки швидкому прогресу в технологіях, штучний інтелект (ШІ) стає все більш революційним інструментом в багатьох сферах діяльності. Один з досить недооцінених, але із потенціалом, що перетворює галузь, напрямків є використання ШІ для оптимізації витрат в побутових та комерційних обставинах [16].

Оскільки технології продовжують розвиватися, ШІ став потужним інструментом для оптимізації управління запасами. Впровадження штучного інтелекту в оптимізацію запасів має багато переваг, які можуть значно знизити операційні витрати та ризики для бізнесу.

Одним із важливих способів допомоги ШІ є максимізація інвестицій у запаси. На відміну від традиційного ведення записів, штучний інтелект може аналізувати величезні набори даних швидше, ніж людина, визначаючи оптимальний рівень запасів між надлишком і браком. Наприклад, інструмент прогнозування використовують попередні дані про продажі, щоб передбачити майбутній попит, враховуючи будь-які майбутні акції чи сезонні розпродажі. Такі системи, керовані штучним інтелектом, гарантують, що підприємства мають лише товари, які користуються попитом, і прибуткові товари, мінімізуючи ризик залишкових запасів або втрачених можливостей продажу.

ШІ також усуває потребу в повторюваній ручній роботі, звільняючи важливий для бізнесу час. Повторювані завдання, такі як кодування замовлення на покупку, перфорація продажів і перевірка залишків на складі, можна автоматизувати, запобігаючи помилкам і затримкам роботи та потенційно економлячи до 20 годин на тиждень. На рисунку 2.3 зображено схематично етапи аналізу витрат та попиту за рахунок використання сервісів штучного інтелекту.

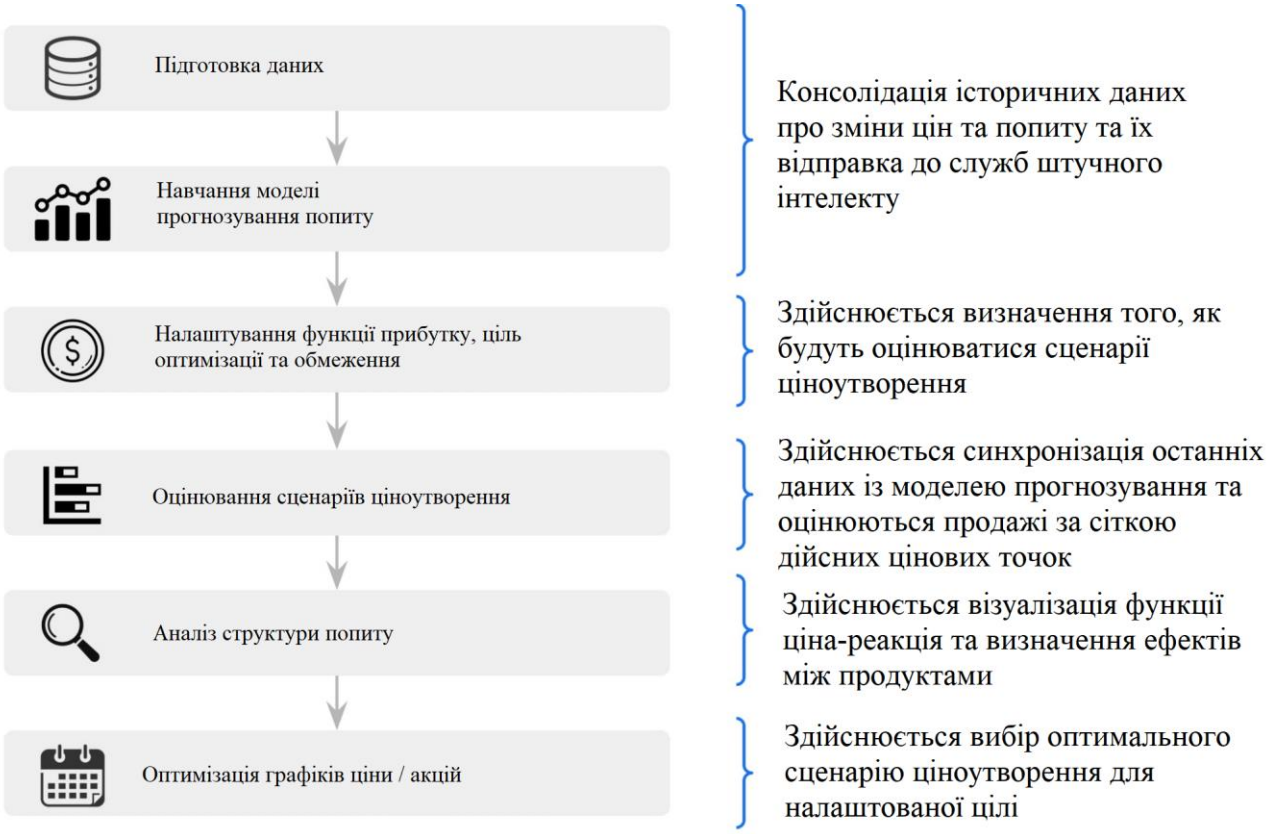

Рисунок 2.3 – Етапи аналізу даних сервісами штучного інтелекту

Ще одна суттєва перевага штучного інтелекту – це його здатність мінімізувати людські помилки. Дрібні помилки можуть серйозно вплинути на прибутковість; однак добре навчений ШІ не робить цих помилок. Завдяки автоматизації процесів і звітів ймовірність людських неточностей значно зменшується [16].

ШІ також сприяє зменшенню витрат на оплату праці, що є особливою перевагою для підприємств, які розвиваються та хочуть уникнути витрат, пов'язаних із наймом додаткового персоналу. ШІ може виконувати багато трудомістких завдань і підвищувати ефективність робочої сили, зберігаючи низькі витрати на робочу силу.

Підвищення ефективності складу – це ще одна перевага штучного інтелекту. ШІ може надати цінну інформацію про запаси та потік товарів, допомагаючи максимально збільшити простір і планування складів. Він також може оптимізувати роботу, забезпечуючи зменшення затримок і пропусків завдань.

Оптимізуючи поповнення запасів і надаючи точні прогнози, ШІ також може оптимізувати грошові потоки. Підприємства можуть інвестувати в продуктивні запаси та зменшити витрати на технічне обслуговування, в результаті чого більше грошей буде доступно для інших сфер бізнесу.

Впровадження штучного інтелекту також може сприяти покращенню виконання замовлень. ШІ може сповіщати компанії в режимі реального часу про низькі запаси, затримки доставки та інші проблеми, зменшуючи ймовірність проблем, спричинених помилками людини. Зменшення кількості повернених продуктів і скарг клієнтів означає зниження витрат, що забезпечує додаткові переваги для підприємств [17].

Таким чином, використання штучного інтелекту для оптимізації запасів пропонує підприємствам безліч переваг. Це максимізує інвестиції в запаси, усуває схильну до помилок ручну роботу, зменшує витрати на оплату праці, максимізує ефективність складу та оптимізує грошовий потік. Крім того, це покращує виконання замовлень, зрештою підвищуючи задоволеність клієнтів і

зменшуючи витрати, пов’язані з поверненнями та скаргами. З усіма цими перевагами штучний інтелект виявився важливим партнером для компаній, які прагнуть покращити свої прибутки.

Нинішні алгоритми ШІ можуть здійснювати глибокі аналітичні розрахунки, які перевершують традиційні методи. Вони мають потужність вивчити, зрозуміти і передбачити витрати на основі існуючих даних, навчаючись виявляти проблемні області, виявляти тенденції і робити обґрунтовані, точні прогнози витрат.

Інноваційність полягає в тому, що ШІ не просто замінює традиційні методи оптимізації витрат, але робить цей процес значно більш ефективним, точним і менш трудомісткім. Він використовує потужність машинного навчання для ідентифікації і виправлення проблемних точок, скорочуючи таким чином витрати та збільшуючи прибуток [17].

Застосування цих методів в мобільному додатку є черговим кроком у цій IT-революції. Мобільний додаток із ШІ може стати унікальним інструментом для користувачів, які бажають аналізувати свої витрати на конкретні категорії чи досягати більшої економії.

Він може пропонувати персоналізовані рекомендації та вказівки на основі індивідуальних витратних звичок користувача. Наприклад, він може виявити, що користувач віддає перевагу придбанню преміальної кави кожного ранку і пропонувати альтернативи для економії. Він може навчитися впізнавати реальний час та сезонність витрат та надавати вам сповіщення, коли виникають несподівані витрати.

В противагу традиційним методам, ШІ може навчатися з обмеженої кількості даних, неперервно збагачуючи своє розуміння витратних моделей та виявляючи можливості для економії в масштабах великих даних.

Запровадження передових функцій в мобільний додаток не тільки покращить його корисність, але й збереже значну суму грошей для користувачів. Використання ШІ як способу оптимізації витрат стає

інструментом майбутнього, об'єднує технології та економіку і допомагає нам жити більш розумно та ефективно.

2.3 Розробка методу інтеграції функції геолокації

Вибір ідеального API геолокації для розробки мобільних додатків має вирішальне значення. Використання геолокації, яка точно визначає координати пристрою, може зробити програму більш інтерактивною та надійною. Інтеграція подібних модулів широко використовується в різних типах програм, таких як соціальні мережі, ігри, електронна комерція, фітнес, комунальні програми та програми для подорожей.

Варіанти використання додатків на основі визначення місця розташування здаються нескінченними. У часи економіки на вимогу та індустрії зростає кількість подібних до Uber, додатків для обслуговування на вимогу, моніторингу стану здоров’я, рішень для керування автопарком і навіть додатків для знайомств, оскільки компанії прагнуть полегшити наше життя все більш творчими способами. Ринок послуг, орієнтованих на визначення місця розташування, продовжує стабільно зростати – очікується, що до 2026 року він становитиме 48 мільярдів доларів США, що зросте за загальним річним темпом зростання на 19,4% у 2021-2026 роках [18].

Недоліком є те, що розробка програми геолокації не проста. Для мобільних додатків, які визначають місцезнаходження, потрібні функції, які, серед іншого, дозволять відстежувати місцезнаходження користувача в режимі реального часу.

Google Maps Geolocation API. Пошук правильної служби геолокації може значно вплинути на продуктивність і інноваціність мобільної програми. Коли справа доходить до надійних служб геолокації, Google Maps стає дуже ефективним інструментом [19].

Розроблений світовим технологічним гігантом Google, цей API базується на надійності та ефективності. Це платформа, яка надає високоточну інформацію про місцезнаходження та пропонує інтуїтивно зрозумілий

інтерфейс, що робить її улюбленою серед початківців і досвідчених розробників.

Завдяки повній базі даних для відстеження місцезнаходження Google Maps Geolocation API є тим самим інструментом, який Google використовує для всіх своїх продуктів, які потребують функцій визначення місцезнаходження. Він надійний, точний і оновлюється в режимі реального часу, надаючи актуальні дані про місцезнаходження, які не мають собі рівних на поточному ринку. На рисунку 2.4 зображено інтерфейс карти, котрий надає Google API.

Рисунок 2.4 – Інтерфейс Google Maps Geolocation API

Плюси використання API геолокації Карт Google:

– Точність. Надзвичайною перевагою використання API геолокації Карт Google – це точність. Це гарантує, що програма може надавати користувачам точні дані про геолокацію, забезпечуючи плавну та ефективну роботу користувача.

– Зручність. Іншою сильною стороною є зручний характер API. Незважаючи на надійність і широкі можливості, він має інтуїтивно зрозумілий інтерфейс. Це робить його доступнішим і легшим для навігації, особливо для розробників, які тільки починають.

– Надійність. Ще один атрибут, у якому є перевага API Google. Він пропонує ефективні оновлення в режимі реального часу, що особливо корисно для додатків, пов’язаних із транспортом, або тих, яким потрібно відстежувати дії та шаблони користувачів.

Незважаючи на свої переваги, Google Maps Geolocation API також має кілька недоліків:

– Вартість використання. Хоча він надає безкоштовний рівень, витрати можуть швидко зростати із збільшенням запитів про місцезнаходження, що може бути недоліком для програм із широкою базою користувачів.

– Конфіденційність. Оскільки API відстежує велику кількість даних про місцезнаходження, це може викликати занепокоєння щодо конфіденційності користувачів. Однак Google гарантує анонімність усіх даних і не може ідентифікувати окремого користувача, що може дещо пом’якшити цю проблему.

У сфері API геолокації Google Maps Geolocation API пропонує поєднання точності, зручності та надійності. Вибір його як основного API для розробки мобільних додатків може відкрити двері для безлічі можливостей, розширюючи та покращуючи взаємодію з додатком. Хоча витрати та питання конфіденційності викликають справедливі занепокоєння, плюси переважують мінуси для більшості вимог до програм, що робить Google Maps Geolocation API лідером у світі послуг геолокації.

IP Geolocation API. IP-геолокація (або Інтернет-геолокація) – це процес визначення фізичного розташування пристрою за допомогою його адреси Інтернет-протоколу (IP).

IP-адреса – це унікальний ідентифікатор, призначений кожному пристрою, підключеному до Інтернету. Коли настільний комп’ютер або мобільний пристрій робить HTTP-запит і викликає API, його IP-адреса надається як частина запиту. Отже, API може бачити його, навіть якщо він не доданий явно [20].

IP Geolocation API працює за допомогою IP-адреси для визначення приблизного географічного розташування пристрою. API використовує комбінацію баз даних, алгоритмів і технологій геолокації для пошуку інформації про місцезнаходження, пов’язане з цією IP-адресою. Ця інформація зазвичай включає країну, регіон, місто та навіть координати широти та довготи пристрою. На рисунку 2.5 зображено приклад інтерфейсу IP Geolocation API.

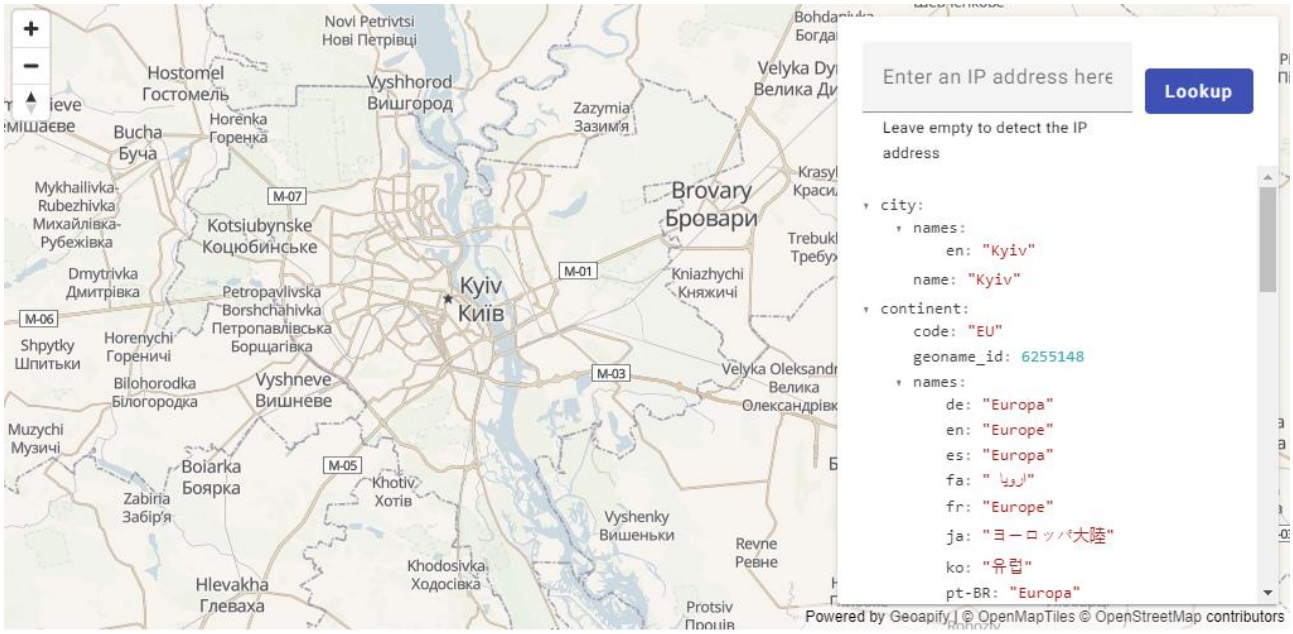

Рисунок 2.5 – Інтерфейс IP Geolocation API

Технологія IP-геолокації часто використовується в різних програмах для налаштування роботи користувачів на основі їхнього географічного розташування, наприклад для відображення локального вмісту, пропозиції персоналізованих угод або виявлення та запобігання шахрайству. API допомагає ідентифікувати географічну інформацію, підключену до пристроїв з підтримкою Інтернету, на основі їхніх IP-адрес. Географічні дані, які повертає цей API, включають країну, регіон, місто, довготу та широту, часовий пояс і навіть поштовий індекс. Цей API спрощує геолокацію, черпаючи дані безпосередньо з IP-адреси користувача без необхідності використання додаткових джерел даних. Він надає повну інформацію про демографічні дані

про місцезнаходження користувачів, що робить його ідеальним інструментом для прийняття стратегічних рішень у процесі розробки програми.

Плюси використання API геолокації IP. Однією з найвидатніших переваг використання API геолокації IP є його простота. Навіть для нових розробників інтегрувати цей API у свої програми відносно просто. Для отримання даних про місцезнаходження потрібна лише IP-адреса користувача, що робить його безпроблемним API геолокації.

Крім того, IP Geolocation API є доступним вибором. Більшість його ключових функцій доступні безкоштовно, пропонуючи розробникам корисний ресурс геолокації без навантаження на бюджет. Така економічна ефективність робить його ідеальним вибором для стартапів, невеликих програм або проектів з обмеженим бюджетом.

Мінуси використання API геолокації IP. Хоча простота API геолокації IP є перевагою, вона також може бути перешкодою. Його залежність виключно від IP-адрес для даних про місцезнаходження може призвести до отримання неточної інформації про місцезнаходження. Отримання точних результатів геолокації за допомогою цього API суттєво залежить від географічного розподілу IP-адрес, який постійно змінюється і не повністю залежить від вас.

Крім того, цей API може мати обмеження в регіонах, де використовуються динамічні IP-адреси – постійна зміна IP-адрес може призвести до того, що API надаватиме неточні або застарілі дані про місцезнаходження. Це може вплинути на результати вашої мобільної програми, особливо якщо ваша програма значною мірою покладається на точну геолокаційну інформацію в реальному часі.

Хоча IP Geolocation API може не мати такої високої точності, як Google Maps Geolocation API, його простота й економічність привабливі, особливо для розробників із обмеженим бюджетом і для програм, які не дуже покладаються на відстеження місцезнаходження в реальному часі. У програмах, де достатньо помірної точності, IP Geolocation API служить ефективним і економічно ефективним рішенням.

HERE Geolocation API. HERE Geolocation – це комплексний сервіс геолокації, який пропонує широкий спектр даних про місцезнаходження. Цей API дозволяє розробникам отримувати доступ до повної географічної інформації, включаючи широту, довготу, назви місць, регіони та пов’язані дані.

HERE Geolocation API отримує дані з безлічі джерел, включаючи з’єднання Wi-Fi, вежі стільникового зв’язку та IP-адреси, надає дані, які рекламуються як надзвичайно точні, конкуруючи з іншими варіантами, доступними на ринку [21]. На рисунку 2.6 зображено приклад інтерфейсу HERE Geolocation API.

Рисунок 2.6 – Інтерфейс HERE Geolocation API

Переваги використання HERE Geolocation API:

– База даних. Однією з вражаючих особливостей HERE Geolocation API є величезна кількість детальних географічних даних, які він надає. Використовуючи таку інформацію, можна персоналізувати та покращити роботу користувача, надаючи програмі потужну перевагу.

– Підтримка стартапів. Крім того, API надає безкоштовний рівень, який підтримує до 250 000 транзакцій на місяць, що виявляється корисним для стартапів і розробників, які тестують воду. Це робить HERE Geolocation API економічно ефективною альтернативою іншим API, які не пропонують безкоштовне використання, і допомагає краще керувати бюджетами проекту.

Недоліки використання HERE Geolocation API:

– Обмежена документація. Незважаючи на численні переваги, API геолокації HERE також має деякі проблеми. Варто зауважити, що він може бути не таким популярним або широко використовуваним, як API геолокації Карт Google. Це може обмежити доступність ресурсів для розробників, посібників з усунення несправностей і прикладів програмування.

– Складність підтримки. Крім того, кілька розробників зазначили, що API може бути дещо складнішим для включення порівняно з іншими. Розробникам можуть знадобитися додаткові години для розуміння та ефективної інтеграції, що може стати перешкодою для тих, хто працює над проектом у стислі терміни.

HERE Geolocation API пропонує доступ до широкого спектру точних географічних даних. Це надійний вибір для програм, де потрібна висока географічна деталізація. Хоча є занепокоєння щодо складнощів у його реалізації та обмежених ресурсів для розробників, його переваги, як-от безкоштовний рівень використання та повна доступність даних, роблять його гідним суперником у сфері послуг геолокації.

Після поглибленого аналізу різних API геолокації стає зрозуміло, що кожен має свої сильні та слабкі сторони. Однак у контексті розробки програми для мобільних пристроїв Android API геолокації Google Maps стає найбільш надійним і надійним рішенням.

Його висока точність позиціонування, легкість інтеграції, багатий доступ до даних і значні ресурси розробника роблять його видатним виконавцем. API Google є, безумовно, найпопулярнішим і широко використовуваним

геолокаційним сервісом, який пропонує багату підтримку спільноти, навчальні посібники та приклади програмування.

Це не означає, що інші API, які ми обговорювали – IP Geolocation API та HERE Geolocation API – не є цінними, але коли йдеться про загальну функціональність і корисність, Google Maps Geolocation API виявляється лідером. Він задовольняє комплексні потреби, типові для життєвого циклу надійної програми для Android, від початкової розробки до підтримки залучення користувачів з часом.

З огляду на це, для прийняття обґрунтованого рішення вкрай важливо враховувати конкретні потреби вашої програми, зокрема бюджетні обмеження та важливість точного відстеження місцезнаходження для основних функцій вашої програми. Але загалом завдяки широкому спектру функцій і перевіреній надійності Google Maps Geolocation API безперечно є найкращим напрямком дій для розробки програм для Android.

2.4 Програмування бази даних

Обробка і зберігання великих обсягів структурованих даних є важливим аспектом у багатьох сучасних програмах. Зокрема, це стає важливим, коли програма повинна мати можливість працювати в автономному режимі, а не тільки підключатися до мережі для перегляду чи обробки даних.

Бібліотека збереження Room в мобільній розробці на Android дозволяє розробникам безпосередньо взаємодіяти з базою даних SQLite. Бібліотека забезпечує додатковий рівень абстракції над SQLite, таким чином дозволяє працювати з базою даних за допомогою об'єктно-орієнтованого підходу. Вона включає в себе декілька ключових компонентів, які спрощують роботу з базою даних [22].

Компоненти Room:

– Database: Ця анотація використовується з абстрактним класом, який розширює RoomDatabase. Вона включає в себе всі таблиці і версію бази даних.

– Entity: Ця анотація використовується з класами, які представляють таблицю в базі даних.

– DAO (Data Access Object): Це інтерфейс, в якому ви визначаєте всі методи для доступу до бази даних. Тут ви встановлюєте SQL-запити.

Переваги Room:

– Room усуває багато рутинного робочого навантаження, пов'язаного з роботою над SQLite, забезпечуючи зручний об’єктно-реляційний відображувач (ORM) і допомагаючи управляти і запитувати бази даних.

– Бібліотека створює корректні SQL запити за вас і допомагає уникнути проблем із SQL-ін'єкціями. Room також підтримує виконання запитів на диспетчері потоків або корутині Kotlin, що допомагає уникнути блокування інтерфейсу користувача.

– Room допомагає дотримуватися принципу розділення відповідальності, що допомагає зробити код вашого додатку більш чистим, простим для розвитку і тестування.

Room не є єдиною бібліотекою для розробки бази даних SQLite на Android, але вона є частиною Android Jetpack і впроваджує та підтримує багато сучасних кращих практик розробки на платформі Android.

Firebase Realtime Database – це хмарна NoSQL база даних, надана платформою Firebase від Google. Вона дозволяє зберігати та синхронізувати дані між користувачами в реальному часі, що робить її відмінним рішенням для розробки додатків з реальним відтворенням даних та спільним доступом [23].

Ключові аспекти Firebase Realtime Database:

– Синхронізація в режимі реального часу: Коли дані змінюються, зміни відразу відображаються на всіх підключених клієнтських пристроях. Це працює незалежно від платформи, чи це Android, iOS, або веб-клієнт.

– Автономна робота: Firebase Realtime Database зберігає локальний кеш на клієнтському пристрої, що дозволяє застосунку продовжувати працювати навіть без підключення до Інтернету. Коли з'єднання відновлюється, всі зміни синхронізуються.

– Безпека: База даних може бути налаштована за допомогою Firebase Security Rules. Ці правила дозволяють вам контролювати, хто має доступ до бази даних, які операції вони можуть виконувати, і навіть валідацію формату даних.

– Масштабування: Firebase Realtime Database розподіляє дані по кількох серверах і локалізаціях, щоб забезпечити високу продуктивність та надійність.

– Інтеграція з іншими сервісами Firebase: Firebase пропонує різноманітні сервіси, які легко інтегрувати з Firebase Realtime Database, включаючи автентифікацію, хмарні повідомлення, аналітику та інше [23].

Все це робить Firebase Realtime Database ідеальним рішенням для створення складних, взаємодіючих в реальному часі приложений на різних платформах. Вона надає потужний, гнучкий набір інструментів для розробника, які працюють відразу з коробки. На рисунку 2.7 зображено схематично Firebase Database.

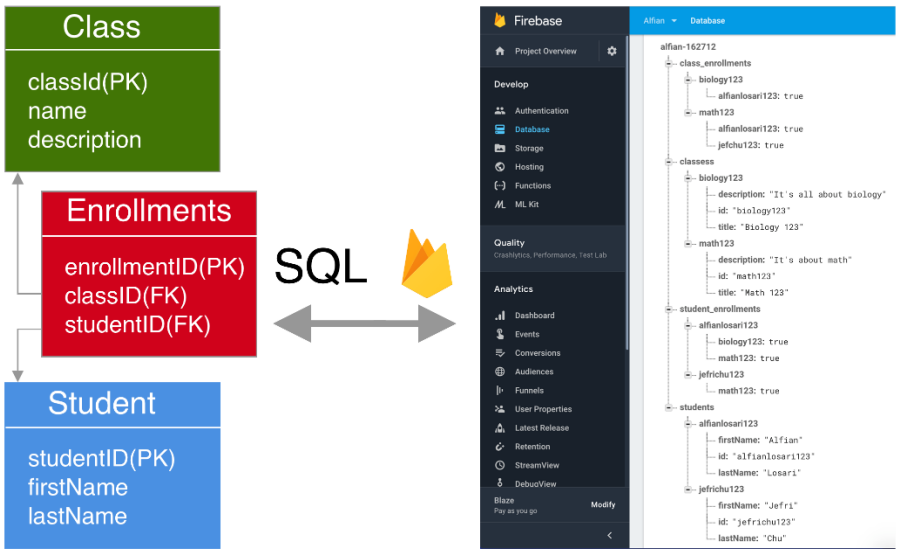

Рисунок 2.7 – Firebase Database

Після всебічного аналізу та вивчення можливостей, вирішено використовувати Room DB для локального зберігання даних у мобільному додатку для оптимізації структури витрат. Room DB вирішує проблему

збереження даних на пристрої користувача, надаючи простий, але потужний інтерфейс для роботи з SQLite базами даних.

Також, з метою забезпечення хмарного зберігання та синхронізації даних між різними пристроями, вирішено використовувати Firebase Database від Google. Firebase Database – це надійне, гнучке та ефективне рішення для реалізації сучасних мобільних додатків, з можливістю роботи в реальному часі та підтримкою автономності.

Обрана комбінація технологій дозволяє забезпечити високу продуктивність, надійність та зручність використання мобільного додатку, а також розширювати його функціональність в майбутньому.

2.5 Розробка блок-схеми алгоритму роботи програми

Розробка деталізованого алгоритму роботи програмного додатку створює необхідну фундаментальну основу для всієї структури процесу розробки. Всебічно вивчений алгоритм вказує на послідовність виконання робочих процедур, пов'язаних з визначеною проблемою, і направляє розробників до кінцевого досягнення мети. Він розкладає великі і складні завдання на керовані, зрозумілі елементи, створюючи сприятливе середовище для ефективного та доступного кодування [24].

Блок-схеми і алгоритми мають важливе значення в процесі візуалізації конкретизованих кроків, потрібних для розробки програми. Це математичні та графічні представлення процесу, які великою мірою полегшують процес програмування.

Алгоритм визначає послідовність і методи виконання завдань, покроково демонструючи процес, який потрібен для досягнення конкретної мети. Він детально описує вимоги до функціонування програми, забезпечуючи поетапну діаграму роботи з відповідними умовами та обробкою помилок.

Блок-схема є графічним інструментом, який демонструє ці алгоритми в більш візуально зручній формі. Він ілюструє процеси в графічному вигляді, пропонуючи візуальне зображення послідовності дій та взаємодії між окремими

компонентами програми. Нижче наведено декілька ключових переваг використання блок-схем у розробці програмного забезпечення:

– Поліпшення зрозумілості: Блок-схеми забезпечують чітке візуальне представлення процесу або системи. Вони подають складну інформацію в легкодоступній та зрозумілій формі, допомагаючи командам краще розуміти, як працює система, і що має відбутись на кожному етапі процесу.

– Ефективне планування: З блок-схемами легше визначити алгоритми, послідовності та умовні оператори, які потрібні для розробки програмного забезпечення. Це дає можливість правильно спланувати розробку та впевнено виконувати кожен наступний крок.

– Зниження помилок: Блок-схеми дозволяють розробникам виявляти та вирішувати проблеми на початкових стадіях розробки, що зменшує кількість помилок і скорочує час на дебагінг в майбутньому.

– Облегшення комунікації та співпраці: За допомогою блок-схем можна легко пояснити алгоритми та процеси колегам, менеджерам проектів і стейкхолдерам. Вони допомагають уникнути непорозумінь та забезпечують команду загальним баченням цілей та завдань.

– Документація: Блок-схеми можуть служити у ролі документації, надаючи загальний огляд системи для майбутніх довідок або модернізації системи в майбутньому.

Блок-схема дозволяє легко ідентифікувати логічну послідовність завдань, відображаючи стратегію і шаги напрацювань у графічній формі. Вона не тільки дає змогу зрозуміти загальний потік роботи програми, але й слугує важливим інструментом для виявлення слабких пунктів у дизайні програми та ідентифікації можливих областей для вдосконалення [24]. На рисунку 2.8 зображена загальна блок-схема алгоритму роботи мобільного додатку з геолокацією обліку витрат.

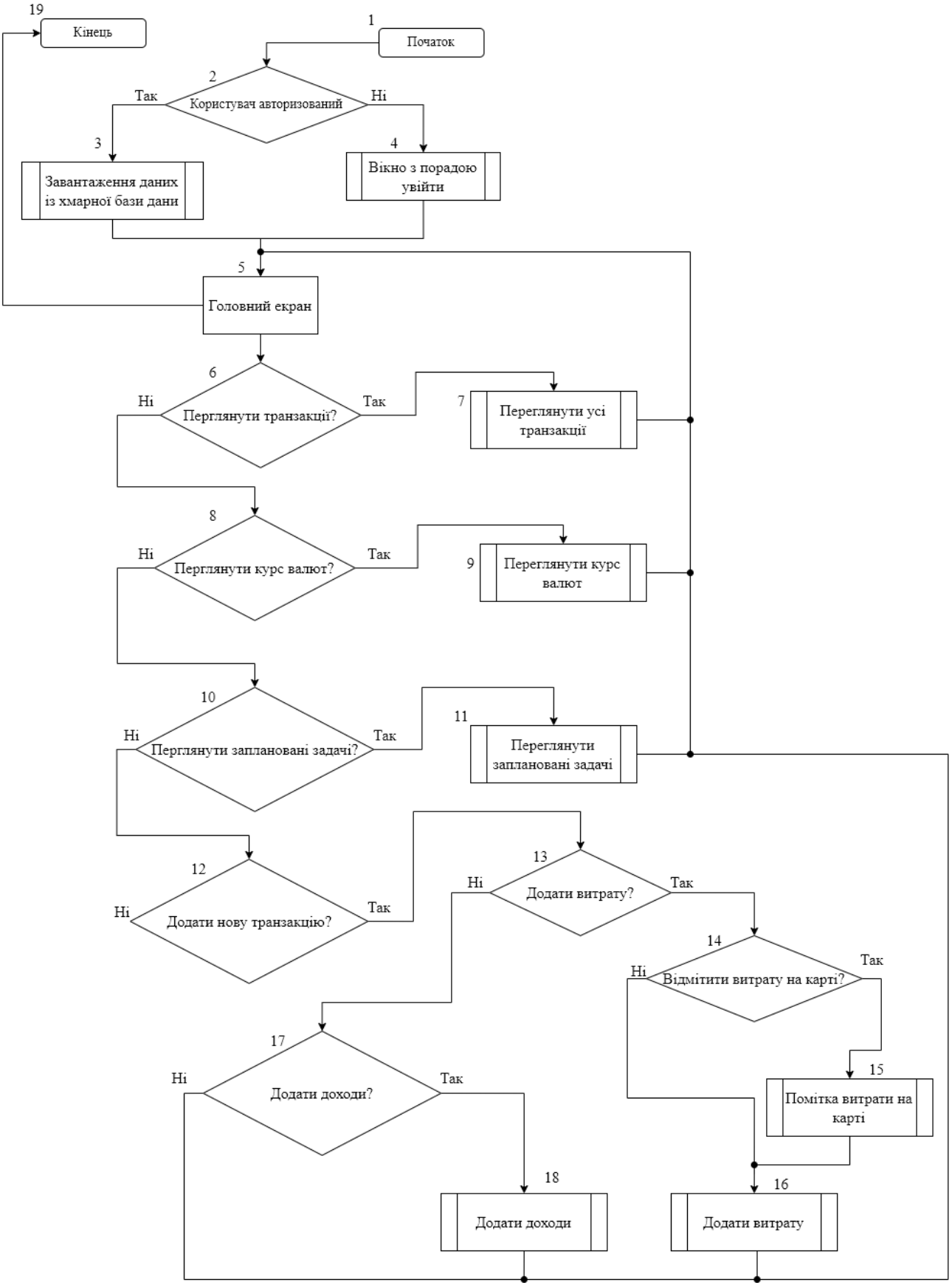

Рисунок 2.8 – Загальна блок-схема алгоритму додатку

На наведеній діаграмі блок-схеми представлені основні етапи функціонування мобільного додатку:

1. Ініціалізація додатку, зокрема завантаження всіх необхідних компонентів та даних з кеш-пам'яті.

2. Виконання перевірки статусу користувача на його авторизацію в додатку.

3. В разі підтвердження авторизації користувача проводиться оновлення даних із бази даних.

4. Якщо користувач не авторизований, йому пропонується увійти або створити обліковий запис у додатку для збереження даних.

5. Після авторизації або реєстрації користувач переходить до головного екрану програми, де він може переглянути свій баланс, доходи, витрати та останні транзакції.

6. Користувачу пропонується вибрати варіант перегляду всіх транзакцій на окремій сторінці.

7. В разі окремого вибору опції перегляду транзакцій відкривається відповідний список у хронологічному порядку.

8. Наступний крок - вибір дії перегляду курсу валют.

9. Відкривається інтерфейс зі списком доступних валют, де користувач може переглянути поточний курс.

10. Користувач має можливість переглянути заплановані задачі.

11. Якщо є бажання переглянути чи додати нову заплановану задачу, користувач може вказати дату, час та підтвердити свої дії. Потім отримує сповіщення з нагадуванням у встановленій добу.

12. Існує можливість створити нову транзакцію.

13. Пропонується варіант обрати вид транзакції: дохід або витрати.

14. В разі вибору витрат, користувачу пропонується вказати місце витрат на карті.

15. Відкривається карта, де користувач може вказати своє місцезнаходження або обрати іншу локацію.

16. Після заповнення форми витрати, користувач зберігає внесену інформацію, після чого може повернутись на головну сторінку програми.

17. Якщо було обрано категорію доходів, користувачу пропонується внести дані у відповідні поля.

18. Після заповнення форми доходів, дія зберігається, а потім здійснюється навігація до головного екрану.

19. На головній сторінці користувач може продовжити роботу з додатком або закінчити її, якщо усі потрібні дії виконані.

2.6 Висновки

1. Розроблено архітектуру програмного забезпечення для оптимізації структури витрат. Ключовою особливістю є інтеграція як локальної, так і хмарної баз даних. Ця архітектура дозволяє користувачеві оновлювати дані про витрати незалежно від доступності інтернет-мережі.

2. Виконано розробку методів оптимізації фінансових витрат за допомогою технологій штучного інтелекту. Це передбачає використання алгоритмів машинного навчання для збору, аналізу та використання даних про витрати користувача з метою розробки індивідуальних рекомендацій щодо оптимізації витрат.

3. Здійснено розробку та інтеграцію функції геолокації в усі аспекти функціонування мобільної програми. Така розробка включена в графічну схему додатку. Це дозволяє обробляти та використовувати геолокаційні дані для глибшого аналізу витрат та пропонувати користувачеві подальший план оптимізації витрат.

4. Розроблено всебічну блок-схему додатку, яка деталізовано відображає функціонал програми і методи навігації. Це в значній мірі спрощує процес розробки коду для мобільного додатку, що пропонує функції відслідковування витрат завдяки геолокації.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ ДЛЯ КОНТРОЛЮ ВИТРАТ

3.1 Аналіз та обґрунтування вибору програмних засобів для розробки

Перед тим, як приступити до створення програмного забезпечення, важливо провести аналіз доступних інструментів та технологій для його розробки. Головним елементом такого процесу є знання мови програмування. Однак обрати найоптимальнішу мову програмування для конкретного проекту є чималим викликом.

Вибір конкретної мови програмування в оптимальному варіанті визначається від майбутнього призначення цієї розробки. Усіх можливостей вибраної мови програмування не потрібно для деяких типів. У той же час, для складних програм може бути необхідність у застосуванні декількох мов програмування [25].

Мова програмування Java. Завдяки тому, що Java була першою офіційною мовою програмування для Android, вона є однією з найчастіше використовуваних мов. Більшість додатків у Play Market розроблено за допомогою Java. Ця мова має сильну підтримку з боку Google та велику онлайн-спільноту, яка допомагає розробникам у вирішенні різноманітних проблем. Проте Java може бути дещо складною для новачків, завдяки таким категоріям як конструктори, винятки нульових покажчиків, впровадження паралельності, перевірені винятки та інше. Додатково, Android Software Development Kit (SDK) може вносити більше складностей. Вибираючи Java, розробники отримують усі переваги розробки на Android, але новачкам можливо варто спочатку обрати щось простіше, а потім уже повернутися до Java [26].

Мова програмування Kotlin. З 2019 року Google прийняла Kotlin офіційною мовою програмування для Android. Kotlin – це універсальна мова програмування, яка може використовуватися як варіант заміни Java для розробки Android-додатків. У 2017 році Kotlin був представлений як додаткова

"офіційна" мова на Java. Kotlin взаємодіє з Java та працює на Java Virtual Machine. Відмінність полягає в тому, що Kotlin видаляє деякі некорисні аспекти Java, такі як винятки нульових покажчиків, та виключає необхідність закінчувати кожне речення крапкою з комою. У порівнянні з Java, Kotlin є більш дружнім для початківців та може служити як ідеальний стартовий пункт для початку розробки Android-додатків [27].

Мова програмування C++. C++ можна втілити для розробки Android-додатків за допомогою Android Native Development Kit (NDK). Проте, повноцінний додаток зробити виключно на C++ не вийде, оскільки NDK передбачено для створення окремих частин коду додатку на її рідній мові, якою є C++. У такий спосіб, NDK допомагає інтегрувати бібліотеки коду C++ при необхідності. Хоч C++ і може бути корисним у певних аспектах розробки Android-додатків, його встановлення може бути складним, а гнучкість обмеженою. Більш того, може виникнути більше помилок, через складність мови програмування. Тому, на балансі між Java та C++, надається перевага Java, адже вона набагато простіша, і її ефективність не відстає від C++, щоб це виправдовувало використання останньої [28].

Мова програмування C#. C# має багато схожостей з Java, що робить його відмінним вибором для розробки Android-додатків. Подібно до Java, C# використовує автоматичне управління пам'яттю (Garbage Collection), що зменшує ймовірність проблем з витоками пам'яті. Крім того, синтаксис C# є більш чистим і спрощеним, ніж у Java, що сприяє більш легкому процесу кодування.

Раніше основним обмеженням використання C# було його обмеження на Windows-системи через залежність від .NET Framework. Однак, із виникненням Xamarin, це вже не проблема. Xamarin, раніше відомий як Mono для Android, представляє собою кросплатформену реалізацію Common Language Infrastructure (CLI) і Common Language Specifications (часто називають "Common Language Runtime (CLR)" або ".NET runtime") [29].

Зараз, за допомогою Xamarin.Android, можна створювати нативні Android-додатки та використовувати однаковий кодову базу для різних платформ. Це забезпечує значну гнучкість для розробників, надаючи їм можливість ефективно створювати і розширювати мобільні додатки на різних платформах.

Мова програмування Python. Python можна задіяти для розробки Android-додатків, хоча Android офіційно не підтримує Python. Це можливо завдяки декільком інструментам, які здатні конвертувати Python програми в Android-пакети, які підтримуються Android-пристроями.

В якості прикладу можна прийняти Kivy – це вільна відкрита бібліотека Python, що використовується для розробки мобільних додатків. Kivy підтримує Android, і зумовлений запровадити експедитивну розробку додатків [30].

Однак, варто відзначити, що використання Kivy може мати деякі обмеження. Конкретно, Kivy не скористається вбудованими перевагами Android-екосистеми, оскільки він не розробляється та не підтримується Google. Це може обмежити деякі функціональні можливості, які були б доступні для додатків, розроблених виключно з використанням офіційно підтримуваних мов програмування Android.

Мова програмування Dart. Dart – це високопродуктивна мова програмування з відкритим вихідним кодом, створена Google, що застосовується в рамках фреймворку Flutter. Dart поступово набирає популярність завдяки своїй спроможності ефективно та оперативно створювати якісні програми для мобільних пристроїв, настільних комп'ютерів та веб-сайтів.

Однією з вирішальних переваг Dart є те, що він був розроблений із метою оптимізації швидкості роботи програм на всіх платформах. Dart прагне полегшити процес створення інтерфейсів користувача для розробників, використовуючи такі ключові функції, як Hot Reload, який дозволяє розробникам відстежувати зміни в реальному часі без перезавантаження програми [31].

Dart визнаний за свою високу продуктивність, оскільки він компілює машинний код як для мобільних пристроїв з процесорами ARM і x64, так і для настільних комп'ютерів і серверів. Він також компілює код у JavaScript для веб-додатків, що забезпечує гнучкість і швидкість роботи на всіх платформах.

Величезна кількість програм, включаючи музичні плеєри, калькуляториб месенджери, ігри та інші, очевидно, може бути створена за допомогою мов програмування, розглянутих вище. Проте, не існує "ідеальної" мови програмування для розробки додатків Android, оскільки кожна з них має свої специфічні особливості і переваги. Тому вибір мови програмування відбувається з урахуванням конкретних цілей і специфікацій проекту.

В ході аналізу різних мов програмування для розробки програмного забезпечення для оптимізації структури витрат, вибір було зроблено на користь Kotlin. Ця мова програмування є офіційно призначеною Google для розробки додатків Android.

Однією з ключових переваг Kotlin є наявність великого обсягу детальної документації та навчальних матеріалів онлайн, що стає незамінним активом, особливо, коли над проектом працює невелика команда розробників. Саме завдяки комбінації цих переваг Kotlin є відмінним вибором для розробки витончених мобільних додатків.

3.2 Вибір середовища розробки

Розробка програмного забезпечення включає в себе різні види діяльності, зокрема: редагування вихідного коду, створення виконуваних файлів (компіляція), налагодження та виправлення помилок, тестування, інтеграція та інше.

Інтегровані середовища розробки (IDE) надають розробникам набір інструментів і утиліт, що дозволяють виконувати всі ці етапи роботи в одному програмному додатку. IDE може інтегрувати різне програмне забезпечення, включаючи текстовий редактор для написання коду, компілятори або інтерпретатори для перетворення вихідного коду на виконуваний код,

налагоджувачі для виявлення та виправлення помилок, а також інструменти для автоматизації побудови аплікації, її тестування та інтеграції.

У результаті, для розробки мобільних додатків було створено ряд інтегрованих середовищ розробки (IDE), кожне з яких має свої особливості та переваги. Вони постійно розвиваються і оновлюються, на ринку постійно з'являються нові продукти, що забезпечують нові можливості розробки, забезпечують зручність і продуктивність розробки і легко інтегруються із іншими інструментами та платформами розробки.

Середовище розробки DroidScript. DroidScript – це повноцінне, високофункціональне мобільне інтегроване середовище розробки (IDE), що спеціально призначене для платформи Android. Однією з ключових переваг DroidScript є те, що він працює офлайн, таким чином, немає необхідності в постійному підключенні до Інтернету. Це особливо важливо для розробників, які постійно переїжджають або працюють у місцях з нестабільним інтернет-з'єднанням [32]. На рисунку 3.1 зображено середовище розробки DroidScript.

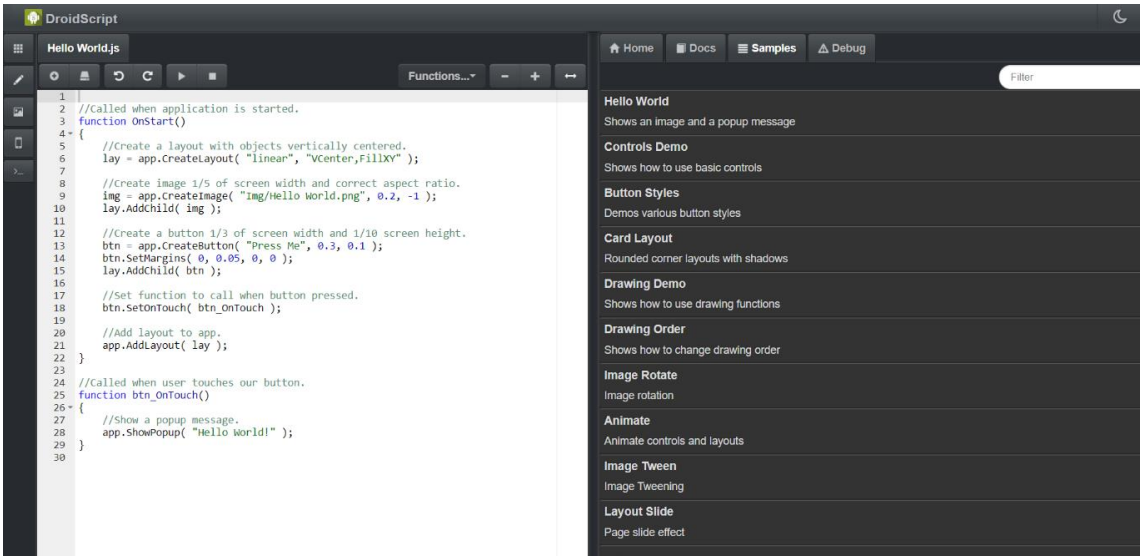

Рисунок 3.1 – Середовище розробки DroidScript

DroidScript приємно використовувати, незалежно від рівня досвіду програміста. Середовище добре підходить як для новачків, так і для досвідчених програмістів. Новачкам буде вельми корисною надана обширна

кількість навчальних матеріалів і туторіалів, що робить процес навчання максимально зручним і ефективним.

Середовище розробки CppDroid. CppDroid є повноцінною, але простою в користуванні IDE для Android-платформи, спеціально призначеною для програмування на мовах C і C++. Спроектована з урахуванням потреб як новачків, так і досвідчених програмістів, CppDroid надає достатньо ресурсів для навчання продуктивного кодування: вбудовані підручники, приклади коду та програми для початківців [33].

CppDroid має налаштовуваний текстовий редактор з підсвічуванням синтаксису та автоматичними підказками для подальшого полегшення процесу програмування. Його можливості включають діагностику помилок та пропозиції щодо виправлення на льоту, тобто в режимі реального часу. На рисунку 3.2 зображено середовище розробки CppDroid.

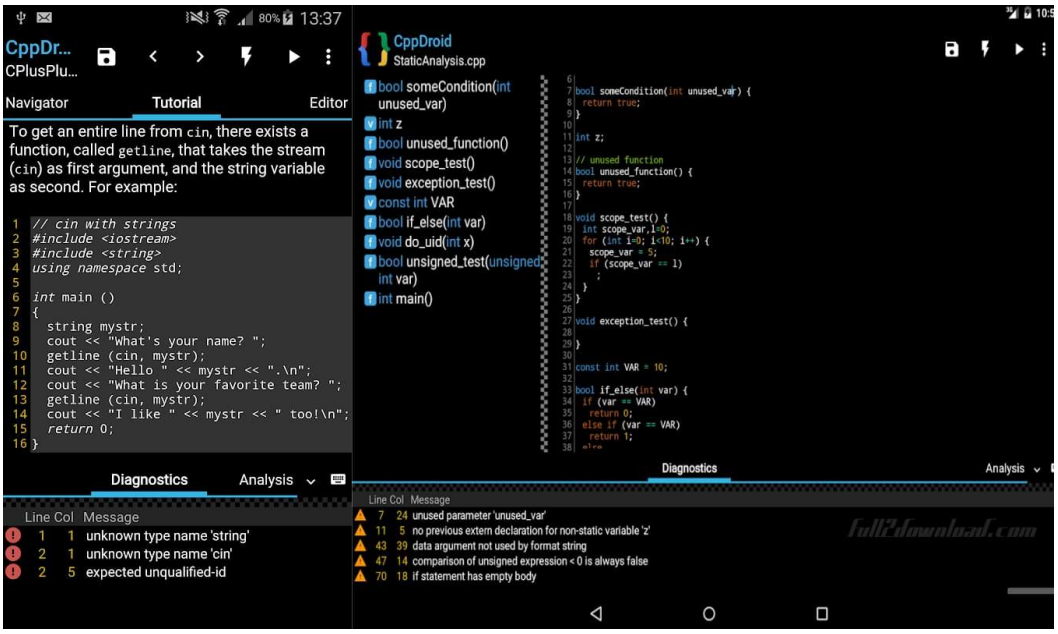

Рисунок 3.2 – Середовище розробки CppDroid

А що надзвичайно важливо, CppDroid працює в автономному режимі завдяки вбудованому компілятору, тобто він може компілювати код, навіть коли пристрій не підключений до Інтернету. Преміум-версія CppDroid надає додаткові переваги, такі як синхронізація з Dropbox та Google Drive, доступ до

великої кількості прикладів та навчальних посібників, а також статичний аналіз коду для забезпечення вищої якості програмного забезпечення.

Середовище розробки Visual Studio. Visual Studio – це визначна та багатофункціональна універсальна середа розробки (IDE), створена під егідою корпорації Microsoft. Вона здобула широку відомість як надійне середовище для розробки проектів на .NET та Windows, однак, це не обмежує її можливостей, і Visual Studio також відмінно підходить для розробки додатків для Android. Інтерфейс середовища розробки представлено на рисунку 3.3.

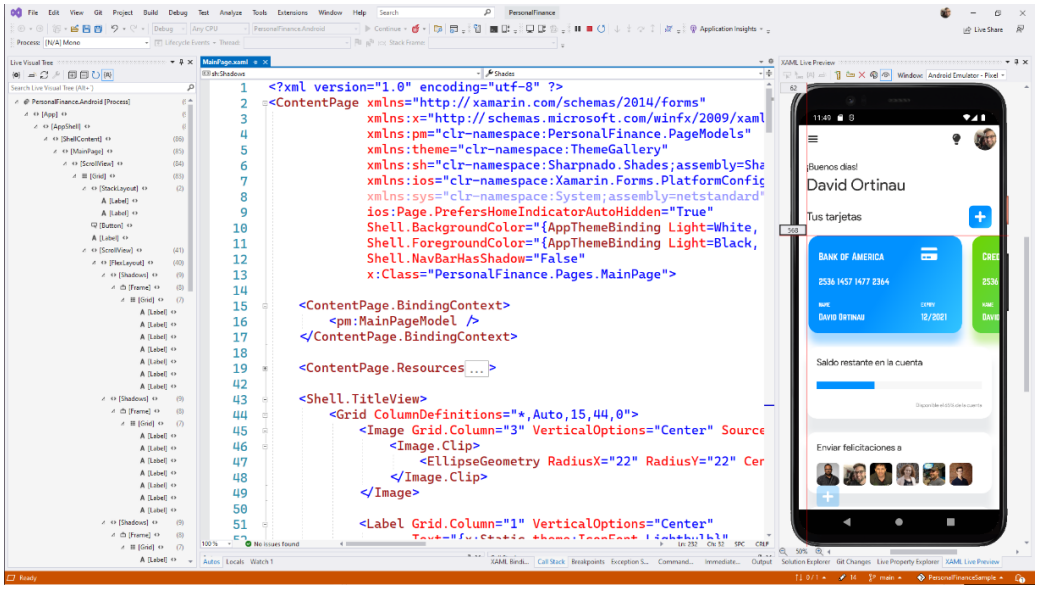

Рисунок 3.3 – Середовище розробки Visual Studio

Однією з унікальних особливостей Visual Studio, яка робить її переферованим вибором для розробки Android-додатків, є її сумісність з Xamarin. Xamarin – це міцний фреймворк, що надає можливість програмістам створювати пристосовані під різні платформи мобільні додатки на мові C#, використовуючи стек технологій .NET, і застосовувати їх у різних середовищах, таких як iOS, Android і Windows [34].

Visual Studio включає гнучкий Android-емулятор, який розробника виставляє умовам тестування додатків без необхідності використання реального пристрою. Емулятор забезпечує різноманітні сценарії використання -

симуляція обертання екрана, сенсорного введення, геолокації тощо, допомагаючи програмісту перевіряти додаток в різних умовах.

Розробникам також належить цінувати глибоку інтеграцію з Android SDK, що забезпечується у Visual Studio. Керування пакетами Android SDK спрощено і здійснюється при помощі Android SDK Manager, що дозволяє легко оновлювати компоненти Android SDK та встановлювати необхідні розробнику зразки коду.

Visual Studio представляє велику кількість готових шаблонів Android-програм, які легшають життя розробникам, оскільки вони вже містять базовий код і пропонують структуру проекту, розроблену за найкращими практиками.

Visual Studio також гармонійно інтегрується з системами контролю версій, такими як Git, що дозволяє співпрацювати над проектами команді розробників, а також підтримує плагіни з Visual Studio Marketplace, що розширюють можливості середовища розробки.

Загалом, Visual Studio виявляється дуже потужною платформою для розробки Android-додатків, яка допомагає розробникам виконувати складні проєкти, використовуючи для цього знайомі їм інструменти та мови програмування.

Середовище розробки Android Studio. Android Studio – це офіційна інтегрована середа розробки (IDE), яку компанія Google створила спеціально для Android-розробки. Ця високопродуктивна платформа надає розробникам функціональний набір преміум-інструментів для побудови якісного Android-додатка.

Android Studio є відомою своїм мощенням для розробки швидкісних програм, ніяк не жертвуючи якості виробу. Вона включає декілька розроблених функцій, які надають цінні переваги розробникам.

Однією з ключових функцій є "Apply Changes", яка дозволяє вносити зміни в додаток без необхідності його повторного запуску; це суттєво прискорює процес розробки. Іншим інноваційним атрибутом є "Intelligent Code Editor" – смарт-редактор коду, який надає успішні рекомендації щодо

оптимізації коду під час введення [35].

Android Studio також включає емулятор, який не лише швидкий, але і універсальний: він має змогу емулювати широкий діапазон Android-пристроїв, що дозволяє легко встановлювати та запускати тестові програми на різних віртуальних пристроях.

Android Studio включає спектр інструментів та фреймворків для тестування програм. Розробник може виконувати тестування прямо на власному пристрої, емуляторі, в системі безперервної інтеграції, або використовуючи Firebase Test Lab.

Іншою позитивною характеристикою Android Studio є можливість підключитися до додаткових файлів вашого проекту, зокрема до коду C/C++. Також, додатковим плюсом є інтеграція з хмарними сервісами Google, що забезпечує ще більш гладку роботу з проектом. На рисунку 3.4 зображено середовище розробки Android Studio.

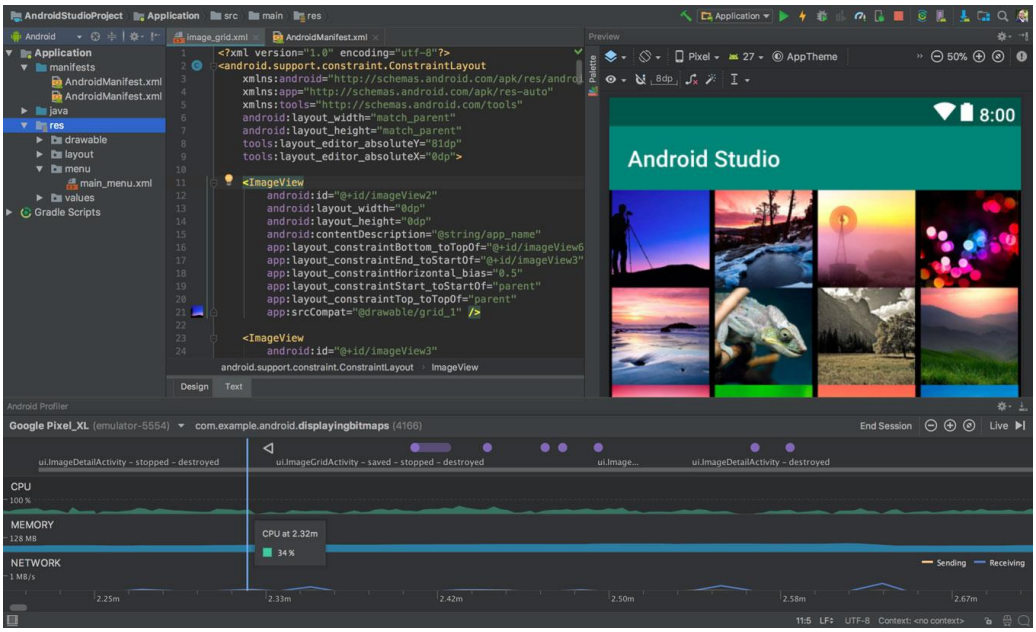

Рисунок 3.4 – Середовище розробки Android Studio

У результаті проведеного аналізу декількох платформ для розробки Android-додатків, виявилися їх різноманітні можливості, швидкодія в процесі компіляції додатку та сумісність із різними мовами програмування. Щоб

успішно дослідити напрацювання, було критично врахувати преференції цих середовищ подання служб, а також плюси та мінуси їхнього використання.

З отриманих висновків було спостережено, що Android Studio, вважається офіційним середовищем розробки від Google, виступає з значними перевагами. Android Studio підтримує використання мови програмування Kotlin, що ідеально підходить для розробки додатків, спеціально пристосованих під платформу Android. Завдяки цим перевагам, Android Studio стає надзвичайно цінним інструментом для розробників, що шукають високошвидкісне та продуктивне універсальне середовище розробки.

Тому, детальний аналіз та порівняння показав, що найкращим вибором для створення програмного забезпечення для оптимізації структури витрат буде середовище розробки Android Studio.

3.3 Програмна реалізація

Для розробки обраного нами мобільного додатку ми вирішили скористатися шаблоном проектування MVVM (Model-View-ViewModel). Цей підхід не лише спрощує процес розробки, але і надає структуру, яка зробить наш код чистішим, більш зрозумілим та легшим для підтримки.

Під час розробки графічного інтерфейсу користувача, робота розподілена на дві основні частини. По-перше, це створення самого інтерфейсу, що включає в себе розмітку або кодування графічного інтерфейсу користувача (GUI). По-друге, це процес побудови бізнес-логіки або так званої "серцевини" додатку, який включає у себе розробку логіки бекенду або моделі даних [36].

Суть шаблону MVVM полягає у наступному: ViewModel слугує перетворювачем значень, роллю якого є керування процесом перетворення даних моделі в таку форму, що з задоволенням і легкістю співпрацює з інтерфейсом користувача. В цьому ракурсі, ViewModel схожа більше на модель, ніж як представлення, контролюючи більшість логіки відображення.

У деяких випадках ViewModel може реалізувати патерн медіатора: організовуючи доступ до бекенду навколо набору юз-кейсів, що підтримуються представленням. На рисунку 3.5 зображено схематично шаблон проектування MVVM.

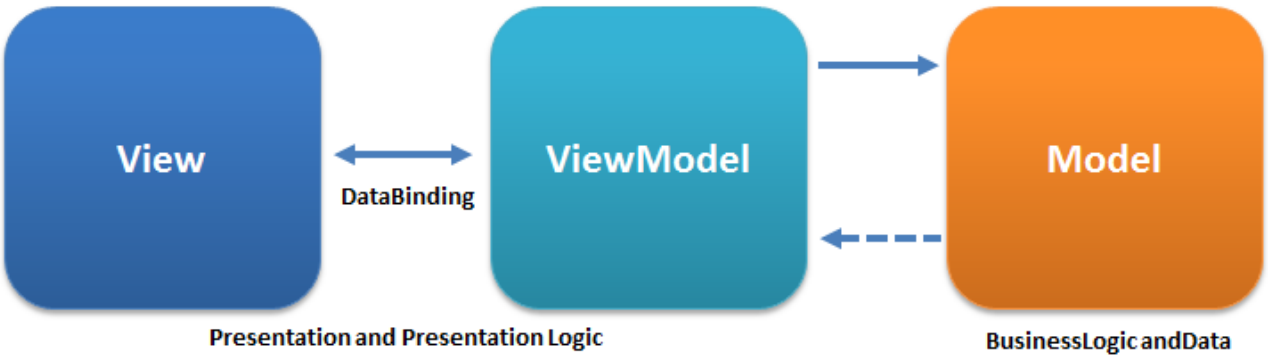

Рисунок 3.5 – Шаблон проектування MVVM

View в шаблоні проектування MVVM представляє собою графічний інтерфейс, який відображає вміст на екрані користувацького пристрою. Це можуть бути як статичні компоненти (текст, кнопки), так і динамічні – анімації, зміна статусу кнопок, виведення даних із бази і тд. Важливо зазначити, що в View абсолютно відсутня бізнес-логіка або логіка обробки даних – його завданням є лише відображення вмісту. Зазвичай, за View в Android відповідають активності (Activity) або фрагменти (Fragment) [36].

View Model в цьому шаблоні відіграє роль посередника між View та Model, беручи на себе обробку даних. View Model відповідає за оформлення даних з манерою, яка найбільш зручна для View. У Android для подібних завдань використовуються вже готові класи, такі як AndroidViewModel або ViewModel, що спрощують проведення зв'язування даних з елементами інтерфейсу.

Model – це сховище даних та бізнес-логіки в вашому додатку, включаючи бази даних, репозиторії для зберігання даних, а також різноманітні класи і методи бізнес-логіки, які обробляють ці дані. Важливо зауважити, що Model працює незалежно від графічного інтерфейсу і не має інформації про те, як саме

і куди його дані вводяться і виводяться. На рисунку 3.6 зображено взаємодію між різними компонентами.

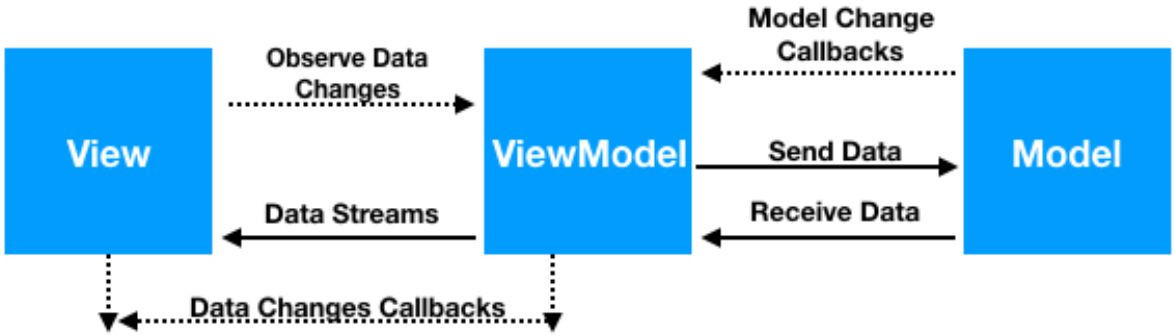

Рисунок 3.6 – Взаємодія у MVVM між різними компонентами

Для успішної реалізації патерну MVVM в додатку, необхідно ретельно опрацювати взаємовідносини між його компонентами. Кожен компонент (View, ViewModel, Model) залежить від іншого, що створює певний ланцюжок залежностей.

При використанні архітектурних компонентів Android, оптимальним рішенням є дотримання моделі, представленої на рисунку 3.7. Стрілки на цьому рисунку показують напрямок зв'язку – від представлення (активності/фрагменту) до моделі.

Це означає, що View знає про існування ViewModel, але не навпаки. Аналогічно, ViewModel знає про Model, але не навпаки. Таким чином, відводиться чітка грань між елементами: у представлення є зв'язок із моделлю представлення (ViewModel), а у моделі представлення є зв'язок із даними (Model). Слід виключно таку послідовність, без винятків.

Дотримання такої архітектурної моделі підвищує надійність програми, спрощує її підтримку та тестування, завдяки чіткими границями і зв’язками між компонентами.

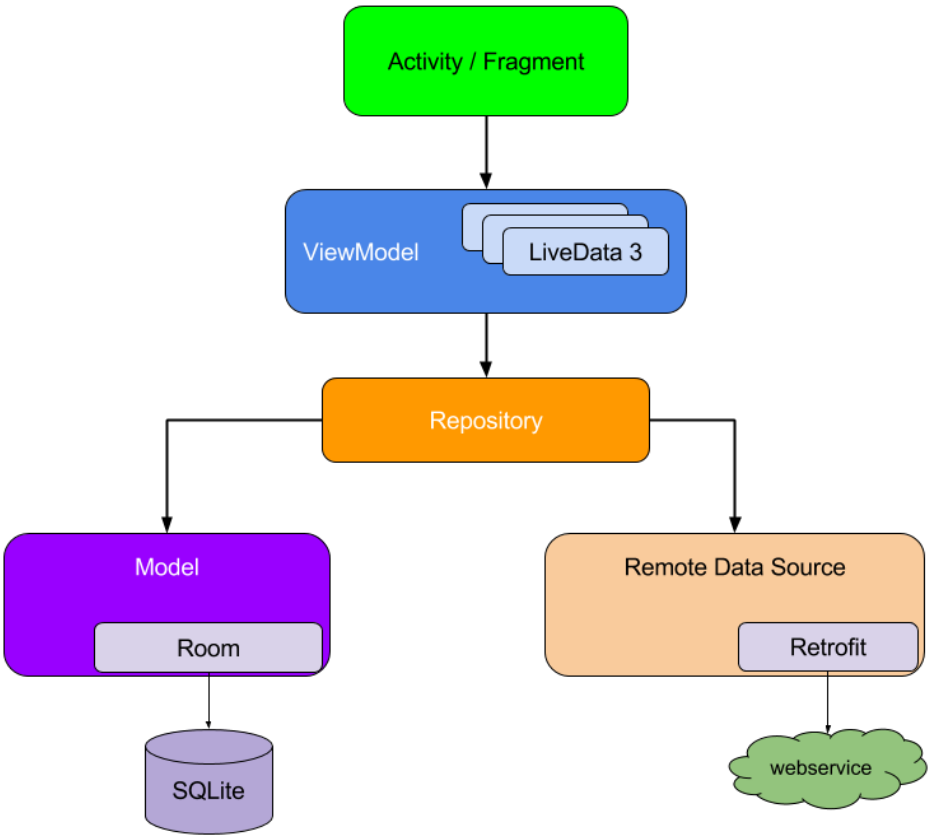

Рисунок 3.7 – Залежності у шаблоні проектування MVVM

Розглянемо конкретний приклад роботи з програмним засобом для оптимізації витрат. При додаванні нової транзакції користувач впроваджує потрібні дані у поля введення, включаючи числове значення – суму транзакції.

На цей процес виконання додавання транзакції реагує спеціальна функція, яка активується відразу після того, як користувач натискає кнопку «Зберегти». Одразу ж після цього, функція отримує числове значення, введене у поле суми транзакції.

Далі відбувається двоступенева валідація цієї інформації: перевіряється, чи не залишилось поле для введення суми порожнім, а потім відбувається перевірка правильності введеної суми. Для прикладу, це може бути перевірка на від'ємне значення, або на нереально велике число. Лістинг функції зображено на рисунку 3.8.

```
private fun actionOnSaveTransactionButton() {
    binding.transactionSaveChangesButton.setOnClickListener {
        val transactionString = binding.transitionValueInput.text.toString()
        if (transactionString.isNotEmpty() && transactionString.toDoubleOrNull() != null) {
            updateDataInDatabase(transactionString.toDouble().makeRounded())
            requireActivity().onBackPressed()
        } else requireContext().showToastLong(getString(R.string.error_transaction_value_empty))
    }
}
```

Рисунок 3.8 – Лістинг функції додавання транзакції до бази даних

Після натискання користувачем кнопки «Зберегти» та успішної валідації суми транзакції, наша функція приступає до додаткового збору даних.

Вона перебирає різні поля введення даних для отримання інформації про транзакцію. По-перше, функція зчитує вибрану користувачем категорію витрат. Наступним кроком є зчитування додаткової інформації про транзакцію, наприклад, користувач може додати опис витрат, але це не є обов’язково. Тоді функція зчитує дату витрати вказану користувачем.

Також неодмінним елементом є поле геолокації, яке зберігає місцезнаходження витрати. Після збору всієї необхідної інформації функція передає її до ViewModel, котра у свою чергу, відправляє отримані дані для збереження до відповідних баз даних. Лістинг коду збереження значення у базу даних зображено на рисунку 3.9.

```
private fun updateDataInDatabase(transactionValue: Double) {
    currentUser?.let { user ->
        val categoryTitle = binding.transitionCategoryInput.text.toString()
        val description = binding.transitionDescriptionInput.text.toString()
        val location = binding.transitionMapInput.text.toString()
        user.expense += transactionValue
        user.currentBalance -= transactionValue
        viewModel.updateUser(user)
    }
}
```

Рисунок 3.9 – Лістинг коду збереження значення у базу даних

Розглянемо процес роботи функції, яка створює нагадування про майбутню покупку в мобільному додатку для оптимізації структури витрат. Програмний код здійснює перевірку даних, а саме: наявність тексту у полі з назвою задачі та коректність дати майбутньої покупки.

Після, функція здійснює процес збереження інформації в базу даних. Потім вона ініціює створення відповідного сповіщення (нотифікації) за допомогою виклику відповідної функції, передаючи туди необхідні параметри.

У випадку, якщо користувачем були надані невірні дані, система згенерує повідомлення про помилку та попросить користувача ввести дані повторно, вже з дотриманням необхідних форматів і критеріїв. На рисунку 3.10 зображено лістинг коду для створення нотифікації.

```
val delay = calendar.timeInMillis - System.currentTimeMillis()
val workManger = WorkManager.getInstance(applicationContext)
workManger.cancelAllWorkByTag(NotificationWorker.NOTIFICATION_TAG)
val workRequest = OneTimeWorkRequest.Builder(NotificationWorker::class.java)
    .setInitialDelay(delay, TimeUnit.MILLISECONDS)
    .setInputData(notificationData.build())
    .addTag(NotificationWorker.NOTIFICATION_TAG + plannerText)
    .build()
workManger.enqueueUniqueWork(
    NotificationWorker.NOTIFICATION_TAG + plannerText,
    ExistingWorkPolicy.APPEND,
    workRequest
)
```

Рисунок 3.10 – Лістинг коду для створення нагадування

Рисунок 3.11 відображає фрагмент коду, що відповідає за етап завантаження категорій з бази даних. Цей процес має велику важливість у контексті взаємодії користувача з програмою при створенні нової транзакції.

При переході користувача до вікна додавання нової транзакції, він отримує можливість або вибрати існуючу категорію для своєї транзакції, або створити нову. Цей вибір здійснюється зі списку категорій, який попередньо був завантажений у пам'ять з бази даних.

```
viewModel.getCategories().observe(viewLifecycleOwner) { list ->
    val listTyped = when (categoryType) {
        TransactionType.Expense -> list.filter { it.type is TransactionType.Expense }
        TransactionType.Income -> list.filter { it.type is TransactionType.Income }
    }
    when (listTyped.isEmpty()) {
        true -> showEmpty()
        false -> {
            (binding.categoryList.adapter as CategoryListAdapter).updateList(listTyped)
            showContext()
        }
    }
}
```

Рисунок 3.11 – Лістинг коду для завантаження категорій із бази даних

Рисунок 3.12 демонструє структуру коду, яка відповідає за обчислення загальної суми витрат та доходів користувача. Це важливий аспект функціоналу додатку, адже він надає користувачеві прозору картину його фінансової активності, допомагаючи у створенні більш повної візуалізації його історії витрат та доходів.

Ця функція працює через звернення до бази даних, де вона реалізує пошук за певним типом транзакцій – витрати або доходи. Вона трансформує цей необхідний об'єм характерної інформації до зручної для використання форми, працюючи із числовими даними, аби вирахувати сумарні значення для відповідних категорій.

```
private fun setUpCategoriesList() {
    viewModel.getCategoriesCount().observe(this) { count ->
        if (count < 1) {
            val list = Category.baseCategories(applicationContext)
            viewModel.insertBaseCategories(list)
            val expense = list.first { it.type == TransactionType.Expense }
            SharedUtil.setLatestCategory(expense)
            val income = list.first { it.type == TransactionType.Income }
            SharedUtil.setLatestCategory(income)
        }
    }
}
```

Рисунок 3.12 – Лістинг коду для обчислення загальної суми витрат та доходів користувача

В рамках підрозділу "Програмна реалізація", був проведений детальний аналіз ключових функцій мобільного додатку. Під час цього аналізу проаналізовано особливості взаємодії між різними екранами додатку і з базою даних, а також загальні умови роботи програмного засобу.

З метою оптимальної організації процесу розробки, було прийнято рішення використовувати шаблон проектування MVVM. Цей шаблон забезпечує чітку структуру та управління взаємодією між різними компонентами додатку, зокрема із екранами користувацького інтерфейсу та базою даних, що сприяє створенню гнучкої, надійної та легко підтримуваної системи.

2.4 Висновки

1. Після глибокого аналізу різноманітних мов програмування та середовищ розробки, що використовуються при створенні мобільних додатків для платформи Android, було зроблено висновок про надзвичайну ефективність мови програмування Kotlin у співпраці з середовищем розробки Android Studio. Це обґрунтовують виключним фокусом цих інструментів на операційну систему Android та відмінні можливості для розробки мобільних додатків будь-якого рівня складності.

2. Застосовано мову програмування Kotlin та Android Studio для розробки методів та програмних засобів, що спрямовані на оптимізацію структури витрат. Таке використання цих ресурсів у розробці дозволило розробити програмні засоби високої якості та ефективності.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

4.1 Вибір методики тестування

Сьогодні мобільні телефони перетворилися на потужні гаджети, завдяки безлічі доступних програм, що значно перевищують їх первісну функціональність зв'язкового пристрою. З масивною колекцією близько 4 мільйонів мобільних додатків, доступних для скачування, користувачі мають вражаючий спектр варіантів.

Цей відповідний високий рівень конкуренції змушує розробників зосередитися не лише на відповідності вимогам користувачів, але й на перевершенні їхніх очікувань із чітким акцентом на якість, зручність використання та безпеку додатка.

Проте, для того, щоб надійно гарантувати успішну реалізацію цих цілей та швидке виходження продукту на ринок, необхідне відповідне планування та організація всього процесу розробки. Розуміння різних типів тестування мобільних додатків і їх особливостей є дуже важливим фактором у розробці. Розуміння цих аспектів допоможе оптимізувати розвиток програмного продукту і дозволить швидше прорватися через життєвий цикл розробки ПЗ.

Функціональне тестування. Функціональне тестування програмного забезпечення цінне у забезпеченні коректності його роботи. Тут головний акцент покладається на обсяг та хід роботи програми з метою підтвердити відповідність її функцій заданим вимогам та технічним специфікаціям [37].

Існує значна кількість стратегій функціонального тестування, і найоптимальнішим шляхом до всебічного вивчення є комбінація ручних методів тестування та автоматизованого підходу.

Серед найпоширеніших стратегій функціонального тестування виділяють методи тестування типу "чорного ящика", коли тестувальнику не потрібно знати подробиці внутрішнього вихідного коду, а його завданням стає перевірка коректної реакції програми на різні вхідні дані.

Давайте поглянемо на декілька поширених методів функціонального тестування:

1. Тестування встановлення – особливо важливе для десктопних та мобільних додатків, де важлива впевненість у коректності їх встановлення;

2. Аналіз межових значень – вивчення реакції програми на крайні значення численних входів;

3. Розміщення еквівалентних класів – об'єднання похожих тестів, щоб зменшити кількість перекриваючих одна одну схожих тестів;

4. Евристичні перевірки – аналіз програмного забезпечення з метою виявлення потенційних місць, де найімовірніше можуть виникнути збої;

5. Модульне тестування – вивчення окремих складових програмного забезпечення на предмет їх коректної роботи;

6. Тестування API – контроль коректної роботи внутрішніх та зовнішніх API з урахуванням усіх аспектів передачі даних та авторизації;

7. Регресійне тестування – тестування, яке забезпечує впевненість, що нові зміни програмного забезпечення не зашкодили звичайному функціонуванню (термін, який часто використовується в контексті автоматизації).

Кожен сценарій функціонального тестування передбачає визначений вихід, який очікується при даному або будь-якому вході. Функціональний тест може бути оформлений у формі сценарію з більш чітким визначенням "пройдено/не пройдено".

Тестування зручності використання. Тестування зручності користувача, що часто називають випробуванням користувацького досвіду, зумовлює перевірку комфорту та зручності використання програмного додатка. Цей процес присвячений повному спектру взаємодії користувача з програмою, ретельно вивчає його досвід в рамках виявлення помилок та розробки рекомендацій щодо поліпшення роботи додатка [38].

Досить важливим є питання реалістичності процесу тестування: проводити його слід практичними користувачами і на реальних пристроях. Це робить можливим швидко виявляти та виправляти проблеми у зручності використання до моменту реального застосування програми.

Виконання цього виду тестування граничить із мистецтвом, адже воно потребує досвідчених тестувальників, в здатності яких спрощувати процес и витягувати інформацію, що найбільше відображає досвід реальних користувачів.

Серед центральних аспектів, на які слід звертати увагу під час проведення тестування, слід відзначити:

1. Якісний дизайн та вдале планування;

2. Інтуїтивність та зрозумілість для користувача;

3. Час реакції – багатьом користувачам важливо, щоб програма запускалася за лічені секунди після відкриття.

Оскільки тестування користувацької зручності має суб'єктивну природу, незабутньо важливо вивчити потреби та технічні прихильності цільового користувача.

Серед авторитетних методів цього виду тестування варто згадати:

1. Розробку глибоко продуманих сценаріїв тестування та анкет для відгуків;

2. Інтеграцію анкет із запитаннями, які стосуються зручності використання, у загальний цикл тестування. Це націлить тестувальників на зміст тестування користувацької зручності, надасть їм доступ до онлайн-анкет і дозволить у рамках доручених їм завдань надати продуктивний зворотній зв’язок;

3. Аналіз відповідей на анкети та зведення підсумків на їх основі. Вірогідно аналізовані дані дозволять розробляти аргументовані рекомендації щодо поліпшення загального сприйняття додатка клієнтами.

Тестування на сумісність. Тестування на сумісність це важливий елемент нефункціонального аналізу, орієнтований на забезпечення правильної роботи вашого мобільного додатка на різних платформах, пристроях, програмах, мережевих середовищах і при різних технічних параметрах обладнання [38].

Цей процес вимагає перевірки ряду ключових аспектів:

1. Сумісність програми з різними операційними системами (Windows, iOS, Android) та їх версіями;

2. Ефективність додатка при роботі із мережами різних типів і параметрів (пропускна здатність, швидкість інтернет-з’єднання);

3. Сумісність програми з основними браузерами (Google Chrome, Safari, Firefox);

4. Забезпечення коректної роботи програми на пристроях з різними технічними характеристиками (розмір екрана, обсяг пам'яті).

Також важливо згадати два типи тестування на сумісність:

Тестування "назад" (backward compatibility testing), спрямоване на визначення сумісності додатка із старими версіями програмного забезпечення;

Тестування "вперед" (forward compatibility testing), здійснює сумісність додатка з новішими, включаючи ті, що лише збираються вийти, версіями програмних платформ.

Тестування продуктивності та навантаження. Тестування продуктивності та навантаження – це важливий етап перевірки ефективності роботи мобільного додатка при певному обсязі завантаження. Ці види тестування відіграють ключову роль у визначенні коректності роботи основних функцій додатка [38].

Цей процес включає в себе перевірку:

1. Продуктивності пристрою: оцінюється швидкість запуску додатка, його вплив на автономність пристрою та споживання оперативної пам'яті;

2. Продуктивності мережі: аналізується час отримання інформації, можливість виникнення затримок або помилок;

3. Ефективність роботи API/серверу: перевіряється швидкість відправлення та формат передачі даних.

Крім цього, справжній додаток повинен бути оснащений надійними механізмами резервного копіювання та відновлення даних. Це надає безпеку, що при втраті інформації з будь-якої причини, її можливо буде повернути. В рамках тестування продуктивності та навантаження ці можливості програми варті формальної перевірки.

Тестування безпеки. Тестування безпеки – це надійний акцент у процесі аналізу кваліфікації програмного додатка. Цей процес спрямований на виявлення потенційних слабкостей додатку, що можуть стати точками зриву для атак зловмисників, а також перевірка на відповідність загальновизнаним стандартам безпеки [38].

Серед ключових аспектів, на які слід звертати увагу під час проведення тестування безпеки, варто відзначити:

1. Автентифікація: перевірка наявності та ефективності системи аутентифікації користувачів, включаючи методи захисту від «брутфорс-атак» та перебору паролів.

2. Авторизація: визначення правил доступу до функціоналу, їх відповідність ролі користувача в системі.

3. Криптографічна безпека: перевірка механізмів шифрування даних, які передаються і зберігаються в додатку.

4. Безпека даних: визначення політик зберігання, опрацювання та передачі персональних даних користувача, виявлення порушень.

5. Митні лінії коду: виявлення вразливостей самого коду додатка.

6. Виявлення вагомих помилок і збоїв, які можуть провокувати витоки інформації.

Все це лише частина завдань, які виконуються під час процесу тестування безпеки, цей список може значно розширюватися в залежності від специфіки додатка та вимог до нього.

Тестування установки. Тестування установки, іноді згадуване як тестування впровадження, є спеціалізованим процесом, що перевіряє коректність встановлення та видалення програми на пристрої користувача. Це незамінний етап для забезпечення гладкого використання додатка від самого початку його життєвого циклу [38].

Серед основних елементів, які перевіряються під час тестування установки, включають:

1. Правильність установки програми: Наскільки коректно і гладко проходить процес установки, чи відповідають відображувані повідомлення фактичному стану процесу, якого вони стосуються.

2. Робота майбутнього оновлення: Чи здатна програма належним чином оновлюватися, не порушуючи при цьому даних користувача та встановлених налаштувань.

3. Процес видалення: Перевіряється, чи безпечно та повністю видаляється програма з системи, не залишаючи "сирітських" файлів та некоректних записів у системному реєстрі.

4. Поведінка при перериванні процесів: Оцінюється робота програми у випадку раптового закриття в процесі установки чи видалення.

5. Випробування помилок та відновлення роботи: Перевірка роботи програми в екстремальних умовах, щоб упевнитися, що користувач завжди має можливість відновлювати додаток після випадкової втрати.

Введення такого виду тестування у план перевірки якості додатка дозволяє забезпечити гладкий процес установки та видалення, при цьому запобігаючи можливим проблемам, що можуть травмувати користувацький досвід.

Тестування локалізації. Тестування локалізації – це процес перевірки правильного адаптування програмного продукту або веб-сайту під конкретну мову та культуру користувача. Цей тип тестування є важливим складником сучасних програмних систем, орієнтованих на багатомовну аудиторію [38].

Основні аспекти над якими працюють під час тестування локалізації включають:

1. Уніфікацію текстового і графічного контенту: переклад текстових елементів додатку, адаптацію зображень та графічних елементів до культурних особливостей країни користувача.

2. Переклад інтерфейсу та документації: переклад текстів меню, повідомлень про помилки, довідників та інших матеріалів користувача.

3. Перевірку формату дати, часу, число, валюти, одиниць виміру: ці формати можуть відрізнятися залежно від країни і культури.

4. Правильність виведення символів кирилиці, виведення тексту справа-наліво для таких мов як арабська чи іврит.

5. Перевірку на використання відповідних шрифтів та їхню щільність.

Проведення тестування локалізації допомагає забезпечити комфортне та зрозуміле використання програмного продукту користувачами з різних країн та культурних середовищ.

Ручне тестування. Тестування мобільних додатків є складним процесом, що включає різноманітні механізми. Ручне тестування – це основний вид тестування програмних продуктів, при якому весь процес перевірки якості продукту контролюється й проводиться людиною. Цей тип тестування застосовується в усіх стадіях роботи програми, включаючи тестування нових функцій, регресійне тестування, тестування сценаріїв користувача, виявлення дефектів та їхню подальшу виправлення.

Головні етапи ручного тестування зазвичай включають:

1. Розробка тестового плану та сценаріїв: Тестувальник розробляє стратегію та план тестування, включаючи визначення цілей тестування, необхідних ресурсів, ризиків та області тестування.

2. Підготовка тестового оточення: На цьому етапі налаштовується середовище для проведення тестів, включаючи налаштування необхідного програмного та апаратного забезпечення.

3. Виконання тестових сценаріїв: Тестувальник запускає програму та виконує тестові сценарії, слідкуючи за відповідністю результатів очікуваним.

4. Оформлення результатів тестування: Збирається інформація про всі виявлені проблеми та подальші дії щодо їх виправлення.

5. Повторне тестування після виправлення помилок: Після усунення знайдених дефектів проводиться повторне тестування, щоб впевнитися в правильності внесених змін.

Основна перевага ручного тестування - це можливість тестувати продукт з точки зору безпосереднього користувача, оцінюючи його зручність, інтуїтивність та загальну якість.

Автоматизоване тестування. Автоматизоване тестування – це процес виконання тестових сценаріїв із використанням спеціалізованого програмного забезпечення, замість традиційного ручного проведення тестів. Цей підхід значно збільшує швидкість та ефективність тестування, забезпечуючи більш точне і консистентне виконання тестових процедур.

Основні аспекти автоматизованого тестування включають:

1. Розробка тестових скриптів: це програми, які автоматично виконують певні кроки, імітуючи дії користувача.

2. Використання спеціалізованих інструментів: є багато інструментів, розроблених спеціально для автоматизованого тестування, як наприклад Selenium, JUnit, TestComplete, Appium тощо.

3. Проведення набору повторюваних тестів: автоматизовані тести ідеально підходять для виконання регресійного тестування, тестування продуктивності, тестування навантаження тощо.

4. Збір та аналіз результатів: автоматизоване тестування зазвичай забезпечує більш швидке та точне підсумування результатів, включаючи детальну інформацію про будь-які виявлені помилки.

Автоматизоване тестування допомагає забезпечити більш швидке та ефективне тестування, при цьому зменшуючи шанс людської помилки. Однак, варто пам'ятати, що ручне тестування все ще важливе для перевірки аспектів

програмного продукту, які вимагають людської уваги та сприйняття, наприклад, зручність використання чи відповідність вимогам користувачів.

Тестування мобільних пристроїв. Тестування мобільних пристроїв – це критичний процес, що передбачає перевірку якості роботи апаратного забезпечення, програмного забезпечення та мобільних додатків, щоб забезпечити найвищу продуктивність, надійність, безпеку та користувацький досвід. З його допомогою можливо забезпечити найвищу стабільність та функціональність мобільних додатків на різних пристроях та операційних системах.

Тестування може бути проведено для різних функцій та сценаріїв, які включають, але не обмежуються:

1. Тестування на переривання: Сучасні мобільні пристрої використовуються в щоденному житті, і переривання, як вхідні дзвінки, повідомлення, сповіщення про заряд батареї, оновлення програм, можуть постійно виникати. Тестування на переривання забезпечує стабільність та коректну роботу додатку під час та після цих переривань.

2. Тестування сервісів на основі геолокації: Додатки, що використовують геолокацію, мають враховувати точність, швидкість та приватність даних користувачів. Тестування таких сервісів дає можливість переконатися, що вони працюють коректно і безпечно де б кінець користувач не знаходився.

3. Тестування біометричних сканерів: Ці тести необхідні для перевірки ефективності біометричних сканерів, таких як сканери відбитків пальців, облич та рівня ДНК або інсуліну для аутентифікації користувачів.

4. Тестування NFC-платежів: Оскільки NFC-платежі стають все більш поширеними, важливо, щоб вони були надійними та безпечними. Тестування цієї технології повинно забезпечувати коректну роботу NFC модуля та взаємодію мобільного пристрою з платіжним терміналом.

Проаналізувавши різні види тестування, прийнято рішення використати наступну стратегію для перевірки працездатності програмного засобу для оптимізації структури витрат. Процес тестування може включати комбінацію

ручного тестування, автоматизованого тестування та тестування установки, щоб забезпечити всебічне та глибоке тестування. Ручне тестування дає можливість перевірити унікальні сценарії використання, а автоматизоване тестування забезпечує швидкість і точність виконання повторюваних тестів. Тестування установки виявляє проблеми, що пов'язані з різними операційними системами, моделями телефонів, версіями браузера та настройками пристрою.

4.2 Тестування програмного додатку

Тестування мобільних додатків – це ключовий етап розробки, який допомагає гарантувати продуктивність, безпеку та зручність користувача. У нашому випадку, було проведено як ручне, так і автоматизоване тестування за допомогою фреймворку JUnit.

Одним з перших етапів тестування було перевірка процесу авторизації користувача. Це важливий компонент, оскільки він безпосередньо впливає на безпеку користувацьких даних. Зокрема, було проведено такі тестування:

1) Реєстрація користувача через соціальні мережі без доступу до інтернету. Це є досить важливим тестуванням, адже досить часто користувачі можуть реєструватися в додатках в умовах повільного або відсутнього Інтернет-з'єднання. Додаток має коректно обробити таку ситуацію і відобразити відповідне повідомлення для користувача. Наприклад, напис "Перевірте своє Інтернет-з'єднання і спробуйте знову". В процесі тестування ми забезпечили, що додаток правильно відповідає на таку ситуацію без аварійних чи фатальних помилок.

2) Помилка реєстрації користувача через соціальні мережі. Іноді, через проблеми з серверами соціальних мереж, їх перевантаження, або через зміни в API мережі, можуть виникати помилки під час здійснення реєстраційних процедур. Важливо, щоб додаток відповідав на такі помилки коректно та інформував користувача про причини та можливі шляхи вирішення проблеми. Під час тестування ми впевнилися, що додаток показує попередження про помилку просто та інтуїтивно зрозумілою мовою.

3) Вхід у програму через соціальні мережі при коректних умовах. Користувачі проводять значну частину часу в соціальних мережах, тому їх інтеграція для реєстрації/входу в додаток значно полегшує цей процес і забезпечує кращий користувацький досвід. Під час тестування було перевірено, що все працює плавно й користувач був належним чином переадресований до головної сторінки додатку.

4) Реєстрація користувача за допомогою номеру мобільного телефону: SMS-верифікація є часто використовуваною технологією, яка допомагає пересвідчитися в тому, що користувач – це реальна людина, а не бот. Важливою частиною цього процесу є введення коду з SMS. Оскільки нерідко додаток може автоматично прочитати цей код або користувачу потрібно ввести його вручну, було проведено тестування на перевірку правильності обробки цих даних та коректного повідомлення про помилки, якщо вони виникнуть.

Для гарантування працездатності функції додавання нових транзакцій в додатку, було проведено наступні тестування:

1) Перевірка активності кнопки збереження при різних станах полів вводу: це перше й важливе тестування в рамках функціоналу створення нової транзакції. У формі додавання нової транзакції користувач має можливість вибрати тип транзакції - "Витрати" або "Доходи", ввести суму та дату транзакції, вибрати відповідну категорію та за бажанням додати місцезнаходження, в якому було здійснено транзакцію.

Важливим моментом тут є перевірка функціональності кнопки збереження. Вона має бути активна лише тоді, коли всі обов'язкові поля - сума, дата та категорія транзакції - заповнено коректно. Якщо ці поля залишені порожніми, або заповнені не вірно, кнопка збереження має бути не активна.

Під час нашого тестування, було здійснено спробу вказати невірну суму транзакції, встановити майбутню дату, вибрати категорію з невідповідним типом транзакції, а також залишити деякі поля пустими. В усіх випадках, кнопка збереження залишалася неактивною, та користувачу відображалися

відповідні помилкові повідомлення, що свідчило про правильну роботу функціоналу.

2) Перевірка роботи кнопок, які відповідають за вибір типу транзакції, була сфокусована на гарантованому точному вибору типу транзакції - витрати або доходи. Це важливий етап, так як ці клавіші визначають, яке значення буде приєднано до нової транзакції, і відповідно впливають на обчислення загальних витрат і доходів користувача.

У ході тестування було створено декілька випадкових транзакцій з коректно вказаними даними в елементах форми вибору. Після цього було натиснуто кнопку "Зберегти". Ми спостерігали за тим, як система веде себе на кожному етапі цієї процедури.

Після повернення на головний екран була проведена перевірка відповідності даних для останньої доданої транзакції в системі до її інформації на екрані головної сторінки, зокрема перевіряли фактичний тип транзакції, який було обрано перед збереженням.

Результати тестування показали, що система коректно обробляла всі введені дані і надійно зберігала їх у системі. Згенерований тип транзакції на головному екрані відповідав вибраному типу під час заповнення форми, що підтверджує працездатність клавіш вибору типу транзакції.

3) Перевірка виконання операції відкриття географічної карти. У формі додавання транзакції надана можливість перегляду мапи за допомогою спеціальної клавіші з іконкою карти. В ході тестування перевірялася реакція системи на натискання цієї клавіші – чи відкривається інтерактивна карта, чи гладко та оперативно відбувається перехід користувача до нового екрану з картою.

Важливо перевірити, чи відповідні клавіші керування на екрані карти розташовані правильно та інтуїтивно зрозумілі користувачу. Також важливо переконатися, що навігаційні засоби (наприклад, кнопки масштабування) працюють коректно.

Результатом тестування став позитивний висновок, адже всі перевірені

функції картографічного модулю виявилися працездатними, користувач безперешкодно переходив до вікна з географічною картою, а сама карта очікувано відповідала на дії користувача.

4) Перевірка функції додавання міток на географічну карту. Коли користувач переходить до перегляду карти з форми додавання транзакції, він має мати змогу використовувати цю функцію для відзначення конкретних місць, в яких була здійснена відповідна транзакція.

Це особливо корисно для відстеження розміщення великих витрат або визначення де користувач найчастіше робить покупки. Інформація, відзначена на карті, також може бути використана для обговорення споживання або використання кредитних карточок для певних транзакцій.

В ході тестування, ми додали декілька міток на карті, відповідного до різних витрат. Після встановлення кожної мітки, ми переконались, що вона зберігається і відображається вірно на карті.

Результати тестування показали, що додавання міток на карту працює без помилок та коректно зберігає своє положення, що свідчить про їх успішну реалізацію в додатку.

5) Перевірка стійкості та коректної реакції програми, якщо користувач відмовив у доступі до геолокації для додатка. Ця перевірка важлива, оскільки відсутність доступу до геолокації може вплинути на деякі функції залежно від їх дизайну та реалізації.

Для тестування цього сценарію, було відкрито додаток і перейдено до картографічного модулю. Наступним кроком було відображення діалогового вікна, яке запитує дозвіл на доступ до геолокації користувача. На цьому етапі було вибрано варіант відхилення цього запиту.

У відповідь на це, програма повинна продовжувати працювати нормально, але з відсутністю можливості відстеження місцезнаходження користувача на карті. Наш тест показав, що програма продовжила працювати в штатному режимі відповідно до очікуваного реакції на цю ситуацію, що свідчить про позитивний результат тестування.

6) Тестування роботи клавіші місцезнаходження на екрані карти. Цей етап перевірки включає запуск додатка, навігацію до картографічного модулю та вибір опції надати дозвіл на використання геолокації, коли додаток запитує цю дозвіл у діалоговому вікні.

Одразу після отримання дозволу на доступ до геолокації, ми перевіряли роботу клавіші місцезнаходження на екрані карти, яка має показувати фактичне місцезнаходження користувача на карті. Для цього була виконана активізація цієї кнопки, що має змусити карту пересуватись до поточного місцеположення користувача.

Результати тестування були позитивними: програма успішно обробила запит на доступ до геолокації, а кнопка місцезнаходження коректно і явно відображала поточне місцезнаходження користувача на географічній карті.

7) Перевірка роботи функції вибору дати для нової транзакції. У формі додавання деталей транзакції наявне спеціальне поле вводу для дати, що має відкривати інтерактивний календар при натисканні.

Під час тестування ми активували це поле вводу, що призвело до відображення діалогового вікна з календарем. В календарі було вибрано конкретну дату та натиснуто кнопку "Ок" для підтвердження вибору.

Після цього ми перевіряли, чи коректно обрана дата відображається в полі вводу дати. Результати тестування показали, що дата вибрана вірно, і відображається у відповідному полі вводу без помилок, що є виявом успішного проходження тестування цієї функції.

8) Перевірка роботи функції вибору категорії транзакції. Цей модуль є важливим елементом форми додавання нової транзакції, оскільки він дозволяє класифікувати кожну транзакцію згідно з її призначенням.

Під час тестування ми активували цю функцію, натиснувши на поле "Категорія". Це відкрило діалогове вікно, що включає динамічний список категорій, в якому користувач може вибрати відповідну категорію.

Після вибору конкретної категорії, ми повернулися до головної форми вводу транзакції і перевіряли, чи правильно обрана категорія тепер

відображається в полі "Категорія". Результати тестування показали, що категорія була вибрана і відображена коректно, що свідчить про успішність тестування цієї функції.

9) Перевірка функції створення нової категорії. Цей тестовий сценарій скерований на перевірку можливості користувача додати власні категорії до системи, що є особливо корисним для більш точної і індивідуальної класифікації транзакцій.

Процес тестування був започаткований з запуску додатку та переходу до форми заповнення даних для нової транзакції. Після цього ми перейшли до вікна вибору категорії, де в правому нижньому куті екрану розташована кнопка для створення нової категорії.

Під час натискання кнопки відкрилося діалогове вікно, в якому були заповнені всі необхідні поля для створення нової категорії. Нова категорія була успішно збережена та відображена в загальному списку категорій.

Результати тестування показали, що додавання нової категорії працює ефективно, що підтверджує заявлену функціональність додатка.

Наступним кроком тестування є перевірка функції «Запланована покупка». Для гарантування працездатності цієї функції було проведено наступні тестування:

1) Перевірка роботи кнопки створення нового плану. Ця перевірка націлена на те, щоб впевнитися, що користувач може легко здійснювати планування своїх транзакцій за допомогою наданих у додатку інструментів.

Процес тестування почався з переходу до розділу планувальника в додатку, де знаходиться клавіша зі знаком "+", що представляє створення нового плану. Відбулося активування цієї клавіші шляхом натискання.

Результат тестування полягає в перевірці відповіді додатка на таку дію: чи спрацьовує перехід до нового вікна, де користувач може внести деталі нового плану. Таким чином, коректне виконання цих операцій вказує на правильну реалізацію функцій додавання нових планів.

2) Перевірка реалізації функції створення нового нагадування. Цей

процес передбачає перевірку можливості користувача зручно планувати майбутні події та отримувати нагадування про них.

Тестування було розпочато з переходу до розділу планувальника в додатку, де була активована клавіша з позначкою "+". Далі було відкрито вікно додавання нового плану, в якому було заповнено всі обов'язкові поля, включно з датою та часом нагадування. Закінчилося тестування активуванням клавіші "Створити нагадування".

Як результат, додаток повинен був повернути нас до головного екрану планувальника з усіма раніше створеними планами, а також у вказаний час повинно було відобразити нагадування. Такий розгортання подій свідчить про коректну роботу запрограмованої функції створення нових нагадувань.

На основі здійсненого тестування було сформовано таблицю 4.1, в якій у нормалізованій формі перелічені усі тестування, їх деталі та результат виконання.

Таблиця 4.1 – Тестування програмного засобу для оптимізації структури витрат

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
1	Реєстрація користувача через соціальні мережі без доступу до інтернету	Перевірка реакції програми на натискання клавіші доступу до соціальних мереж користувачем без інтернет-з'єднання. Здійснюється перевірка на реакцію програми на відсутність доступу до мережі. Програма згенерує повідомлення про помилку або ж може виникнути помилка без обробки, що призведе до закриття програми.	Повідомлення для користувача про відсутність мережі	Успішне виконання

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
2	Помилка реєстрації користувача через соціальні мережі	Перевірка реакції програми на відмову соціальної мережі надати дані для реєстрації користувача. Тестування поведінки, як програма обробляє помилку повернену соціальною мережею: чи інформує користувача або ж здійснює аварійне завершення роботи.	Після тестування, користувач отримує повідомлення про помилку.	Успішне виконання
3	Вхід у програму через соціальні мережі при коректних умовах	Перевірка реакції програми на натискання клавіші соціальної мережі користувачем при наявності інтернет-з'єднання	Отримання відповіді з даними від обраної соціальної мережі.	Успішне виконання
4	Реєстрація користувача за допомогою мобільного телефону	Тестування процесу введення номеру мобільного телефону та натискання кнопки підтвердження. Після цього очікуємо перехід до екрану введення SMS-коду. Вводимо код вручну або дозволяємо системі автоматично зчитати його з SMS. Тоді натискаємо кнопку підтвердження, система виконує запит до бази даних і перевіряє правильність коду. Тест включає введення як вірних, так і невірних даних.	Успішна реєстрація при правильному введенні даних або повідомлення про помилку при неправильному введенні.	Успішне виконання

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
5	Перевірка активності кнопки збереження при різних станах полів вводу	Перевірка вікна додавання транзакції, яке містить клавіші вибору типу транзакції (витрати, доходи), поле для введення суми транзакції, дати, категорії та локації. Ключовими є поле суми, дата та категорія транзакції.	Після тестування при введенні помилкових даних очікується отримання повідомлення про помилку.	Успішне виконання
6	Перевірка роботи кнопок, які відповідають за вибір типу транзакції	Вибираємо тип транзакції та заповнюємо всі поля форми. Після цього натискаємо на кнопку збереження.	Після тестування очікуємо перехід на головну сторінку та здійснюємо порівняння останнього доданого елементу: чи співпадають вказані дані і чи вірний тип транзакції.	Успішне виконання
7	Перевірка виконання операції відкриття географічної карти	В формі для транзакції є клавіша із піктограмою карти. Відбувається її активація.	Після тестування очікується перехід користувача до екрану із картою.	Успішне виконання
8	Перевірка функції додавання міток на географічну карту	Тестування можливості встановлення міток на карті для відображення місць здійснення витрат.	Після активації карти очікується поява міток із позначенням категорії витрат.	Успішне виконання
9	Перевірка стійкості та коректної реакції програми,	Відтворюємо сценарій, коли додаток відкривається та переходить до карти; при виникненні вікна	Після тестування програма продовжує працювати, але без	Успішне виконання

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
	якщо користувач відмовив у доступі до геолокації для додатка	запиту на геолокацію, вибираємо варіант заборонити доступ до геолокації.	відображення актуального місцезнаходження на карті.	
10	Перевірка роботи функції вибору категорії транзакції	У формі заповнення транзакції знаходиться поле "Категорія". Під час тестування активуються це поле, відкривається вікно зі списком категорій, звідки обирається потрібна.	Після тестування у полі "Категорія" відображається обрана категорія.	Успішне виконання
11	Перевірка функції створення нової категорії	Для перевірки функціональності відкриваємо додаток, переходимо до форми транзакції, потім до вікна вибору категорії. Вибираємо опцію додавання нової категорії, заповнюємо всі поля і зберігаємо її.	Після тестування у вікні вибору категорії відображається нова додана категорія.	Успішне виконання
12	Перевірка роботи кнопки створення нового плану	Необхідно перейти до основного розділу додатку – вікна планувальника. Користувач повинен бачити головний елемент керування - кнопку з піктограмою «+», що символізує процес створення або додавання нового елементу.	Після тестування очікується перехід до вікна для створення нового плану.	Успішне виконання

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
13	Перевірка реалізації функції створення нового нагадування	Перевірка функції створення нагадувань. Для цього переходимо до вікна планувальника, натискаємо на кнопку «+»б заповнюємо всі полі додавання нового плану, вказуємо дату та час. Активуємо кнопку "Створити нагадування".	Після тестування сподіваємось на перехід до вікна з уже доданими планами і отримання нагадування у вказаний час.	Успішне виконання
14	Перевірка роботи функції вибору дати для нової транзакції	Перевірка роботи функції вибору дати для нової транзакції. В формі додавання деталей транзакції активуємо поле вводу дати, яке викликає відкриття інтерактивного календаря. В календарі вибираємо конкретну дату та підтверджуємо вибір, натискаючи на кнопку "Ок".	Обрана дата коректно відображається в полі вводу дати, що свідчить про успішне тестування цієї функції.	Успішне виконання

Тестування, описане вище, було успішно проведено. Безперечно, як і в будь-якому процесі розробки, виникли деякі помилки, які потребували корекції. Однак, після виправлення цих помилок, всі тести були успішно пройдені.

4.3 Розробка інструкції користувача

Для публікації нашого програмного засобу, розробленого з метою оптимізації фінансових витрат користувача, в публічний доступ, нам необхідно створити зрозумілу та деталізовану інструкцію користувача.

Інструкція користувача є критичним компонентом успішного розгортання будь-якого програмного забезпечення. Зокрема:

1. Зручність користувача: Інструкція допомагає користувачам краще розуміти функціонал та можливості програмного засобу, що сприяє комфортнішому та ефективнішому його використанню.

2. Зменшення обсягу служби підтримки: Якщо користувачам надається зрозуміла інструкція, вони зможуть самостійно розв'язувати багато питань, що зменшує навантаження на службу підтримки.

3. Навчання: Інструкції можуть бути використані як навчальні матеріали для користувачів, що тільки починають користуватися додатком.

Перша частина повинна містити детальний опис процесу запуску мобільного додатку. Для початку користувачу потрібно натиснути на іконку з назвою «Budget Manager».Запуск програми відкриває головну сторінку додатку, де зразу ж представлені основні функції і розділи. На рисунку 4.1 зображено головну сторінку додатку.

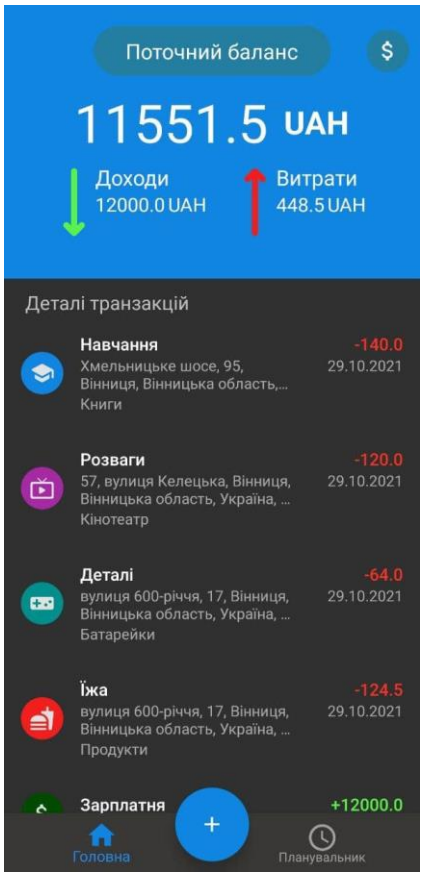

Рисунок 4.1 – Головна сторінка додатку

У нижній частині головного екрану зображена навігаційна панель з рядом важливих клавіш. Серед них – кнопки переходу "на головну сторінку" та "на сторінку планувальника". Біля них розташована універсальна клавіша зареєстрована під символом «+», що дає користувачу можливість додавати нові плани або транзакції прямо з головного екрану (див. рис. 4.2).

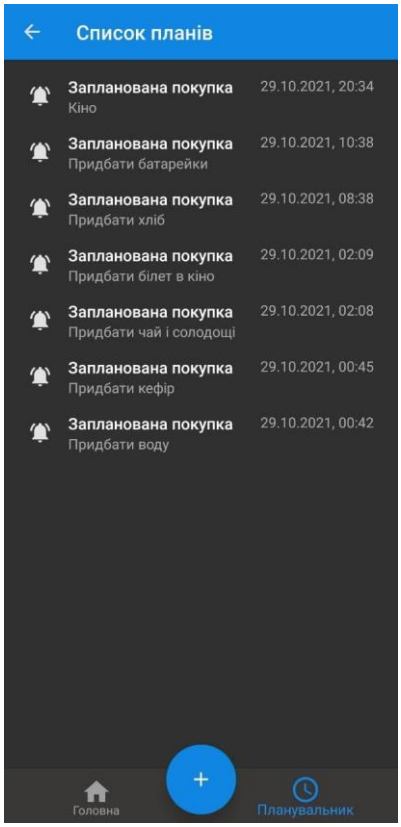

Рисунок 4.2 – Сторінка планувальника

Кнопка із символом "plus", або "+", забезпечує різні функції залежно від поточної сторінки, на якій перебуває користувач:

1. Якщо користувач перебуває на головній сторінці, яка є центральним місцем навігації, клавіша "+" ініціює перехід на сторінку додавання нової транзакції. Ця процедура заощаджує час користувача, дозволяючи йому швидко та легко реєструвати нові витрати або прибутки без необхідності переходити через багаторівневу систему меню або декілька вкладок (див. рис. 4.3).

2. Проте, якщо користувач перебуває на сторінці планувальника, що є спеціально відведеною секцією для створення та відслідковування майбутніх

фінансових зобов'язань та цілей, то клавіша "+" відкриває сторінку додавання нового плану. Сюди користувач може вводити нові події, ставити нагадування та розташовувати свої фінансові плани в календарі (див. рис. 4.4).

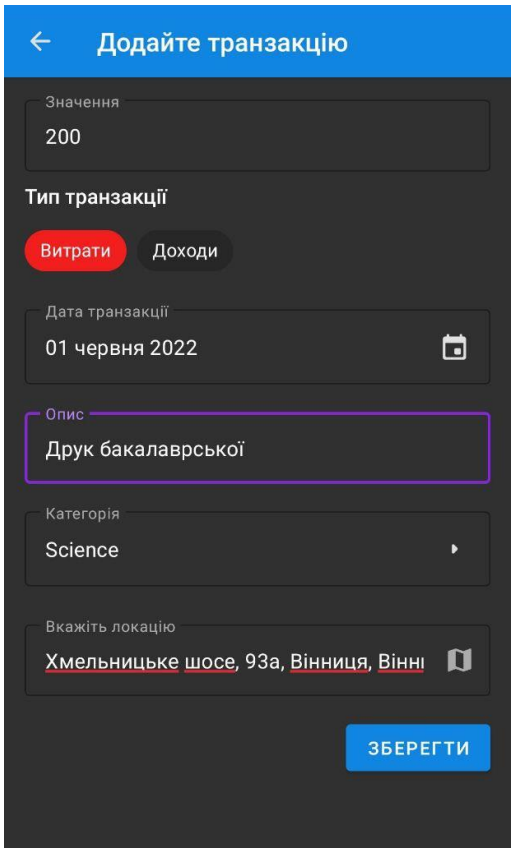

Рисунок 4.3 – Сторінка додавання транзакції

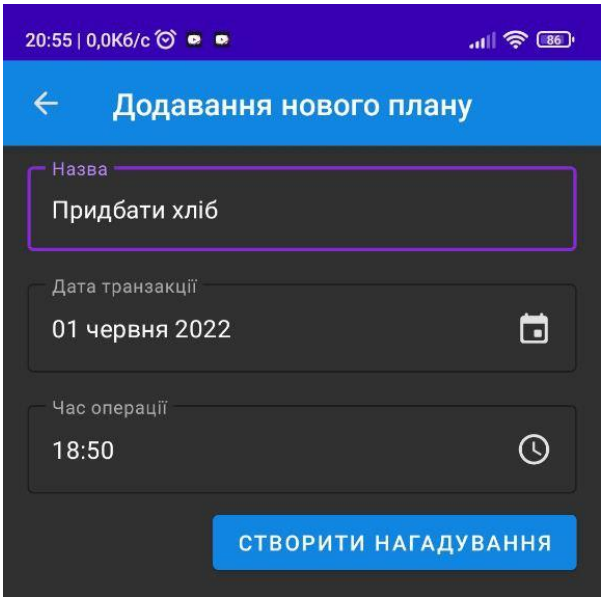

Рисунок 4.4 – Сторінка створення нового плану

Екран додавання нової транзакції в "Budget Manager" ретельно планований, щоб максимізувати ефективність та зручність для користувача. Цей екран включає ряд ключових елементів:

1. Клавіші вибору типу транзакції: дозволяють користувачам вибрати, чи реєструють вони витрати чи дохід.

2. Клавіша для внесення дати транзакції: дає можливість вибору точної дати транзакції з календаря.

3. Поле для опису транзакції: тут користувач може включити будь-яку додаткову інформацію про транзакцію.

4. Клавіша для вибору категорії: користувач може вибрати з представленого списку категорій або додати нову.

Якщо користувач вирішить використати клавішу "Категорія", його буде напралено на екран із загальним списком доступних категорій, з яких він може вибрати (див. Рис. 4.5). Кожна категорія має свій унікальний значок та назву, що допомагає користувачу легко пізнати і вибрати потрібну.

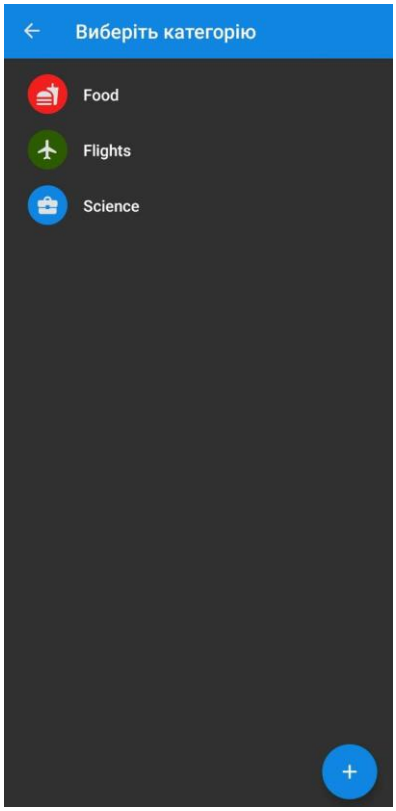

Рисунок 4.5 – Сторінка вибору категорії

За бажанням користувача вказати детальнішу геолокаційну інформацію про транзакцію, додаток "Budget Manager" передбачає використання функції карти.

Після натискання на клавішу "Карта", користувача автоматично перенаправляє на новий екран із інтерактивною картою (див. рис. 4.6). На цьому екрані користувач має можливість використання пошуку для вказівки точної локації або обрати місце прямо на карті, натискаючи на потрібне місце.

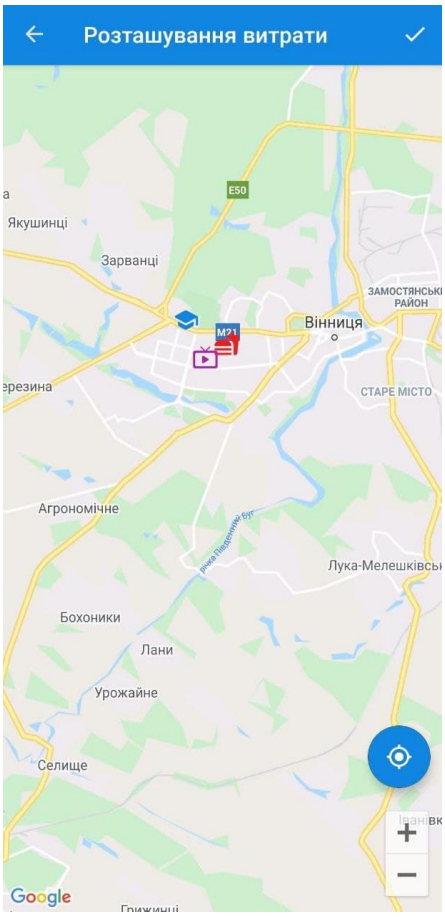

Рисунок 4.6 – Карта у мобільному додатку

Крім того, якщо користувач надає дозвіл на доступ до функції геолокації пристрою, програма "Budget Manager" може автоматично визначати та вказувати поточне місце перебування користувача на карті за допомогою певної мітки. Це особливо зручно для реєстрації транзакцій у режимі реального часу.

Після завершення процесу розробки вичерпної інструкції користувача можна висловити певну впевненість у тому, що мобільний додаток "Budget Manager" характеризується високою ступенем потребам користувача.

Особливістю інтерфейсу цього додатку є його простота та доступність. Кожен елемент дизайну, від кнопок до меню, створюється з метою максимального спрощення навігації та використання.

Більше того, схема керування додатком розроблено таким чином, що користувачам, незалежно від їх технічної грамотності або досвіду використання подібних додатків, не потрібно мати спеціальних навиків для його використання. Це означає, що користувачі будь-якого рівня можуть без проблем впровадити цей додаток у свої повсякденні фінансові режими.

4.4 Висновки

1. Було упроваджено вибір методології тестування. Найбільш популярні методології були досліджені та аналізовані. Для тестування мобільного додатку, який використовує розроблені методи та програмні засоби для оптимізації структури витрат, були обрані методи ручного тестування, автоматизованого тестування та тестування установки.

2. Було проведено детальне тестування мобільного додатку. Всі протестовані функції відпрацювали належним чином без помилок або проблем, свідчить про готовність додатку до використання в реальних умовах для обліку та оптимізації витрат.

3. Заключним етапом стала розробка інструкції користувача. Цей важливий документ, наповнений вичерпними поясненнями та вказівками, допоможе користувачам легко навчитися керувати додатком та використовувати всі його можливості на повну потужність.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» є оцінювання науково-технічного рівня та

рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 **[39].**

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуютьс я у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	4	4
2. Ринкові переваги (наявність аналогів)	4	3	4
3. Ринкові переваги (ціна продукту)	3	4	3
4. Ринкові переваги (технічні властивості)	4	3	3
5. Ринкові переваги (експлуатаційні витрати)	3	4	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	40	40	40
Середньоарифметична сума балів $СБ_с$	40		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 **[39]**.

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» становить 40 балів, що, відповідно до

таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов'язані з проведенням науково-дослідної роботи на тему «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою **[39]**:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \qquad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$$З_о = 50000{,}00 \cdot 65 / 21 = 154761{,}90 \text{ грн.}$$

Проведені розрахунки зведемо до таблиці.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	50000	2380,95	65	154761,90
Інженер-розробник програмного забезпечення	30000	1428,57	65	92857,14
UI/UX-дизайнер	15000,00	714,29	65	46428,57
Всього				294047,62

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \quad (5.2)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду **[39]**;

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих

об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$T_р$ – середнє число робочих днів в місяці, приблизно $T_р$ = 21 дні;

$t_{зм}$ – тривалість зміни, год.

$С_1 = 6700,00 \cdot 1,5 \cdot 1,65 / (21 \cdot 8) = 98,71$ грн.

$З_{р1} = 98,71 \cdot 6,00 = 592,23$ грн.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Налаштування операційної системи	6	4	1,5	98,71	592,23
Налаштування локальної мережі для обладнання	12	5	1,7	111,87	1342,39
Налаштування додаткових сервісів для розробки	16	5	1,7	111,87	1789,86
Тестування системи	50	5	1,7	111,87	5593,30
Всього					9317,79

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 12%.

$З_{дод} = (294047,62 + 9317,79) \cdot 12 / 100\% = 36403,85$грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$З_н$ = (294047,62+ 9317,79+ 36403,85) · 22 / 100% = 74749,23 грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (5.6)$$

де $Н_j$ – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1$ = 3 · 200,00 · 1,1 - 0,000 · 0,00 = 660,0 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од, грн	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір	200	3	0	0	660
Папір для записів	110	1	0	0	121
Канцелярське приладдя (набір офісного працівника)	175	2	0	0	385
Всього					1166

5.2.4 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \,, \qquad (5.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1{,}10…1{,}12$);

k – кількість найменувань устаткування.

$B_{спец} = 50000 \cdot 1 \cdot 1{,}1 = 55000$ грн.

Отримані результати зведемо до таблиці

Таблиця 5.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Системний блок	1	50000	55000
Монітор	2	9000	19800
Комплектуючі (миша, клавіатура), набір	1	3000	3300
Роутер	1	3000	3300
Всього			81400

5.2.5 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10...1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 6000,00 \cdot 1 \cdot 1,12 = 6720$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 5.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Операційна система Microsoft Windows 10 Pro for Workstations	1	6000	6720
Всього			6720

5.2.6 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (55000{,}00 \cdot 3) / (5 \cdot 12) =$ 2750,00 *грн.*

Таблиця 5.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Системний блок	55000	5	3	2750,00
Монітор	19800	5	3	990,00
Комплектуючі (миша, клавіатура), набір	3300	2	3	412,50
Роутер	3300	4	3	206,25
ОС Windows 11	6720	3	3	560,00
Всього				4918,75

5.2.7 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e = 7,50$ грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,15 \cdot 520,0 \cdot 7,50 \cdot 0,95 / 0,97 = 572,94$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 5.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Споживання системного блоку	0,15	520	572,94
Споживання моніторів	0,14	520	534,74
Споживання периферійних пристроїв	0,01	520	38,20
Споживання інтернет обладнання	0,02	520	76,39
Всього			1222,27

5.2.8 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та

приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{Н_{св}}{100\%}, \quad (5.11)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (294047,62+ 9317,79) · 20 / 100% = 60673,08 грн.

5.2.9 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%.

$B_{сп}$ = (294047,62+ 9317,79) · 30 / 100% = 91009,62грн.

5.2.10 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (294047,62+ 9317,79) · 50 / 100% = 151682,70грн.

5.2.11 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}\ , \qquad (5.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$В_{нзв}$ = (294047,62+ 9317,79) · 120 / 100% = 364 038,49 грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}\ . \qquad (5.15)$$

$В_{заг}$= 1177349,40 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}\ , \qquad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,8.

ЗВ = 1177349,40/ 0,8 = 1471686,75 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 6000 користувачів;

2-й рік – 8000 користувачів;

3-й рік – 6000 користувачів.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 15000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 23 000 грн за доступ до користування протягом року;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 1000 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою **[39]:**

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо $\rho = 25\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 = (1000,00 \cdot 15000,00 + 24000,00 \cdot 6000) \cdot 0,83 \cdot 0,25 \cdot (1-0,18/100\%) =$ 27151635 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2 = (1000,00 \cdot 15000,00 + 24000,00 \cdot (6000+8000)) \cdot 0,83 \cdot 0,25 \cdot (1-0,18/100\%) =$ 59938515 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 = (1000,00 \cdot 1500,00 + 240000,00 \cdot (6000+8000+6000)) \cdot 0,83 \cdot 0,25 \cdot (1-0,18/100\%) =$ 84528675 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,25$;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 27151635/(1+0,25)^1 + 59938515/(1+0,25)^2 + 84528675/(1+0,25)^3 =$$

103360639,20 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \tag{5.19}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=2;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1471686,75грн.

$PV = k_{інв} \cdot ЗВ = 2 \cdot 1471686,75 = 2943373,494$грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 103360639,20 грн;

PV – теперішня вартість початкових інвестицій, 2943373,494грн.

$E_{абс} = ПП - PV = 103360639,20 - 2943373,494 = 100417265,71$грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.21)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 100417265,71грн;

PV – теперішня вартість початкових інвестицій, 2943373,494грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+ 100417265,71/ 2943373,494)^{1/3} = 2,27.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.22)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,11;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,18.

$\tau_{мін}$ = 0,11+0,18 = 0,29 < 2,27 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» доцільно [40].

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (5.23)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 2,27 = 0,44 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» становить 40 балів, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вищий середнього).

Також термін окупності становить 0,44 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту».

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи було вдало досягнуто важливої мети – успішно розроблено методи та програмні засоби для оптимізації структури витрат.

З огляду на важливість зрозумілого й зручного представлення цих методів для користувача, було прийнято вирішальне рішення створити інтуїтивно зрозумілий та легкий у використанні мобільний додаток, спеціально розроблений для операційної системи Android.

Цей додаток було написано на потужній та гнучкій мові програмування Kotlin, яка забезпечує розробку високо функціональних та надійних мобільних програм. Кінцевим результатом став додаток, що орієнтований на надання користувачам сучасних та прогресивних методів та програмних засобів, розроблених для ефективної оптимізації структури витрат.

Протягом виконання цієї роботи було досягнуто значних результатів:

1. Аналіз існуючих додатків: Проведений детальний аналіз різних мобільних додатків для обліку витрат виявив основні переваги та недоліки кожного з них. Зокрема, було виявлено, що більшість додатків не надає можливості створювати геолокацію витрат. Ця функція допомагає користувачам краще аналізувати свої витрати та розробляти більш ефективні стратегії збереження коштів.

2. Аналіз штучного інтелекту: Зроблено детальний аналіз потенціалу штучного інтелекту для оптимізації витрат. Розроблено методи, що використовують штучний інтелект для аналізу витрат, виявлення тенденцій та прогнозування майбутніх витрат. Це може направити користувачів до кращих фінансових рішень та допомогти їм економити кошти.

3. Функціональні характеристики: Основним фокусом при розробці додатку було впровадження потужних функцій геолокації, що дозволяють користувачам відстежувати свої розходи з урахуванням місця. Цей підхід може

допомогти користувачам виявити патерни в своїх витратах та визначити потенційні області для економії.

4. Архітектура додатку: Розроблена архітектура додатку передбачає використання як локальної, так і хмарної бази даних, що може гарантувати постійний доступ до даних користувача, навіть без підключення до Інтернету. Крім того, цей підхід дозволяє зберігати дані безпечно і робити резервне копіювання для запобігання можливих втрат.

5. Вибір технології: Для розробки додатку було вибрано середовище розробки Android Studio та мова програмування Kotlin, яка забезпечує високу продуктивність та гнучкість для розробки сучасних мобільних додатків.

6. Розробка додатку: Мовою програмування Kotlin було розроблено повноцінний код додатку, що включає всі задумані функції і можливості.

7. Тестування додатку: Проведено ретельне тестування всіх функцій додатку, включаючи геолокацію витрат, що дозволило переконатися в його надійності та ефективності.

8. Інструкція користувача: В якості зручного посібника для користувачів було створено повну інструкцію, що включає детальні пояснення та керівництва по всіх аспектах використання додатку.

9. У рамках виконання роботи було успішно проведено комерційний аудит науково-технічної розробки, що включав оцінювання потенційних ринків, аналіз конкуренції та формулювання стратегій комерціалізації.

10. Виконано докладний розрахунок витрат на проведення науково-дослідної роботи, включаючи витрати на матеріали, обладнання, оплату праці науковців, а також інші пов'язані витрати. Це дало докладне уявлення про фінансові ресурси, що необхідні для успішного впровадження розробки.

Загалом, цей проект є успішною розробкою нового мобільного додатку, що повністю відповідає потребам користувачів у простому та ефективному інструменті для оптимізації структури витрат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cost Accounting – Concept, Types & Methods [Електронний ресурс]. URL: https://cleartax.in/s/cost-accounting.

2. Лисенко Г.Л. Методичні вказівки до оформлення курсових проектів (робіт) у Вінницькому національному технічному університеті /Уклад. Г.Л. Лисенко, А.Г. Буда, Р.Р. Обертюх, – Вінниця: ВНТУ, 2006. – 60 с.

3. Розробка мобільного додатку з геолокацією обліку витрат [Електронний ресурс] / Майданюк В.П., Ярмола В. С. // Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, Вінниця: Вінницький Національний Технічний Університет. – 2022. URL: https://conferences.vntu.edu.ua/index.php/ all-fitki/all-fitki-022/paper/view/15099

4. Програмне забезпечення для контролю витрат з інтеграцією модуля геолокації [Електронний ресурс] / Майданюк В.П., Ярмола В. С. // XVI Міжнародна науково-практична конференція, Одеса: Одеський Національний Технічний Університет. – 2023. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/38161/Збірник%20тез%20конференції%20ITIA%20–%202023-pages.pdf?sequence=1&isAllowed=y

5. Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту [Електронний ресурс] / Майданюк В.П., Ярмола В. С. «Електронні інформаційні ресурси: створення, використання, доступ» // Міжнародна науково-практична інтернет-конференція, Вінниця: Вінницька академія безперервної освіти. – 2023. URL: https://drive.google.com/file/d/1eAb_QXS6CvA-e1ikKxerTqvw6P1IPRhc/view/

6. Топ-10 додатків для контролю особистих фінансів [Електронний ресурс]. URL: https://womo.ua/top-10-dodatkiv-dlya-kontrolyu-osobistih-finansiv/.

7. Гаманець – облік витрат і доходів, бюджет. [Електронний ресурс]. URL: https://play. google.com/store/apps/details?id=com.vitvov. jc&hl=uk

8. 1Money – облік витрат, бюджет. [Електронний ресурс]. URL: https://play.google.com/ store/apps/details?id=org.pixelrush.moneyiq

9. Пархоменко О.В. Використання гнучких методологій розробки програмного забезпечення у підготовці майбутніх програмістів – Київ, 2021, – 282с.

10. Ітеративна модель. [Електронний ресурс]. URL: https://qalight.ua/baza-znaniy/iterativna-model-iterative-model/

11. What is the Agile methodology? [Електронний ресурс]. URL:

https://www.atlassian.com/agile

12. SDLC – Waterfall Model. [Електронний ресурс]. URL: https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm

13. Application architecture. [Електронний ресурс]. URL: https://www.techtarget. com/searchapparchitecture/definition/application-architecture

14. Explain Algorithm and Flowchart with Examples. [Електронний ресурс]. URL: https://www.edrawsoft.com/explain-algorithm-flowchart.html

15. Діаграма послідовності (Sequence Diagrams). [Електронний ресурс]. URL: https://www.maxzosim.com/sequence-diagrams/

16. How can artificial intelligence help you identify cost savings opportunities? [Електронний ресурс]. URL: https://www.linkedin.com/advice/1/how-can-artificial-intelligence-help-you-identify-cost.

17. Top 6 Areas Generative AI can help in saving costs in Manufacturing. [Електронний ресурс]. URL: https://www.linkedin.com/pulse/top-6-areas-generative-ai-can-help-saving-costs-manufacturing/

18. Geolocation APIs Choosing the Right Service for Mobile App Development. [Електронний ресурс]. URL: https://utilitiesone.com/geolocation-apis-choosing-the-right-service-for-mobile-app-development

19. Geocoding API overview. [Електронний ресурс]. URL: https://developers.google.com/maps/documentation/geocoding/overview

20. IP Geolocation API. [Електронний ресурс]. URL: https://www.geoapify.com/ip-geolocation-api

21. Geocoder API Developer Guide. [Електронний ресурс]. URL: https://www.here.com/docs/bundle/geocoder-api-developer-guide/page/topics/what-is.html

22. Save data in a local database using Room. [Електронний ресурс]. URL: https://developer.android.com/training/data-storage/room

23. Firebase Database. [Електронний ресурс]. URL: https://firebase.google.com/docs/database

24. What is a block diagram? [Електронний ресурс]. URL: https://miro.com/diagramming/what-is-a-block-diagram/

25. Top Programming Languages for Android App Development. [Електронний ресурс]. URL: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/

26. Java. [Електронний ресурс]. URL: https://uk.wikipedia.org/wiki/Java

27. Kotlin for Android. [Електронний ресурс]. URL: https://kotlinlang.org/

docs/android-overview.html

28. Can C++ Be Used for Android App Development? [Електронний ресурс]. URL: https://groovetechnology.com/blog/can-c-be-used-for-android-app-development/

29. Xamarin [Електронний ресурс]. URL: https://learn. microsoft.com/ru-ru/xamarin/android/get-started/hello-android

30. Building Android Apps With Python. [Електронний ресурс]. URL: https://towardsdatascience.com/building-android-apps-with-python-part-1

31. Dart overview. [Електронний ресурс]. URL: https://dart.dev/overview

32. DroidScript – JavaScript IDE. [Електронний ресурс]. URL: https://droid script.org

33. Android IDEs for Developers. [Електронний ресурс]. URL: https://ncube. com/blog/android-ides-for-developers

34. Mobile app development. [Електронний ресурс]. URL: https://visual studio.microsoft.com/vs/features/mobile-app-development/

35. Meet Android Studio. [Електронний ресурс]. URL: https://developer. android.com/studio/intro

36. MVVM Clean Architecture Pattern in Android with Use Cases. [Електронний ресурс]. URL: https://medium.com/@ami0275/mvvm-clean-architecture-pattern-in-android-with-use-cases-eff7edc2ef76

37. Різниця між функціональним і нефункціональним тестуванням. [Електронний ресурс]. URL: https://testlio.com/blog/whats-difference-functional-nonfunctional-testing/

38. Типи та підходи мобільного тестування. [Електронний ресурс]. URL: https://testlio.com/blog/mobile-testing-types-and-approaches/

39. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

40. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

Додаток А

(обов'язковий)

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"19" 09 2022 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту» за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доцент Майданюк В.П.
"19" 09 2023 р.

Виконав:
студент гр. ЗПІ-22м Ярмола В.С.
"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту».

Галузь застосування – контроль фінансів, оптимізація витрат, збереження локацій.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від «18» вересня 2023 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є оптимізація структури витрат за допомогою розробки нових методів та програмних засобів, що базуються на принципах штучного інтелекту і забезпечують геолокацію місць витрат.

Основними задачами дослідження є:

1. Аналіз відомих мобільних додатків для контролю витрат;
2. Розробка алгоритму роботи для програмного засобу оптимізації структури витрат;
3. Вибір програмних засобів для вирішення поставлених завдань;
4. Розробка коду програмного продукту;
5. Тестування роботи програмного продукту.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР:

1. Explain Algorithm and Flowchart with Examples. [Електронний ресурс]. URL: https://www.edrawsoft.com/explain-algorithm-flowchart.html

2. Application architecture. [Електронний ресурс]. URL: https://www.techtarget.com/searchapparchitecture/definition/architecture

3. Типи та підходи мобільного тестування. [Електронний ресурс]. URL: https://testlio.com/blog/mobile-testing-types-and-approaches/

4. Розробка мобільного додатку з геолокацією обліку витрат [Електронний ресурс] / Майданюк В.П., Ярмола В. С. // Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії, Вінниця: Вінницький Національний Технічний Університет. – 2022. URL: https://conferences.vntu.edu.ua/index.php/ all-fitki/all-fitki-022/paper/view/15099

5. Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту [Електронний ресурс] / Майданюк В.П., Ярмола В. С. «Електронні інформаційні ресурси: створення, використання, доступ» // Міжнародна науково-практична інтернет-конференція, Вінниця: Вінницька академія безперервної освіти. – 2023. URL: https://drive.google.com/ file/d/1eAb_QXS6CvA-e1ikKxerTqvw6P1IPRhc/view/

6. Різниця між функціональним і нефункціональним тестуванням. [Електронний ресурс]. URL: https://testlio.com/blog/whats-difference-functional-nonfunctional-testing/

7. How can artificial intelligence help you identify cost savings opportunities? [Електронний ресурс]. URL: https://www.linkedin.com/advice/1/how-can-artificial-intelligence-help-you-identify-cost.

8. Geolocation APIs Choosing the Right Service for Mobile App Development. [Електронний ресурс]. URL: https://utilitiesone.com/geolocation-apis-choosing-the-right-service-for-mobile-app-development

5. Технічні вимоги

Модель розробки – каскадна; шаблон проектування – MVVM; операційна система – Android; середовище розробки –Android Studio; мова програмування Kotlin.

6. Конструктивні вимоги

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

1. Пояснювальна записка до МКР;
2. Технічне завдання;
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Обґрунтування вибору методу розробки та постановка задач	20.09.2023–30.09.2023
2	Розробка методів та алгоритмів реалізації програмних засобів	01.10.2023–10.10.2023
3	Аналіз і вибір мови програмування та середовища розробки	11.10.2023–14.10.2023
4	Розробка програмних засобів для контролю витрат	15.10.2023–14.11.2023
5	Тестування програмного засобу	15.11.2023–18.11.2023
6	Економічна частина	19.11.2023–26.11.2023
7	Оформлення матеріалів до захисту МКР	27.11.2023–01.12.2023

11. Порядок контролю та прийняття

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Методи та програмні засоби оптимізації структури витрат з використанням штучного інтелекту**

Тип роботи: кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, ЗПІ-22м

Науковий керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.

Unicheck	
Оригінальність	96,4 %
Схожість	3,6 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Ярмола В.С.

Керівник роботи _____ Майданюк В.П.

Додаток В

(довідниковий)

Лістинг програмного коду для розробки методів та програмних засобів оптимізації структури витрат

```
package com.pet.project.budget.manager.di.models

import androidx.lifecycle.*
import com.pet.project.budget.manager.di.repositories.ApplicationRepository
import com.pet.project.budget.manager.model.entity.*
import com.pet.project.budget.manager.util.PermissionStatus
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import javax.inject.Inject

class ApplicationViewModel(private val repository: ApplicationRepository) :
    ViewModel() {
    fun getCategories() = repository.getAllCategories()
    fun getAllTransactions() = repository.getAllFullTransactions()
    fun getCurrentUser() = repository.getCurrentUser()
    fun getUsersCount() = repository.getUsersCount()
    fun getCategoriesCount() = repository.getCategoriesCount()

    fun insertBaseCategories(categories: List<Category>) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.insertBaseCategories(categories)
        }
    }

    fun insertCategory(category: Category) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.insertNewCategory(category)
        }
    }

    fun insertTransaction(transaction: Transaction) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.insertNewTransaction(transaction)
        }
    }

    fun insertUser(user: User) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.insertUser(user)
        }
    }

    fun updateUser(user: User) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.updateUser(user)
        }
    }

    private val itemBackground = MutableLiveData<Int>()
    val itemBackgroundLiveData: LiveData<Int> = itemBackground
```

```
    fun setCategoryItemBackground(color: Int) {
        itemBackground.postValue(color)
    }

    private val categoryIconName = MutableLiveData<String>()
    val categoryIconLiveData: LiveData<String> = categoryIconName

    fun setCategoryIconName(iconName: String) {
        categoryIconName.postValue(iconName)
    }

    private val selectedCategoryMutableLiveData = MutableLiveData<Category>()
    val selectedCategoryLiveData: LiveData<Category> =
selectedCategoryMutableLiveData

    fun setSelectedCategory(category: Category) {
        selectedCategoryMutableLiveData.postValue(category)
    }

    fun getAllPlans() = repository.getAllPlans()

    fun insertPlan(plan: Planner) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.insertPlan(plan)
        }
    }

    var exchangeRate = repository.exchangeRate
    fun loadCurrencyFromApi() {
        repository.loadCurrencyFromApi()
    }

    fun getAllPlaces() = repository.getAllPlaces()

    fun insertPlace(place: Place) {
        viewModelScope.launch(Dispatchers.IO) {
            repository.insertPlace(place)
        }
    }

    private val _permissionLiveData = MutableLiveData<PermissionStatus>()
    val permissionLiveData: LiveData<PermissionStatus> = _permissionLiveData

    fun setPermissionStatus(status: PermissionStatus) {
        _permissionLiveData.postValue(status)
    }

    class ViewModelFactory @Inject constructor(private val repository:
ApplicationRepository) :
        ViewModelProvider.Factory {
        @Suppress("UNCHECKED_CAST")
        override fun <T : ViewModel> create(modelClass: Class<T>): T {
            require(modelClass == ApplicationViewModel::class.java)
            return ApplicationViewModel(repository) as T
        }
    }
}


package com.pet.project.budget.manager.di.repositories

import android.util.Log
import androidx.lifecycle.LiveData
```

```
import androidx.lifecycle.MutableLiveData
import com.pet.project.budget.manager.database.dao.*
import com.pet.project.budget.manager.model.entity.*
import com.pet.project.budget.manager.rest.ExchangeRateService
import com.pet.project.budget.manager.rest.model.Rate
import com.pet.project.budget.manager.rest.model.Resource
import io.reactivex.rxjava3.android.schedulers.AndroidSchedulers
import io.reactivex.rxjava3.core.Observer
import io.reactivex.rxjava3.disposables.Disposable
import io.reactivex.rxjava3.schedulers.Schedulers
import java.text.SimpleDateFormat
import java.util.*
import javax.inject.Inject
import com.pet.project.budget.manager.model.entity.Currency as ExchangeCurrency

class ApplicationRepository @Inject constructor(
    private val transactionDao: TransactionDao,
    private val categoryDao: CategoryDao,
    private val userDao: UserDao,
    private val plannerDao: PlannerDao,
    private val fullTransaction: TransactionWithCategoryDao,
    private val exchangeRateService: ExchangeRateService,
    private val placeDao: PlaceDao
) {
    fun getAllFullTransactions() = fullTransaction.getAllTransactions()

    fun getAllCategories() = categoryDao.getAllCategories()
    fun getCategoriesCount() = categoryDao.getCategoriesCount()

    fun getCurrentUser() = userDao.getCurrentUser()
    fun getUsersCount() = userDao.getUsersCount()

    suspend fun insertNewCategory(category: Category) {
        categoryDao.insert(category)
    }

    suspend fun insertNewTransaction(transaction: Transaction) {
        transactionDao.insert(transaction)
    }

    suspend fun insertBaseCategories(categories: List<Category>) {
        categoryDao.insertBaseCategories(categories)
    }

    suspend fun insertUser(user: User) {
        userDao.insert(user)
    }

    suspend fun updateUser(user: User) {
        userDao.update(user)
    }

    fun getAllPlans() = plannerDao.getAllPlans()

    suspend fun insertPlan(plan: Planner) {
        plannerDao.insert(plan)
    }

    private val _exchangeRate = MutableLiveData<Resource>()
    val exchangeRate: LiveData<Resource> = _exchangeRate

    fun loadCurrencyFromApi() {
        exchangeRateService.getHistoricalExchangeRate(previousDay())
            .subscribeOn(Schedulers.io())
```

```
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(getObserverForHistoricalRate())
    }

    private fun getObserverForHistoricalRate() = object : Observer<Rate> {
        override fun onNext(item: Rate) {
            exchangeRateService.getLatestExchangeRate()
                .subscribeOn(Schedulers.io())
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe(getObserverForCurrentRate(item))
        }

        override fun onError(e: Throwable) {
            val message = "Error while loading historical rate: ${e.message}"
            _exchangeRate.postValue(Resource.Error(message))
            Log.d(TAG, message)
        }

        override fun onSubscribe(d: Disposable) {
            Log.d(TAG, "Start loading historical rate")

        }

        override fun onComplete() {
            Log.d(TAG, "Complete loading historical rate")
        }
    }

    private fun getObserverForCurrentRate(historicalRate: Rate) = object :
Observer<Rate> {
        override fun onNext(currentRate: Rate) {
            val list = mutableListOf<ExchangeCurrency>()
            for (value in currentRate.map) {
                val oldRate = historicalRate.map[value.key] ?: 0f
                list.add(
                    ExchangeCurrency(
                        value.key, value.value,
                        value.value - oldRate
                    )
                )
            }
            _exchangeRate.postValue(Resource.Success(list))
        }

        override fun onError(e: Throwable) {
            val message = "Error while loading current rate: ${e.message}"
            _exchangeRate.postValue(Resource.Error(message))
            Log.d(TAG, message)
        }

        override fun onSubscribe(d: Disposable) {
            Log.d(TAG, "Start loading current rate")

        }

        override fun onComplete() {
            Log.d(TAG, "Complete loading current rate")
        }
    }

    fun getAllPlaces() = placeDao.getAllPlaces()

    suspend fun insertPlace(place: Place) {
        placeDao.insert(place)    }
```

```
    private fun previousDay(): String {
        val time = System.currentTimeMillis() - 24 * 60 * 60 * 1000
        val sdf = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())
        return sdf.format(time)
    }

    companion object {
        private const val TAG = "ApplicationRepository"
    }
}
package com.pet.project.budget.manager.model.entity

import android.content.Context
import android.os.Parcelable
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import com.pet.project.budget.manager.R
import com.pet.project.budget.manager.util.getColorById
import com.pet.project.budget.manager.util.getImageName
import kotlinx.parcelize.Parcelize

@Entity(tableName = Category.CATEGORY_TABLE)
@Parcelize
data class Category(
    @PrimaryKey
    @ColumnInfo(name = "title")
    var title: String,
    @ColumnInfo(name = "type")
    var type: TransactionType,
    @ColumnInfo(name = "color")
    var backgroundColor: Int,
    @ColumnInfo(name = "icon_id")
    var iconId: String
) : Parcelable {
    companion object {
        const val CATEGORY_TABLE = "category_table"

        fun baseCategories(context: Context): List<Category> = listOf(
            Category(
                title = context.getString(R.string.food),
                backgroundColor = context.getColorById(R.color.expense_color),
                iconId = context.getImageName(R.drawable.ic_food),
                type = TransactionType.Expense
            ),
         Category(
                title = context.getString(R.string.salary),
                backgroundColor = context.getColorById(R.color.income_color),
                iconId = context.getImageName(R.drawable.ic_money),
                type = TransactionType.Income
            )
        )
    }
}
package com.pet.project.budget.manager.model.entity

import android.os.Parcelable
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import com.google.android.gms.maps.model.LatLng
import kotlinx.parcelize.Parcelize

@Entity(tableName = Place.PLACE_TABLE)
```

```
@Parcelize
data class Place(
    @PrimaryKey(autoGenerate = true)
    val id: Long = 0,
    var latitude: Double,
    var longitude: Double,
    var address: String,
    @ColumnInfo(name = "icon_name")
    var iconName: String,
    @ColumnInfo(name = "icon_color")
    var iconColor: Int
) : Parcelable {
    fun getLocation() = LatLng(latitude, longitude)

    companion object {
        const val PLACE_TABLE = "place_table"
    }
}


package com.pet.project.budget.manager.model.entity

import android.os.Parcelable
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey
import kotlinx.parcelize.Parcelize
import java.util.*

@Entity(tableName = Transaction.TRANSACTION_TABLE)
@Parcelize
data class Transaction(
    @PrimaryKey(autoGenerate = true)
    var id: Long = 0,
    @ColumnInfo(name = "value")
    var value: Double = 0.0,
    @ColumnInfo(name = "date")
    var date: Date,
    @ColumnInfo(name = "type")
    var type: TransactionType,
    @ColumnInfo(name = "category_id")
    var categoryId: String,
    @ColumnInfo(name = "description")
    var description: String,
    @ColumnInfo(name = "location")
    var location: String = ""
) : Parcelable {
    companion object {
        const val TRANSACTION_TABLE = "transaction_table"
    }
}


package com.pet.project.budget.manager.ui.home.main.rate

import android.view.LayoutInflater
import android.view.ViewGroup
import android.widget.TextView
import androidx.core.content.ContextCompat
import androidx.databinding.BindingAdapter
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.pet.project.budget.manager.R
```

```
import com.pet.project.budget.manager.databinding.ExchangeRateListItemBinding
import com.pet.project.budget.manager.model.entity.Currency
import com.pet.project.budget.manager.util.DiffCallBack

class CurrencyAdapter :
RecyclerView.Adapter<CurrencyAdapter.CurrencyViewHolder>() {
    private var items: List<Currency> = emptyList()

    inner class CurrencyViewHolder(private val binding:
ExchangeRateListItemBinding) :
        RecyclerView.ViewHolder(binding.root) {
        fun bind(item: Currency) {
            binding.currency = item
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
CurrencyViewHolder =
        CurrencyViewHolder(
            ExchangeRateListItemBinding.inflate(
                LayoutInflater
                    .from(parent.context), parent, false
            )
        )

    override fun onBindViewHolder(holder: CurrencyViewHolder, position: Int) =
        holder.bind(items[position])

    override fun getItemCount(): Int = items.size

    fun updateList(list: List<Currency>) {
        val diffCallback = DiffCallBack(this.items, list)
        val diffResult = DiffUtil.calculateDiff(diffCallback)
        diffResult.dispatchUpdatesTo(this)
        items = list
    }
}

@BindingAdapter("android:exchangeColor")
fun TextView.setColorOfExchangeText(value: Float) {
    if (value > 0f)
        setTextColor(
            ContextCompat.getColor(
                context,
                R.color.income_color
            )
        )
    else if (value < 0f)
        setTextColor(
            ContextCompat.getColor(
                context,
                R.color.expense_color
            )
        )
}
package com.pet.project.budget.manager.ui.home.main

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.fragment.app.Fragment
import androidx.fragment.app.activityViewModels
import by.kirich1409.viewbindingdelegate.viewBinding
import com.pet.project.budget.manager.R
```

```
import com.pet.project.budget.manager.databinding.FragmentHomeBinding
import com.pet.project.budget.manager.di.models.ApplicationViewModel
import com.pet.project.budget.manager.ui.home.main.rate.ExchangeRateActivity

class HomeFragment : Fragment(R.layout.fragment_home) {
    private val viewBinding by viewBinding(FragmentHomeBinding::bind)
    private val viewModel: ApplicationViewModel by activityViewModels()
    override fun onStart() {
        super.onStart()
        viewModel.getCurrentUser().observe(viewLifecycleOwner) { user ->
            viewBinding.user = user
        }
        viewBinding.transactionList.adapter = TransactionListAdapter()
        viewModel.getAllTransactions().observe(viewLifecycleOwner) {
transactionList ->
            (viewBinding.transactionList.adapter as
TransactionListAdapter).updateList(
                transactionList
            )
        }
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        with(viewBinding) {
            rateBackground.setOnClickListener {
                startActivity(Intent(requireActivity(),
ExchangeRateActivity::class.java))
            }
        }
    }
}


package com.pet.project.budget.manager.ui.home.planner.add

import android.app.DatePickerDialog
import android.app.TimePickerDialog
import android.os.Bundle
import androidx.activity.viewModels
import androidx.appcompat.app.AppCompatActivity
import androidx.work.Data
import androidx.work.ExistingWorkPolicy
import androidx.work.OneTimeWorkRequest
import androidx.work.WorkManager
import by.kirich1409.viewbindingdelegate.viewBinding
import com.pet.project.budget.manager.R
import com.pet.project.budget.manager.appComponent
import com.pet.project.budget.manager.databinding.ActivityAddNewPlanBinding
import com.pet.project.budget.manager.di.models.ApplicationViewModel
import com.pet.project.budget.manager.model.entity.Planner
import com.pet.project.budget.manager.util.NotificationWorker
import com.pet.project.budget.manager.util.showToastLong
import dagger.Lazy
import java.text.SimpleDateFormat
import java.util.*
import java.util.concurrent.TimeUnit
import javax.inject.Inject

class AddNewPlanActivity : AppCompatActivity(R.layout.activity_add_new_plan) {
    private val binding by viewBinding(ActivityAddNewPlanBinding::bind)

    @Inject
```

```
    internal lateinit var viewModelFactory:
Lazy<ApplicationViewModel.ViewModelFactory>
    private val viewModel: ApplicationViewModel by viewModels {
        viewModelFactory.get()
    }
    private val calendar = Calendar.getInstance()
    private val dateFormatter = SimpleDateFormat("dd MMMM yyyy",
Locale.getDefault())
    private val timeFormatter = SimpleDateFormat("HH:mm", Locale.getDefault())

    override fun onCreate(savedInstanceState: Bundle?) {
        appComponent.inject(this)
        super.onCreate(savedInstanceState)
        initializeAppBar()
        setCurrentDateAndTimeInLayouts()
        actionOnDateSetButton()
        actionOnTimeSetButton()
        setOnCreateReminderButton()
        setContentView(binding.root)
    }

    @Suppress("kotlin:S1151")
    private fun setOnCreateReminderButton() {
        binding.plannerCreateNotification.setOnClickListener {
            val plannerText = binding.plannerTitleInput.text.toString()
            when (plannerText.isNotEmpty()) {
                true -> {
                    viewModel.insertPlan(
                        Planner(
                            title = plannerText,
                            date = calendar.time
                        )
                    )
                    val notificationData = Data.Builder()
                    notificationData.putString(
                        NotificationWorker.NOTIFICATION_MESSAGE_KEY,
                        plannerText
                    )
                    val delay = calendar.timeInMillis -
System.currentTimeMillis()
                    val workManger = WorkManager.getInstance(applicationContext)

workManger.cancelAllWorkByTag(NotificationWorker.NOTIFICATION_TAG)
                    val workRequest =
OneTimeWorkRequest.Builder(NotificationWorker::class.java)
                        .setInitialDelay(delay, TimeUnit.MILLISECONDS)
                        .setInputData(notificationData.build())
                        .addTag(NotificationWorker.NOTIFICATION_TAG +
plannerText)
                        .build()
                    workManger.enqueueUniqueWork(
                        NotificationWorker.NOTIFICATION_TAG + plannerText,
                        ExistingWorkPolicy.APPEND,
                        workRequest
                    )
                    onBackPressed()
                }
                false ->
applicationContext.showToastLong(getString(R.string.planner_text_empty))
            }
        }
    }

    private fun setCurrentDateAndTimeInLayouts() {
```

```
        binding.dateSelectorInput.setText(dateFormatter.format(calendar.time))
        binding.timeSelectorInput.setText(timeFormatter.format(calendar.time))
    }

    private fun initializeAppBar() {
        binding.appBarLayout.toolbar.apply {
            setNavigationIcon(R.drawable.ic_arrow_back)
            setTitle(R.string.add_new_plan)
            setNavigationOnClickListener {
                onBackPressed()
            }
        }
    }

    private fun actionOnDateSetButton() {
        binding.dateSelectorInput.setOnClickListener {
            DatePickerDialog(
                this@AddNewPlanActivity,
                createDataPicker(),
                calendar.get(Calendar.YEAR),
                calendar.get(Calendar.MONTH),
                calendar.get(Calendar.DAY_OF_MONTH)
            ).show()
        }
    }

    private fun actionOnTimeSetButton() {
        binding.timeSelectorInput.setOnClickListener {
            TimePickerDialog(
                this@AddNewPlanActivity,
                createTimePicker(),
                calendar.get(Calendar.HOUR_OF_DAY),
                calendar.get(Calendar.MINUTE),
                true
            ).show()
        }
    }

    private fun createDataPicker() =
        DatePickerDialog.OnDateSetListener { _, year, monthOfYear, dayOfMonth ->
            calendar.set(Calendar.YEAR, year)
            calendar.set(Calendar.MONTH, monthOfYear)
            calendar.set(Calendar.DAY_OF_MONTH, dayOfMonth)

binding.dateSelectorInput.setText(dateFormatter.format(calendar.time))
        }

    private fun createTimePicker() =
        TimePickerDialog.OnTimeSetListener { _, hour, minute ->
            calendar.set(Calendar.HOUR_OF_DAY, hour)
            calendar.set(Calendar.MINUTE, minute)
            calendar.set(Calendar.SECOND, 0)

binding.timeSelectorInput.setText(timeFormatter.format(calendar.time))
        }
}

package com.pet.project.budget.manager.util

import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Context
```

```
import android.content.Intent
import android.media.RingtoneManager
import android.os.Build
import androidx.core.app.NotificationCompat
import androidx.core.app.NotificationManagerCompat
import androidx.work.CoroutineWorker
import androidx.work.WorkerParameters
import com.pet.project.budget.manager.R
import com.pet.project.budget.manager.ui.home.HomeActivity
import kotlinx.coroutines.coroutineScope

class NotificationWorker(private val context: Context, parameters:
WorkerParameters) :
    CoroutineWorker(context, parameters) {

    override suspend fun doWork(): Result = coroutineScope{
        createNotificationChannel()
        val notificationMessage = inputData.getString(NOTIFICATION_MESSAGE_KEY)
?: ""
        with(NotificationManagerCompat.from(applicationContext)) {
            notify(NOTIFICATION_ID, createNotification(notificationMessage))
        }
        Result.success()
    }

    private fun createNotification(message: String): Notification {
        val mainActivity = Intent(applicationContext,
HomeActivity::class.java).apply {
            flags = Intent.FLAG_ACTIVITY_CLEAR_TOP or
Intent.FLAG_ACTIVITY_CLEAR_TOP
        }
        val pendingIntent = PendingIntent.getActivity(
            context, 0, mainActivity,
            if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M)
                PendingIntent.FLAG_IMMUTABLE
            else PendingIntent.FLAG_UPDATE_CURRENT
        )
        val defaultSoundUri =
RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION)
        val title = context.getString(R.string.budget_manager_reminder)
        return NotificationCompat.Builder(applicationContext, CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_notifications)
            .setContentTitle(title)
            .setContentText(message)
            .setAutoCancel(true)
            .setSound(defaultSoundUri)
            .setContentIntent(pendingIntent).build()
    }

    private fun createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            val name = context.getString(R.string.channel_name)
            val descriptionText =
context.getString(R.string.channel_description)
            val importance = NotificationManager.IMPORTANCE_DEFAULT
            val channel = NotificationChannel(
                CHANNEL_ID,
                name,
                importance
            ).apply {
                description = descriptionText
            }

            val notificationManager =
```

```
                context.getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager
            notificationManager.createNotificationChannel(channel)
        }
    }

    companion object {
        private const val CHANNEL_ID = "com.pet.project.budget.manager"
        private const val NOTIFICATION_ID = 121
        const val NOTIFICATION_MESSAGE_KEY = "Worker notification message key"
        const val NOTIFICATION_TAG = "Work manager task: "   }}
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".ui.home.planner.add.AddNewPlanActivity">

    <androidx.coordinatorlayout.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <include
            android:id="@+id/appBarLayout"
            layout="@layout/app_toolbar" />

        <androidx.core.widget.NestedScrollView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <com.google.android.material.textfield.TextInputLayout
                    style="@style/Widget.TextInputLayout.Design"
                    android:hint="@string/title">

                    <com.google.android.material.textfield.TextInputEditText
                        android:id="@+id/planner_title_input"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:inputType="text"
                        tools:ignore="TextContrastCheck" />

                </com.google.android.material.textfield.TextInputLayout>

                <com.google.android.material.textfield.TextInputLayout
                    android:id="@+id/date_selector_field"
                    style="@style/Widget.TextInputLayout.Design"
                    android:hint="@string/transaction_date"
                    app:hintAnimationEnabled="false">

                    <com.google.android.material.textfield.TextInputEditText
                        android:id="@+id/date_selector_input"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:cursorVisible="false"
                        android:drawableEnd="@drawable/ic_today"
                        android:focusableInTouchMode="false"
                        android:inputType="text"
                        tools:ignore="TextContrastCheck" />
```

```
                </com.google.android.material.textfield.TextInputLayout>

                <com.google.android.material.textfield.TextInputLayout
                    android:id="@+id/time_selector_field"
                    style="@style/Widget.TextInputLayout.Design"
                    android:hint="@string/transaction_time"
                    app:hintAnimationEnabled="false">

                    <com.google.android.material.textfield.TextInputEditText
                        android:id="@+id/time_selector_input"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:cursorVisible="false"
                        android:drawableEnd="@drawable/ic_clock"
                        android:focusableInTouchMode="false"
                        android:inputType="text"
                        tools:ignore="TextContrastCheck" />

                </com.google.android.material.textfield.TextInputLayout>

                <com.google.android.material.button.MaterialButton
                    android:id="@+id/planner_create_notification"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="end"
                    android:layout_marginEnd="16dp"
                    android:layout_marginBottom="16dp"
                    android:backgroundTint="@color/background"
                    android:text="@string/create_reminder"
                    android:textColor="@color/white"
                    tools:ignore="TextContrastCheck" />
            </LinearLayout>

        </androidx.core.widget.NestedScrollView>
    </androidx.coordinatorlayout.widget.CoordinatorLayout>
</layout>

<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools">

    <data>

        <variable
            name="category"
            type="com.pet.project.budget.manager.model.entity.Category" />
    </data>

    <androidx.constraintlayout.widget.ConstraintLayout
style="@style/Layout.ListItem.SingLine">

        <View
            android:id="@+id/image_background"
            style="@style/Layout.ListItem.Image"
            android:background="@drawable/ic_round_grey"
            viewBackground="@{category.backgroundColor}"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <ImageView
            android:id="@+id/category_item_image"
            android:layout_width="24dp"
```

```
            android:layout_height="24dp"
            android:contentDescription="@string/category_icon"
            imageUrl="@{category.iconId}"
            android:src="@drawable/ic_money"
            app:layout_constraintBottom_toBottomOf="@id/image_background"
            app:layout_constraintEnd_toEndOf="@id/image_background"
            app:layout_constraintStart_toStartOf="@id/image_background"
            app:layout_constraintTop_toTopOf="@id/image_background"
            tools:ignore="ImageContrastCheck" />

        <TextView
            android:id="@+id/category_item_title_text"
            style="@style/Text.ListItem.Primary.OneLine"
            android:layout_width="0dp"
            android:layout_marginStart="16dp"
            android:layout_marginEnd="8dp"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toEndOf="@id/image_background"
            app:layout_constraintTop_toTopOf="parent"
            tools:text="@{category.title, default=@string/category_title}" />
    </androidx.constraintlayout.widget.ConstraintLayout>
</layout>

<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context=".ui.transaction.add.AddTransactionFragment">

    <androidx.coordinatorlayout.widget.CoordinatorLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <include
            android:id="@+id/barContainer"
            layout="@layout/app_toolbar" />

        <androidx.core.widget.NestedScrollView
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            app:layout_behavior="@string/appbar_scrolling_view_behavior">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:orientation="vertical">

                <com.google.android.material.textfield.TextInputLayout
                    style="@style/Widget.TextInputLayout"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_marginStart="16dp"
                    android:layout_marginTop="12dp"
                    android:layout_marginEnd="16dp"
                    android:hint="@string/hint_value">

                    <com.google.android.material.textfield.TextInputEditText
                        android:id="@+id/transition_value_input"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:inputType="numberDecimal"
                        android:maxLines="1"
                        tools:ignore="TextContrastCheck" />
```

```
</com.google.android.material.textfield.TextInputLayout>

<TextView
    style="@style/Text.ListItem.Primary"
    android:layout_marginStart="16dp"
    android:text="@string/transaction_type" />

<include
    android:id="@+id/chip_group"
    layout="@layout/view_transaction_types" />

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/date_selector_field"
    style="@style/Widget.TextInputLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="16dp"
    android:hint="@string/transaction_date"
    app:hintAnimationEnabled="false">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/date_selector_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:cursorVisible="false"
        android:drawableEnd="@drawable/ic_today"
        android:focusableInTouchMode="false"
        android:inputType="text"
        tools:ignore="TextContrastCheck" />

</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    style="@style/Widget.TextInputLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="16dp"
    android:hint="@string/description">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/transition_description_input"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="text"
        tools:ignore="TextContrastCheck" />

</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/transition_category_field"
    style="@style/Widget.TextInputLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="16dp"
    android:layout_marginBottom="8dp"
    android:hint="@string/category"
    app:hintAnimationEnabled="false">
```

```
                    <com.google.android.material.textfield.TextInputEditText
                        android:id="@+id/transition_category_input"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:cursorVisible="false"
                        android:drawableEnd="@drawable/ic_arrow_right"
                        android:focusableInTouchMode="false"
                        android:inputType="text"
                        tools:ignore="TextContrastCheck" />
                </com.google.android.material.textfield.TextInputLayout>

                <com.google.android.material.textfield.TextInputLayout
                    android:id="@+id/transition_map_field"
                    style="@style/Widget.TextInputLayout"
                    android:layout_width="match_parent"
                    android:layout_height="wrap_content"
                    android:layout_marginStart="16dp"
                    android:layout_marginTop="8dp"
                    android:layout_marginEnd="16dp"
                    android:layout_marginBottom="8dp"
                    android:hint="@string/select_location"
                    app:endIconDrawable="@drawable/ic_map"
                    app:endIconMode="custom"
                    app:hintAnimationEnabled="false">

                    <com.google.android.material.textfield.TextInputEditText
                        android:id="@+id/transition_map_input"
                        android:layout_width="match_parent"
                        android:layout_height="wrap_content"
                        android:inputType="text"
                        tools:ignore="TextContrastCheck" />
                </com.google.android.material.textfield.TextInputLayout>

                <com.google.android.material.button.MaterialButton
                    android:id="@+id/transaction_save_changes_button"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_gravity="end"
                    android:layout_marginEnd="16dp"
                    android:layout_marginBottom="16dp"
                    android:backgroundTint="@color/background"
                    android:text="@string/save"
                    android:textColor="@color/white"
                    tools:ignore="TextContrastCheck" />
            </LinearLayout>

        </androidx.core.widget.NestedScrollView>

    </androidx.coordinatorlayout.widget.CoordinatorLayout>
</layout>
```

Додаток Г

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОДИ ТА ПРОГРАМНІ ЗАСОБИ ОПТИМІЗАЦІЇ СТРУКТУРИ ВИТРАТ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

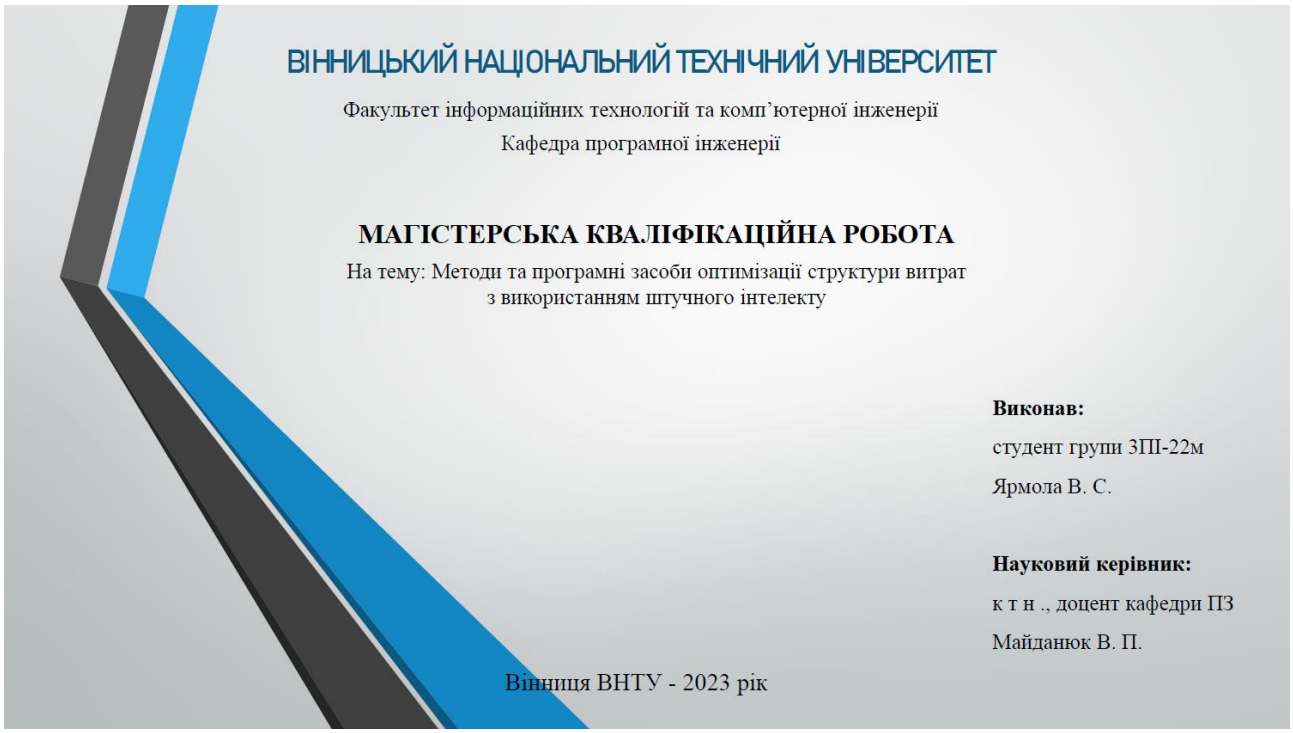

Рисунок Г.1 – Слайд №1

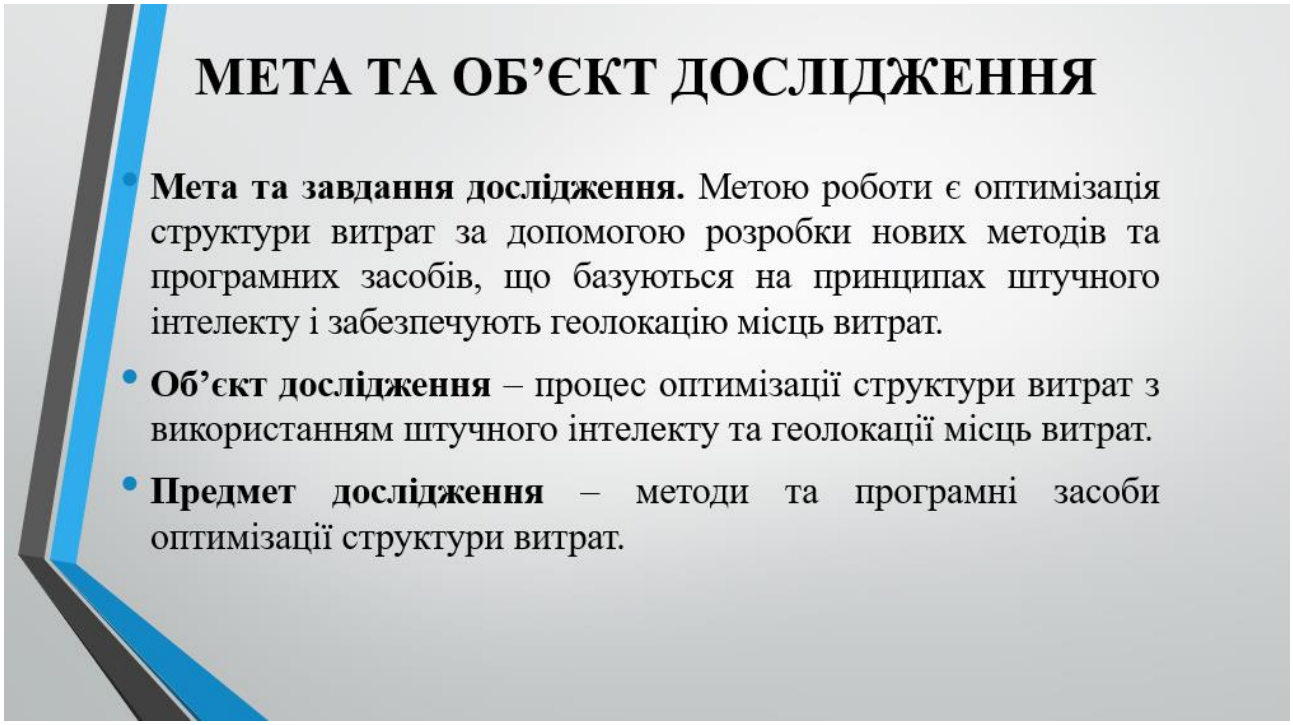

Рисунок Г.2 – Слайд №2

Наукова новизна отриманих результатів.

- Вперше запропоновано метод оптимізації структури витрат, особливість якого полягає у використанні сервісів штучного інтелекту, що дозволяє підвищити точність прогнозування та збільшити ефективність процесу оптимального розподілу ресурсів.
- Подальшого розвитку отримав метод обліку витрат, у який на відміну від існуючих, інтегровано функції геолокації, що розширює можливості для аналізу даних щодо витрат і полегшує користувачам процес вибору способів оптимізації витрат.

Практична цінність отриманих результатів.

- Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено мовою програмування Kotlin мобільні програмні засоби для оптимізації структури витрат з інтеграцією сервісів штучного інтелекту та геолокації місць витрат.

Рисунок Г.3 – Слайд №3

РОЗРОБКА МЕТОДУ ОПТИМІЗАЦІЇ ВИТРАТ ЗА ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ

Для того щоб програмний засіб міг аналізувати дані про доходи та витрати користувача за допомогою штучного інтелекту та рекомендувати можливі шляхи для економії чи оптимізації витрат, було розроблено наступний метод:

1) Збір і підготовка даних;

2) Аналіз даних із застосуванням ШІ;

3) Генерація рекомендацій;

4) Повернення інформації до додатку;

5) Здійснення дій на основі аналізу.

Ключовим фактором успіху методу є здатність алгоритмів ШІ точно аналізувати витрати та підраховувати ефективні, призначені для користувача рекомендації, які враховують їхні індивідуальні фінансові умови та цілі.

Рисунок Г.4 – Слайд №4

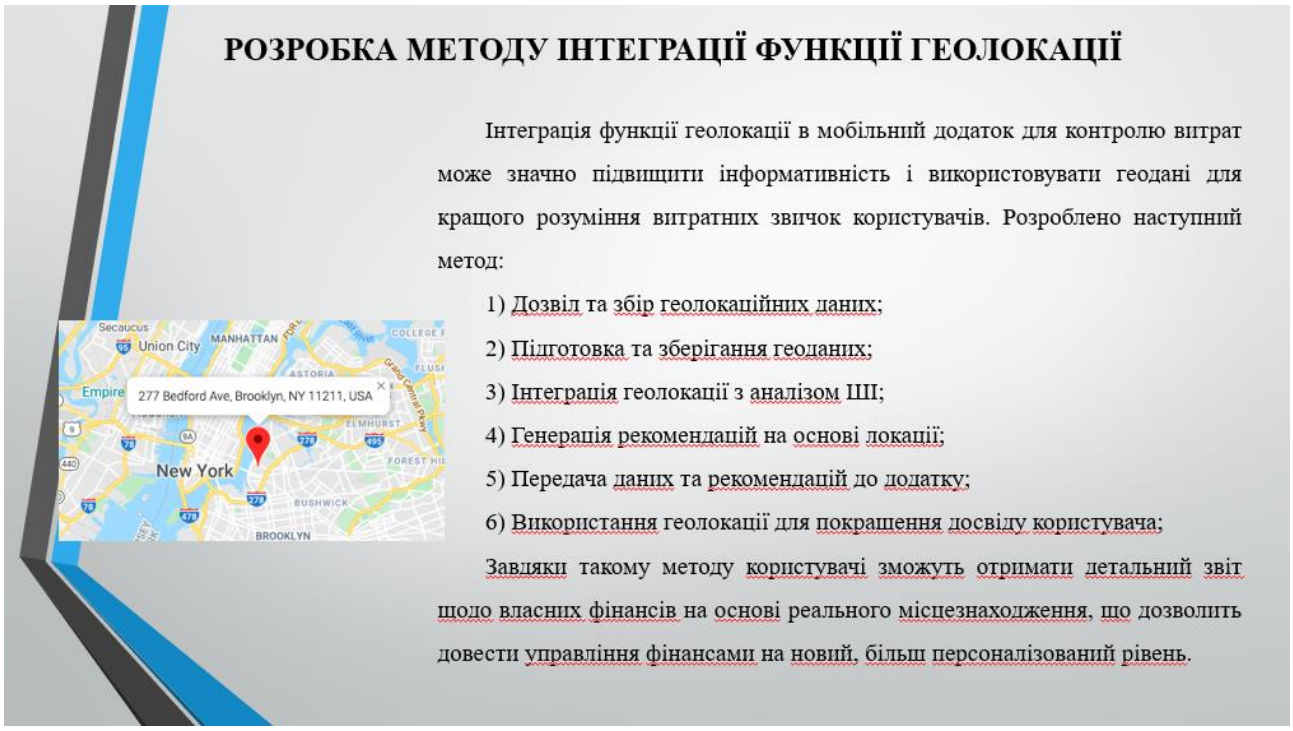

Рисунок Г.5 – Слайд №5

Рисунок Г.6 – Слайд №6

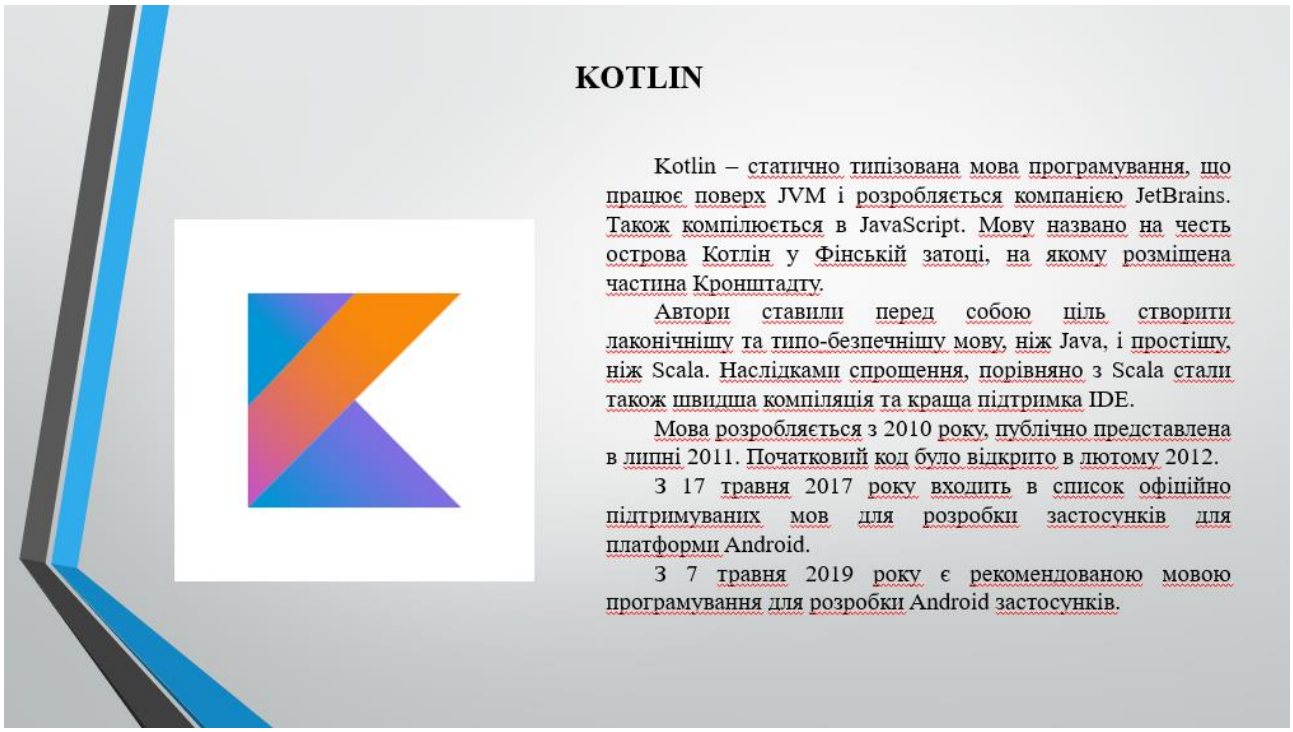

Рисунок Г.7 – Слайд №7

Рисунок Г.8 – Слайд №8

Рисунок Г.9 – Слайд №9

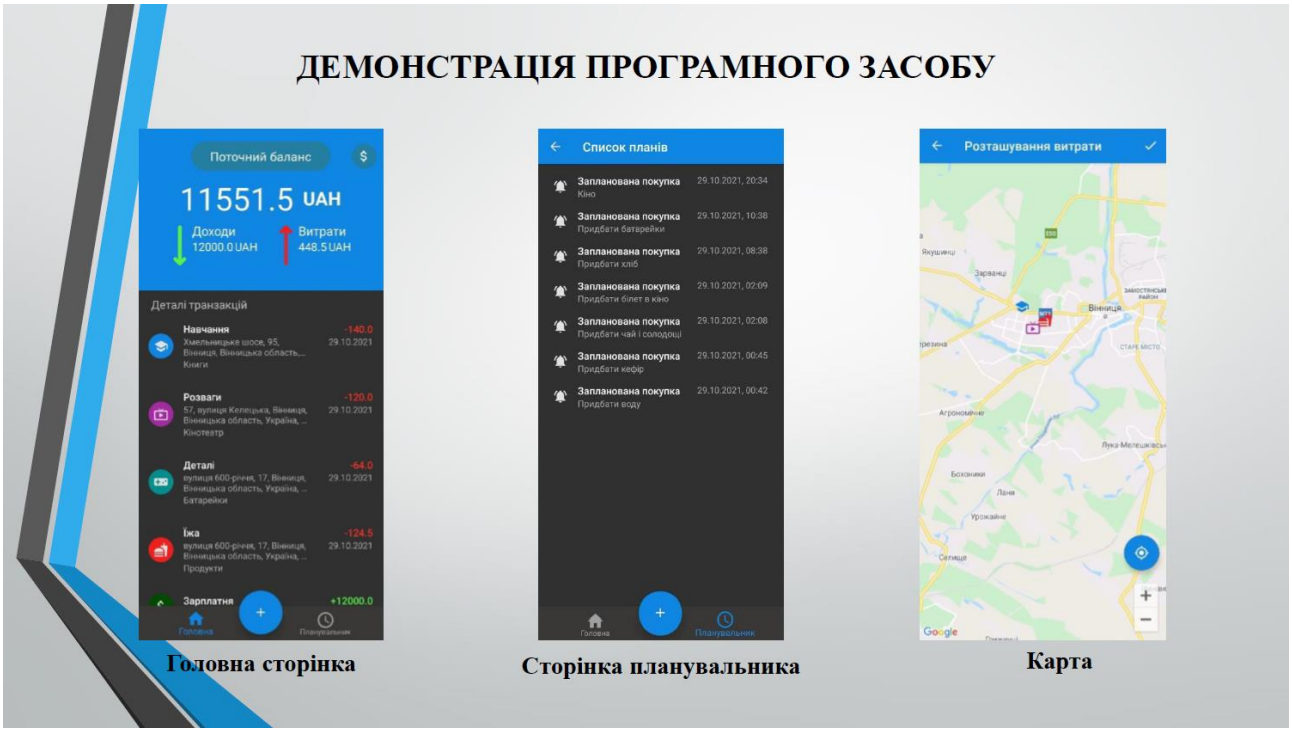

Рисунок Г.10 – Слайд №10

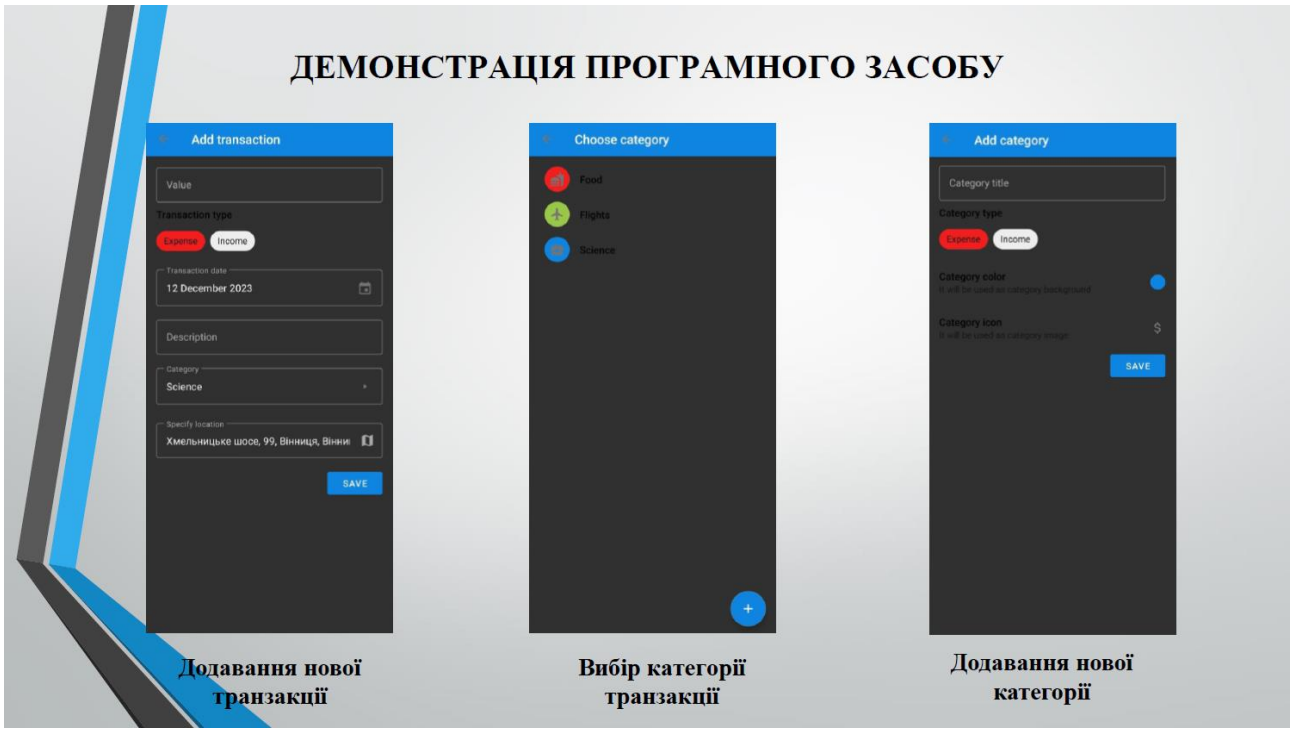

Рисунок Г.11 – Слайд №11

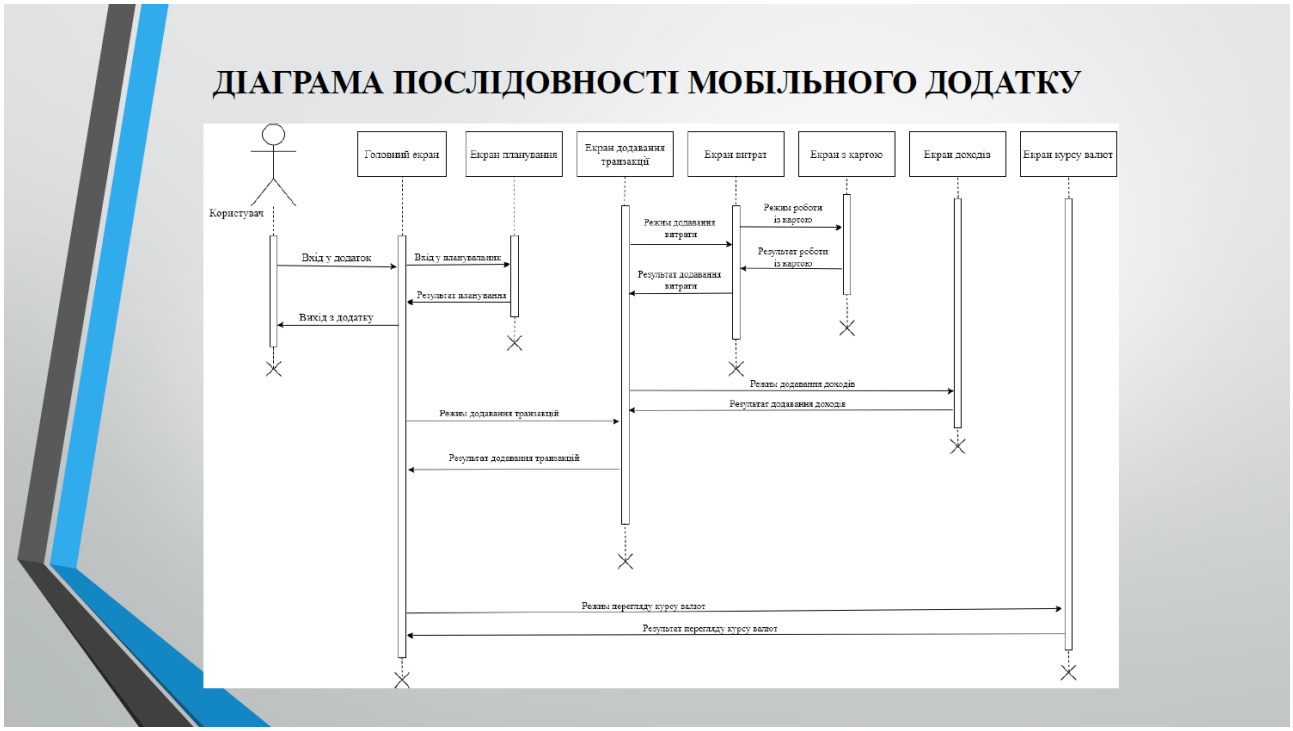

Рисунок Г.12 – Слайд №12

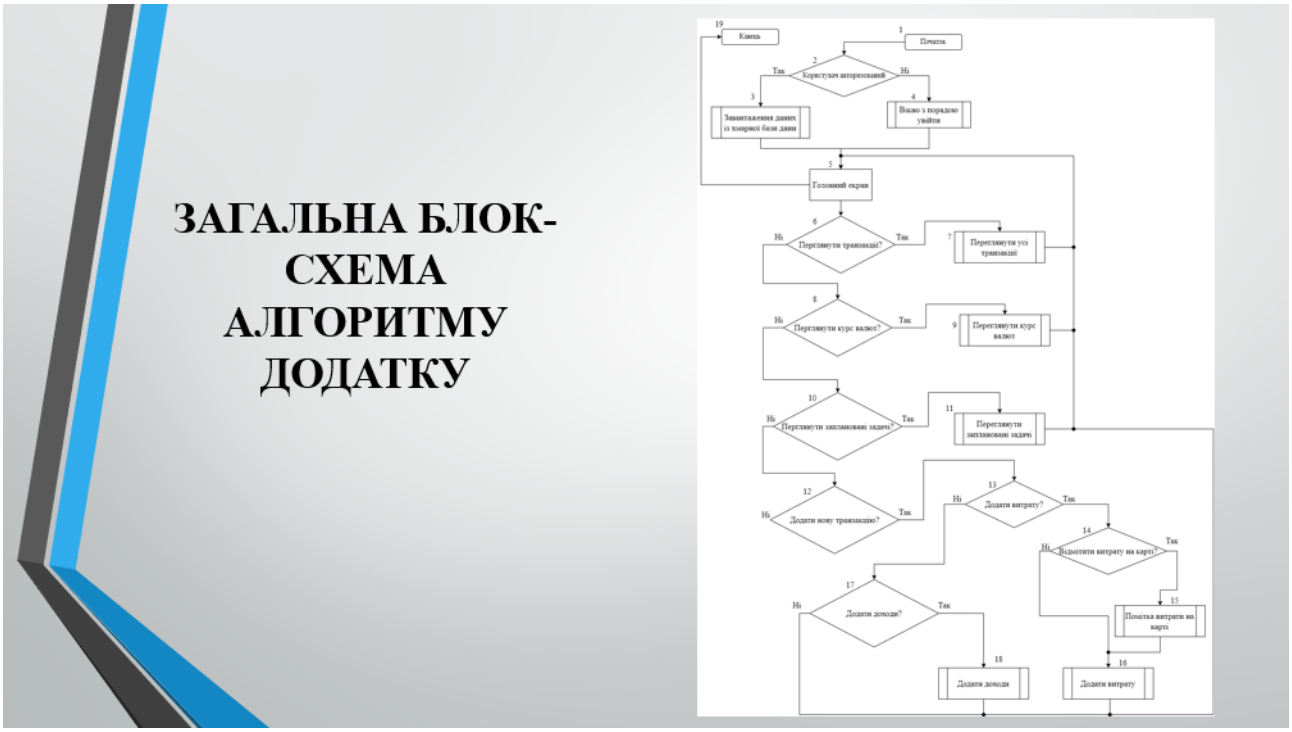

Рисунок Г.13 – Слайд №13

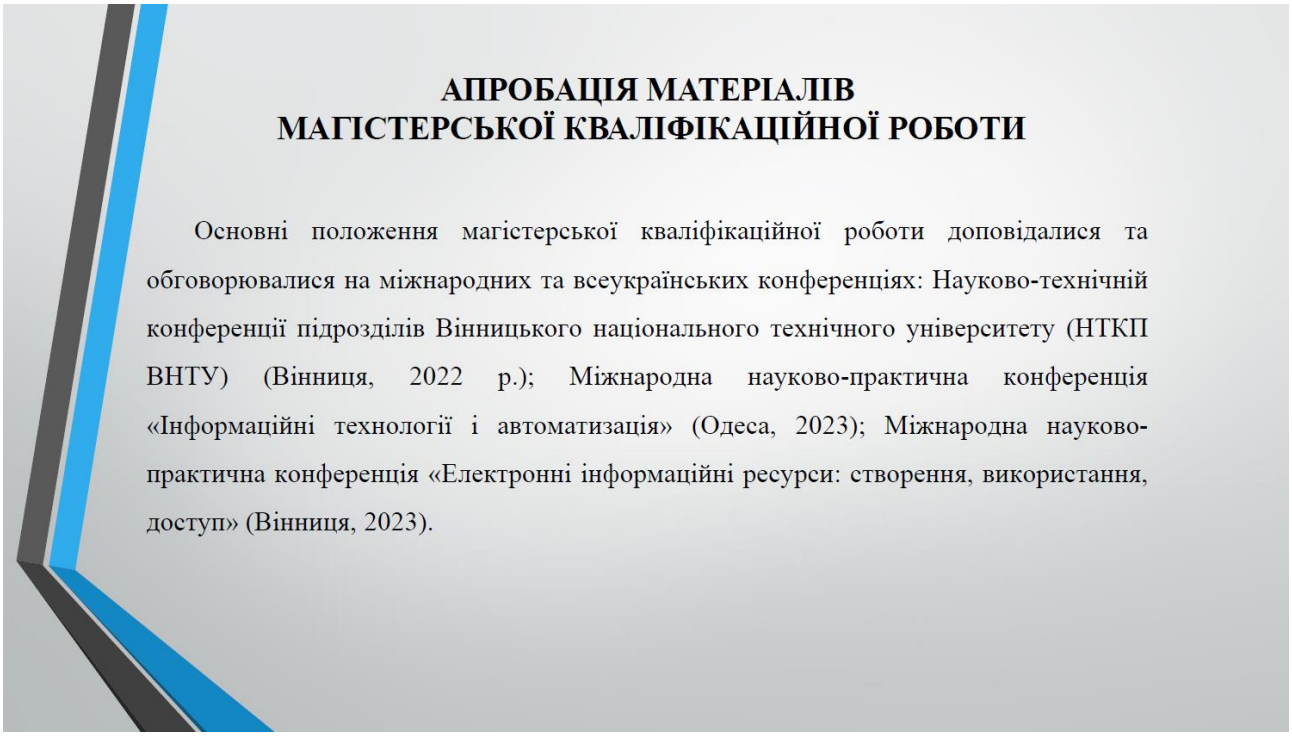

Рисунок Г.14 – Слайд №14

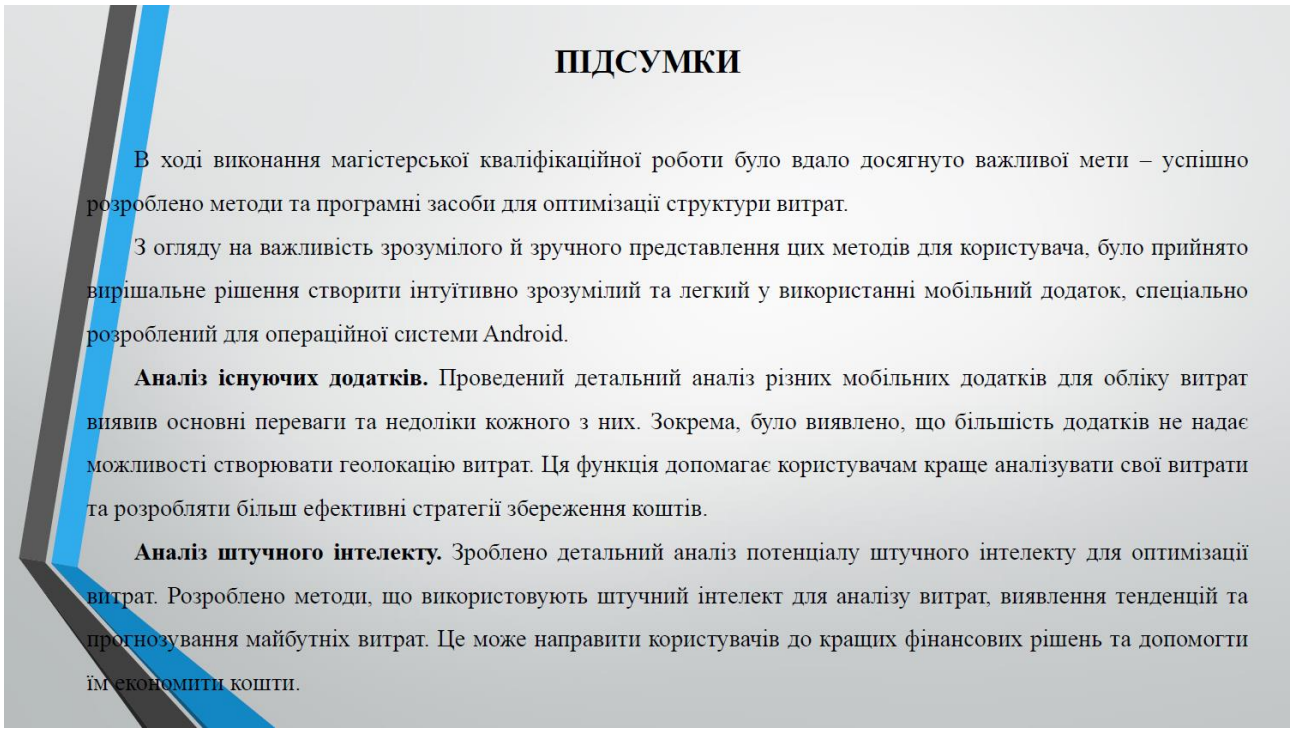

Рисунок Г.15 – Слайд №15