Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

Бакалаврська кваліфікаційна робота на тему:

«РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ З ГЕОЛОКАЦІЄЮ ОБЛІКУ ВИТРАТ»

Виконав: студент 4 курсу, групи 2ПІ-18б спеціальності 121 – Інженерії програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Ярмола В. С.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри ПЗ
Майданюк В. П.
(прізвище та ініціали)
«____»_____2022 р.

Рецензент: к.т.н., доцент кафедри КН
Колесницький О. К.
(прізвище та ініціали)
«____»_____2022 р.

Допущено до захисту
Зав. кафедри ПЗ Романюк О. Н.
«___»_____2022р.

Вінниця ВНТУ – 2022 рік

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
25 березня 2022 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Ярмолі Віталію Сергійовичу

1. Тема роботи – «Розробка мобільного додатку з геолокацією обліку витрат»
Керівник роботи: Майданюк Володимир Павлович, к. т. н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 24 березня 2022 р. № 66

2. Термін подання студентом роботи : 13 червня 2022 р.

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

4.Зміст розрахунково-пояснювальної записки: вступ; обґрунтування вибору методу розробки та постановка задач; розробка архітектури та алгоритмів мобільного додатку з геолокацією обліку витрат; розробка мобільного додатку з геолокацією обліку витрат; тестування мобільного додатку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): титульний слайд; мета, об'єкт та предмет дослідження; актуальність розробки мобільного додатку з геолокацією обліку витрат; реалізовані функції у додатку; технології; firebase database; діаграма послідовності мобільного додатку; схематична архітектура додатку; шаблон проектування; демонстрація додатку; підсумки; фінальний слайд.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	завдання прийняв
1–4	к.т.н., доц. каф ПЗ Майданюк В.П.		

7. Дата видачі завдання 25 березня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту	Строк виконання етапів роботи	Примітка
1	Обґрунтування вибору методу розробки та постановка задач	26.03.22 – 01.04.22	Виконано
2	Розробка архітектури та алгоритмів програмного продукту	02.04.22 – 07.04.22	Виконано
3	Аналіз і вибір мови програмування та середовища розробки	08.04.22 – 11.04.22	Виконано
4	Розробка програмного продукту	12.04.22 – 01.05.22	Виконано
5	Тестування програми	02.05.22 – 10.05.22	Виконано
6	Оформлення матеріалів до захисту БДР	11.05.22 – 10.06.22	Виконано

Студент _____ **Ярмола В. С.**

(підпис) (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ **Майданюк В. П.**

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 84 сторінок формату А4, на яких є 26 рисунків та 1 таблиця, список використаних джерел містить 15 найменувань.

Метою бакалаврської дипломної роботи є підвищення точності обліку фінансів споживачем за рахунок можливості позначення на карті місць витрат.

Запропоновано метод обліку витрат, особливість якого полягає у визначенні місця розташування об’єктів, де виконувались витрати і позначенні їх на карті, що збільшує обсяг даних для аналізу витрат і таким чином полегшує користувачам вибір шляхів оптимізації витрат.

У розділі обґрунтування вибору методу розробки та постановка задач було здійснено аналіз предметної області, проведено аналіз аналогів, обрано модель життєвого циклу для розробки додатку.

У розділі розробка архітектури та алгоритмів мобільного додатку з геолокацією обліку витрат розроблено загальну архітектуру додатку, розроблено базу даних та блок схему алгоритму роботи програми.

У розділі розробка мобільного додатку з геолокацією обліку витрат здійснено аналіз та обґрунтування вибору програмних засобів для розроби, обрано середовище розробки, наведено приклади програмної реалізації.

У розділі тестування мобільного додатку обґрунтовано вибір методики тестування. Для підтвердження ефективності та правильності функціонування розробленого мобільного додатку проведено ручне та автоматизоване тестування. Розроблено інструкцію користувача.

Ключові слова: Android, Android Studio, Kotlin, геолокація, мобільний додаток, програма.

ANNOTATION

The bachelor's graduate thesis consists of 84 A4 pages, which have 26 figures and 1 table, the list of sources used contains 15 titles.

The purpose of the bachelor's thesis is to increase the accuracy of financial accounting by the consumer due to the possibility of marking on the map of places of expenditure.

A cost accounting method has been proposed, which features the location of objects where costs were incurred and marked on a map, which increases the amount of data for cost analysis and thus makes it easier for users to choose ways to optimize costs.

In the section of substantiation of the choice of development method and statement of tasks the analysis of the subject area was carried out, the analysis of analogues was carried out, the model of a life cycle for application development was chosen.

In the section development of architecture and algorithms of the mobile application with geolocation of cost accounting the general architecture of the application is developed, the database and the block diagram of the algorithm of work of the program are developed.

In the section development of a mobile application with geolocation of cost accounting the analysis and substantiation of the choice of software for development is carried out, the development environment is chosen, examples of software implementation are given.

In the section of testing the mobile application, the choice of testing method is substantiated. Manual and automated testing was conducted to confirm the effectiveness and correct operation of the developed mobile application. User manual developed.

Keywords: Android, Android Studio, application, Kotlin, geolocation, mobile application.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Контроль витрат – це метод зниження витрат бізнесу шляхом управління та аналізу фінансових даних. Збір витрат у консолідованому форматі дозволяє організаціям робити більш точні та обґрунтовані прогнози, знати, де можна мінімізувати витрати, і визначати сфери перевитрат.

Ось чому контроль витрат і управління постачальниками взаємопов’язані, оскільки оптимізація взаємодії з постачальниками може призвести до значної економії витрат для бізнесу.

Зрештою, мета контролю витрат – надати структуру, яка призначена для покращення видимості та контролю над витратами.

Управління витратами й контроль – це сфера, що вимагає багато даних та аналізу, і тому не дивно, чому розробники програмного забезпечення знайшли тут застосування для автоматизації. Програмне керування оптимізує процес кількома способами: стомливі обчислення та інші кількісні завдання виконуються майже миттєво й захищені від людських помилок за допомогою програмного забезпечення. Аналіз набагато простіше, коли дані чітко подаються через інформаційні панелі, прості у використанні і представлені у зручному інтерфейсі користувача. Таким чином стандартизація даних відбувається автоматично, що може бути корисно для проєктів у міжнародному масштабі, які мають використовувати декілька валют. Інтеграція сторонніх розробників з іншими рішеннями для управління допомагає ефективніше об’єднувати дані [1].

Цілі обліку витрат повністю відповідають цілям особи, групам осіб чи компанії, оскільки обидві спрямовані на зниження витрат і підвищення прибутковості та конкурентоспроможності компанії, мабуть, найважливішими з них є наступні:

1. Точне визначення механізмів збору, обліку і класифікації витрат.
2. Контроль над елементами витрат.

3. Допомога у прийнятті рішень за допомогою відповідної звітності про витрати.

4. Планування на майбутнє за допомогою обліку витрат, що забезпечує складання бюджету.

5. Ціноутворення на продукти та визначення цільової калькуляції витрат, а потім максимізації прибутку.

Однак, відомі програмні рішення обліку витрат мало уваги приділяють питанням геолокації витрат, тобто визначення розташування об'єктів, де витрачались кошти. Геолокація особливо важлива для звичайних споживачів, які бажають контролювати свої витрати, оскільки вони часто виконують безсистемні витрати в різних місцях по мірі необхідності. Крім того, важливим є реалізація програмного продукту контролю витрат як мобільного додатку, оскільки це дає можливість контролю в реальному часі і відповідні технічні засоби є у кожної людини.

Тому тема роботи «Розробка мобільного додатку з геолокацією обліку витрат» є вкрай актуальною.

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

Мета та завдання дослідження. Метою роботи є підвищення точності обліку фінансів споживачем за рахунок можливості позначення на карті місць витрат.

Основними задачами дослідження є:

- аналіз відомих мобільних додатків для контролю витрат;
- розробка алгоритму роботи мобільного додатку з геолокацією обліку витрат;
- вибір програмних засобів для вирішення поставлених завдань;
- розробка коду програмного продукту;
- тестування роботи програмного продукту.

Об'єкт дослідження – процес обліку витрат з можливістю позначення на карті місць витрат.

Предмет дослідження – методи та мобільні програмні засоби з геолокацією обліку 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 обліку витрат, особливість якого полягає у визначенні місця розташування об'єктів, де виконувались витрати і позначенні їх на карті, що збільшує обсяг даних для аналізу витрат і таким чином полегшує користувачам а вибір шляхів оптимізації витрат.
2. Подальшого розвитку отримав метод обліку витрат, у якому, на відміну від існуючих, використано мобільну платформу на основі мови програмування Kotlin, що дозволило підвищити продуктивність на 10%.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено мовою програмування Kotlin мобільні програмні засоби для обліку витрат з їх геолокацією.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованій праці, опублікованих у співавторстві, автору належать такі результати: розробка моделі та інтерфейсу мобільного додатку обліку витрат [2].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.).

Публікації. Основні результати досліджень опубліковано в одній статті у матеріалах конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) (Вінниця, 2022 р.) .

1 ОБҐРУНТУВАННЯ ВИБОРУ МЕТОДУ РОЗРОБКИ ТА ПОСТАНОВКА ЗАДАЧ

1.1 Аналіз предметної області і сучасного стану питання

Більшість людей бажають витрачати менше та економити більше, але щоденна реальність роботи, сімейного життя та інших зобов’язань робить це непрактичним, навіть, неможливим. Важко знайти час чи мотивацію для контролю особистих фінансів, а деякі люди взагалі цього не роблять.

З цих та багатьох інших причин програми контролю фінансів стають все більш популярними протягом останнього десятиліття. Ці програми полегшують контроль за витратами та накопиченнями, пов’язуючи облікові записи з додатком та відстежуючи грошовий потік [2].

Сьогодні практично в кожного є смартфон, який допомагає нам у роботі, навчанні, розвагах. Проте вони також є корисною річчю для збереження та управління інформацією, тому правильним кроком буде відмовитись від старомодних записів буденної інформації, такої як запис витрат на паперах і користуватись новітніми технологіями не лише в освітніх цілях а й для модернізації буденності [3].

Розробка журналу витрат призначена для того, щоб полегшити фінансові розрахунки. За допомогою цього програмного продукту можна допомогти людям контролювати свої витрати. Журнал витрат допоможе швидко, зручно і ефективно додавати витрати і доходи користувача, залежно від здійснених покупок або отриманих грошей.

Додаток покаже, куди зникають гроші, а також допоможе накопичити та заощадити кошти. Управління фінансами стане максимально простим.

Для розробки даного програмного продукту був проведений аналіз існуючих реалізацій. Було виявлено різні переваги та недоліки аналогічних програм. Після порівняння існуючих реалізацій було сплановано власний програмний продукт, виявлено базові алгоритми, які будуть застосовані та максимізують ефективність програми.

1.2 Порівняльний аналіз аналогів

Аналогом додатку менеджер рецептів є мобільний додаток «Гаманець – облік витрат і доходів, бюджет» (рисунок 1.1).

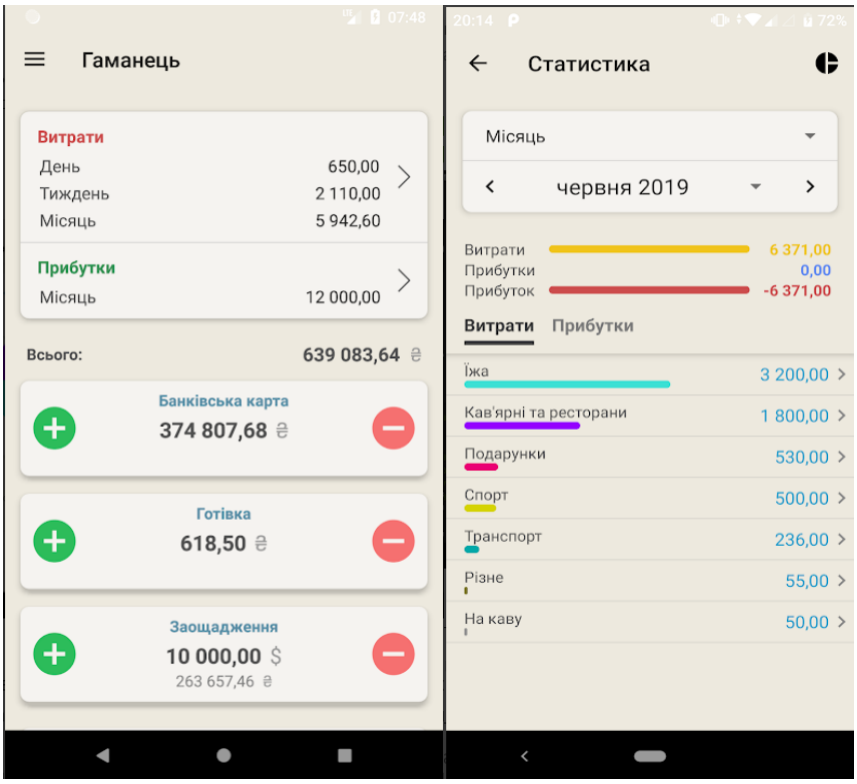

Рисунок 1.1 – Додаток «Гаманець – облік витрат і доходів, бюджет»

Цей додаток допоможе швидко, зручно і ефективно додавати витрати і доходи в момент здійснення покупки або отримання грошей . Додаток покаже, куди зникають гроші, а також допоможе накопичувати ти заощаджувати кошти. Управління фінансами стане максимально простим. Якщо записувати всі витрати протягом дня, то додаток покаже, куди пішли ваші гроші у зрозумілому графічному інтерфейсі та в діаграмі. Також є можливість дивитись статистику витрат і доходів за день, тиждень, місяць, рік і за вказаний період [4].

Ще одним аналогом є мобільний додаток «1Money – облік витрат, бюджет» (рисунок 1.2). Цей додаток покаже, куди зникають гроші. Найчастіше облік фінансів заняття не з приємних і вимагає конкретних навичок і вмінь, тому цей додаток допоможе легко вводити витрати прямо на ходу [5].

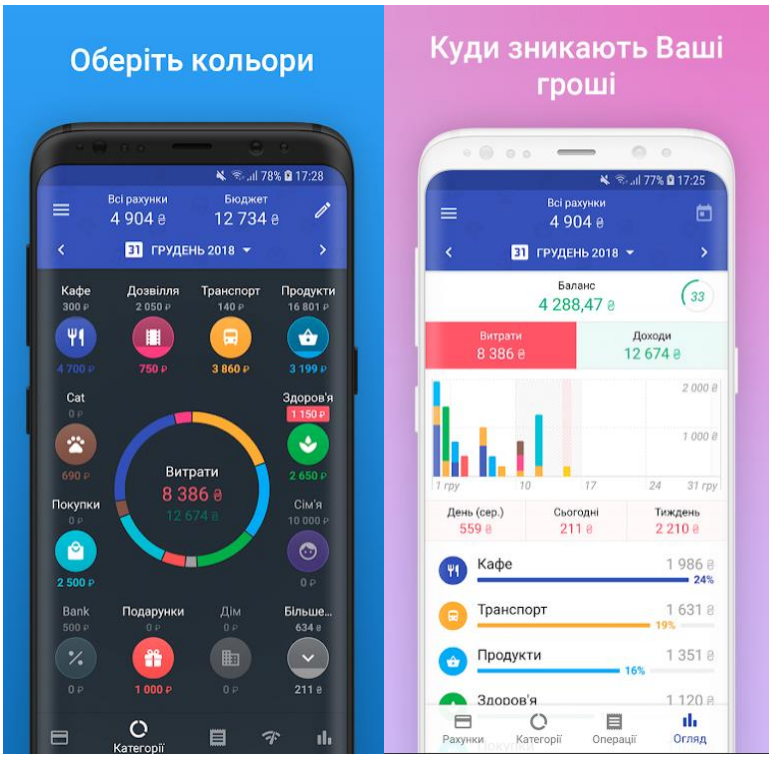

Рисунок 1.2 – Додаток «1Money – облік витрат, бюджет»

Отже, більшість аналогів програми «журнал витрат» існують у вільному доступі, мають красивий та функціональний дизайн. Проте кожна з програм має певні недоліки. А також у порівняні з аналогами, додаток який розробляється, матиме можливість відмічати витрати на карті. Тому є актуальною задачею розробити мобільний додаток з геолокацією контролю витрат.

Готовий продукт повинен мати зрозумілий користувачу інтерфейс, можливість додавання нових витрат та доходів, можливість переглядати раніше додані витрати, видаляти та редагувати їх. А також цей додаток повинен мати можливість додавати, редагувати, та видаляти категорії.

1.3 Вибір моделі життєвого циклу для розробки додатку

При розробці будь-якого програмного додатку розробник чи команда розробників обирає модель за якою буде розроблятися додаток. Такі моделі включать в себе процеси розробки, експлуатації та супровід програмного додатку, а також починаючи від визначення вимог до припинення використання програмного додатку.

Моделі розробки програмного забезпечення – це в основному набір конкретних інструментів і підходів, які використовуються протягом життєвого циклу проєкту. Їх цілі полягають у підвищенні продуктивності при одночасному зниженні операційних витрат, а це означає, що модель розробки програмного забезпечення, яку обирають, часто може вплинути на проєкт або порушити його. Щоб уникнути несподіваних збоїв у потоці, потрібно розглянути компоненти життєвого циклу розробки програмного забезпечення (SDLC) і те, як вони можуть вплинути на ваш вибір моделей розробки програмного забезпечення [6].

Дотримання процесу SDLC у розробці програмного забезпечення допоможе виконати проєкт за графіком і в рамках бюджету.

Ітераційна модель

Ітераційні моделі програмного процесу починаються з меншого набору вимог. Розробники використовують ці вимоги для аналізу та поступового впровадження функцій. Після цього ці функції оцінюються та тестуються для визначення нових вимог. Цей цикл називається «ітерацією». На рисунку 1.3 зображено принцип ітераційної моделі.

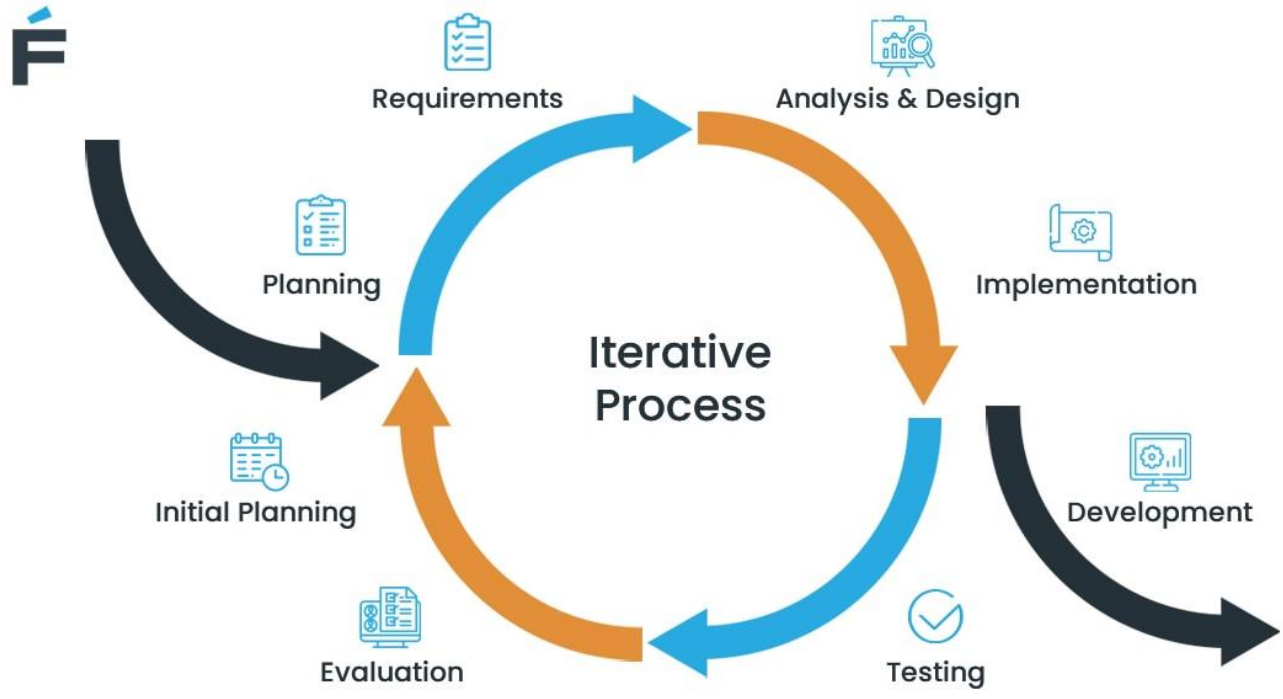

Рисунок 1.3 – Принцип ітераційної моделі

Цей життєвий цикл повторюється багато разів, вводячи нові варіанти дизайну, кодування та зміни вимог.

- Переваги:
- Кодування починається на ранніх етапах
- Економічний спосіб впровадження змін вимог
- Спрощене управління, оскільки виробництво поділено на менші частини (ітерації)
- Помилки та дефекти легко виявити на попередніх етапах SDLC

Недоліки:

- Важко аналізувати ризики
- Потенційні проблеми з дизайном та архітектурою на пізніх етапах
- Занадто залежить від базового плану
- Модель із великими ресурсами

Підходить для:

- Великих проєктів з кількома модулями (наприклад, веб-сервіси або мікросервіси)
- Проєктів з чітко визначеними цілями та завданнями

AGILE модель

Agile зосереджується на безперервних циклах випуску та міжфункціональному розвитку [5]. Це один з найбільш гнучких і популярних типів моделей SDLC. За даними CollabNet 2019 State of Agile Report, рушійними факторами його визнання є зниження витрат на 27 відсотків і підвищення продуктивності. Agile дуже схожий на Iterative, де команди працюють з повторюваними кроками. Однак, на відміну від Iterative, Agile не покладається на базовий план. План разом із вимогами постійно змінюється протягом життєвого циклу. Найпопулярніші підгрупи та методи в Agile: Scrum та Kanban. На рисунку 1.4 зображено схематично Agile модель.

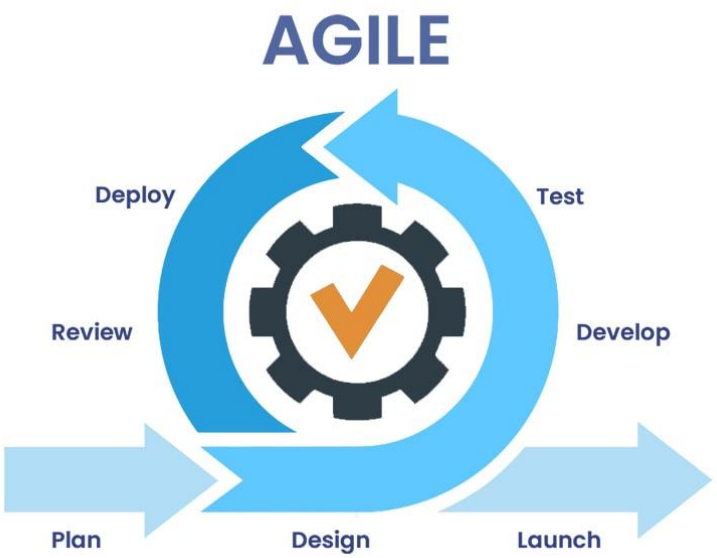

Рисунок 1.4 – Agile модель

Переваги:

- Легко змінювати вимоги та базовий план
- Швидкий випуск прототипу
- Зосередьтеся на спілкуванні між розробниками та клієнтом
- Залучення клієнта інтегровано в життєвий цикл
- Постійна оцінка та зворотній зв'язок

Недоліки:

- Важко оцінити кінцеву вартість виробництва
- Можливі конфлікти архітектури через постійні зміни вимог

Підходить для:

- Великі та менші проєкти
- Аутсорсинг та керовані IT-послуги
- Додавання нових функцій до робочого прототипу

Водоспадна модель

Водоспад є одним із найстаріших типів методологій життєвого циклу розробки програмного забезпечення [5]. Він зосереджений на лінійному підході, коли фази SDLC слідують одна за одною послідовно. Результат кожної фази

впливає на вхід наступної. Тому повернутися до попередніх етапів набагато важче і дорожче. На рисунку 1.5 зображено принцип водоспадної моделі.

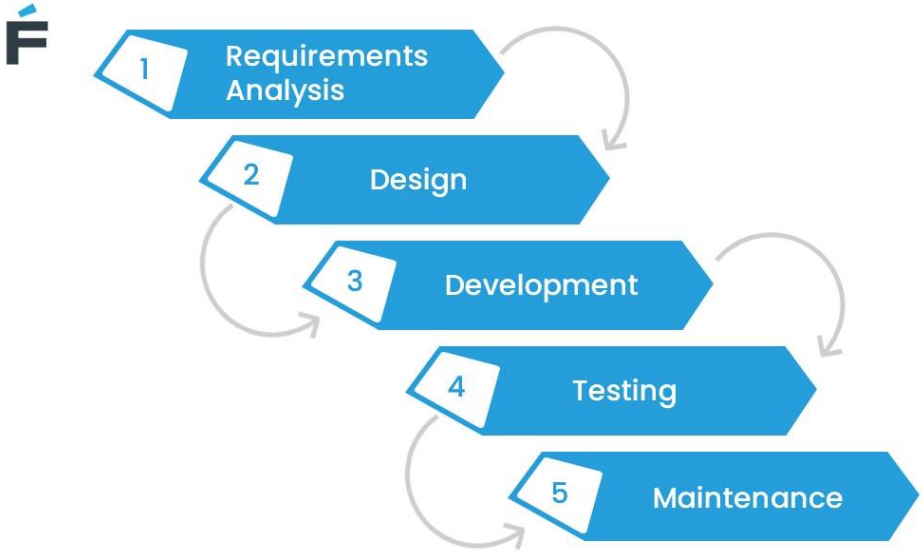

Рисунок 1.5 – Принцип водоспадної моделі

Переваги:

- Прості та легко керовані
- Чітко визначені результати та віхи
- Легке визначення пріоритетів завдань

Недоліки:

- Недостатня універсальність ітераційного та Agile підходів.
- Фази не можуть перекриватися.
- Забирають багато часу
- Занадто дорого, щоб повернутися до попередніх етапів.

Підходить для:

- менших та середніх проєктів з чітко визначеними вимогами

Серед розглянутих вище моделей життєвого циклу найбільш прийнятним буде водоспадна модель. Оскільки маючи чітке технічне завдання та невеликий за розмірами проєкт каскадна модель буде швидшою у виконанні ніж спіральна

модель. Таким чином кожен етап розробки буде поетапним, повернення до попередніх етапів не передбачено.

1.4 Постановка задачі розробки

Оскільки завданням дипломної роботи є розробка мобільного додатку з геолокацією обліку витрат, можна виділити такі задачі для виконання:

- проаналізувати наявні засоби для розрахунку коштів;
- визначити ключові модулі та поняття;
- розробити структуру БД;
- проаналізувати наявні хмарні середовища для створення бази даних;
- розробити серверний частину.

1.5 Висновки

1. Аналіз додатків обліку витрат показав, що функціонал відомих реалізацій не задовольняє потреби багатьох користувачів, зокрема, через відсутність функції геолокації витрат, що підтверджує доцільність розробки додатку обліку витрат з їх геолокацією.
2. Визначено функціональні характеристики додатку для обліку витрат, що розробляється. Зокрема, додано можливість створення позначок на карті місць розташування об’єктів здійснення витрат коштів, що полегшує користувачам вибір шляхів оптимізації витрат.

2 РОЗРОБКА АРХІТЕКТУРИ ТА АЛГОРИТМІВ МОБІЛЬНОГО ДОДАТКУ З ГЕОЛОКАЦІЄЮ ОБЛІКУ ВИТРАТ

2.1 Розробка загальної архітектури додатку

Архітектура програми – це структурна карта того, як збираються програмні засоби організації та як ці програми взаємодіють одна з одною, щоб задовольнити вимоги бізнесу або користувачів. Архітектура програми допомагає забезпечити масштабованість та надійність програм, а також допомагає підприємствам виявляти недоліки у функціональності. Загалом, архітектура програми визначає, як програми взаємодіють з сутностями, такими як проміжне програмне забезпечення, бази даних та інші програми. Архітектура додатків зазвичай дотримується принципів розробки програмного забезпечення, які є загальноприйнятими серед його прихильників, але можуть не мати офіційних галузевих стандартів [7].

Переваги архітектури додатків. Загалом, архітектура додатків допомагає IT та бізнес-планувальникам працювати разом, щоб мати правильні технічні рішення для досягнення бізнес-цілей. Точніше, архітектура програми:

- Зменшує витрати за рахунок виявлення надмірностей, таких як використання двох незалежних баз даних, які можна замінити однією;
- Підвищує ефективність, виявляючи недоліки, такі як основні послуги, до яких користувачі не можуть отримати доступ через мобільні додатки;
- Створює корпоративну платформу для доступності додатків та сторонньої інтеграції;
- Дозволяє створювати сумісні модульні системи, простіші у використанні та обслуговуванні;
- Допомагає архітектору «побачити загальну картину» та узгодити стратегії програмного забезпечення із загальними бізнес-цілями організації.

Діаграма варіантів використання. Як найвідоміший тип діаграм поведінкових типів UML, діаграми варіантів використання дають графічний огляд акторів, задіяних у системі, різних функцій, необхідних цим акторам, і того, як ці різні функції взаємодіють. Це чудова відправна точка для будь-якого обговорення

проєкту, оскільки можливо легко визначити головних учасників та основні процеси системи. На рисунку 2.1 зображено діаграму використання для додатку з геолокацією обліку витрат.

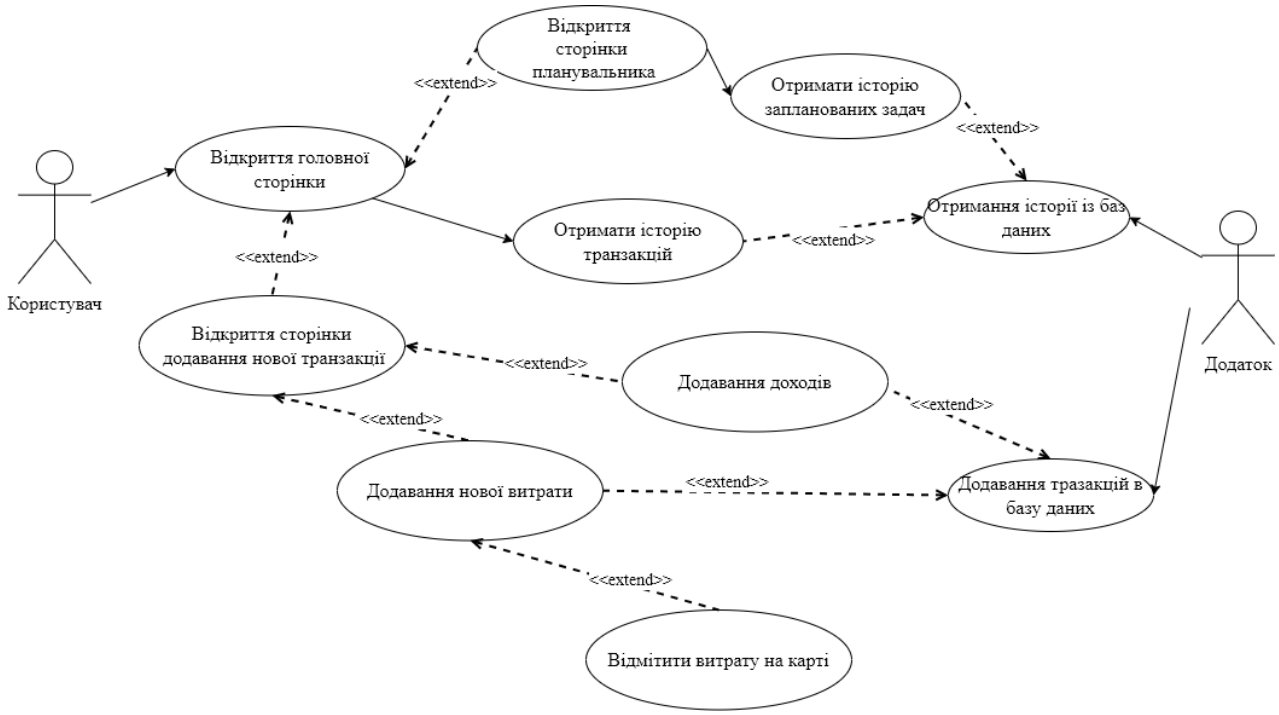

Рисунок 2.1 – Діаграма варіантів використання

Діаграма послідовності – це тип діаграми взаємодії, оскільки вона описує, як і в якому порядку група об’єктів працює разом. Ці діаграми використовуються розробниками програмного забезпечення та бізнес-фахівцями, щоб зрозуміти вимоги до нової системи або документувати існуючий процес. Діаграми послідовності іноді називають діаграмами подій або сценаріями подій [8].

Переваги діаграм послідовності:

- Представити деталі варіанту використання UML.
- Змоделювати логіку складної процедури, функції або операції.
- Подивитися, які об’єкти та компоненти взаємодіють один з одним, щоб завершити процес.
- Планувати та розуміти детальну функціональність існуючого або майбутнього сценарію.

Існує два типи діаграм послідовності: діаграми UML і діаграми на основі коду. На рисунку 2.2 зображено діаграму послідовності додатку.

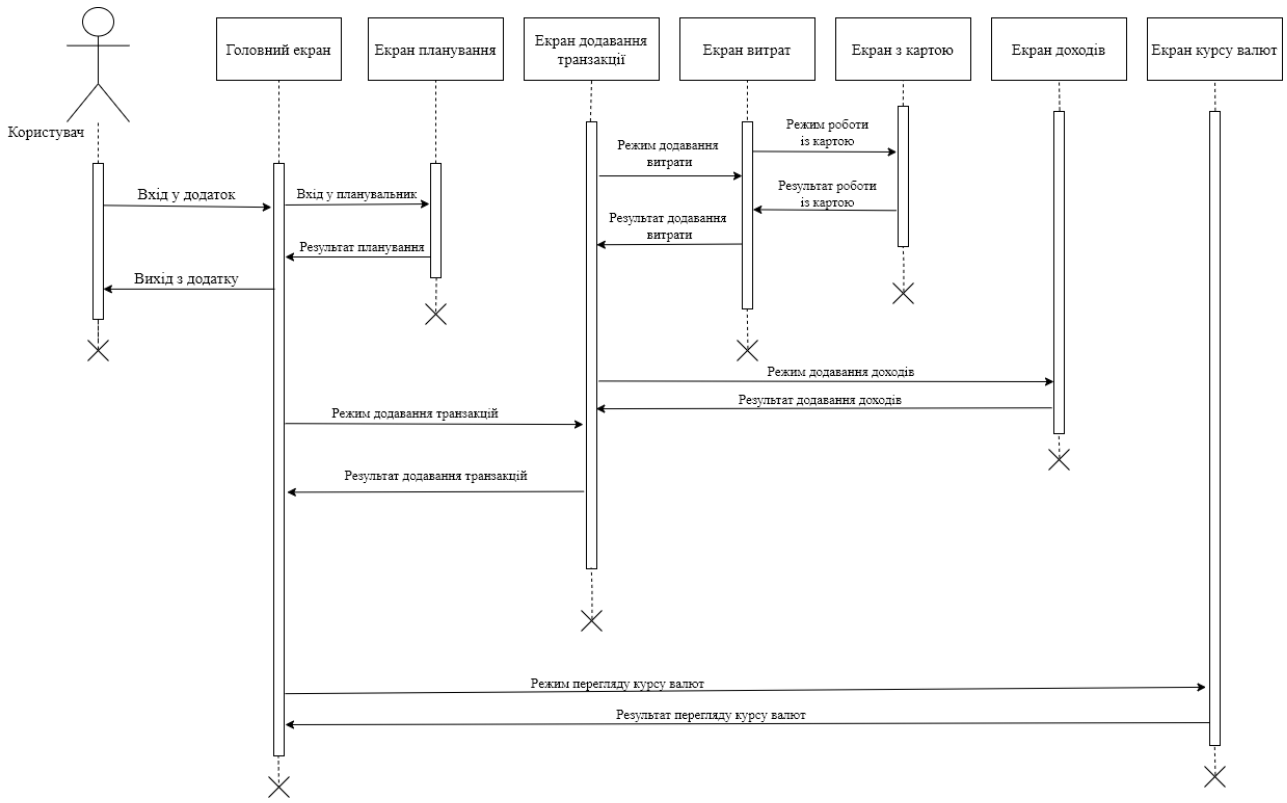

Рисунок 2.2 – Діаграма послідовності мобільного додатку

Об’єктами діаграми послідовності є: «Користувач», «Головний екран», «Екран планування», «Екран додавання транзакції», «Екран затрат», «Екран з картою», «Екран доходів», «Екран курсу валют». На діаграмі можна побачити можливі варіанти використання додатку. Користувач із головного екран має можливість переглядати витрати та доходи, перейти на екран планувальника, добавити нові транзакції, відкрити екран про курс валют.

Представлені діаграми описують загальну архітектуру мобільного додатку з геолокацією витрат для того, аби потім було зрозуміло, що потрібно реалізовувати.

2.2 Програмування бази даних

Програми, які обробляють нетривіальні обсяги структурованих даних, можуть отримати велику користь від збереження цих даних локально.

Найпоширенішим варіантом використання є кешування відповідних фрагментів даних, щоб, коли пристрій не може отримати доступ до мережі, користувач все ще міг переглядати цей вміст у автономному режимі. Бібліотека збереження Room забезпечує рівень абстракції над SQLite, щоб забезпечити вільний доступ до бази даних, використовуючи всю потужність SQLite. Зокрема, Room надає такі переваги: Перевірка SQL-запитів під час компіляції. Зручні анотації, які зводять до мінімуму повторюваний і схильний до помилок шаблонний код. Оптимізовані шляхи міграції бази даних [9].

База даних Firebase Realtime — це хмарна база даних. Дані зберігаються у форматі JSON і синхронізуються в режимі реального часу з кожним підключеним клієнтом. При створені міжплатформенних програм усі клієнти спільно використовують один екземпляр бази даних реального часу й автоматично отримують оновлення з найновішими даними.

Замість типових запитів HTTP база даних Firebase Realtime використовує синхронізацію даних — щоразу, коли дані змінюються, будь-який підключений пристрій отримує це оновлення протягом мілісекунд. Забезпечте спільну роботу та захоплюючий досвід, не замислюючись про мережевий код.

Доступ до бази даних Firebase Realtime можна отримати безпосередньо з мобільного пристрою або веб-переглядача; немає потреби в сервері додатків. Безпека та перевірка даних доступні через правила безпеки бази даних Firebase Realtime, правила на основі виразів, які виконуються під час читання або запису даних [10].

Завдяки базі даних Firebase Realtime Database можна підтримувати потреби в даних додатку в масштабі, розділяючи дані між кількома екземплярами бази даних в одному проєкті Firebase. Також є можливість спростити автентифікацію за допомогою Firebase Authentication і контролювати доступ до даних у кожній базі даних за допомогою спеціальних правил баз даних Firebase Realtime для кожного екземпляра бази даних. На рисунок 2.4 зображено схематично Firebase Database.

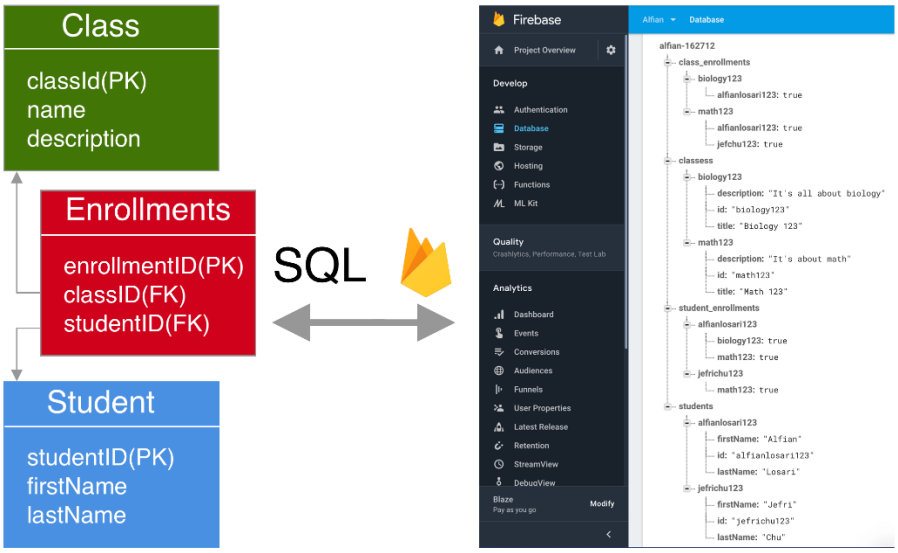

Рисунок 2.4 – Firebase Database

В результаті дослідження, для розробки мобільного додатку з геолокацією обліку витрат було вирішено використовувати Room DB для збереження даних у пристрої користувача. Для хмарного збереження даних було обрано готове рішення Firebase Database від компанії Google.

2.3 Розробка блок-схеми алгоритму роботи програми

Одним із важливих етапів розробки додатку – побудова алгоритму його роботи. Алгоритмом у програмуванні, можна вважати рецептом, який описує конкретні досягнення, необхідні для вирішення проблеми або досягнення мети. Алгоритми прості для розуміння по тій причині, що поставлена проблема розбивається на дрібніші шматки або кроки. Таким чином, програмісту легше перетворити його на справжню програму. Алгоритми та блок-схеми – це два різні інструменти, які допомагають створювати нові програми, особливо в комп'ютерному програмуванні. Алгоритм — це покроковий аналіз процесу, тоді як блок-схема пояснює кроки програми у графічному вигляді.

На рисунку 2.5 зображена загальна блок-схема алгоритму роботи мобільного додатку з геолокацією обліку витрат.

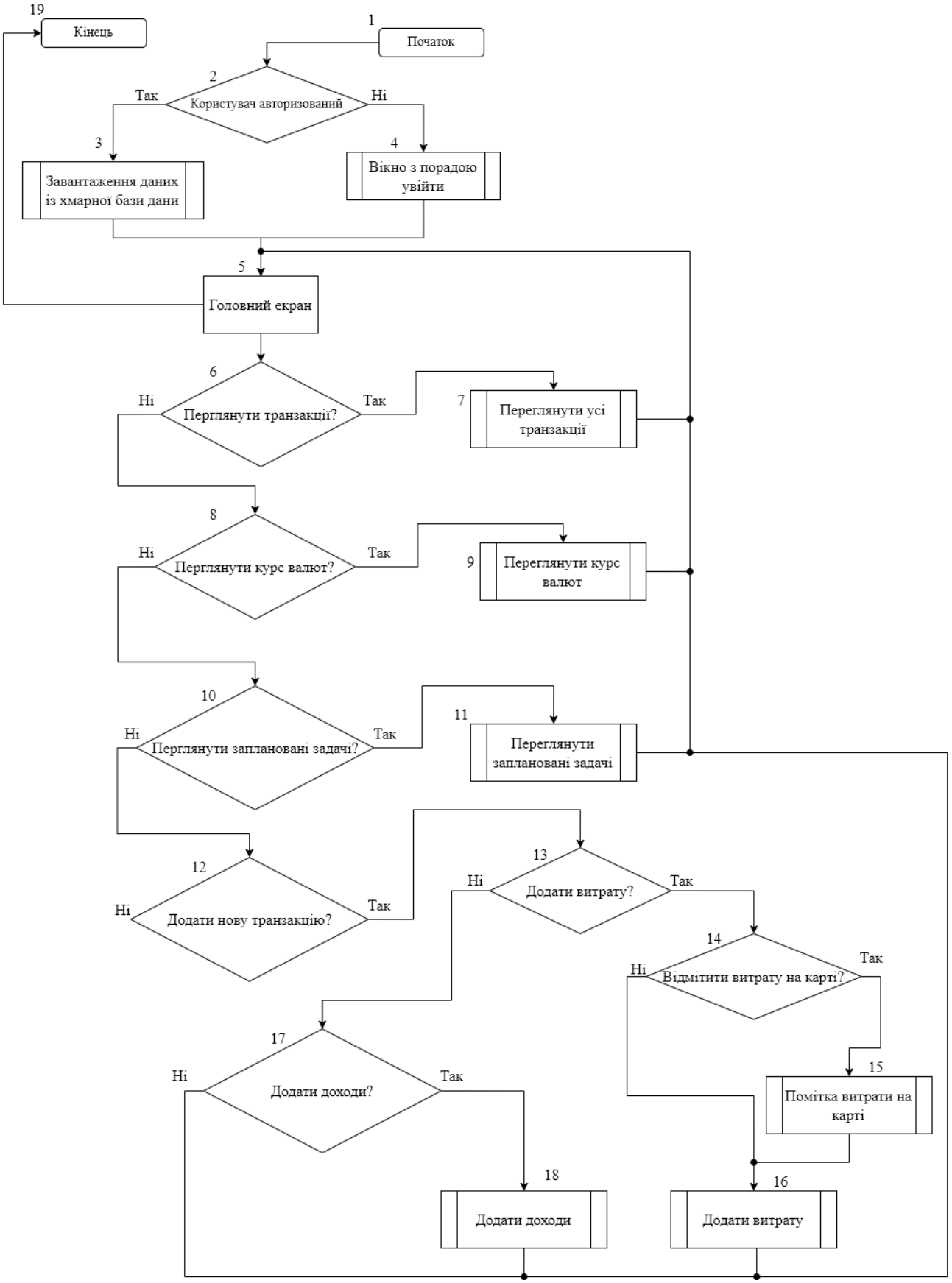

Рисунок 2.5 – Загальна блок-схема алгоритму додатку

На представленій блок схемі, схематично, представлені кроки роботи мобільного додатку:

1. Перший крок роботи додатку, це його запуск, а саме завантаження всіх компонентів та даних із кешу;
2. Запуск перевірки на авторизацію користувача;
3. Якщо користувач авторизований, відбувається оновлення записів із бази даних;
4. У випадку відсутності авторизації, користувачу пропонується увійти чи зареєструватися у додатку, аби записи були збережені у надійному місці.
5. Відкриття головного екрану програми, де користувач може побачити наявний баланс грошей, доходи та витрати і список останніх транзакцій.
6. Користувачу дається вибір переглядати список усіх транзакцій.
7. У випадку коли користувач обере варіант побачити транзакцій, йому відкриється екран із списком усіх операцій, у порядку від останніх.
8. Вибір переглянути курс валют. Користувач обирає чи потрібно йому здійснити таку операцію.
9. Відкривається вікно із списком валют, що дає користувачу можливість дізнатися стан конкретної валюти.
10. Вибір переглянути заплановані задачі. Користувач обирає чи потрібно йому здійснити таку операцію.
11. Користувач може переглянути заплановані задачі, а також створити нову задачу, вказати дату та час і підтвердити дію. В результаті операції користувач отримає нотифікацію із повідомленням у вказаний час.
12. Користувач здійснює вибір створені нової транзакції. Користувач обирає чи потрібно йому здійснити таку операцію.
13. Перед користувачем стоїть вибір, яку категорію транзакції обрати.
14. Якщо користувач обрав категорію витрати, перед ним стоїть вибір чи потрібно йому помічати позицію витрати на карті.
15. Користувачу відкривається вікно із картою, де вказуються його локація у вигляді категорії витрати, також він може вказати інше місце на карті.

16. Користувач коли заповнив поля у вікні витрати, натискає на клавішу «Зберегти» і в результаті транзакція зберігається, а вікно додавання витрати закривається і в результаті користувачу буде видно головний екран.
17. Якщо користувач обрав категорію «Доходи», він приймає рішення про заповнення полів у вікні.
18. Користувач заповнює відповідні поля у вікні «Доходи», натискає клавішу «Зберегти», в результаті відбудеться навігація до головного вікна.
19. У головному вікні користувач може продовжити свою роботу у додатку, або закрити його у випадку, коли він завершив свою роботу.

2.4 Висновки

1. Розроблено архітектуру мобільного додатку з геолокацією обліку витрат, особливістю якої є використання як локальної так і хмарної баз даних, що надає можливість поповнювати базу даних витрат навіть в умовах відсутності мережі.
2. Розроблено загальну блок-схему додатку, яка деталізує функції програми, способи навігації та спрощує розробку коду мобільного додатку з геолокацією обліку витрат.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ З ГЕОЛОКАЦІЄЮ ОБЛІКУ ВИТРАТ

3.1 Аналіз та обґрунтування вибору програмних засобів для розробки

Перед початком роботи над створенням мобільного додатку потрібно зробити аналіз засобів розробки мобільних додатків. Для розробки мобільного додатку потрібно знати мову програмування. Але може бути важко вибрати найкращу мову (або мови) для проекту.

Все залежить від того, який додаток планується розробити. Для деяких додатків розробникам мобільних пристроїв можуть не знадобитися всі доступні функції певної мови. В інших ситуаціях для однієї програми може знадобитися більше однієї мови.

Java. Спочатку Java була офіційною мовою для розробки додатків для Android, отже, вона також є найбільш використовуваною мовою. Багато програм у Play Market створено на Java, і це також найбільш підтримувана мова Google. Java має чудову онлайн-спільноту для підтримки в разі будь-яких проблем. Однак Java є складною мовою для початківців, оскільки вона містить такі складні теми, як конструктори, винятки нульових покажчиків, паралельність, перевірені винятки тощо. Крім того, Android Software Development Kit (SDK) підвищує складність на новий рівень. Загалом, Java – чудова мова, щоб відчути всі переваги розробки додатків для Android. Однак це може бути трохи складним для новачків, які вважають за краще почати з чогось простішого, а потім повернутися до цього [11].

Kotlin. Тепер Kotlin є офіційною мовою для розробки додатків для Android, оголошеної Google у 2019 році. Kotlin – це кросплатформна мова програмування, яка може використовуватися як альтернатива Java для розробки додатків для Android. У 2017 році він також був представлений як вторинна «офіційна» мова Java. Kotlin може взаємодіяти з Java і працює на віртуальній машині Java. Єдина суттєва відмінність полягає в тому, що Kotlin видаляє зайві функції Java, такі як винятки нульового покажчика. Це також усуває необхідність закінчувати кожен рядок крапкою з комою. Коротше кажучи, Kotlin набагато простіше для початківців

у порівнянні з Java і його також можна використовувати як «точку входу» для розробки додатків для Android.

C++ можна використовувати для розробки додатків Android за допомогою набору Android Native Development Kit (NDK). Однак програму не можна створити повністю за допомогою C++, а NDK використовується для реалізації частин програми у рідному коді C++. Це допомагає використовувати бібліотеки коду C++ для програми за потреби. Хоча C++ у деяких випадках корисний для розробки додатків для Android, але його набагато складніше налаштувати і він менш гнучкий. Це також може призвести до більшої кількості помилок через підвищену складність. Отже, краще використовувати Java, ніж C, оскільки він не забезпечує достатнього виграшу, щоб компенсувати необхідні зусилля [11].

C# дуже схожий на Java, тому він ідеально підходить для розробки додатків Android. Як і Java, C# також реалізує збір сміття, тому є менше шансів витоку пам'яті. І C# також має більш чистий і простий синтаксис, ніж Java, що робить кодування з ним порівняно простіше. Раніше найбільшим недоліком C# було те, що він міг працювати тільки в системах Windows, оскільки використовував .NET Framework. Однак цю проблему вирішив Xamarin. Android (раніше Mono для Android) — це кросплатформна реалізація загальномовної інфраструктури. Тепер, Xamarin. Інструменти Android можна використовувати для написання нативних програм Android і обміну кодом між різними платформами.

Python можна використовувати для розробки додатків Android, навіть якщо Android не підтримує розробку на Python. Це можна зробити за допомогою різних інструментів, які перетворюють програми Python в пакети Android, які можуть працювати на пристроях Android. Прикладом цього є Kivy, бібліотека Python з відкритим кодом, яка використовується для розробки мобільних додатків. Він підтримує Android, а також заохочує швидку розробку додатків. Однак недоліком цього є те, що для Kivy не буде вбудованих переваг, оскільки він не підтримується в оригінальному форматі.

Dart — це мова програмування з відкритим вихідним кодом, яка керує фреймворком Flutter, який сьогодні стає все більш популярним через його здатність

створювати красиві та продуктивні програми для Інтернету, настільних комп’ютерів і мобільних пристроїв за менший час. Ключова перевага Dart полягає в тому, що він розроблений Google як мова, оптимізована для клієнтів для швидких програм на будь-якій платформі. Dart в основному зосереджується на тому, щоб полегшити розробку інтерфейсу користувача для розробників за допомогою таких функцій, як гаряче перезавантаження, що дозволяє розробникам миттєво бачити зміни під час роботи над програмою. Dart також відомий своєю високою продуктивністю, він компілює машинний код ARM і x64 для мобільних пристроїв, настільних комп’ютерів і серверів. І до JavaScript для веб-програм .

Існує багато програм, таких як месенджери, музичні плеєри, ігри, калькулятори тощо, які можна створити за допомогою вищевказаних мов. І немає мови, яку можна було б назвати «правильною мовою» для розробки додатків для Android. Зробити правильний вибір мови можна лише на основі цілей і переваг для кожного окремого проекту.

В результаті дослідження різних мов програмування, було обрано Kotlin, для розробки мобільного додатку з геолокацією обліку витрат, так як Kotlin є офіційною мовою Google для розробки додатків для android. Перевагою використання Kotlin для розробки мобільних додатків є те, що в інтернеті міститься багато документації, що це є зручно, коли над додатком працює невелика команда.

3.2 Вибір середовища розробки

Найпоширенішими видами діяльності з написання програмного забезпечення є редагування вихідного коду, створення виконуваних файлів і налагодження.

IDE – це набір програм, у якому ви можете виконувати всі ці дії в одному місці. В IDE. можете виконати весь життєвий цикл розробки програмного забезпечення. В результаті для розробки мобільних додатків було створено багато різних інтегрованих середовищ розробки (IDE). З'являється багато нового, а старі розвиваються.

Середовище розробки DroidScript. DroidScript – це мобільна IDE для Android,

яка не вимагає підключення до Інтернету та не використовує хмару, тому її можна використовувати з будь-якого місця [12] .

DroidScript є дружнім до нових користувачів, тих, хто тільки навчається, і має багато навчальних посібників. Він також містить багато плагінів і розширень для більш досвідчених користувачів. На рисунку 3.1 зображено середовище розробки DroidScript.

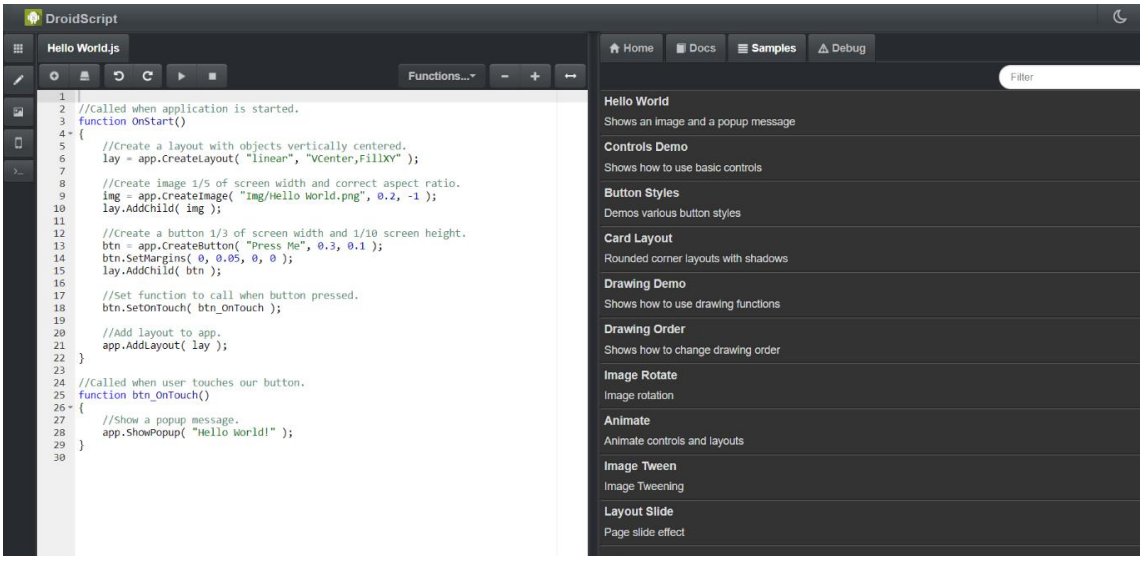

Рисунок 3.1 – Середовище розробки DroidScript

Середовище розробки CppDroid. CppDroid – це проста IDE для Android, орієнтована на програмування на C/C++. Як і раніше згадані IDE, CppDroid містить приклади та підручники для початківців. При першому запуску програми CppDroid, вона завантажить необхідні бібліотеки та створить програму «Hello World» [12].

CppDroid має настроюваний редактор з розумним підсвічуванням синтаксису. Він містить діагностику та виправлення в режимі реального часу. Має вбудований комп’ютер, що дозволяє працювати в автономному режимі. Преміальна версія надає доступ до Dropbox, підтримки Google Drive, прикладів і навчальних посібників, а також статичного аналізу. На рисунку 3.2 зображено середовище розробки CppDroid.

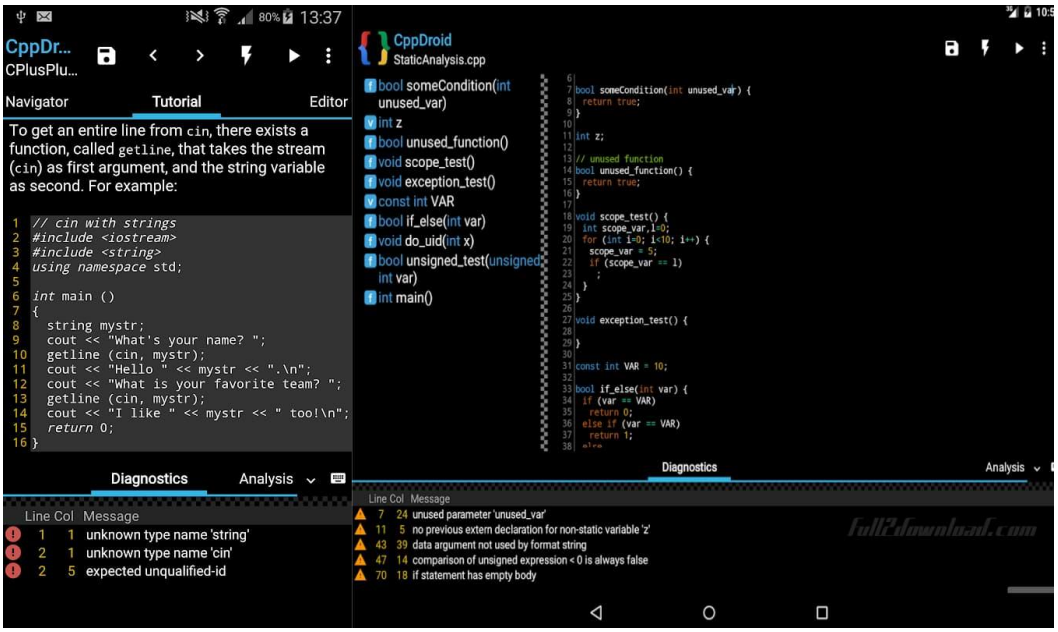

Рисунок 3.2 – Середовище розробки CppDroid

Середовище розробки Visual Studio. Visual Studio – один із найкращих інтегрованих середовищ розробки. Microsoft зробила це, і в останні роки, після інтеграції з Xamarin, він став одним із кращих середовищ для розробки Android . Середовище розробки представлено на рисунку 3.3.

Є інструменти для швидкого кодування та можливість тестування різних функцій. IDE дає можливість швидко виправляти помилки та прискорювати швидкість програми. Це найефективніша IDE.

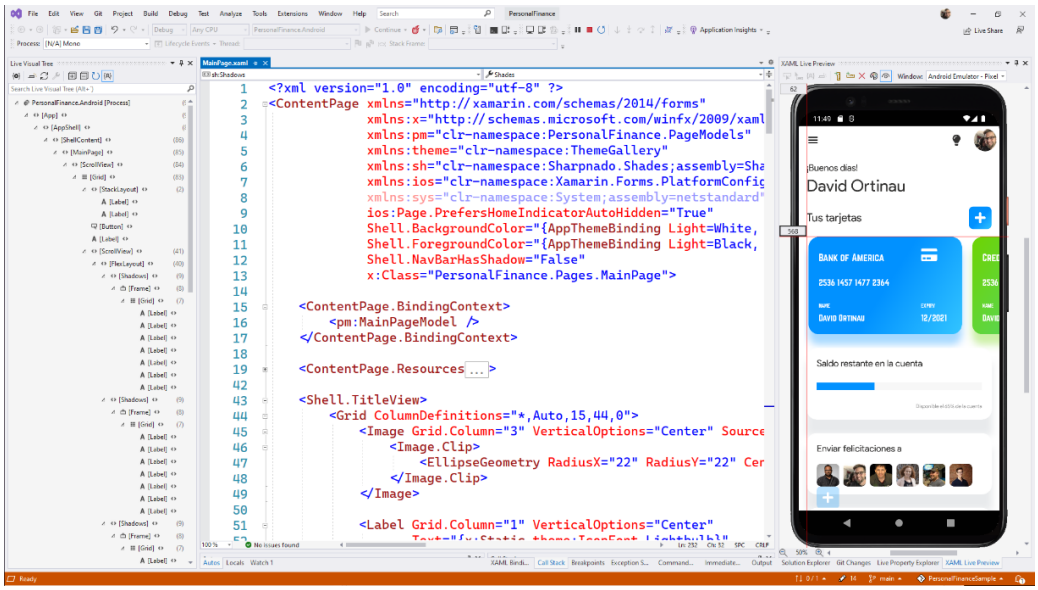

Рисунок 3.3 – Середовище розробки Visual Studio

Середовище розробки Android Studio. Android Studio є офіційною IDE для Android. Це набір програмного забезпечення, створений компанією Google і має всі вбудовані інструменти для створення високоякісного додатка для Android.

Android Studio здебільшого відома своєю здатністю прискорювати процес розробки, не втрачаючи при цьому ніякої якості.

Деякі з найкращих функцій:

- Застосувати зміни – що дозволяє вносити зміни в програму без необхідності перезапускати програму;
- Інтелектуальний редактор коду – отримуйте пропозиції щодо кращого коду під час введення;
- Швидкий і багатофункціональний емулятор – дозволяє дуже швидко встановлювати та запускати тестові програми для різних пристроїв Android;
- Інструменти та фреймворки тестування. Виконуйте тести на своєму пристрої чи емуляторі, у середовищі безперервної інтеграції або в тестовій лабораторії Firebase;

Серед інших функцій Android Studio дозволяє підключатися до інших файлів проекту, наприклад C/C++. Також є інтеграція з хмарою Google. На рисунку 3.4 зображено середовище розробки Android Studio.

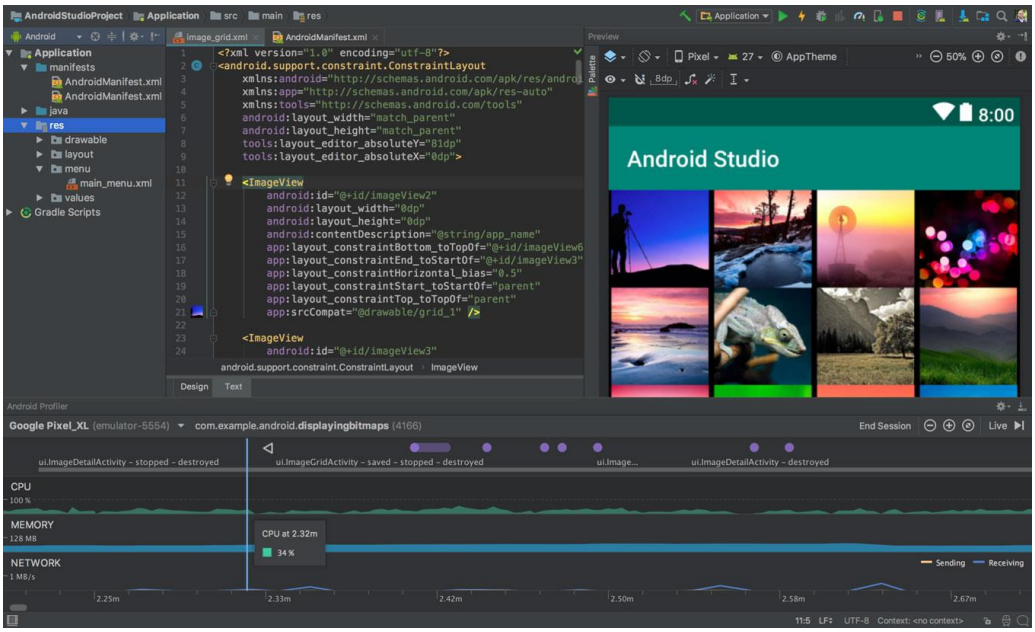

Рисунок 3.4 – Середовище розробки Android Studio

В результаті дослідження різних середовищ розробки додатків на android, було розглянуто їх можливості, швидкість компіляції додатку та яку мову програмування вони підтримують. Було виділено їх переваги та недоліки. Із отриманих даних було визначено, що Android Studio – є офіційним середовищем розробки від Google, він підтримує мову програмування Kotlin і є найбільш орієнтований для розробки під платформу Android.

Тому для створення мобільного додатку з геолокацією обліку витрат було обрано середовище розробки Android Studio.

3.3 Програмна реалізація

Для розробки мобільного додатку було опрано шаблон проектування MVVM (Model-View-ViewModel). Розробка графічного інтерфейсу для користувачів ділиться на дві частини. Перша – це робота з мовою розмітки або кодом GUI.

Друга – розробка бізнес-логіки або логіки бекенда (модель даних). Частина View model у MVVM – це конвертер значень. Це означає, що view model відповідає за конвертування об'єктів даних із моделі в такий вид, щоб з об'єктами було легко працювати. Якщо дивитися з цього боку, то view model це швидше модель, ніж уявлення [13]. Вона контролює більшу частину логіки відображення. Модель представлення може реалізовувати патерн медіатор. Для цього організується доступ до логіки бекенда навколо набору юз-кейсів, що підтримуються представленням. На рисунку 3.5 зображено схематично шаблон проектування MVVM.

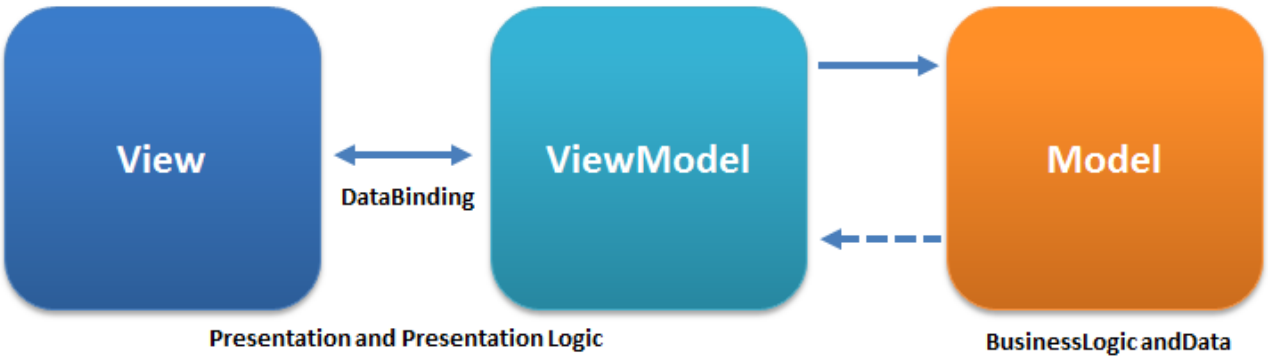

Рисунок 3.5 – Шаблон проектування MVVM

View містить структурне визначення, що користувачі отримають на екранах. Розробник може помістити сюди статичний та динамічний вміст (анімацію та зміну станів). Тут може бути ніякої логіки докладання. У view може бути активність або фрагмент.

View model. Цей компонент пов'язує модель та уявлення. Відповідає за керування посиланнями даних та можливих конверсій. Тут з'являється біндінг. В Android про це не потрібно турбуватися, тому що можна використовувати клас AndroidViewModel або ViewModel.

Model. Це рівень бізнес-даних і він не пов'язаний з жодним особливим графічним уявленням. В Android, згідно з “чистою” архітектурою, модель може містити базу даних, репозиторії та класи бізнес-логіки. На рисунку 3.6 зображено взаємодію між різними компонентами.

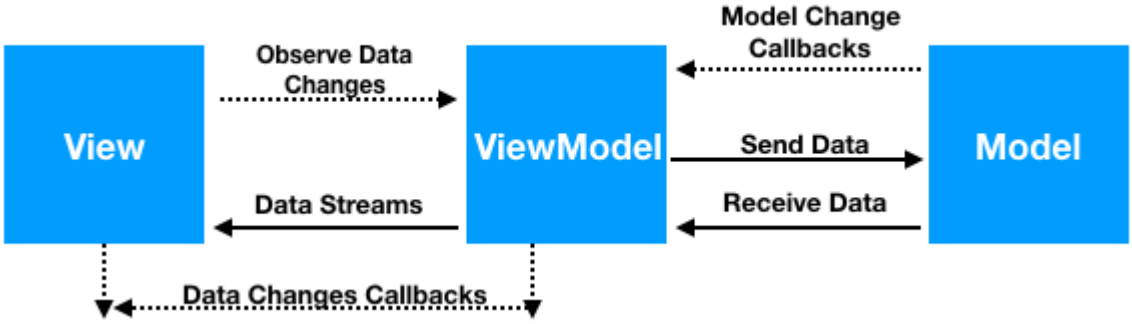

Рисунок 3.6 – Взаємодія у MVVM між різними компонентами

Щоб реалізувати патерн MVVM, важливо почати з компонентів, яким потрібний інший компонент. Це і є залежність. А з моменту появи компонента архітектури, логічне загальне рішення – реалізувати Android-програми за допомогою моделі, яка зображена на рисунку 3.7. Стрілки, які зображені на рисунку, ведуть від представлення (активності/фрагменту) до моделі.

А це означає, що View знає про View-Model, а не навпаки, і View Model знає про Model, і не навпаки. Тобто у вистави буде зв'язок з моделлю уявлення, а у моделі уявлення буде зв'язок з моделлю. Суворо в такому порядку, ніяк інакше. Завдяки такій архітектурі програма легко підтримувати та тестувати.

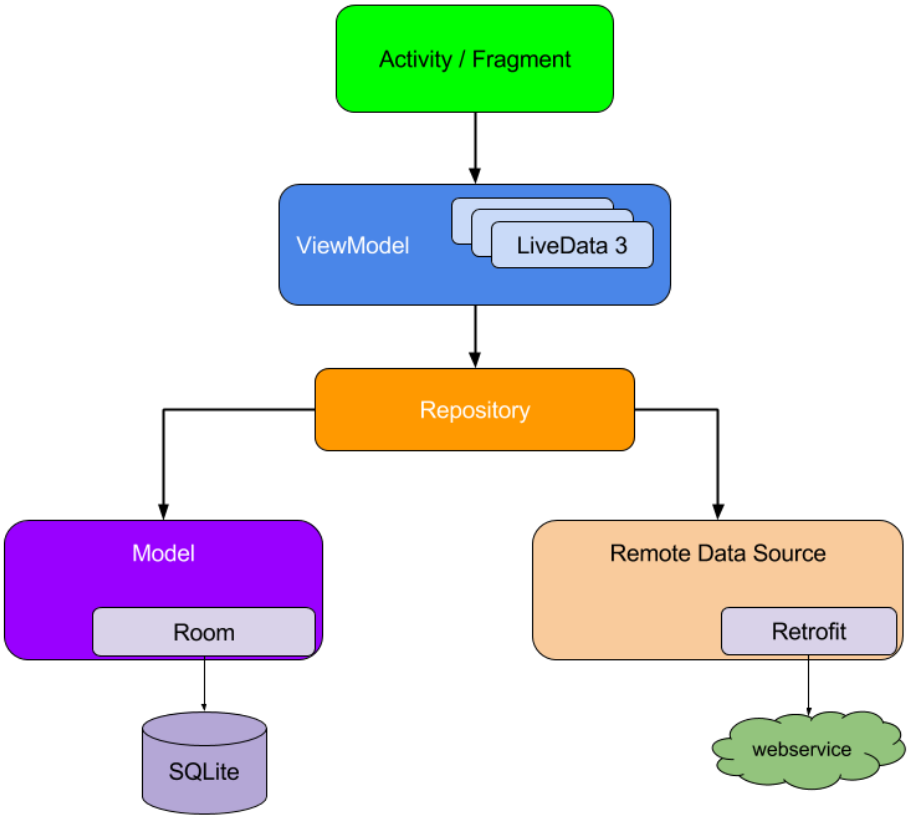

Рисунок 3.7 – Залежності у шаблоні проектування MVVM

Розглянемо приклад додавання транзакції у мобільному додатку з геолокацією обліку фінансів. Функція викликається, коли користувач натискає на клавішу «Зберегти». Функція отримує значення із поля де вказується сума транзакції, перевіряє чи є в полі якесь значення, потім провіряє значення на його коректність. Лістинг функції зображено на рисунку 3.8.

```
private fun actionOnSaveTransactionButton() {
    binding.transactionSaveChangesButton.setOnClickListener { it: View!
        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

Якщо вимоги функції виконуються після виклику на кнопку «Зберегти», функція продовжує збирати дані із полів. Функція отримує значення вибраної категорії, потім опис транзакції, який можна додати за бажанням, далі значення вказаної дати. Також сканується поле із локацією, у якому зберігаються локація витрати. Усі зібрані дані передаються у ViewModel, котра потім публікує значення у базах 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()
        when (getSelectedTransactionType()) {
            TransactionType.Expense -> {
                viewModel.insertTransaction(
                    Transaction(
                        value = transactionValue,
                        date = date,
                        type = TransactionType.Expense,
                        categoryId = categoryTitle,
                        description = description,
                        location = location
                    )
                )
                user.expense += transactionValue
                user.currentBalance -= transactionValue
                viewModel.updateUser(user)
            }
            TransactionType.Income -> {
                viewModel.insertTransaction(
                    Transaction(
                        value = transactionValue,
                        date = date,
                        type = TransactionType.Income,
                        categoryId = categoryTitle,
                        description = description,
                    )
                )
                user.income += transactionValue
                user.currentBalance += transactionValue
                viewModel.updateUser(user)
            }
        }
    }
}
```

Рисунок 3.9 – Лістинг коду збереження значення у базу даних

Функція для створення нагадування про майбутню покупку, працює коли користувач вказав у поле із заголовком задачі не порожнє і дата вказана правильно. Після цього функція зберігає значення у базі даних, потім викликає функцію для створення нотифікації, передає туди коректні параметри. В разі некоректних даних, користувач отримає повідомлення, що потрібно ввести коректні дані. На рисунку 3.10 зображено лістинг коду для створення нотифікації.

```
private fun setOnCreateReminderButton() {
    binding.plannerCreateNotification.setOnClickListener { it: View!
        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( tag: NotificationWorker.NOTIFICATION_TAG + plannerText)
                    .build()
                workManger.enqueueUniqueWork(
                     uniqueWorkName: NotificationWorker.NOTIFICATION_TAG + plannerText,
                    ExistingWorkPolicy.APPEND,
                    workRequest
                )
                onBackPressed()
            }
            false -> applicationContext.showToastLong("The plan title cannot be empty")
        }
    }
}
```

Рисунок 3.10 – Лістинг коду для створення нагадування

На рисунку 3.11 зображено лістинг коду для завантаження категорій із бази даних. Коли користувач відкриває вікно додавання транзакцій, він може вибрати категорію, або створити нову, тому ця функція завантажує категорії у пам’ять.

```
private fun initializeList() {
    binding.categoryList.adapter = CategoryListAdapter()
    (binding.categoryList.adapter as CategoryListAdapter).onClickListener =
        { selectedCategory ->
            viewModel.setSelectedCategory(selectedCategory)
            sharedUtil.setLatestCategory(selectedCategory)
            requireActivity().onBackPressed()
        }

    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

В підрозділі «програмна реалізація» було проведено аналіз важливих функцій додатку, було обрано шаблон проектування MVVM, для того щоб правильно працювати із екранами і базою даних.

3.4 Висновки

1. Аналіз мов програмування та середовищ розробки для створення мобільних додатків на платформі Android показав, що переваги має мова програмування Kotlin та середовищ розробки Android Studio, оскільки ця мова та середовище розробки спеціально створені для операційної системи Android і дозволяють розробляти додатки будь-якої складності.
2. Мовою програмування Kotlin з використанням середовищ розробки Android Studio розроблено код мобільного додатку з геолокацією обліку витрат.

4 ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

4.1 Вибір методики тестування

Завдяки безлічі програм, доступних на ринку, мобільні телефони вийшли далеко за межі своєї колись виключно утилітарної цінності. Майже 4 мільйони мобільних додатків доступні для завантаження, тому клієнти мають широкий вибір варіантів. Цей висококонкурентний попит і пропозиція означає, що потрібно переконатися, що якість, зручність використання та безпека додатка не лише відповідають очікуванням, але й перевершують їх.

Але потрібна організація та планування, щоб гарантувати, що буде можливість швидше пройти життєвий цикл розробки програмного забезпечення і, зрештою, швидше вийти на ринок. Знання типів тестування мобільних додатків та їх функцій допоможе у цьому.

Функціональне тестування. Функціональне тестування програмного забезпечення гарантує, що програма працює правильно. Цей тип тестування зосереджується на головній меті та ході програми, гарантуючи, що всі його функції відповідають вимогам і відповідають специфікаціям [14].

Існує безліч стратегій функціонального тестування, і найкращий спосіб забезпечити охоплення функціональним тестуванням – це поєднання ручного та автоматичного тестування.

Найпоширенішими стратегіями функціонального тестування є методи тестування «чорного ящика», коли тестувальнику не потрібно переглядати внутрішній вихідний код, а перевіряє функціональність шляхом тестування різних комбінацій вхідних даних [15].

Нижче наведено деякі з поширених методів функціонального тестування:

- Тестування встановлення – для настільних або мобільних додатків, тестування правильного встановлення;
- Аналіз межових значень – перевірка меж числових входів;
- Розподіл еквівалентності – групування тестів разом, щоб зменшити перекриття подібних функціональних тестів;

- Вгадування помилок – оцінка, де функціональні проблеми, найімовірніше, будуть виявлені, і перевірка їх більш детально, ніж в інших областях;
- Модульне тестування – тестування, що виконується на найменшому рівні програмного забезпечення, не як функціонує система в цілому, а чи правильно працює кожен блок;
- Тестування API – перевіряє, що внутрішні та зовнішні API функціонують належним чином, включаючи передачу даних та авторизацію;
- Регресійне тестування – тести, які проводяться для перевірки того, що нові зміни програмного забезпечення не мали негативного впливу на існуючу функціональність (найпоширеніша методика автоматизації).

Усі функціональні тести мають певний вихід, який очікується від будь-якого входу. Усі функціональні тести можуть бути написані в сценарії з дуже чітким критерієм успішно/не пройдено.

Тестування зручності використання, відоме як тестування користувацького досвіду, перевіряє, наскільки зручним є додаток з точки зору простоти використання та інтуїтивності. В ідеалі тестування юзабіліті зосереджується навколо всього досвіду роботи з клієнтами, що керується додатком, із статистикою, яка включає виявлення помилок та рекомендації щодо способів покращення взаємодії з клієнтами як у програмі, так і поза нею [14].

Важливо, щоб тестування юзабіліті додатків проводилося реальними людьми на реальних пристроях, щоб швидко виявляти та виправляти проблеми з зручністю використання до випуску програми.

Цей тип тестування – це більше мистецтво, ніж наука, і вимагає кваліфікованих тестувальників QA для зручності використання, щоб полегшити тестування та отримати інформацію, яка відображає реальних користувачів або клієнтів програми.

Потрібно завжди мати на увазі наступні ознаки, які варто завжди тестувати:

- Гарне планування та дизайн
- Інтуїтивно зрозумілий
- Час реакції

Більшість користувачів віддають перевагу програмам, які запускаються протягом двох-трьох секунд після відкриття програми? Оскільки тестування юзабіліті є суб'єктивним, потрібно розуміти цільових кінцевих користувачів та їх уподобання.

Інші найкращі методи тестування юзабіліті включають:

- Продумана настройка сценаріїв тесту юзабіліті та анкет відгуків.
- Інтеграція анкет щодо зручності використання в циклі тестування, щоб тестувальники розуміли інструкції з тестування юзабіліті, могли отримати доступ до онлайн-анкет і надати зворотний зв'язок у рамках своїх завдань тестування.
- Аналіз результатів та підсумок відгуків із корисною інформацією та рекомендаціями щодо покращення загального досвіду клієнтів.

Тестування на сумісність – це тип нефункціонального тестування, який має вирішальне значення, щоб переконатися, що ваш мобільний додаток працює на різних операційних системах, пристроях і програмах, мережевих середовищах і з конкретними внутрішніми специфікаціями обладнання.

Зокрема, ви повинні знати, якщо:

- Програма сумісна з різними операційними системами та їх різними версіями (iOS, Android, Windows тощо).
- Додаток добре працює з різними мережами та їх параметрами (пропускна здатність, робоча швидкість тощо)
- Програма сумісна з різними браузерами (Google, Firefox, Safari тощо)
- Програма сумісна з різними пристроями (розмір екрана, сховище даних тощо)

Також слід розглянути два типи тестування на сумісність:

- Назад: тестування поведінки мобільного додатка зі старішими версіями програмного забезпечення
- Наперед: тестування поведінки мобільного додатка з новими версіями програмного забезпечення, включаючи бета-версію.

Тестування продуктивності та навантаження перевіряє, наскільки добре мобільний додаток працює під певним робочим навантаженням. Ці тести є важливими, щоб переконатися, що додаток працює правильно.

Тести продуктивності та навантаження перевіряють наступне:

- Продуктивність пристрою: час запуску, споживання батареї, споживання пам'яті
- Продуктивність мережі: затримки або помилки в отриманні інформації
- Продуктивність API / сервера: як швидко і в якому форматі передаються дані

Крім того, додаток повинен мати вбудовані функції резервного копіювання та відновлення, які зберігають або відновлюють дані користувача, які можуть бути втрачені з будь-якої причини. Використовуючи цей метод тестування можна перевірити цю здатність.

Тестування безпеки. 80 відсотків користувачів «видаляють програму через безпеку». Тому дуже важливо розуміти та поважати тестування безпеки.

Від Tinder до програм для подорожей, деякі програми запитують особисту інформацію користувача. Якщо розроблюваний додаток зберігає таку інформацію, потрібно обов’язково повинні гарантувати конфіденційність, автентичність та цілісність програми. Ось чому тестування QA має надавати пріоритет безпеці даних і тестувати поведінку додатків за різними схемами дозволів пристрою.

Тестування установки. Також відоме як тестування впровадження, тестування встановлення виконується, щоб перевірити, чи правильно встановлюється та видаляється програмне забезпечення.

Крім того, тестування встановлення гарантує, що оновлення також безперервні та безпомилкові. Це включає розуміння того, що станеться, якщо користувач не оновить додаток.

Тестування локалізації. Від перекладу кількома мовами до конвертації в місцеві валюти та дотримання місцевих правил та юридичних вимог важливо забезпечити доступність додатка та його використання на різноманітних ринках. Ось тут на допомогу приходить тестування на локалізацію.

Споживачі зазвичай пропускають попередні програми, графічні елементи або елементи інтерфейсу яких не відповідають їх культурі, мові чи доступності пристрою – вони очікують бездоганний інтерфейс, локалізований відповідно до їхніх потреб і вподобань.

У той же час тестування локалізації продовжує залишатися проблемою, оскільки половина всіх команд QA не має необхідного покриття тестування та доступу до ресурсів, необхідних для тестування локалізації.

Ручне тестування. Тестування мобільних додатків – це складний процес, що включає різноманітні механізми та різні набори навичок.

Команди QA використовують ручне тестування, щоб переконатися, що кінцевий продукт дійсно працює, як задумано. Маючи особливу роль, ручне тестування використовується для вивчення випадків використання, які можуть бути не такими очевидними. Досвідчені тестувальники, проводять багато тестувань, щоб протестувати програму, що розробляється і часто знаходять випадки, коли якась функція працює неналежним чином.

Крім того, не завжди можна автоматизувати деякі типи тестів. До таких належать:

- Тести фізичного інтерфейсу
- Комплексні тести
- Дослідницьке тестування

Автоматизоване тестування. Як було зазначено раніше, є деякі випадки, коли ручне тестування є кращим варіантом. Однак деякі тести QA або занадто виснажливі, або занадто складні для тестувальників. Ось чому розумно виконане автоматизоване тестування разом із ручними тестами може допомогти забезпечити якість та швидше випускати кращі продукти.

Кілька найкращих методів автоматизованого тестування та проблем включають:

- Продуманий дизайн, збірка та підтримка точних тестових сценаріїв;
- Узгодження та інтеграція існуючих інженерних робочих процесів із автоматизованим процесом тестування;

- Створення та обслуговування системи автоматизації тестування, включаючи інфраструктуру;
- Керування тестовими запусками та налаштуваннями;
- Ретельні огляди для підтвердження результатів тестів і дефектів.

Тестування мобільних пристроїв. Мобільні програми не існували б без апаратного забезпечення та операційних систем. Отже, потрібно подумати про тестування мобільних пристроїв, щоб забезпечити якість програмного та апаратного забезпечення. Для мобільних пристроїв існує кілька типів тестування, зокрема:

- Переривання – тестування переривань оцінює, як програма реагує на переривання та чи повертається до попереднього стану. Поширені перебої в мобільних додатках включають втрату заряду акумулятора, вхідний телефонний дзвінок або текстове повідомлення, сповіщення та оновлення програм.
- Служби на основі місцезнаходження (LBS) – використовуючи геодані з мобільного пристрою, послуги на основі місцезнаходження надають інформацію в режимі реального часу, розваги або безпеку. Вони також використовуються споживачами, щоб «зареєструватися», коли відчувають життя в дорозі, наприклад, відвідують місце кафе або під час відвідування концерту.
- Біометричні – мобільні пристрої часто включають біометричні датчики, які включають розпізнавання обличчя, відбитків пальців і геометрію руки, розпізнавання райдужної оболонки і навіть рівень ДНК або інсуліну.
- NFC-платежі – Near Field Communications (NFC) дозволяє мобільним пристроям зв’язуватися з платіжним терміналом, що дозволяє здійснювати безконтактні платежі.

Враховуючи особливості різних видів тестування було вирішено використовувати наступні методики: ручне тестування, автоматизоване тестування та тестування установки. Вибрані види тестування є найбільш доречними для розробки мобільного додатку з геолокацією обліку витрат .

4.2 Тестування програмного додатку

Для початку було проведено ручне тестування та автоматизоване за допомогою Junit тестів. З початку було проведено, тестування авторизації користувача.

Авторизація користувача. Реєстрація у додатку відбувається завдяки соціальним мережам або підтвердженням смс на номер. Для перевірки реєстрації потрібно провести наступні тестування:

1. Реєстрація користувача через соціальні мережі без доступу до мережі. Відбувається перевірка, що буде в результаті натиску користувача на клавіші із соціальними мережами. Чи програма покаже, користувачу повідомлення про те, що користувач немає доступу до мережі чи можливо трапиться таке, що програма немає обробки такої помилки і в результаті закриється. В результаті тестування розроблюваного додатку, користувачу було показано повідомлення про відсутність мережі.
2. Помилка реєстрації користувача через соціальні мережі. Потрібно здійснити перевірку, що буде в результаті, якщо соціальна мережа, по тій чи іншій причині, не надасть дані для реєстрації для реєстрації, а знову ж таки поверне помилку, як програма відреагує на такий результат. Чи буде повідомлення про помилку чи аварійне завершення програми. Після проведення тестування додатку, користувачу було показано повідомлення про помилку.
3. Вхід у програму через соціальні мережі, якщо всі умови виконуються. Було проведено тестування коли доступ до інтернету наявний, і користувач після натиску на одну із клавіш соціальних мереж. В результаті було отримано відповідь із даними від соціальної мережі. Було відкрито головну сторінку мобільного додатку.
4. Реєстрація користувача за допомогою мобільного телефону. Користувачу доступне поле, де він вказує номер мобільного та кнопка підтвердження. Після натиску на неї користувача пересилає на слідуючий екран із полем для введення смс, для підтвердження номеру. Користувач вводить , номер

із смс або система автоматично зчитує цифри із останнього смс. Користувач натискає клавішу підтвердити і програма здійснює запит до бази даних, де здійснюється перевірка на правильність вводу номеру із смс. Під час тестування було проведено декілька операції, а саме введення коректних даних та неправильних. В обох випадках програма відпрацювала успішну.

Для перевірки додавання нової транзакції у додаток потрібно наступні тестування:

1. Перевірка чи активна кнопка збереження, коли поля не заповнені або вказані не вірні дані. У вікні додавання, транзакції доступні клавіші для вибору типу транзакції: витрати та доходи, далі поле для суми транзакції, дата транзакції, категорія та локація. Важливими є поле суми, дата та категорія транзакції. В результаті тестування, при вказаних не коректних даних було показано повідомлення про помилку.
2. Перевірка працездатності клавіш вибору типу транзакції. Було проведено тестування, вибрано тип і заповнено усі поля форми, після цього здійснено натискання клавіші збереження. Після навігації на головну сторінку, було здійснено перевірку із останнім доданим елементом, чи всі вказані дані збігаються і чи тим транзакції вірний. Тестування було здійснено успішно.
3. Перевірка можливості відкриття карти. У формі для заповнення транзакції є клавіша із позначенням карти. Потрібно провести тестування чи буде здійснена навігація користувача до екрану із картою. Результатом тестування є успішним.
4. Перевірка на можливість додавання мітки на карту. Коли користувач відкриває вікно карти, потрібно перевірити чи є можливість ставити мітки, в яких були здійснені витрати. Здійснюється тестування цієї функціональності, результат тестування успішний.
5. Перевірка на працездатність програми, якщо користувач не дав дозвіл для геолокації у додатку. Здійснюється тестування, відкривається додаток,

здійснюється навігація на карту, коли з’являється діалогове вікно із дозволом на геолокацією. Для перевірки роботи вибирається варіант із забороною геолокації у додатку. В результаті тестування програма продовжує свою роботу, але в наслідок скасування геолокації, користувач не зможе побачити своє місцезнаходження на карті. Результат тестування успішний.

6. Тестування клавіші місцезнаходження в екрані карти. Здійснюється тестування, відкривається додаток, здійснюється навігація на карту, коли з’являється діалогове вікно із дозволом на геолокацією, обирається варіант надати дозвіл. Після цього здійснюємо натиск на клавішу місцезнаходження. В результаті тестування програма показала приблизне місцезнаходження користувача.
7. Перевірка функції вказування дати транзакції. У формі для заповнення транзакції є поле з датою. Здійснюємо натиск на клавішу вказування дати, з’являється вікно із календарем. Вибираємо значення і натискаємо клавішу «ОК». В результаті тестування в полі дати з’явиться обране значення. Тестування успішне.
8. Перевірка функції вибору категорії. У формі для заповнення транзакції є поле із категорією. Під час тестування було здійснено натиск на поле «Категорія». Відкрилося вікно із категоріями. У вікні наявний список категорій. Здійснюємо натиск на категорію. Відбувається навігація на сторінку із формою заповнення транзакції. В результаті тестування у полі «Категорія» з’являється обране значення.
9. Перевірка додавання нової категорії. Для перевірки працездатності функції запускається додаток. Здійснюється навігація переходу до форми заповнення транзакції. Потім навігація до вікна вибору категорії. У правому нижньому куті додатку наявна клавіша додавання нової категорії. Здійснюється натиск на клавішу і відкривається вікно створення нової категорії. Заповнюються усі поля, категорія зберігається. В результаті тестування у вікні вибору категорії з’являється нова категорія.

Для тестування функції «Запланована покупка» потрібно здійснити наступні тестування.

1. Перевірка функціонування клавіші додавання нового плану. Для перевірки функції було здійснено навігацію до вікна планувальника. Здійснено натиск на клавішу із символом «+». В результаті тестування повинна відбутися навігація до вікна додавання нового плану.
2. Перевірка функції створення нового нагадування. Для перевірки цієї функції було здійснено навігація до вікна планувальник. Натиснута клавіша «+». У вікні додавання нового плану заповнені всі поля та вказані дата та час. Здійснення натиск на клавішу «Створити нагадування». В результаті тестування здійснюється навігація до вікна із раніше доданими планами. І у вказаний час має з’явиться нагадування.

Описані тестування, їх опис та очікуваний результат записано до таблиці 4.1.

Таблиця 4.1 – Тестування мобільного додатку із геолокацією обліку витрат

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
1	Реєстрація користувача через соціальні мережі без доступу до мережі	Відбувається перевірка, що буде в результаті натиску користувача на клавіші із соціальними мережами. Чи програма покаже, користувачу повідомлення про те, що користувач немає доступу до мережі чи можливо трапиться таке, що програма немає обробки такої помилки і в результаті закриється.	В результаті тестування розроблюваного додатку, користувачу має з’явитися повідомлення про відсутність мережі.	Тест успішний
2	Помилка реєстрації користувача через соціальні мережі	Здійснення перевірки, що буде в результаті, якщо соціальна мережа, по тій чи іншій причині, не надасть дані для реєстрації для реєстрації, а знову ж таки поверне	Після проведення тестування додатку, користувачу було показано	Тест успішний

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
		помилку, як програма відреагує на такий результат. Чи буде повідомлення про помилку чи аварійне завершення програми.	повідомлення про помилку	
3	Вхід у програму через соціальні мережі, якщо всі умови виконуються	Проведення тестування коли доступ до інтернету наявний, і користувач після натиску на одну із клавіш соціальних мереж.	В результаті має прийти відповідь із даними від соціальної мережі.	Тест успішний
4	Реєстрація користувача за допомогою мобільного телефону	Тестування заповнення номеру мобільного та кнопки підтвердження. Після натиску на клавішу, має відбутися навігація на слідуючий екран із полем для введення смс, для підтвердження номеру. Вказується номер із смс або система автоматично зчитує цифри із останнього смс. Здійснюється натиск на клавішу підтвердження і програма здійснює запит до бази даних, де здійснюється перевірка на правильність вводу номеру із смс. Під час тестування було проведено декілька операції, а саме введення коректних даних та неправильних.	В результаті тестування у разі правильний даних, реєстрація буде здійснена в іншому випадку буде повідомлення про помилку.	Тест успішний

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
5	Перевірка чи активна кнопка збереження, коли поля не заповнені або вказані не вірні дані	У вікні додавання, транзакції доступні клавіші для вибору типу транзакції: витрати та доходи, далі поле для суми транзакції, дата транзакції, категорія та локація. Важливими є поле суми, дата та категорія транзакції.	В результаті тестування, при вказаних не коректних даних має з’явитися повідомлення про помилку	Тест успішний
6	Перевірка працездатності клавіш вибору типу транзакції	Вибирається тип і заповнюються усі поля форми, після цього здійснюється натискання клавіші збереження.	В результаті тестування має здійснитися навігація на головну сторінку та проведена перевірка із останнім доданим елементом, чи всі вказані дані збігаються і чи тим транзакції вірний.	Тест успішний
7	Перевірка можливості відкриття карти	У формі для заповнення транзакції є клавіша із позначенням карти. Здійснюється натиск на іконку із картою.	В результаті тестування буде навігація користувача до екрану із картою.	Тест успішний
8	Перевірка на можливість додавання мітки на карту	Перевірка чи є можливість ставити мітки, в яких були здійснені витрати.	Після натиску на карту має з’явитися мітки із символом категорії.	Тест успішний
9	Перевірка на працездатність програми, якщо	Здійснюється тестування, відкривається додаток, здійснюється навігація на карту, коли	В результаті тестування програма продовжує	Тест успішний

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
	користувач не дав дозвіл для геолокації у додатку	з’являється діалогове вікно із дозволом на геолокацією. Для перевірки роботи вибирається варіант із забороною геолокації у додатку.	свою роботу, але без можливості бачити місце-знаходження на карті.	
10	Перевірка функції вибору категорії	У формі для заповнення транзакції є поле із категорією. Під час тестування здійснюється натиск на поле «Категорія». Відкривається вікно із категоріями. У вікні наявний список категорій. Здійснюється натиск на категорію. Відбувається навігація на сторінку із формою заповнення транзакції.	В результаті тестування у полі «Категорія» з’являється обране значення.	Тест успішний
11	Перевірка додавання нової категорії	Для перевірки працездатності функції запускається додаток. Здійснюється навігація переходу до форми заповнення транзакції. Потім навігація до вікна вибору категорії. У правому нижньому куті додатку наявна клавіша додавання нової категорії. Здійснюється натиск на клавішу і відкривається вікно створення нової категорії. Заповнюються усі поля, категорія зберігається.	В результаті тестування у вікні вибору категорії з’являється нова категорія.	Тест успішний

Продовження таблиці 4.1

№	Назва тесту	Опис тестування	Очікуваний результат	Результат тестування
12	Перевірка функціонування клавіші додавання нового плану	Для перевірки функції було здійснено навігацію до вікна планувальника. Здійснено натиск на клавішу із символом «+»	В результаті тестування повинна відбутися навігація до вікна додавання нового плану	Тест успішний
13	Перевірка функції створення нового нагадування	Для перевірки цієї функції було здійснено навігація до вікна планувальник. Натиснута клавіша «+». У вікні додавання нового плану заповнені всі поля та вказані дата та час. Здійснення натиск на клавішу «Створити нагадування».	В результаті тестування здійснюється навігація до вікна із раніше доданими планами. І у вказаний час має з’явиться нагадування.	Тест успішний

Було проведено тестування, які було описано вище, всі тести відпрацювали успішно. Звісно як у будь-якій розробці були помилки, що стало причиною провести роботу над помилками. Коли все було полагоджено, всі тести пройшли успішно.

4.3 Розробка інструкції користувача

Для публікації мобільного додатку з геолокацією обліку витрат в публічний доступ потрібно розробити інструкцію користувача.

Для того щоб запустити мобільний додаток потрібно натиснути на іконку з назвою «Budget Manager». Після запуску мобільного додатку буде відображено головну сторінку додатку (рис. 4.1).

У нижній частині екрану зображена навігаційна панель, на якій представлені клавіші – «на головну сторінку», «на сторінку планувальника» та універсальна клавіша із символом «+» (рис. 4.2).

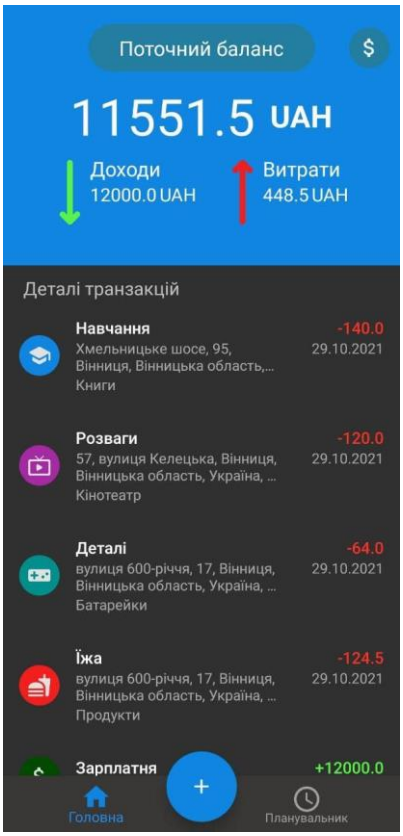

Рисунок 4.1 – Головна сторінка додатку

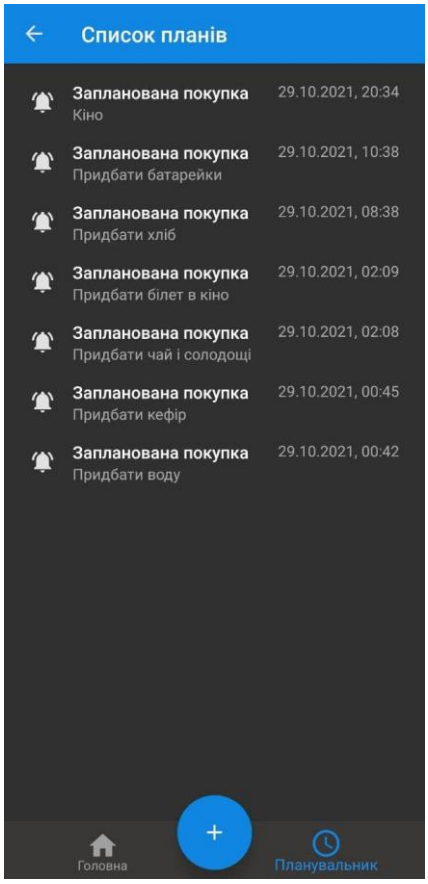

Рисунок 4.2 – Сторінка планувальника

Клавіша «+» працює так:

1. Якщо користувач знаходиться на головній сторінці, то виконується перехід на сторінку додавання нової транзакції (рис. 4.3).
2. Якщо користувач знаходиться на сторінці планувальника, то відкриється сторінка додавання нового плану (рис. 4.4).

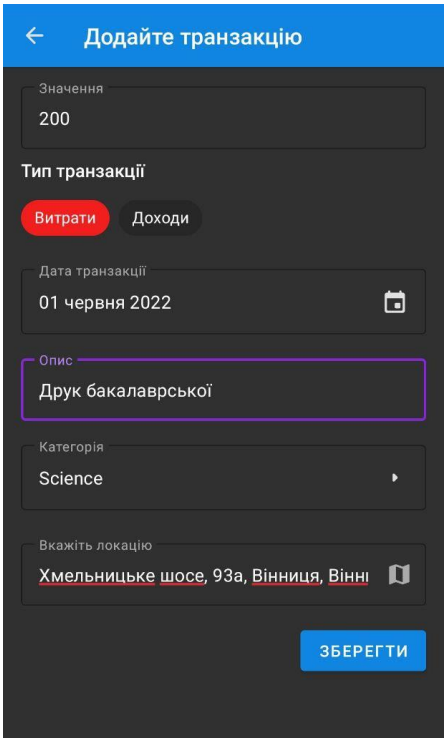

Рисунок 4.3 – Сторінка додавання транзакції

На екрані додавання нової транзакції представлено клавіші вибору типу транзакції, клавіша для внесення дати транзакції, поле для опису транзакції, клавіша вибору категорії та поле для введення локації та клавіша для переходу на карту.

Якщо користувач натисне на клавішу категорія, відкриється екран із списком категорій (рис. 4.5).

Після вибору клавіші «карти» відкриється вікно із картою (рис. 4.6), у якому користувач зможе вказати локацію або використати мітку, яку програма вказала завдяки геолокації.

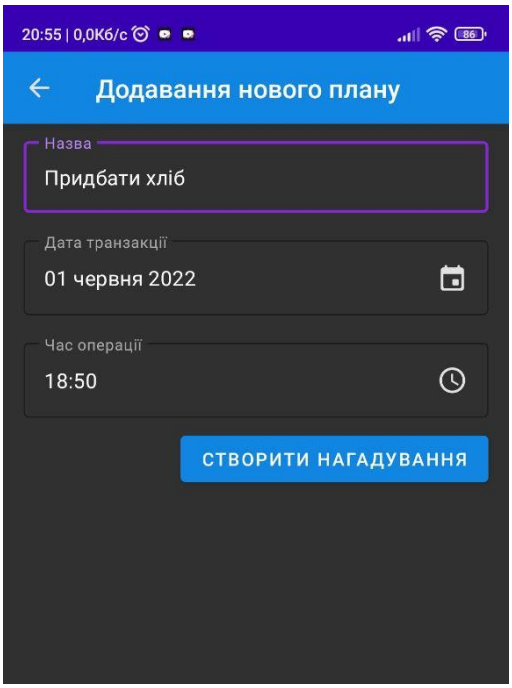

Рисунок 4.4 – Сторінка створення нового плану

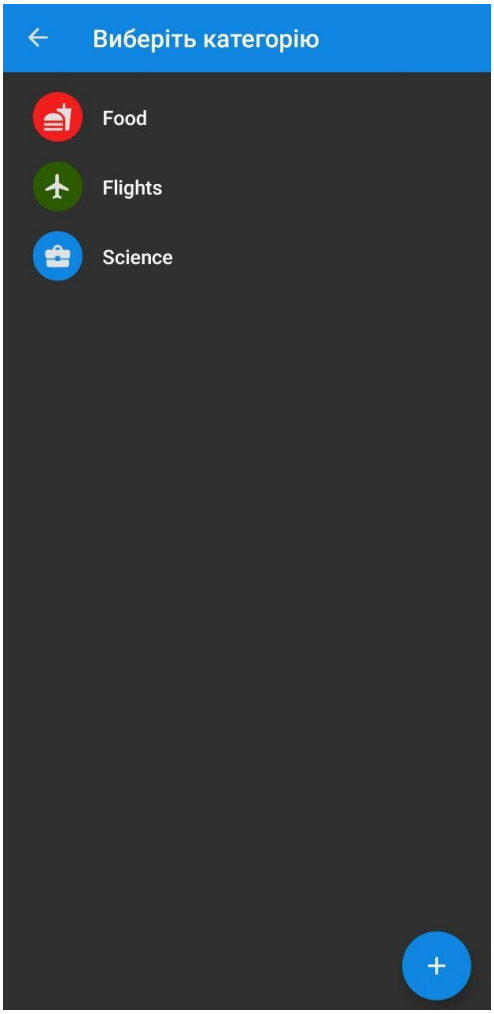

Рисунок 4.5 – Сторінка вибору категорії

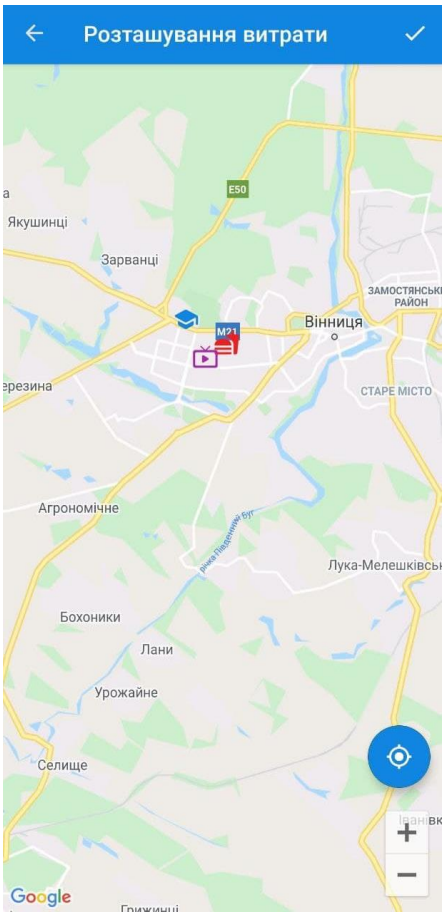

Рисунок 4.6 – Карта у мобільному додатку

В результаті розробки інструкції користувача зрозуміло, що мобільний додаток має зручний інтерфейс користувача, тому для керування ним не потрібно мати спеціальних навиків.

4.4 Висновки

1. Здійснено вибір методики тестування. Наведено найбільш популярні методики. Обрано наступні методики для тестування мобільного додатку із геолокацією обліку витрат: ручне тестування, автоматизоване тестування та тестування установки.
2. Проведено тестування мобільного додатку, описані тестування, які було проведено. Всі тести було успішно відпрацювали, отже додаток можна використовувати для обліку витрат.
3. Розроблено інструкцію користувача.

ВИСНОВКИ

У роботі мовою Kotlin розроблено мобільний додаток з геолокацією обліку витрат для ОС Android. Результати, отримані при виконанні роботи:

- аналіз додатків обліку витрат показав, що функціонал відомих реалізацій не задовольняє потреби багатьох користувачів, зокрема, через відсутність функції геолокації витрат, що підтверджує доцільність розробки додатку обліку витрат з їх геолокацією;
- визначено функціональні характеристики додатку для обліку витрат, що розробляється. Зокрема, додано можливість створення позначок на карті місць розташування об’єктів здійснення витрат коштів, що полегшує користувачам вибір шляхів оптимізації витрат;
- розроблено архітектуру мобільного додатку з геолокацією обліку витрат, особливістю якої є використання як локальної так і хмарної баз даних, що надає можливість поповнювати базу даних витрат навіть в умовах відсутності мережі;
- аналіз мов програмування та середовищ розробки для створення мобільних додатків на платформі Android показав, що переваги має мова програмування Kotlin та середовищ розробки Android Studio, оскільки ця мова та середовище розробки спеціально створені для операційної системи Android і дозволяють розробляти додатки будь-якої складності;
- мовою програмування Kotlin з використанням середовищ розробки Android Studio розроблено код мобільного додатку з геолокацією обліку витрат.
- тестування додатку показало його повну працездатність та відповідність завданню на роботу.
- розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

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

2. Ярмола В.С. Розробка мобільного додатку з геолокацією обліку витрат [Електронний ресурс] / Майданюк В.П., Ярмола В. С. // LI 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

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

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

5. 1Money – облік витрат, бюджет. [Електронний ресурс]. URL: https://play.google.com/ store/apps/details?id=org.pixelrush.moneyiq

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

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

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

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

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

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

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

13. Піт Браун. Silverlight. Практичне керівництво. – Пітер, 2012. – 816с.

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

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

ДОДАТОК А

(обов’язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп’ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедрою ПЗ

д.т.н., проф.

_____ О. Н. Романюк

"25" березня 2022 р.

Технічне завдання

на бакалаврську дипломну роботу «Розробка мобільного додатку з геолокацією обліку витрат» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доцент В.П. Майданюк

"__" _____ 2022 р.

Виконав:

_____ студент гр. 2ПІ-18б В.С. Ярмола

"__" _____ 2022 р.

Вінниця – 2022 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мобільного додатку з геолокацією обліку витрат».

Галузь застосування – контроль фінансів, збереження локацій.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №66 від «24» березня 2022 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є підвищення точності обліку фінансів споживачем за рахунок можливості позначення на карті місць витрат.

Основними задачами дослідження є:

- аналіз відомих мобільних додатків для контролю витрат;
- розробка алгоритму роботи мобільного додатку з геолокацією обліку витрат;
- вибір програмних засобів для вирішення поставлених завдань;
- розробка коду програмного продукту;
- тестування роботи програмного продукту.

4. Вихідні дані для проведення НДР

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

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/application-architecture

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

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

5. Технічні вимоги

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

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

1. Пояснювальна записка до БДР;
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 дипломного проєкту	Строк виконання етапів роботи
1	Обґрунтування вибору методу розробки та постановка задач	26.03.22 – 01.04.22
2	Розробка архітектури та алгоритмів програмного продукту	02.04.22 – 07.04.22
3	Аналіз і вибір мови програмування та середовища розробки	08.04.22 – 11.04.22
4	Розробка програмного продукту	12.04.22 – 01.05.22
5	Тестування програми	02.05.22 – 10.05.22
6	Оформлення матеріалів до захисту БДР	11.05.22 – 10.06.22

10. Порядок контролю та прийняття

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК Б

(обов’язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мобільного додатку з геолокацією обліку витрат

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Майданюк В.П.

Оригінальність	94,5%
Схожість	5,5%

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.

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 мобільного додатку з геолокацією обліку 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

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ З ГЕОЛОКАЦІЄЮ ОБЛІКУ ВИТРАТ

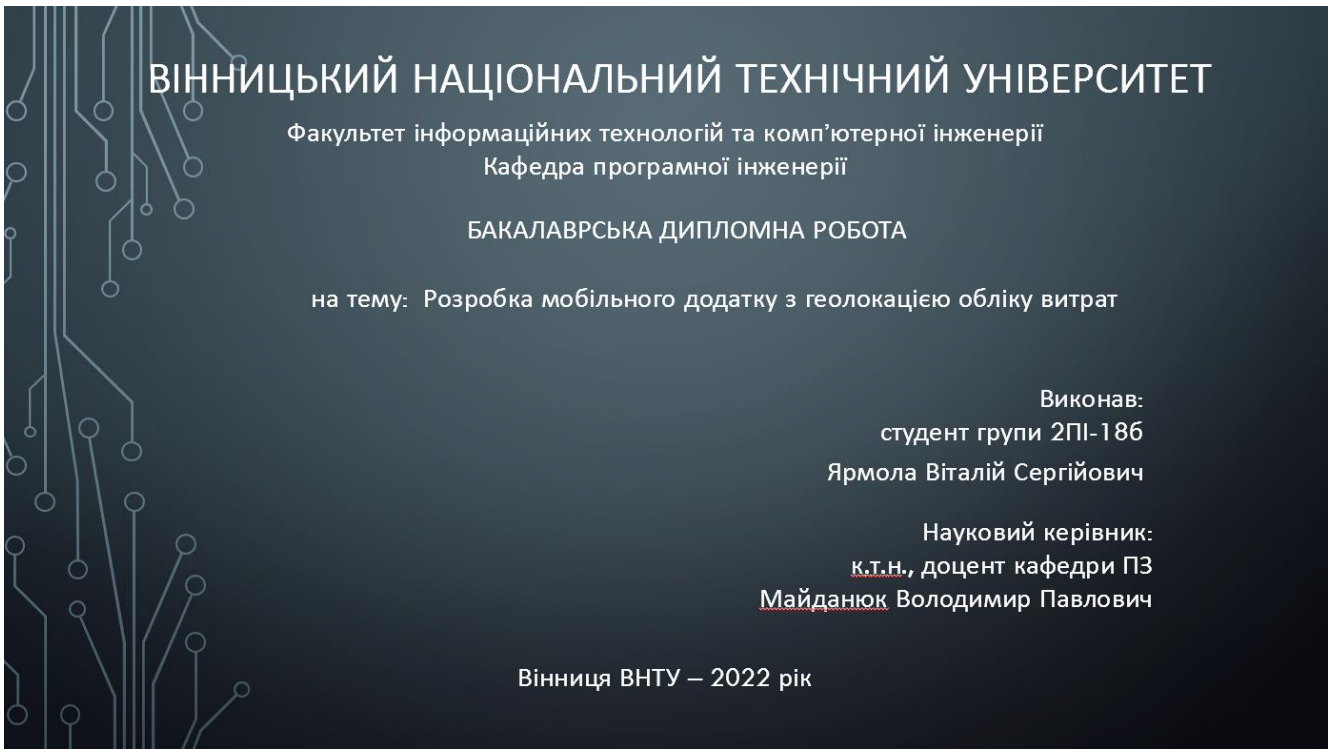

Рисунок Г.1 – Титульний слайд

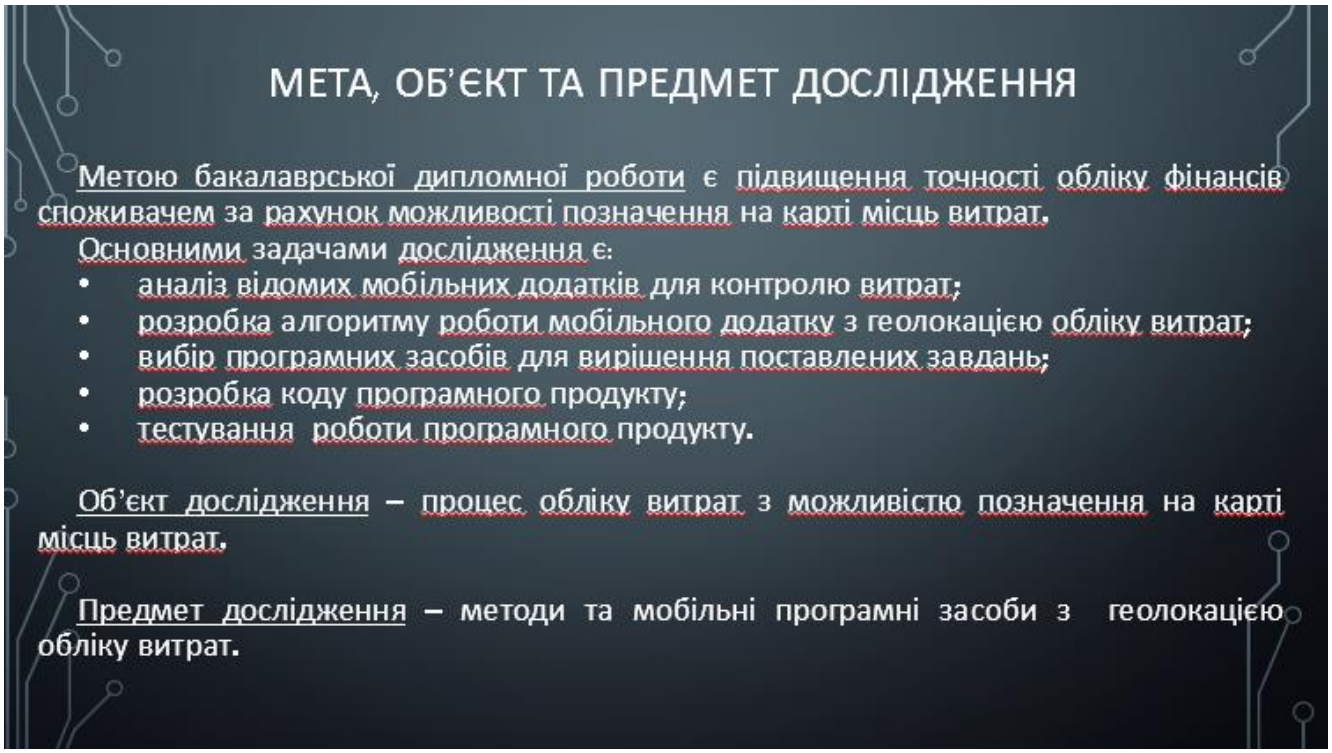

Рисунок Г.2 – Мета, об'єкт та предмет дослідження

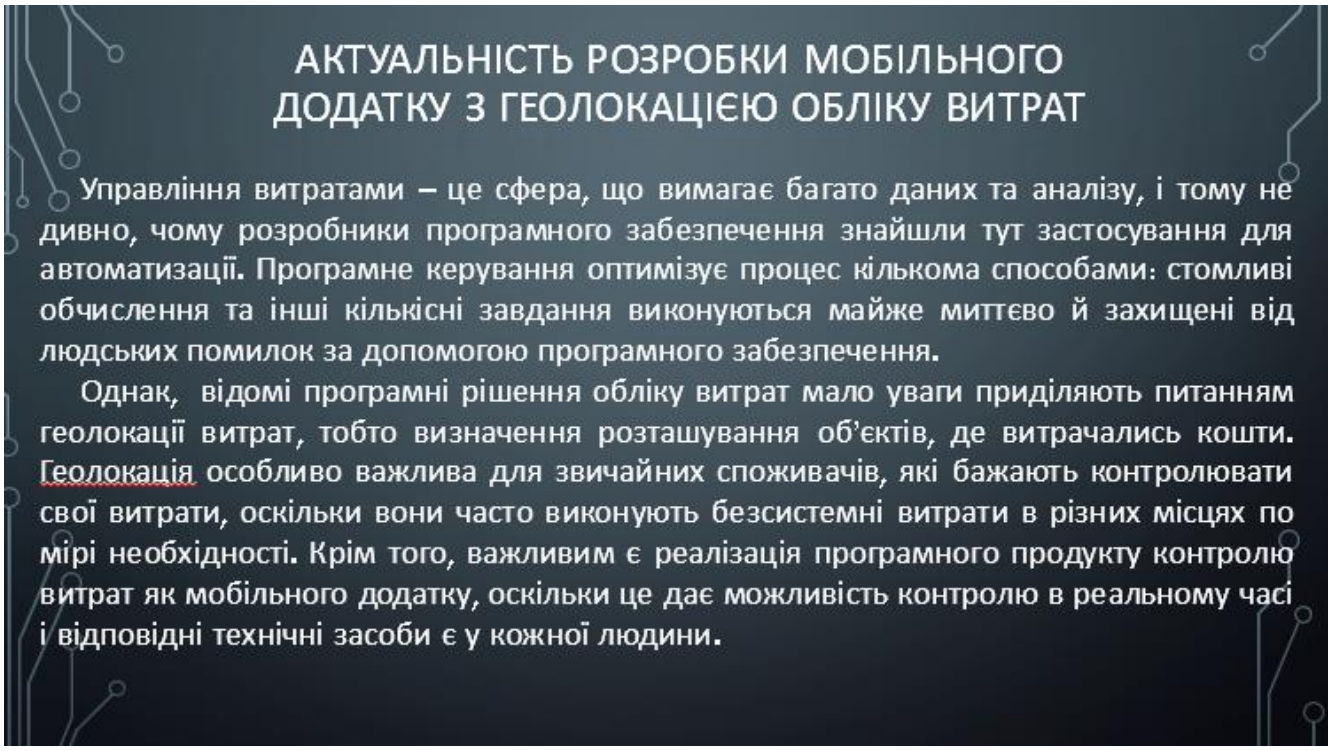

Рисунок Г.3 – Актуальність розробки мобільного додатку з геолокацією обліку витрат

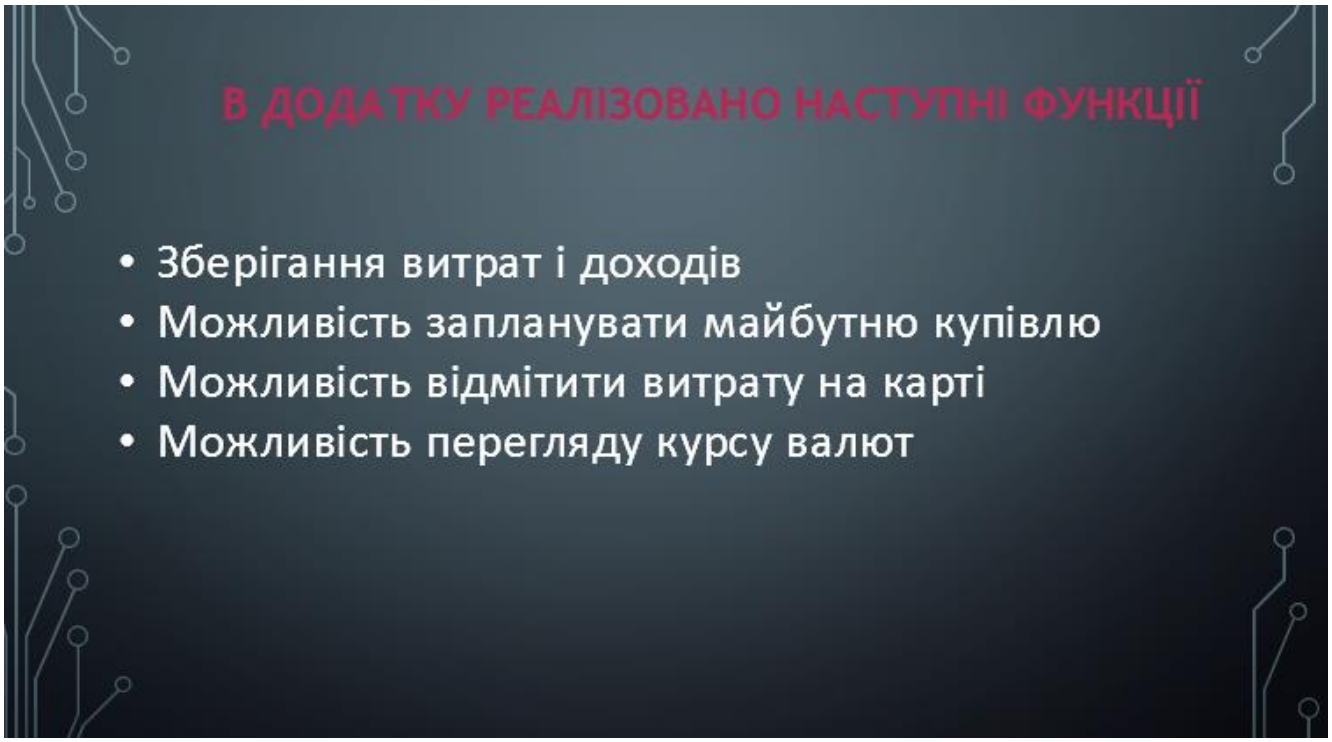

Рисунок Г.4 – Реалізовані функції у додатку

Рисунок Г.5 – Технології

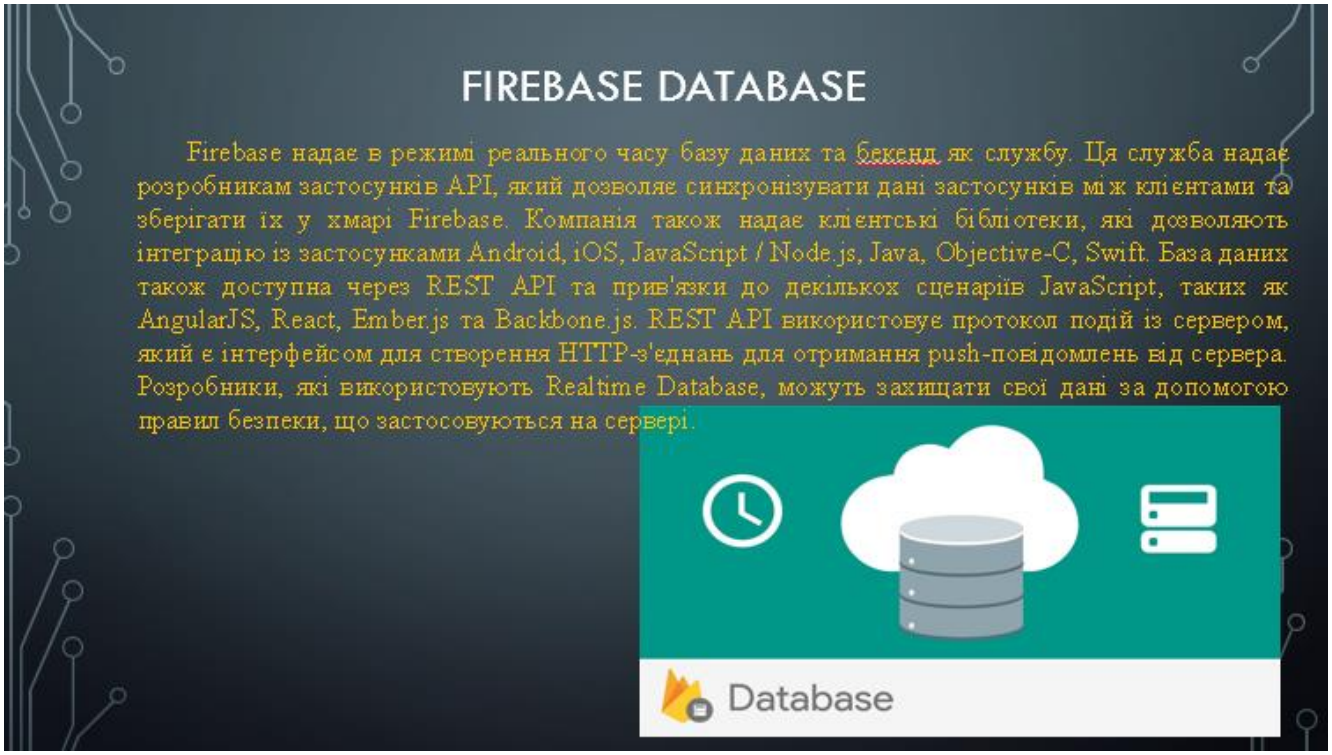

Рисунок Г.6 – Firebase Database

Рисунок Г.7 – Діаграма послідовності мобільного додатку

Рисунок Г.8 – Схематична архітектура додатку

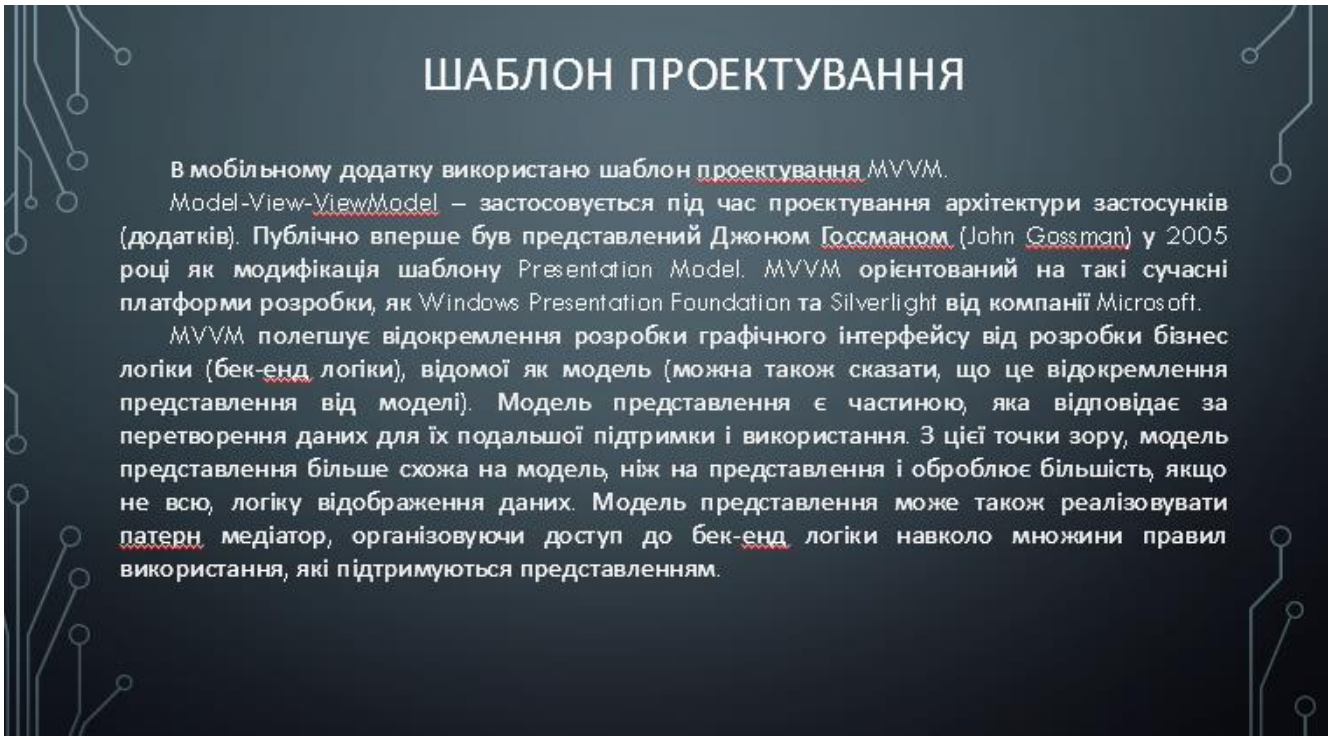

Рисунок Г.9 – Шаблон проектування

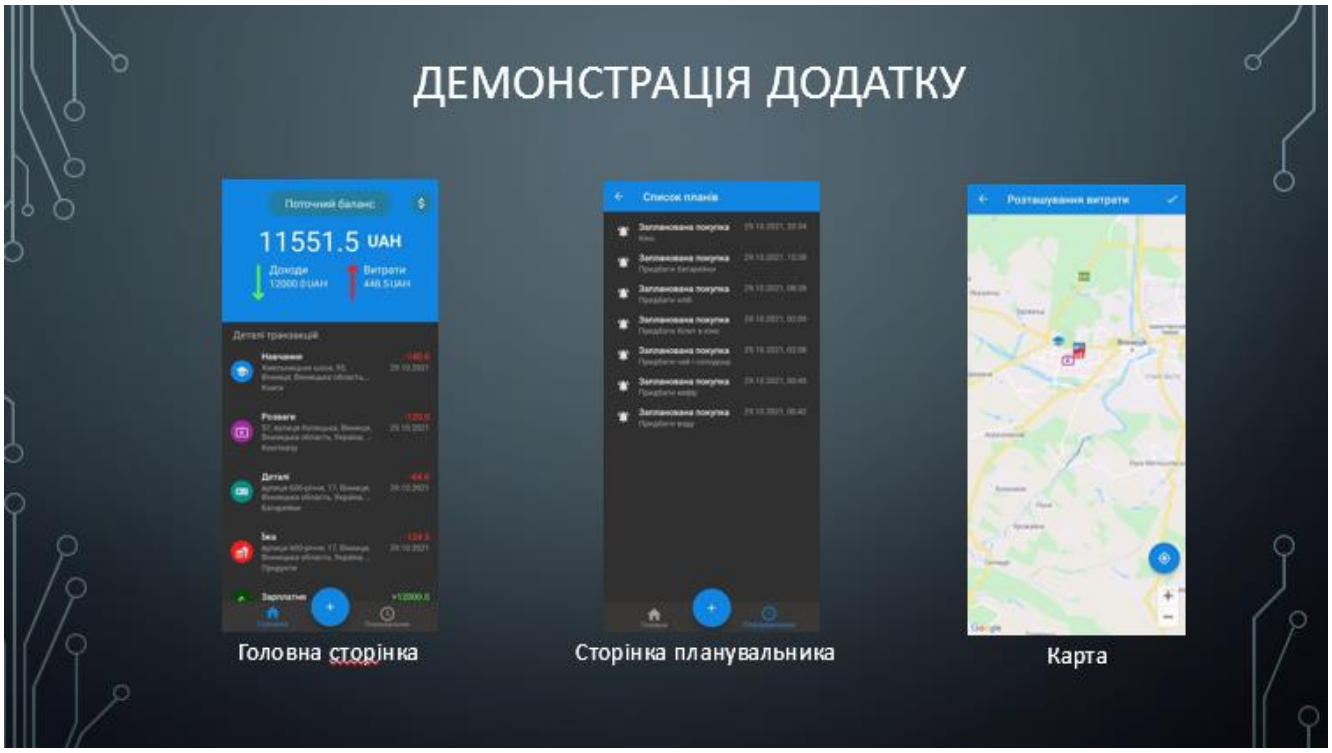

Рисунок Г.10 – Демонстрація додатку

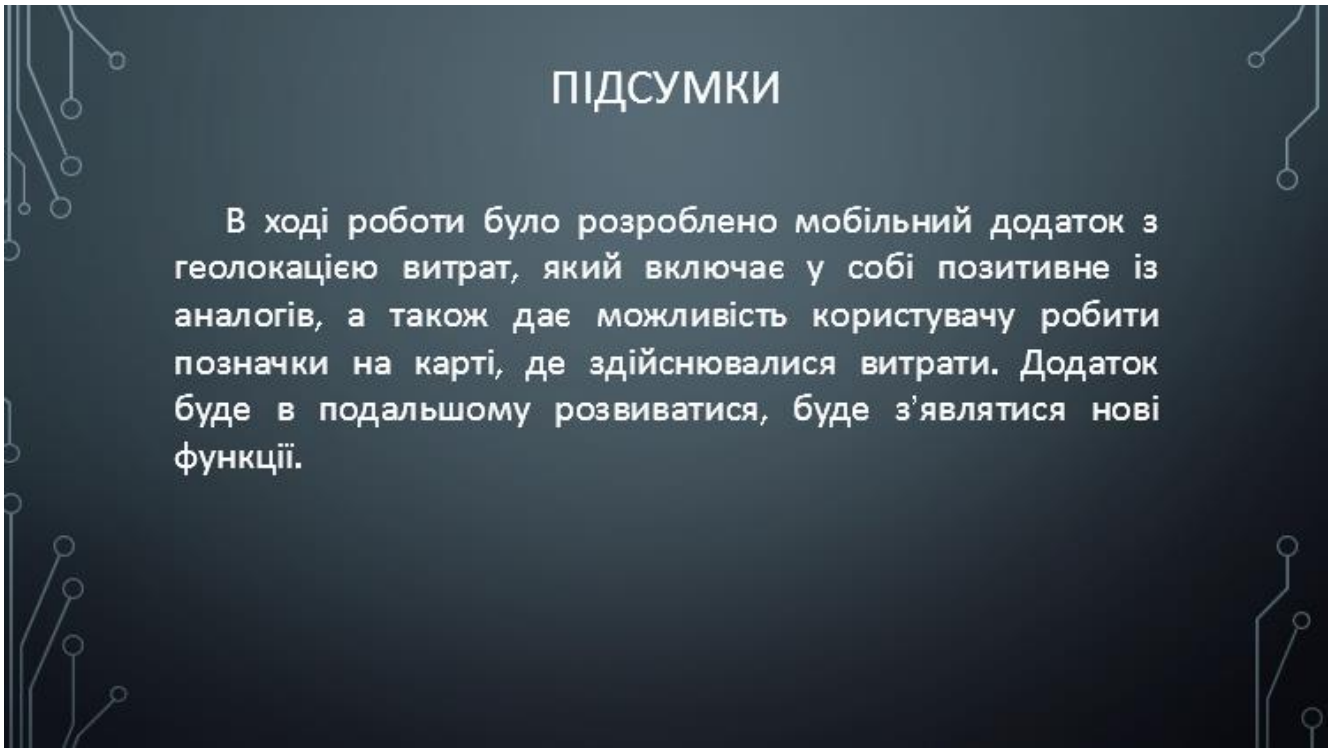

Рисунок Г.11 – Підсумки

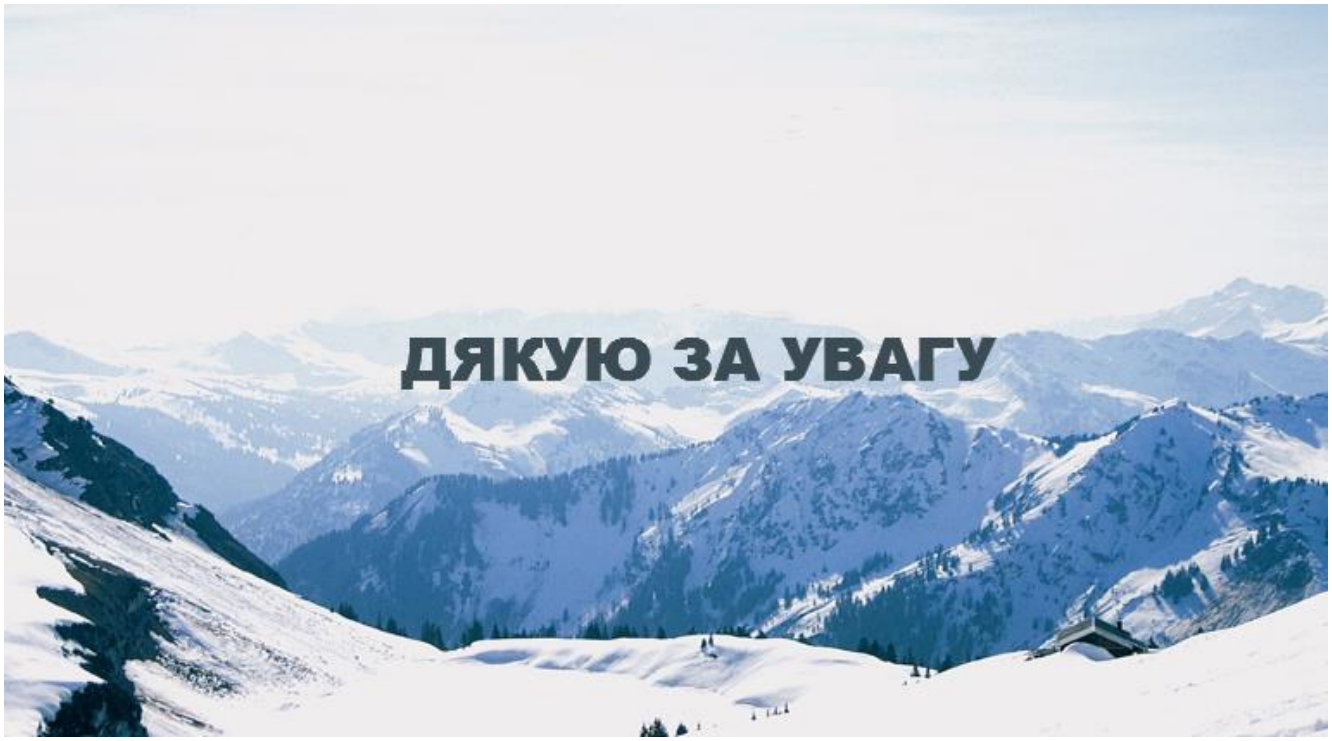

Рисунок Г.12 – Фінальний слайд