Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Дипломна робота

бакалавра

з теми: “РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ КОНТРОЛЮ ПЕРСОНАЛЬНИХ ФІНАНСІВ”

Виконала: студентка 4 курсу, KN1-B20 групи
спеціальності 122 “Комп’ютерні науки”
Матраєва Ельвіра Миколаївна
Керівник:
Моцик Ростислав Васильович
доцент кафедри комп’ютерних наук,
кандидат педагогічних наук, доцент
Рецензент:
Сморжевський Ю.Л.
доцент кафедри математики,
кандидат педагогічних наук, доцент

Кам’янець-Подільський, 2024 р.

АНОТАЦІЯ

Тема: розробка мобільного додатку для контролю персональних фінансів.

Виконавець: студентка 4 курсу бакалаврського рівня вищої освіти групи KN1-B20 Матраєва Ельвіра Миколаївна.

Науковий керівник: Моцик Ростислав Васильович, кандидат педагогічних наук, доцент кафедри комп'ютерних наук.

Консультант: Сморжевський Юрій Людвігович, кандидат педагогічних наук, доцент кафедри математики.

В даній дипломній роботі розглянуто різноманітні інструменти, які допомагають планувати бюджет та підвищувати фінансову грамотність. Використання мобільних застосунків для контролю фінансів допомагає знизити ризик зловживання кредитними картками та уникнути банкрутства.

Проведено аналіз трьох мобільних застосунків, доступних у Google Play, для контролю персональних фінансів та досліджено їх переваги і недоліки.

Описано роботу, можливості та переваги середовища програмування Android Studio, бібліотеки Room, мови програмування Kotlin і бази даних SQLite. Також процес розробки, логіку і надано фрагменти коду програми.

Проведено тестування застосунку, показано роботу програми і надані рекомендації із її використання.

Ключові слова: фінансова грамотність, мобільний застосунок, нейронна мережа, вебсторінка, аналіз даних.

ANNOTATION

Subject: development of a mobile application for controlling personal finances.

Performer: student of the 4th year of the bachelor's level of higher education, group KN1-B20 Matrayeva Elvira Mykolaivna.

Academic supervisor: Rostislav Motsyk, Candidate of Pedagogical Sciences, Associate Professor of the Department of Computer Sciences.

Consultant: Yury Smorzhevsky, Candidate of Pedagogical Sciences, Associate Professor of the Department of Mathematics.

This qualification paper examines various tools that help plan the budget and increase financial literacy. Using mobile apps to manage your finances helps reduce the risk of credit card abuse and avoid bankruptcy.

An analysis of three mobile applications available on Google Play for controlling personal finances was carried out and their advantages and disadvantages were investigated.

The operation, capabilities and advantages of the Android Studio programming environment, the Room library, the Kotlin programming language and the SQLite database are described. Also, the development process, logic and code fragments of the program are provided.

The application was tested, the operation of the program was shown and recommendations for its use were provided.

Keywords: financial literacy, mobile application, neural network, web page, data analysis.

ВСТУП

Розвиток інтернет-банкінгу та стрімкий ріст безготівкових платежів призвели до безконтрольних витрат, що зумовило створення програм для контролю фінансів. Мобільні додатки для контролю фінансів надають можливість швидко та зручно вести облік витрат і доходів. Споживачі все більше отримують доступ до онлайн-платформ та інтернет-банкінгу, що забезпечує розвиток мобільних додатків для фінансового управління [1]. Мобільні застосунки дозволяють користувачам з електронними гаманцями та картками в режимі реального часу відстежувати свої витрати та проводити фінансові транзакції. Розробники програм пропонують різноманітні інструменти, які допомагають планувати бюджет та підвищувати фінансову грамотність. Багато застосунків мають можливість автоматичного імпортування фінансових транзакцій з банківських рахунків, що спрощує процес внесення даних. Використання мобільних додатків для контролю фінансів допомагає знизити ризик зловживання кредитними картками та уникнути банкрутства [2]. Інформація, отримана з мобільних програм для фінансового управління, дає можливість розуміти, які витрати необхідно скоротити, що дозволяє економити кошти. Оскільки мобільні додатки доступні на всіх типах смартфонів, вони є універсальним інструментом для кожного, хто бажає ефективно управляти своїми фінансами.

Актуальність теми: правильне та ефективне контролювання фінансів є важливим елементом повсякденного життя, особливо у період економічних криз, тому розробка мобільних додатків для контролю персональних фінансових операцій є актуальною і важливою темою для дослідження.

Мета роботи: розробити мобільний застосунок для контролю фінансів.

Завдання:

- розглянути аналогічні програмні розробки;

- проаналізувати існуючі засоби розробки мобільних додатків;

- визначити особливості та переваги використання мобільних додатків для контролю фінансів;

- розробити мобільний застосунок для контролю фінансів та перевірити його ефективність;

- реалізувати базу даних для застосунку;

- протестувати програму.

Об’єкт дослідження: процес розробки мобільних додатків, зокрема Android застосунку з базою даних SQLite.

Предмет дослідження: розробка прототипу мобільного додатку для контролю персональних фінансів.

Структура роботи. Дипломна робота бакалавра складається зі вступу, двох розділів, висновку, списку використаних джерел.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ЗАСОБІВ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ ТА ОПИС ВИКОРИСТАНИХ ІНСТРУМЕНТІВ

1.1 Дослідження засобів розробки мобільних застосунків

На сьогоднішній день існує багато засобів розробки мобільних додатків. Серед найбільш популярних можна виділити Android Studio, Xcode, React Native, Flutter та Ionic. Android Studio – це безкоштовне інтегроване середовище розробки для розробки додатків під Android. Xcode – це IDE для створення додатків під iOS, доступна лише для MacOS, що вимагає від розробників купівлі дорогих пристроїв MacBook. React Native – це фреймворк для розробки мобільних застосунків, заснований на JavaScript у вільному доступі для всіх розробників. Flutter – це середовище для розробки мобільних додатків, який базується на програмуванні з використанням мови Dart, що дозволяє розробляти кросплатформенні програми для Android та iOS. Ionic – це фреймворк для розробки хмарних додатків, який заснований на веб-технологіях, таких як HTML, CSS та JavaScript. Один з найбільших недоліків простих фреймворків, таких як Ionic, полягає в тому, що вони не можуть створювати додатки з використанням функцій, що є специфічними для деяких платформ. Flutter та React Native, навпаки, дозволяють створювати мобільні додатки, які працюють на обох платформах. Android Studio та Xcode забезпечують повний доступ до функцій платформи та обробці даних, що забезпечує більшу продуктивність та ефективність розробки. У порівнянні з IOS, Android Studio має дуже широке коло функцій для розробки додатків. Одним з ключових переваг Flutter є можливість розробляти додатки на багатьох різних платформах, таких як Android, iOS, Windows, MacOS тощо. React Native також має чимало переваг, наприклад, дозволяє відображення додатків на iOS та Android одночасно. Ionic відмінно підходить для розробки маленьких проектів або маленьких додатківвізиток.

Від вибору засобу розробки залежить продуктивність, складність та вартість розробки мобільного додатку.

1.2 Огляд середовища програмування Android Studio

Android Studio – офіційне інтегроване середовище розробки (IDE) для операційної системи Android, створене на основі програмного забезпечення IntelliJ IDEA від JetBrains і розроблене спеціально для розробки Android [4]. Окрім потужного редактора коду та інструментів розробника IntelliJ, Android Studio пропонує ще більше функцій, які підвищують продуктивність під час створення програм для Android, наприклад:

- швидкий і багатофункціональний емулятор;
- уніфіковане середовище, де можна розробляти для всіх пристроїв Android;
- шаблони коду та інтеграція GitHub, які допоможуть вам створювати загальні функції програми та імпортувати зразок коду; - гнучка система збирання на основі Gradle; - підтримка C++ і NDK [5].

Середовище Android Studio призначене як для одного розробника мобільних застосунків, так і для великих міжнародних організацій з різними системами управління версіями. Досвідчені розробники можуть вибрати інструменти, які найбільше підходять для масштабних проектів. Програми для Android розробляються в Android Studio за допомогою Kotlin, Java або C++ [6].

Для створення нового проекту потрібно обрати шаблон додатку, що буде генерувати основний код програми. Далі, все, що потрібно зробити – це вибрати назву і пакет для додатку. В Android Studio є панель інструментів в головному вікні для відображення різних компонентів вашого проекту, таких як файли, вікна редактора, список емуляторі і т.д. Редактор коду має потужний інструмент для редагування, що включає в себе автозавершення

кодування, прискорення відлагодження, систему контролю версій і багато іншого. Консоль Android Studio дає можливість виводити консольні повідомлення, що надходять з коду або відлагодження, зручно розгорнути всі діагностичні повідомлення, закриваючи шлях для помилкових рішень. Для тестування програми на різних пристроях та версіях операційної системи достатньо встановити та настроїти емулятор.

У результаті огляду Android Studio можна зробити висновок, що це є потужний інструмент для розробки Android-додатків, який містить багато корисних функцій та інструментів для розробки та тестування. Особливо варто відзначити можливості автоматизації процесу розробки, таких як швидкість компіляції та підтримка вбудованих інструментів розробки інтерфейса користувача. Крім того, підтримка гнучкої системи плагінів допомагає розширити функціональність Android Studio залежно від потреб користувача. Загалом, Android Studio має великий потенціал для використання в розробці Android-додатків і може стати невід'ємною частиною роботи розробника.

1.3 Огляд мови програмування Kotlin

Kotlin – це мова програмування високого рівня, яка розробляється компанією JetBrains [7]. Вона є мультиплатформенною мовою, тому розробники можуть використовувати Kotlin для написання програм для різних операційних систем, включаючи Android, iOS, Windows, Linux та MacOS.

Kotlin – це статично типізована мова програмування, що робить її більш надійною та безпечною, а також дозволяє розробникам швидко виявляти й усувати помилки. Вона має вбудовану підтримку функціонального програмування, що дозволяє писати чистий та зрозумілий код. Kotlin – це проект із відкритим кодом, доступний безкоштовно за

ліцензією Apache 2.0. Код для проекту відкрито розробляється на GitHub переважно командою JetBrains за участі Google та інших [8].

Використовуючи Kotlin для розробки Android, можна отримати такі переваги:

- менше коду в поєднанні з більшою читабельністю.
- менше типових помилок (згідно з внутрішніми даними Google, програми, створені за допомогою Kotlin, мають на 20% менше шансів вийти з ладу);
- підтримка Kotlin у бібліотеках Jetpack (Jetpack Compose – рекомендований сучасний інструментарій Android для створення інтерфейсу користувача в Kotli, розширення KTX додають функції мови Kotlin, як-от співпрограми, функції розширення, лямбда-вирази та іменовані параметри до існуючих бібліотек Android);
- підтримка мультиплатформенної розробки (Kotlin Multiplatform дозволяє розробляти не лише Android, але й iOS, серверні та веб-програми. Деякі бібліотеки Jetpack вже є мультиплатформенними. Compose Multiplatform, декларативна структура інтерфейсу користувача JetBrains на основі Kotlin і Jetpack Compose, дає змогу ділитися інтерфейсами користувача між платформами – iOS, Android, настільним комп’ютером та Інтернетом.);
- зріла мова і середовище (З моменту створення в 2011 році Kotlin постійно розвивався не лише як мова, але й як ціла екосистема з надійними інструментами. Тепер він плавно інтегрований в Android Studio і активно використовується багатьма компаніями для розробки додатків Android.);
- взаємодія з Java (є можливість використовувати Kotlin разом із мовою програмування Java у своїх програмах без необхідності переносити весь свій код на Kotlin [9].

Ще однією з переваг Kotlin є його синтаксис, який дуже схожий на Java, що дозволяє розробникам легко переходити з Java на Kotlin та навпаки.

Однак, Kotlin також має свої унікальні особливості, такі як нулівська безпека, функціональні типи даних та Lambda-вирази.

Kotlin має велику спільноту розробників та багато корисних інструментів, таких як IntelliJ IDEA, Android Studio, та засоби розробки на Kotlin веб-додатків. Ця мова програмування заслужила довіру великих компаній, таких як Google, Netflix, та Pinterest, що вже успішно використовують Kotlin в своїх проектах.

Усі ці переваги зробили Kotlin однією з найшвидше зростаючих мов програмування. Вона стала популярною серед розробників, які шукають нові методи програмування інноваційних додатків на різних платформах.

1.4 SQLite та бібліотека Room

SQLite – реляційна система керування базами даних [10]. Втілена у вигляді бібліотеки мовою C, яка реалізує невелику, швидку, самодостатню, високонадійну, повнофункціональну систему баз даних SQL. SQLite є найбільш використовуваним механізмом баз даних у світі. Вона вбудовано в усі мобільні телефони та більшість комп’ютерів і поставляється разом із незліченною кількістю інших програм, якими люди користуються щодня [11].

Наведемо деякі переваги використання SQLite для зберігання даних.

1. Легкість використання: SQLite – це база даних, яка дуже легко встановлюється на всіх платформах, на яких можна виконувати програми. Програмісти можуть працювати з SQLite, не маючи спеціалізованих знань у сфері баз даних.

2. Швидкість: SQLite – це дуже швидкий інструмент, який може ефективно зберігати великі об'єми даних. Він працює без сервера баз даних та не потребує підтримки бази даних.

3. Кросплатформність: SQLite може використовуватися на різних платформах, включаючи Windows, Mac OS, Linux, Android та iOS.

Room – це бібліотека Android, яка забезпечує рівень абстракції над SQLite, щоб забезпечити вільний доступ до бази даних, одночасно використовуючи всю потужність SQLite [12]. Основні переваги:

- оптимізований підхід до роботи з базами даних SQLite, що дозволяє зменшити час створення та управління базою даних;
- уніфікований та зручний підхід до створення запитів до бази даних

SQLite;

- легко налаштовується та масштабується. Room може працювати з будьякими розмірами бази даних, а також з будь-якою кількістю таблиць;
- робота з Room легка та зрозуміла. За допомогою анотацій можна швидко та просто створити модель даних та таблиці для бази;
- повний контроль над часом життя бази даних та її складових.

Принципові складові Room:

1. Database – основна складова Room, яка інкапсулює SQLite базу даних. Клас повинен містити абстрактні методи для кожної таблиці в базі даних, анотовані спеціальними анотаціями Entity.

2. Entity – це клас-модель, який відображає сутності у таблицях бази даних. Анотація Entity дозволяє вказати назву таблиці у базі даних та назви її полів.

3. DAO (Data Access Object) - складова Room, яка дозволяє створювати запити до бази даних і виконувати CRUD операції (створення, читання, оновлення та видалення). Інтерфейс повинен містити анотовані SQL-запити.

4. TypeConverters – це складова, яка дозволяє конвертувати складні типи даних в базі даних SQLite в простіші. Наприклад, перетворити списки в

рядки. Дозволяє значно зменшити кількість SQL-коду в додатку та полегшити роботу зі складними типами.

Отже, Room дозволяє значно скоротити час на розробку програм, що використовують бази даних. Це повноцінний інструмент, який забезпечує всім необхідним для роботи з локальними базами даних, зменшує кількість коду і збільшує швидкість роботи додатку.

1.5 Огляд та аналіз аналогічних програмних розробок

На даний момент існує багато програм для комп'ютерів та смартфонів, які використовуються для контролю витрат та ведення обліку фінансів. Скачати подібні додатки для мобільних пристроїв можна на платформах Google Play Store для користувачів операційної системи Android та App Store для користувачів Iphone.

Одним із популярних додатків доступних у Google Play Store є «Wallet» (рис. 1.1). Розробником додатку є компанія «BudgetBakers.com». Даний додаток дозволяє записувати витрати та розподіляти їх за категоріями, доходи, перекази, коригувати баланс, створювати список покупок і надає статистику грошових операцій.

Ще одним популярним додатком у Google Play Store який використовують для обліку витрат є «Журнал витрат» (рис. 1.2). Розробником є «Vitaliy Vovchok». Дана програма має менший функціонал, ніж попередня, але у неї присутні всі необхідні функції, що робить її простішою у використанні.

Третім популярним мобільним додатком для обліку витрат є «Monefy» (рис. 1.3). Розробником є «Reflectly». Даний додаток надає можливість вести облік своїх доходів та витрат, створювати різні категорії для витрат, а також вибирати зручний перегляд інформації в графічному вигляді.

Дані додатки призначені для ведення обліку фінансових операцій та витрат. Порівняння програм:

1. Можливості: Wallet та Monefy мають більше можливостей, таких як планування бюджету, аналіз витрат за категоріями, підтримку кількох валют тощо. Журнал витрат має менше можливостей і в основному призначений для того, щоб зберігати записи про витрати, але у ньому також присутні підтримка кількох валют та формування звіту в таблицях.

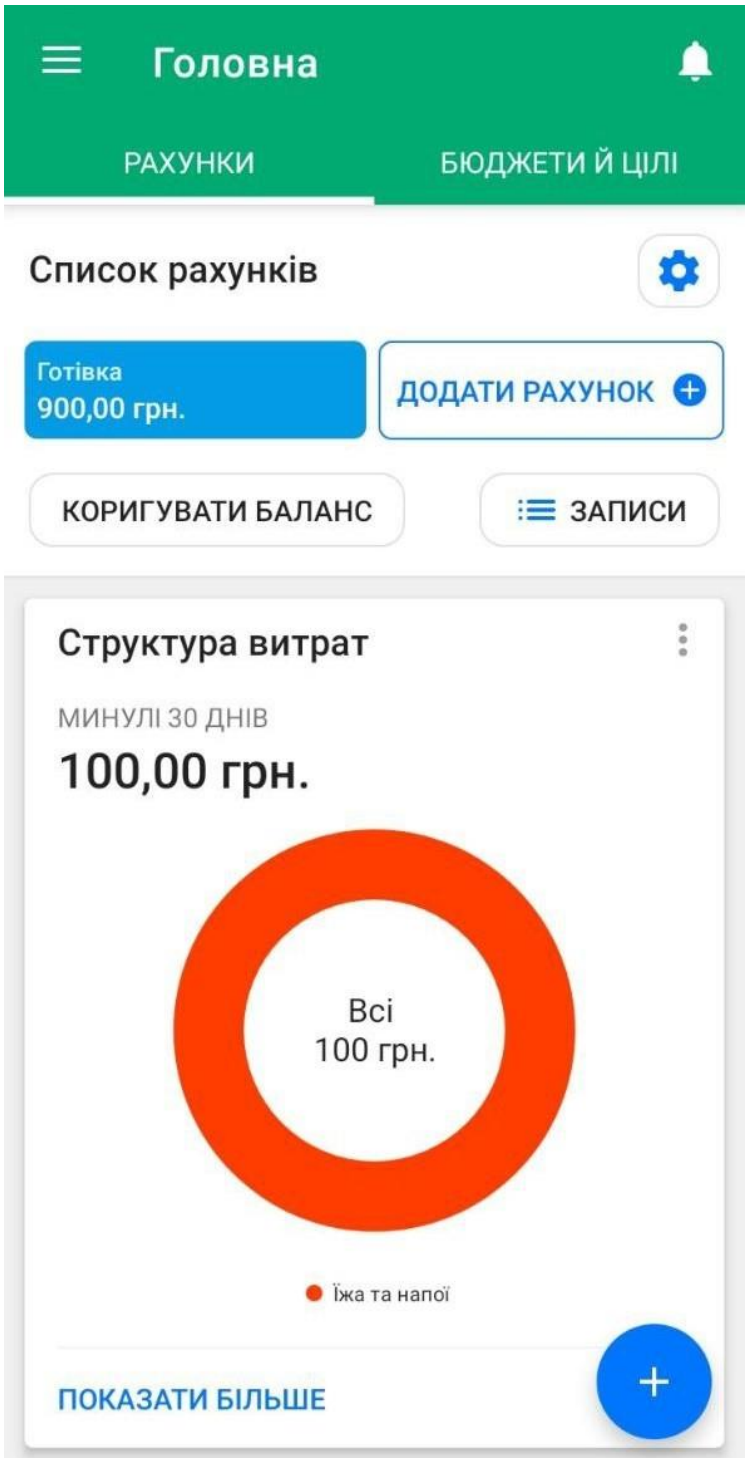


Рисунок 1.1 - Додаток «Wallet»

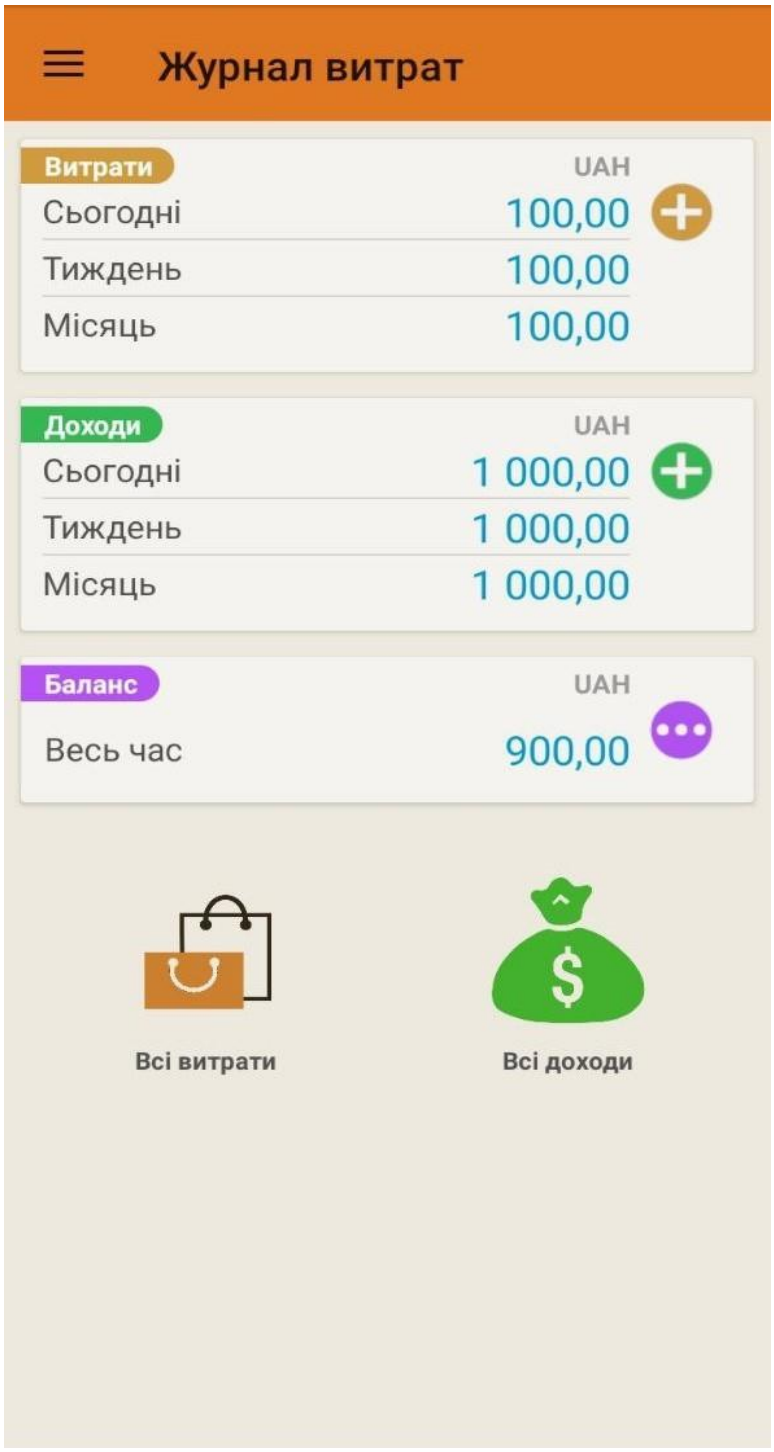


Рисунок 1.2 - Додаток «Журнал витрат»

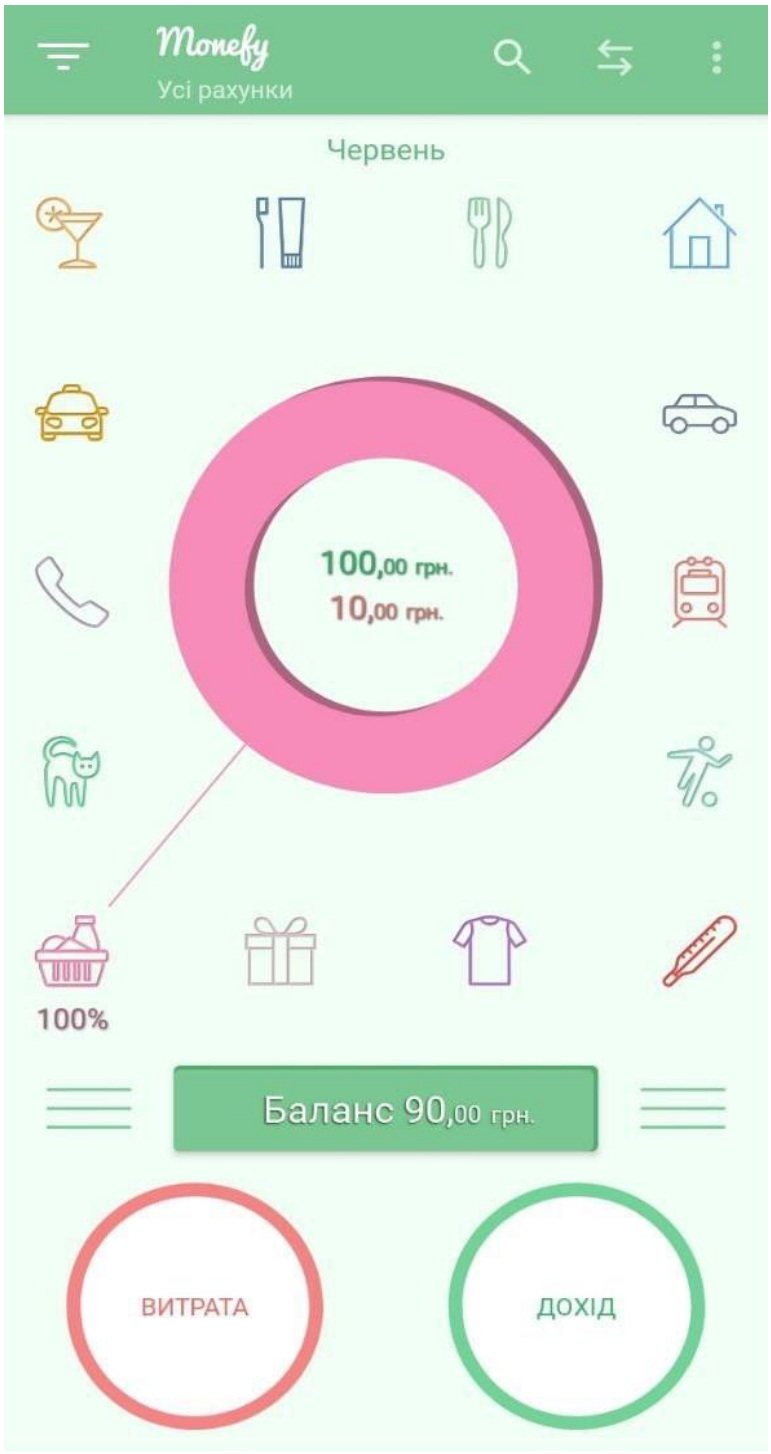


Рисунок 1.3 - Додаток «Monefy»

2. Дизайн та інтерфейс: Wallet і Monefy мають більш сучасний, естетичний дизайн та зручний інтерфейс. Журнал витрат має більш простий дизайн.

3. Мультиплатформність: Wallet доступний для використання на різних платформах, включаючи Android, IOS та веб-версії. Журнал витрат і Monefy можна скачати лише у Google Play, тому вони доступні для використання тільки на платформі Android.

4. Спільний доступ: Wallet дозволяє ділитися доступом до додатку з різними користувачами, що дуже зручно, коли ведеться спільний бюджет. Журнал витрат та Monefy не мають спільного доступу.

5. Ціна: Wallet, Monefy та Журнал витрат є безкоштовними додатками, проте для отримання додаткових можливостей у Wallet, необхідно підписатися на річний платний пакет. Журнал витрат також надає додаткові послуги такі як

експортування даних в таблиці Excel, відключення реклами та додаткові віджети за платну підписку.

6. Безпека: Wallet, Monefy та Журнал витрат надають можливість зберігати дані у хмарному сховищі якщо ви підєднали акаунт Google до застосунку, що дозволяє зберегти дані при видалені програми та відновити при повторному скачуванні, але лише у Журнала витрат є вбудована функція захисту паролем, що підвищує безпеку додатку перед несанкціонованим доступом.

Наявність основних функцій застосунків представлено у таблиці 1.1

Отже, існує декілька відмінностей між додатками, в тому числі і в тому, як вони ставляться до податків та бюджетування. Якщо потрібен більш інтуїтивно - зрозумілий та простий додаток для ведення записів про витрати, то Журнал витрат може бути кращим вибором. З іншого боку, якщо є потреба у розширених можливостях та більш сучасному дизайні, тоді Wallet чи Monefy може бути кращим вибором.

Таблиця 1.1

Функції	**Monefy**	**Wallet**	**Журнал витрат**
Платні функції	Є	Є	Є
Категорії витрат	Є	Є	Є
Синхронізація даних	У про-версії	Є	Є
Режим розподілу витрат	Є	Є	Немає
Планування витрат	Є	Є	Є
Аналітика по витратах	Є	Є	Немає
Графіки витрат	Є	Є	Є
Відстеження банківських операцій	Немає	Є	Немає
Підтримка декількох валют	Є	Є	Немає

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

2.1 Постановка задачі, призначення і вимоги до мобільного застосунку

Для досягнення поставленої мети слід розв’язати такі задачі:

- створення екрану запуску;
- створення інтерфейсу всіх екранів програми;
- підключення і створення бази даних;
- реалізація основного функціоналу додатку;
- тестування програми.

Мобільний застосунок призначений для індивідуального користування. Він не потребує підключення до мережі та повинен містити інтуїтивно-зрозумілий інтерфейс. Основним функціоналом програми мусить бути додавання, зберігання даних та надання статистики по доданих даних за певний період.

2.2 Вибір моделі розробки програми

Для розробки даного програмного продукту була вибрана водоспадна модель. Водоспадна модель життєвого циклу ПЗ – послідовний метод розробки програмного забезпечення, названий так через діаграму, схожу на водоспад [13]. Дана модель передбачає послідовну послідовність етапів розробки, які починаються з формулювання вимог і закінчується тестуванням та технічною підтримкою. Кожна стадія має бути завершена до переходу до наступної, а створювані на ній робочі продукти після їх перевірки та затвердження повинні бути заморожені і передані на наступну стадію як еталон [14]. Основні етапи, які виконуються в рамках каскадної моделі розробки, включають:

- ✓ формулювання вимог – у цьому етапі визначається потреби користувача, розробляються специфікації, які описують функціональні та нефункціональні вимоги до продукту;
- ✓ проєктування – на цьому етапі розробляється детальний план проєкту, проєктуються архітектура, компоненти програми та інтерфейси;
- ✓ реалізація – цей етап передбачає написання коду згідно з проєктувальним планом;
- ✓ тестування – на цьому етапі програмне забезпечення перевіряється на відповідність вимогам та на наявність помилок;
- ✓ впровадження – після успішного проходження випробувань, програмне забезпечення готове до впровадження;
- ✓ технічна підтримка – після впровадження програмного забезпечення можуть виникнути проблеми або бажання користувачів отримати додаткову допомогу. На цьому етапі забезпечується технічна підтримка користувачів [15].

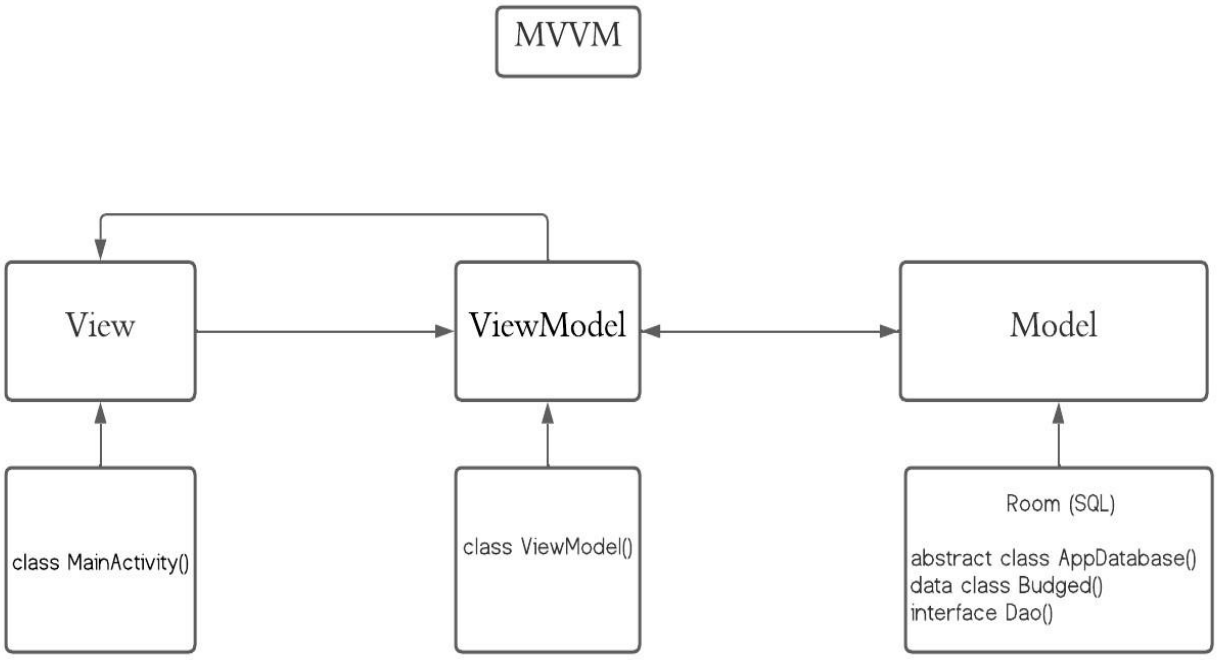
Основною перевагою каскадної моделі є те, що вона структурована та передбачувана [16]. Недоліком є те, що вона не дозволяє змінювати вимоги під час розробки, що може призвести до проблем на пізніших етапах. Також, через послідовний характер моделі, не завжди можна одночасно працювати на всіх етапах, що збільшує час розробки.

2.3 Загальний опис проєкту

2.3.1 Архітектура

У розробці мобільного додатку була використана архітектура MVVM (рис. 2.1). Model-View-ViewModel – шаблон проєктування, що застосовується під час проєктування архітектури застосунків [17]. Дана модель базується на підході Model-View-Controller, але додає до нього ViewModel. ViewModel є проміжним сполучником між відображенням та моделлю даних і виступає у

вигляді форми або набору даних, що інтерпретуються і відображаються у відповідних View-рівнях [18]. Архітектура дозволяє розгортати функціональні додатки у конфігураціях, що легко масштабуються, і може бути знайома розробникам, які працюють з більш традиційними підходами до дизайну додатків. MVVM дозволяє вдосконалювати розробку додатків шляхом зниження кількості коду, який треба писати, якщо використовуються інші підходи. Обов'язки різних шарів повністю відокремлені, що дає можливість розробляти кожен шар незалежно. Це спрощує підтримку та змінність системи в майбутньому. Підхід MVVM дозволяє досить просто підключати моделі даних та розгортати систему в складних середовищах, потребуючих взаємодії з іншими системами. Він дозволяє об'єднувати код, щоб отримати більш компактну версію програми, яку можна з легкістю



підтримувати та змінювати[19].

Рисунок 2.1 - Схема архітектури MVVM

2.3.2 База данних

У проєкті використовується база даних SQLite та бібліотека Room для роботи з ОС Android. Для підключення та побудови бази даних було створено абстрактний клас AppDatabase у якому описано підключення бази даних, клас даних Budget у якому задаються колонки та інтерфейс Dao у якому задано усі запити, які необхідно виконати до бази даних. База даних складається з однієї таблиці (Табл. 2.1).

Таблиця 2.1

id	products	cafe	clothe	car	medicine	entertainment	other	balanc

2.4 Обґрунтування вибору інструментальних засобів розробки

Для розробки мобільного додатку було обрано середовище програмування Android Studio та мова програмування Kotlin. Для реалізації бази даних використовувались бібліотека Room та мова програмування SQL.

Android Studio та мова програмування Kotlin є найбільш популярними та ефективними інструментами для розробки мобільних додатків для операційної системи Android [20]. Kotlin є простою та інтуїтивно зрозумілою мовою програмування, яка значно скорочує час розробки, зменшує кількість помилок та офіційно підтримується компанією Google, що гарантує постійну підтримку та оновлення мови. Android Studio є потужним інструмент для розробки Androidдодатків, який містить багато корисних функцій та інструментів для розробки та тестування. Android Studio має повну інтеграцію з Kotlin, що дозволяє швидко створювати та редагувати код [21].

SQLite – це база даних, яка дуже легко встановлюється на всіх платформах, на яких можна виконувати програми [22]. Програмісти можуть працювати з SQLite, не вивчаючи спеціалізованих знань у сфері баз даних. SQLite – це дуже швидкий інструмент, який може ефективно зберігати великі

об'єми даних. Він працює без сервера баз даних та не потребує підтримки бази даних.

Отже, використання Android Studio та мови програмування Kotlin дозволяє збільшити продуктивність розробки мобільних додатків та забезпечити їх більшу стабільність та швидкість роботи.

2.5 Особливості реалізації програми

Під час створення додатку спочатку було розроблено екран завантаження на якому відображається назва застосунку. Для цього було реалізовано клас

SplashActivity в якому під`єднано файл xml у якому описано інтерфейс екрану (Рис. 2.2).

Індикатор завантаження запрограмовано у співпрограмі в кінці виконання якої відкривається головний екран (рис. 2.3).

Далі було підключено та реалізовано базу даних. Для цього спочатку було підключено необхідні бібліотеки у файлі build.gradle (рис. 2.4, рис. 2.5).

Підключення бази даних реалізовано у абстрактному класі AppDatabase, який наслідується від RoomDatabase (рис. 2.6). Над ініціалізацією класу присутня анотація Database в дужках якої зазначено у якому класі описано структуру таблиці бази даних та версію бази даних.

За колонки бази даних та назву таблиці відповідає клас даних Budget (рис. 2.7). У сьомому рядку коду використовується анотація Entity за допомогою якої задається назва таблиці. Анотація PrimaryKey, із властивістю autoGenerate, використовується для автоматичного додавання номеру рядка запису в таблицю даних. Анотація ColumnInfo використовується для позначення типу даних та назви стовпця.

```
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/purple_500"
    tools:context=".SplashActivity">
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Budget"
        android:textColor="@color/white"
        android:textSize="55sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.4" />
    <ProgressBar
        android:id="@+id/progress_bar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="250dp"
        android:layout_height="wrap_content"
        android:scaleY="3"
        android:layout_marginTop="24dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Рисунок 2.2 - Опис інтерфейсу екрана завантаження

```
import android.animation.ObjectAnimator
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.example.budget.databinding.ActivitySplashBinding
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.delay
import kotlinx.coroutines.launch

class SplashActivity : AppCompatActivity() {
    lateinit var binding: ActivitySplashBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivitySplashBinding.inflate(layoutInflater)
        setContentView(binding.root)

        CoroutineScope(Dispatchers.Main).launch { this: CoroutineScope
            binding.progressBar.max = 1000
            val value = 1000
            ObjectAnimator.ofInt(binding.progressBar,  propertyName: "progress", value).setDuration(2000).start()
            delay( timeMillis: 2000)
            startActivity(Intent( packageContext: this@SplashActivity,MainActivity::class.java))
        }
    }
}
```

Рисунок 2.3 - Реалізація класу SplashActivity

Інтерфейс Dao містить усі запити до бази даних, які будуть використовуватися додатком під час роботи (рис. 2.8, 2.9). Анотація Query означає, що в дужках повинен бути запит до бази даних на мові SQLite. Під анотацією розташована функція за допомогою якої ми передаємо або отримуємо дані. Кожна функція перед оголошенням повинна містити ключове слово suspend для коректного виклику у співпрограмі.

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-kapt'
}
android {
    namespace 'com.example.budget'
    compileSdk 33
    defaultConfig {
        applicationId "com.example.budget"
        minSdk 29
        targetSdk 33
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    buildFeatures{
        viewBinding true
    }
}
```

Рисунок 2.4 - Підключення необхідних бібліотек для роботи

Щоб запрограмувати головний екран було створено клас MainActivity. Спочатку було підключено файл з макетом, який відповідає за інтерфейс, ініціалізовано екземпляри класів AppDatabase та viewModel для роботи з базою даних. Для відображення поточної дати був використаний екземпляр класу Calendar та його функція time, також був заданий спеціальний формат

При відкриті додатку програма перевіряє чи перший раз за сьогодні ви її відкриваєте. Якщо це перший раз, як ви користуєтеся додатком взагалі, чи перший за сьогодні, то програма робить запис у базу даних, де добавляє сьогоднішню дату та присвоює кожній іншій колонці значення нуль. Даний алгоритм реалізовано у співпрограмі (Рис. 2.11).

```
dependencies {
    implementation("androidx.room:room-runtime:2.5.1")
    annotationProcessor("androidx.room:room-compiler:2.5.1")
    kapt("androidx.room:room-compiler:2.5.1")
    implementation("androidx.room:room-ktx:2.5.1")
    implementation("androidx.room:room-rxjava2:2.5.1")
    implementation("androidx.room:room-rxjava3:2.5.1")
    implementation("androidx.room:room-guava:2.5.1")
    testImplementation("androidx.room:room-testing:2.5.1")
    implementation("androidx.room:room-paging:2.5.1")

    implementation('androidx.lifecycle:lifecycle-livedata-ktx:2.6.1')
    implementation('androidx.lifecycle:lifecycle-runtime-ktx:2.6.1')
    implementation ('androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.1')

    implementation("ir.mahozad.android:pie-chart:0.7.0")
    implementation 'androidx.core:core-ktx:1.9.0'
    implementation 'androidx.appcompat:appcompat:1.6.1'
    implementation 'com.google.android.material:material:1.8.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
    testImplementation 'junit:junit:4.13.2'
    androidTestImplementation 'androidx.test.ext:junit:1.1.5'
    androidTestImplementation 'androidx.test.espresso:core:3.5.1'
}
```

Рисунок 2.5 - Підключення необхідних бібліотек для роботи

```
import android.content.Context
import androidx.room.Database
import androidx.room.Room
import androidx.room.RoomDatabase

@Database(entities = [Budget::class], version = 1)
abstract class AppDatabase: RoomDatabase() {
    abstract fun dao(): Dao

    companion object{

        @Volatile
        private var INSTANCE: AppDatabase? = null
        fun getDatabase(context: Context): AppDatabase{
            val tempInstance = INSTANCE
            if(tempInstance != null){
                return tempInstance
            }

            synchronized( lock: this){
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java,
                     name: "app_database"
                ).build()
                INSTANCE = instance
                return instance
            }
        }
    }
}
```

Рисунок 2.6 - Реалізація класу AppDatabase

```
import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "budget_table")
data class Budget (
    @PrimaryKey(autoGenerate = true) val id: Int?,
    @ColumnInfo(name = "date") val date: String?,
    @ColumnInfo(name = "products") val products: Float?,
    @ColumnInfo(name = "home") val home: Float?,
    @ColumnInfo(name = "cafe") val cafe: Float?,
    @ColumnInfo(name = "clothes") val clothes: Float?,
    @ColumnInfo(name = "car") val car: Float?,
    @ColumnInfo(name = "medicine") val medicine: Float?,
    @ColumnInfo(name = "entertainment") val entertainment: Float?,
    @ColumnInfo(name = "other") val other: Float?,
    @ColumnInfo(name = "balance") val balance: Float?
        )
```

Рисунок 2.7 - Реалізація класу даних Budget

Запис рядка у базу даних відбувається за допомогою допоміжних класів ViewModel (Рис. 2.12) і Repository (Рис. 2.13), використовуючи архітектуру MVVM.

Передача даних з бази даних в поля балансу та витрат відображено на (Рис. 2.14). Реалізація інтерфейсу даних полів відбувається у файлі activity_main.xml (Рис. 2.15).

Для реалізації та заповнення діаграми з бази даних зчитується рядок який відповідає поточній даті, далі загальна сума витрат ділиться на суму потрібної категорії та додається до діаграми. Також реалізована перевірка, чи є невикористана категорія, для правильного виведення діаграми (Рис. 2.16).

```
@Dao
interface Dao {
    @Query("SELECT date FROM budget_table")
    suspend fun getDate(): List<String>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insert(budget: Budget)

    @Query("UPDATE budget_table SET products= :productu WHERE date LIKE :locdate")
    suspend fun updateProducts(productu: Float, locdate: String)

    @Query("UPDATE budget_table SET home= :hom WHERE date LIKE :locdate")
    suspend fun updateHome(hom: Float, locdate: String)

    @Query("UPDATE budget_table SET cafe= :caf WHERE date LIKE :locdate")
    suspend fun updateCafe(caf: Float, locdate: String)

    @Query("UPDATE budget_table SET clothes= :cloth WHERE date LIKE :locdate")
    suspend fun updateClothes(cloth: Float, locdate: String)

    @Query("UPDATE budget_table SET car= :ca WHERE date LIKE :locdate")
    suspend fun updateCar(ca: Float, locdate: String)

    @Query("UPDATE budget_table SET medicine= :med WHERE date LIKE :locdate")
    suspend fun updateMedicine(med: Float, locdate: String)

    @Query("UPDATE budget_table SET entertainment= :entr WHERE date LIKE :locdate")
    suspend fun updateEntertainment(entr: Float, locdate: String)

    @Query("UPDATE budget_table SET other= :oth WHERE date LIKE :locdate")
    suspend fun updateOther(oth: Float, locdate: String)

    @Query("UPDATE budget_table SET balance= :balan WHERE date LIKE :locdate")
    suspend fun updateBalance(balan: Float, locdate: String)

    @Query("SELECT * FROM budget_table WHERE date LIKE :today")
    suspend fun getCategory(today: String): Budget

    @Query("SELECT * FROM budget_table WHERE date LIKE :locdate")
    fun readBalance(locdate: String): Flow<List<Budget>>
```

Рисунок 2.8 - Реалізація інтерфейсу Dao

```
    @Query("SELECT products FROM budget_table WHERE id> :startId")
    suspend fun getProductsForWeek(startId: Int): List<Float>

    @Query("SELECT home FROM budget_table WHERE id> :startId")
    suspend fun getHomeForWeek(startId: Int): List<Float>

    @Query("SELECT cafe FROM budget_table WHERE id> :startId")
    suspend fun getCafeForWeek(startId: Int): List<Float>

    @Query("SELECT clothes FROM budget_table WHERE id> :startId")
    suspend fun getClothesForWeek(startId: Int): List<Float>

    @Query("SELECT car FROM budget_table WHERE id> :startId")
    suspend fun getCarForWeek(startId: Int): List<Float>

    @Query("SELECT medicine FROM budget_table WHERE id> :startId")
    suspend fun getMedicineForWeek(startId: Int): List<Float>

    @Query("SELECT entertainment FROM budget_table WHERE id> :startId")
    suspend fun getEntertainmentForWeek(startId: Int): List<Float>

    @Query("SELECT other FROM budget_table WHERE id> :startId")
    suspend fun getOtherForWeek(startId: Int): List<Float>

    @Query("SELECT balance FROM budget_table WHERE id> :startId")
    suspend fun getBalanceForWeek(startId: Int): List<Float>
}
```

Рисунок 2.9 - Реалізація інтерфейсу Dao

Бібліотека для побудови діаграми була взята з репозиторію GitHub [23]. На рисунку 2.17 описано інтерфейс діаграми.

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding
    private lateinit var appDb: AppDatabase
    private lateinit var viewModel: ViewModel
    var toDate:String = ""
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        appDb = AppDatabase.getDatabase( context: this)
        viewModel = ViewModelProvider( owner: this)[ViewModel::class.java]

        val calendar: Calendar = Calendar.getInstance()
        var locDate: String = DateFormat.getDateInstance().format(calendar.time).toString()
        binding.toDate.text = locDate
```

Рисунок 2.10 - Ініціалізація змінних та визначення дати

```
CoroutineScope(Dispatchers.Main).launch { this: CoroutineScope
    val listDate: List<String> = appDb.dao().getDate()
    if (listDate.isNotEmpty()) {
        toDate = listDate.last().toString()

        if (locDate == toDate) {
        } else {
            val insrt =
                Budget( id: null,locDate, products: 0f, home: 0f, cafe: 0f, clothes: 0f, car: 0f, medicine: 0f, entertainment: 0f, other: 0f, balance: 0f)
            viewModel.insert(insrt)
        }
    } else{
        val insrt =
            Budget( id: null, locDate, products: 0f, home: 0f, cafe: 0f, clothes: 0f, car: 0f, medicine: 0f, entertainment: 0f, other: 0f, balance: 0f)
        viewModel.insert(insrt)
    }
}
```

Рисунок 2.11 - Реалізація алгоритму

```
import android.app.Application
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class ViewModel(application: Application): AndroidViewModel(application) {
    private val repository: Repository
    init {
        val dao = AppDatabase.getDatabase(application).dao()
        repository = Repository(dao)
    }
    fun insert(budget: Budget){
        viewModelScope.launch(Dispatchers.IO){ this: CoroutineScope
            repository.insert(budget)
        }
    }
}
```

Рисунок 2.12 - Реалізація класу ViewModel

```
class Repository(private val dao: Dao) {
    suspend fun insert(budget: Budget){
        dao.insert(budget)
    }
}
```

Рисунок 2.13 - Реалізація класу Repository

```
appDb.dao().readBalance(locDate).asLiveData().observe( owner: this){ it: List<Budget>!
    it.forEach { it: Budget
        binding.tvBalance.text = it.balance.toString() + "грн."
    }
}
appDb.dao().readBalance(locDate).asLiveData().observe( owner: this){ it: List<Budget>!
    it.forEach { it: Budget
        var sum: Float? = it.products!! + it.home!! + it.cafe!! + it.clothes!! + it.car!! + it.medicine!! + it.entertainment!! + it.other!!
        binding.tvCosts.text = sum.toString() + "грн."
    }
}
```

Рисунок 2.14 - Заповнення полів балансу та витрат

Назва додатку та кнопка меню знаходяться на верхній панелі інструментів головного екрану. Кнопка меню відкриває боковий екран навігації на якому розміщено кнопки, які відповідають за відкриття екранів із статистикою за певний період (Рис. 2.18). У рядках 118 та 123 реалізовано кнопки відкриття екранів редагування балансу та запису витрат.

```
<TextView
    android:id="@+id/tv_costs"
    android:layout_width="150dp"
    android:layout_height="40dp"
    android:layout_marginTop="48dp"
    android:gravity="end"
    android:text="200"
    android:textSize="24sp"
    android:textColor="@color/black"
    app:layout_constraintEnd_toStartOf="@+id/add_costs_button"
    app:layout_constraintHorizontal_bias="0.863"
    app:layout_constraintStart_toEndOf="@+id/costs"
    app:layout_constraintTop_toBottomOf="@+id/toolbar" />

<TextView
    android:id="@+id/tv_balance"
    android:layout_width="150dp"
    android:layout_height="40dp"
    android:layout_marginTop="15dp"
    android:layout_marginEnd="136dp"
    android:gravity="end"
    android:text="10000"
    android:textColor="@color/black"
    android:textSize="24sp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="@+id/tv_costs"
    app:layout_constraintTop_toBottomOf="@+id/tv_costs" />
```

Рисунок 2.15 - Реалізація інтерфейсу полів балансу та витрат

```
val pieChart = binding.pieChart
appDb.dao().readBalance(locDate).asLiveData().observe( owner: this){ it: List<Budget>!
    it.forEach { it: Budget
        val sum: Float? = it.products!! + it.home!! + it.cafe!! + it.clothes!! + it.car!! + it.medicine!! + it.entertainment!! + it.other!!
        val pr = PieChart.Slice( fraction: floor( x: (it.products!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 253,  green: 72,  blue: 133))
        val ho = PieChart.Slice( fraction: floor( x: (it.home!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 142,  green: 42,  blue: 159))
        val ca = PieChart.Slice( fraction: floor( x: (it.cafe!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 147,  green: 227,  blue: 54))
        val cl = PieChart.Slice( fraction: floor( x: (it.clothes!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 104,  green: 0,  blue: 36))
        val car = PieChart.Slice( fraction: floor( x: (it.car!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 0,  green: 255,  blue: 231))
        val med = PieChart.Slice( fraction: floor( x: (it.medicine!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 255,  green: 0,  blue: 0))
        val ent = PieChart.Slice( fraction: floor( x: (it.entertainment!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 255,  green: 198,  blue: 26))
        val oth = PieChart.Slice( fraction: floor( x: (it.other!!*100/ sum!! /100)*1000)/1000,Color.rgb( red: 133,  green: 125,  blue: 125))
        val myList = ArrayList<PieChart.Slice>()
        if(it.products >0 && pr !in myList){
            myList.add(pr)
        }
        if(it.home >0 && ho !in myList){
            myList.add(ho)
        }
        if(it.cafe >0 && ca !in myList){
            myList.add(ca)
        }
        if(it.clothes >0 && cl !in myList){
            myList.add(cl)
        }
        if(it.car >0 && car !in myList){
            myList.add(car)
        }
        if(it.medicine >0 && med !in myList){
            myList.add(med)
        }
        if(it.entertainment >0 && ent !in myList){
            myList.add(ent)
        }
        if(it.other >0 && oth !in myList){
            myList.add(oth)
        }
        pieChart.slices = myList
    }
}
```

Рисунок 2.16 - Заповнення діаграми

Основні задачі, які необхідно було реалізувати у класі додавання витрат це введення та зчитування суми витрати та збереження її у колонці відповідної категорії бази даних. Введення та збереження суми витрат описано у функції setNumber (Рис. 2.19).

```
<ir.mahozad.android.PieChart
    android:id="@+id/pieChart"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginStart="20dp"
    android:layout_marginEnd="20dp"

    android:layout_marginBottom="140dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent" />
```

Рисунок 2.17 - Реалізація інтерфейсу діаграми

Збереження суми витрати у колонку бази даних, яка відповідає певній категорії зображено на рисунку 2.20. Після натиснення на значок певної категорії запускається функція у якій зчитується число з поля введення суми витрати і записується у тимчасову змінну, яка пізніше виступає параметром функції, яка оновлює рядок бази даних з новим значенням.

Інтерфейс екрана додавання витрат описано у файлі activity_add_costs.xml. Реалізація категорій зображено на (Рис. 2.21), а цифрової клавіатури (Рис. 2.22).

За кожна категорію відповідає картинка, яка розміщена на екрані за рахунок макету сітки, який розділений на два рядки та чотири стовпці.

Екран корегування балансу реалізовувався із ініціалізації необхідних змінних, панелі інструментів та визначення поточної дати (Рис. 2.23). За введення суми відповідає функція setNumber, а збереження суми балансу та закриття екрану описано у функціоналі кнопки «корегувати» (Рис. 2.24).

Для перегляду статистики за певний проміжок часу потрібно натиснути на кнопку «меню» та вибрати кнопку з написом «тиждень», «місяць», «півріччя», «рік» чи «весь час» (Рис. 2.25). Функціонал роботи кожної кнопки

схожий, тому розглянемо роботу кнопки «тиждень». Натиснувши на кнопку, відкривається екран статистики за тиждень. Роботу даного екрану описано в класі WeekActivity.

```
val toolbar = binding.toolbar as Toolbar
setSupportActionBar(toolbar)
supportActionBar?.title = "Budget"
toolbar.setNavigationOnClickListener{ it: View!
    binding.drawer.openDrawer(GravityCompat.START)
}

binding.addCostsButton.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, AddCostsActivity::class.java)
    startActivity(intent)
}
binding.editBalance.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, CorrectBalanceActivity::class.java)
    startActivity(intent)
}
binding.btnWeek.setOnClickListener { it: View!
    val intent = Intent( packageContext: this,WeekActivity::class.java)
    startActivity(intent)
}
binding.btnMonth.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, MonthActivity::class.java)
    startActivity(intent)
}
binding.btnHalfYear.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, HalfYearActivity::class.java)
    startActivity(intent)
}
binding.btnYear.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, YearActivity::class.java)
    startActivity(intent)
}
binding.btnAllTime.setOnClickListener { it: View!
    val intent = Intent( packageContext: this, AllTimeActivity::class.java)
    startActivity(intent)
}
```

Рисунок 2.18 - Програмування кнопок головного екрана

```
private fun setNumber(){
    binding.oneB.setOnClickListener{ setTextField("1")}
    binding.twoB.setOnClickListener{ setTextField("2")}
    binding.threeB.setOnClickListener{ setTextField("3")}
    binding.fourB.setOnClickListener{ setTextField("4")}
    binding.fiveB.setOnClickListener{ setTextField("5")}
    binding.sixB.setOnClickListener{ setTextField("6")}
    binding.sevenB.setOnClickListener{ setTextField("7")}
    binding.eightB.setOnClickListener{ setTextField("8")}
    binding.nineB.setOnClickListener{ setTextField("9")}
    binding.zeroB.setOnClickListener{ setTextField("0")}
    binding.btnPoint.setOnClickListener{ setTextField(".")}
    binding.backspaceB.setOnClickListener{ it: View!
        val str = binding.tvResult.text.toString()
        if(str.isNotEmpty()) {
            binding.tvResult.text = str.substring(0, str.length - 1)
        }else {
            binding.tvResult.text = ""
        }
    }
    binding.backspaceB.setImageResource(R.drawable.ic_backspace)
}
private fun setTextField(str: String){
    binding.tvResult.append(str)
}
```

Рисунок 2.19 - Реалізація функції setNumber

Основний функціонал роботи екрану поміщено у співпрограму. Спочатку з бази даних зчитуються дані за сім днів та знаходиться їхня сума (Рис. 2.26).

На рисунках 2.27 та 2.28 запрограмовано діаграму, яка показує статистику витрат за тиждень. Змінним задається значення, отримане з бази даних у відсотках. Потім кожна із змінних перевіряється на те, чи не дорівнює нулю, та додається до списку, який відображає діаграма.

```
binding.imageProducts.setOnClickListener{ it: View!
    var sum: Float = binding.tvResult.text.toString().toFloat()
    var budget: Budget
    lifecycleScope.launch(Dispatchers.IO) { this: CoroutineScope
        budget = appDb.dao().getCategory(locDate)
        val tem = budget.products

        appDb.dao().updateProducts( productu: sum+ tem!!, locDate)
        delay( timeMillis: 2000)
    }
    finish()
}
binding.imageHome.setOnClickListener{...}
binding.imageRestoran.setOnClickListener{...}
binding.imageClothes.setOnClickListener{...}
binding.imageTransport.setOnClickListener{...}
binding.imageMedicine.setOnClickListener{...}
binding.imageEntertainment.setOnClickListener{...}
binding.imageOther.setOnClickListener{ it: View!
    val sum = binding.tvResult.text.toString().toFloat()
    var budget: Budget
    lifecycleScope.launch(Dispatchers.IO) { this: CoroutineScope
        budget = appDb.dao().getCategory(locDate)
        val tem = budget.other
        appDb.dao().updateOther( oth: sum+tem!!, locDate)
        delay( timeMillis: 2000)
    }
    finish()
}
```

Рисунок 2.20 - Збереження суми витрати в базу даних

```
<ImageView
    android:id="@+id/imageProducts"
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:layout_row="0"
    android:layout_column="0"
    android:layout_marginStart="10dp"
    android:layout_marginTop="5dp"
    app:srcCompat="@drawable/ic_shopping" />
<ImageView...>

<ImageView...>
<ImageView...>
<ImageView...>
<ImageView...>
<ImageView...>

<ImageView
    android:id="@+id/imageOther"
    android:layout_width="70dp"
    android:layout_height="70dp"
    android:layout_row="1"
    android:layout_column="3"
    android:layout_marginEnd="10dp"
    android:layout_gravity="end"
    android:layout_marginTop="5dp"
    app:srcCompat="@drawable/ic_launcher_other" />
```

Рисунок 2.21 - Реалізація інтерфейсу категорій

```
<TextView
    android:id="@+id/seven_b"
    android:layout_width="80dp"
    android:layout_height="70dp"
    android:layout_marginStart="36dp"
    android:layout_marginBottom="24dp"
    android:background="@color/white"
    android:gravity="center"
    android:text="7"
    android:textSize="50sp"
    app:layout_constraintBottom_toTopOf="@+id/four_b"
    app:layout_constraintEnd_toStartOf="@+id/eight_b"
    app:layout_constraintHorizontal_bias="0.02"
    app:layout_constraintStart_toStartOf="parent" />
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<TextView...>
<ImageView
    android:id="@+id/backspace_b"
    android:layout_width="80dp"
    android:layout_height="70dp"
    android:layout_marginEnd="36dp"
    android:layout_marginBottom="28dp"
    android:background="@color/white"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.941"
    app:layout_constraintStart_toEndOf="@+id/zero_b" />
```

Рисунок 2.22 - Реалізація інтерфейсу цифрової клавіатури

```
import ...

class CorrectBalanceActivity : AppCompatActivity() {
    private lateinit var binding: ActivityCorrectBalanceBinding
    private lateinit var appDb: AppDatabase
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityCorrectBalanceBinding.inflate(layoutInflater)
        setContentView(binding.root)
        appDb = AppDatabase.getDatabase( context: this)

        val toolbar = binding.toolbar
        setSupportActionBar(toolbar)
        supportActionBar?.title = " "
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        val calendar: Calendar = Calendar.getInstance()
        val locDate: String = DateFormat.getDateInstance().format(calendar.time).toString()
        setNumber()
```

Рисунок 2.23 - Ініціалізація змінних в класі CorrectBalanceActivity

Далі формується статистика за рахунок присвоювання змінним готових значень із бази даних, або використовуючи функції знаходження мінімальних чи максимальних значень у списках (Рис. 2.29 , 2.30).

Після закінчення роботи з функціоналом та інтерфейсом кожного екрана у файлі AndroidManifest.xml повинні бути записані усі наявні екрани додатку та їхня ієрархія (Рис. 2.31). Також тут описані назва додатку та його значок (Рис. 2.32).

```
    binding.btnCorrect.setOnClickListener { it: View!
        val sum: Float = binding.tvResult.text.toString().toFloat()
        var budget: Budget
        lifecycleScope.launch(Dispatchers.IO) { this: CoroutineScope
            budget = appDb.dao().getCategory(locDate)
            val tem = budget.balance
            appDb.dao().updateBalance( balan: sum+tem!!, locDate)
            delay( timeMillis: 2000)
        }
        finish()
    }
}
private fun setNumber(){
    binding.oneB.setOnClickListener{ setTextField("1")}
    binding.twoB.setOnClickListener{ setTextField("2")}
    binding.threeB.setOnClickListener{ setTextField("3")}
    binding.fourB.setOnClickListener{ setTextField("4")}
    binding.fiveB.setOnClickListener{ setTextField("5")}
    binding.sixB.setOnClickListener{ setTextField("6")}
    binding.sevenB.setOnClickListener{ setTextField("7")}
    binding.eightB.setOnClickListener{ setTextField("8")}
    binding.nineB.setOnClickListener{ setTextField("9")}
    binding.zeroB.setOnClickListener{ setTextField("0")}
    binding.btnPoint.setOnClickListener{ setTextField(".")}
    binding.backspaceB.setOnClickListener{...}
    binding.backspaceB.setImageResource(R.drawable.ic_backspace)
}
private fun setTextField(str: String){
    binding.tvResult.append(str)
}
```

Рисунок 2.24 - Програмування кнопки «корегувати» та функція setNumber

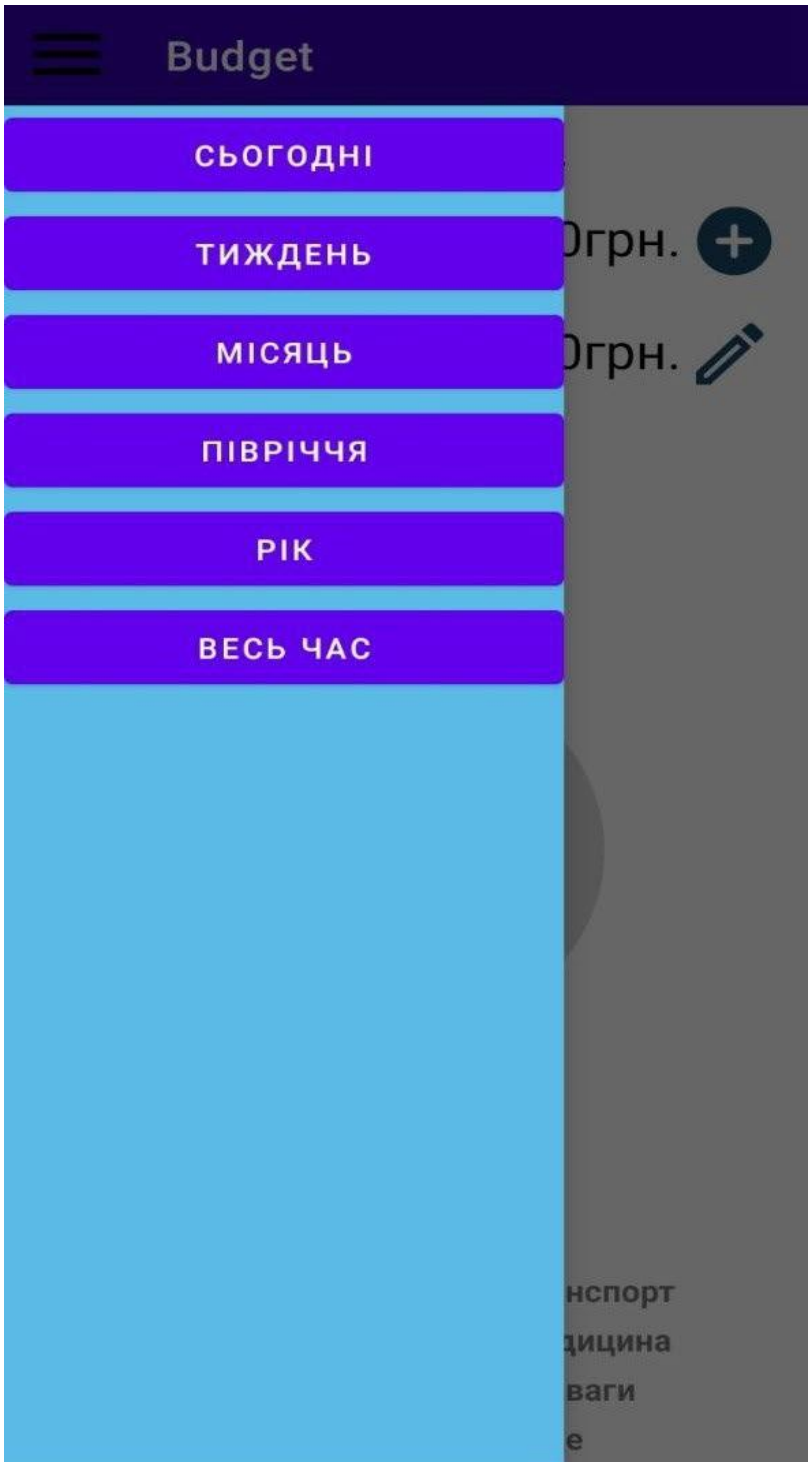


Рисунок 2.25 - Робота кнопки «Меню»

```
val listOfProducts = appDb.dao().getProductsForWeek(starId)
val sumProducts = listOfProducts.sum()

val listOfHome = appDb.dao().getHomeForWeek(starId)
val sumHome = listOfHome.sum()

val listOfCafe = appDb.dao().getCafeForWeek(starId)
val sumCafe = listOfCafe.sum()

val listOfClothes = appDb.dao().getClothesForWeek(starId)
val sumClothes = listOfClothes.sum()

val listOfCar = appDb.dao().getCarForWeek(starId)
val sumCar = listOfCar.sum()

val listOfMedicine = appDb.dao().getMedicineForWeek(starId)
val sumMedicine = listOfMedicine.sum()

val listOfEntertainment = appDb.dao().getEntertainmentForWeek(starId)
val sumEntertainment = listOfEntertainment.sum()

val listOfOther = appDb.dao().getOtherForWeek(starId)
val sumOther = listOfOther.sum()
```

Рисунок 2.26 - Зчитування даних за сім днів з бази даних

```
val sum: Float = sumProducts + sumHome + sumCafe + sumClothes +
        sumCar + sumMedicine + sumEntertainment + sumOther
val pr = PieChart.Slice(
     fraction: floor( x: (sumProducts*100/ sum /100)*1000) /1000,
    Color.rgb( red: 253,  green: 72,  blue: 133))
val ho = PieChart.Slice(
     fraction: floor( x: (sumHome*100/ sum /100)*1000) /1000,
    Color.rgb( red: 142,  green: 42,  blue: 159))
val ca = PieChart.Slice(
     fraction: floor( x: (sumCafe*100/ sum /100)*1000) /1000,
    Color.rgb( red: 147,  green: 227,  blue: 54))
val cl = PieChart.Slice(
     fraction: floor( x: (sumClothes*100/ sum /100)*1000) /1000,
    Color.rgb( red: 104,  green: 0,  blue: 36))
val car = PieChart.Slice(
     fraction: floor( x: (sumCar*100/ sum /100)*1000) /1000,
    Color.rgb( red: 0,  green: 255,  blue: 231))
val med = PieChart.Slice(
     fraction: floor( x: (sumMedicine*100/ sum /100)*1000) /1000,
    Color.rgb( red: 255,  green: 0,  blue: 0))
val ent = PieChart.Slice(
     fraction: floor( x: (sumEntertainment*100/ sum /100)*1000) /1000,
    Color.rgb( red: 255,  green: 198,  blue: 26))
val oth = PieChart.Slice(
     fraction: floor( x: (sumOther*100/ sum /100)*1000) /1000,
    Color.rgb( red: 133, green: 125, blue: 125))
```

Рисунок 2.27 - Присвоєння змінним значень із бази даних

```
val myList = ArrayList<PieChart.Slice>()
if(sumProducts >0 && pr !in myList){
    myList.add(pr)
}
if(sumHome >0 && ho !in myList){
    myList.add(ho)
}
if(sumCafe >0 && ca !in myList){
    myList.add(ca)
}
if(sumClothes >0 && cl !in myList){
    myList.add(cl)
}
if(sumCar >0 && car !in myList){
    myList.add(car)
}
if(sumMedicine >0 && med !in myList){
    myList.add(med)
}
if(sumEntertainment >0 && ent !in myList){
    myList.add(ent)
}
if(sumOther >0 && oth !in myList){
    myList.add(oth)
}
pieChart.slices = myList
```

Рисунок 2.28 - Перевірка на додатне значення змінних та заповнення діаграми

```
binding.tvTotalCosts.text = sum.toString() + "грн."
val listOfBalance = appDb.dao().getBalanceForWeek(starId)
binding.maxBalance.text = listOfBalance.max().toString() + "грн."
binding.minBalance.text = listOfBalance.min().toString() + "грн."
val m = listOfCateg.max()
binding.maxCategory.text = when(m){
    sumProducts -> "Продукти"
    sumHome -> "Дім"
    sumCafe -> "Кафе та ресторани"
    sumClothes -> "Одяг"
    sumCar -> "Транспорт"
    sumMedicine -> "Медицина"
    sumEntertainment -> "Розваги"
    sumOther -> "Інше"
    else -> " "
}
```

Рисунок 2.29 - Формування статистики

```
val tempMinList = kotlin.collections.ArrayList<Float>()
for(i in listOfCateg){
    if(i != 0.0f){
        tempMinList.add(i)
    }
}
val minn = tempMinList.min()
binding.minCategory.text = when(minn){
    sumProducts -> "Продукти"
    sumHome -> "Дім"
    sumCafe -> "Кафе та ресторани"
    sumClothes -> "Одяг"
    sumCar -> "Транспорт"
    sumMedicine -> "Медицина"
    sumEntertainment -> "Розваги"
    sumOther -> "Інше"
    else -> " "
}
```

Рисунок 2.30 - Формування статистики

```
        <activity
            android:name=".MonthActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity" />
        <activity
            android:name=".CorrectBalanceActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity" />
        <activity
            android:name=".AddCostsActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity" />
        <activity
            android:name=".WeekActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity" />
        <activity
            android:name=".MainActivity"
            android:exported="false" />
        <activity
            android:name=".SplashActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Рисунок 2.31 - Ієрархія екранів в програмі

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/an
    xmlns:tools="http://schemas.android.com/tools">

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rule
        android:fullBackupContent="@xml/backup_rules"
        android:icon="@mipmap/ic_launcher"
        android:label="Budget"
        android:supportsRtl="true"
        android:theme="@style/Theme.Budget"
        tools:targetApi="31">
        <activity
            android:name=".AllTimeActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity"/>
        <activity
            android:name=".YearActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity"/>
        <activity
            android:name=".HalfYearActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity"/>
        <activity
            android:name=".MonthActivity"
            android:exported="false"
            android:parentActivityName=".MainActivity" />
```

Рисунок 2.32 - Опис назви та значка додатка

2.6 Організація тестування та налагодження програми

Головна мета тестування програми – це виявлення помилок в її роботі.

Тестування програм та систем – це спосіб семантичної перевірки програми, який полягає в опрацюванні програмою послідовності різноманітних контрольних наборів тестів з відомими результатами [24]. Тестування ПЗ може надавати об'єктивну, незалежну інформацію про якість

ПЗ, ризики відмови, як для користувачів, так і для замовників [25]. Це забезпечує якість продукту та задоволення користувачів, а також зменшує витрати на виправлення помилок після релізу.

Тестування також допомагає забезпечити відповідність програмного забезпечення вимогам, що ставляться до нього. У більш технічному вимірі, тестування допомагає виявити недоліки та помилки, розкриває проблемні місця та некоректну поведінку програми. Це дозволяє програмістам поліпшувати код та забезпечувати оптимальну роботу ПЗ.

Відкривши програму запускається екран завантаження (Рис. 2.33) та головний екран програми (Рис. 2.34).

На головному екрані відображається сьогоднішня дата та пуста діаграма сьогоднішніх витрат. Для того, щоб записати витрати, потрібно натиснути на значок плюс, який знаходиться у правій верхній частині екрану. Натиснувши на плюс, відкриється екран додавання витрат (Рис. 2.35). Ввівши необхідну суму (Рис. 2.36), потрібно натиснути на категорію, до якої відноситься ваша витрата. Тоді програма збереже зміни і перенаправить вас на головний екран, на діаграмі якого відобразиться зміна статистики ваших витрат (Рис. 2.37).

Під значком додавання витрат знаходиться значок корегування балансу. Натиснувши на цей значок відкриється екран у якому можна коригувати ваш поточний баланс (Рис. 2.38). Ввівши необхідну суму (Рис. 2.39) потрібно натиснути на кнопку корегувати і вас поверне на головний екран (Рис. 2.40).

У верхньому лівому куті знаходиться значок меню. Натиснувши на нього, відкриється список, у якому можна вибрати період, за який ви хочете отримати статистику витрат (Рис. 2.41). Для прикладу на рисунку 2.42 зображену статистику, яку було отримано за сім днів користування додатком.

Рисунок 2.33 - Запуск програми

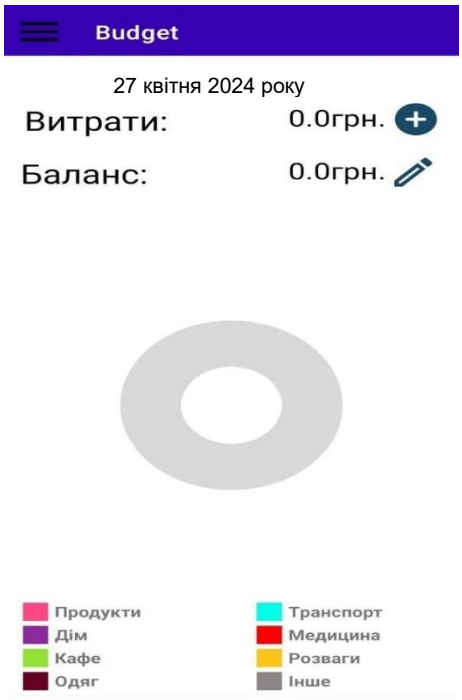


Рисунок 2.34 - Головний екран

Рисунок 2.35 - Екран додавання витрат

Рисунок 2.36 - Введення суми витрати

Рисунок 2.37 - Зміна статистики витрат

Рисунок 2.38 - Екран корегування балансу

Рисунок 2.39 - Введення балансу

2.7 Рекомендації з використання та впровадження програми

Рисунок 2.40 - Головний екран

Мобільний застосунок призначений для щоденного запису витрат та надання статистики за такі проміжки часу, як день, тиждень, місяць, півріччя, рік та загальний час користування програмою з моменту встановлення. Статистика витрат в програмі представлена у форматі діаграм, що дає можливість швидко оцінити розподіл витрат за певним періодом часу.

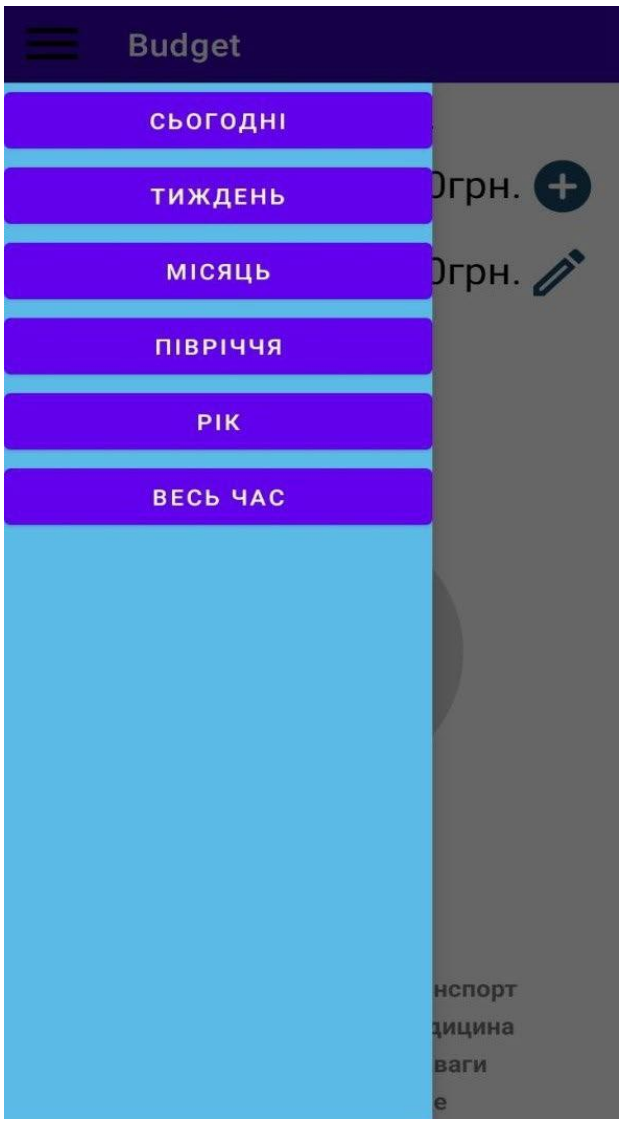


Рисунок 2.41 - Меню

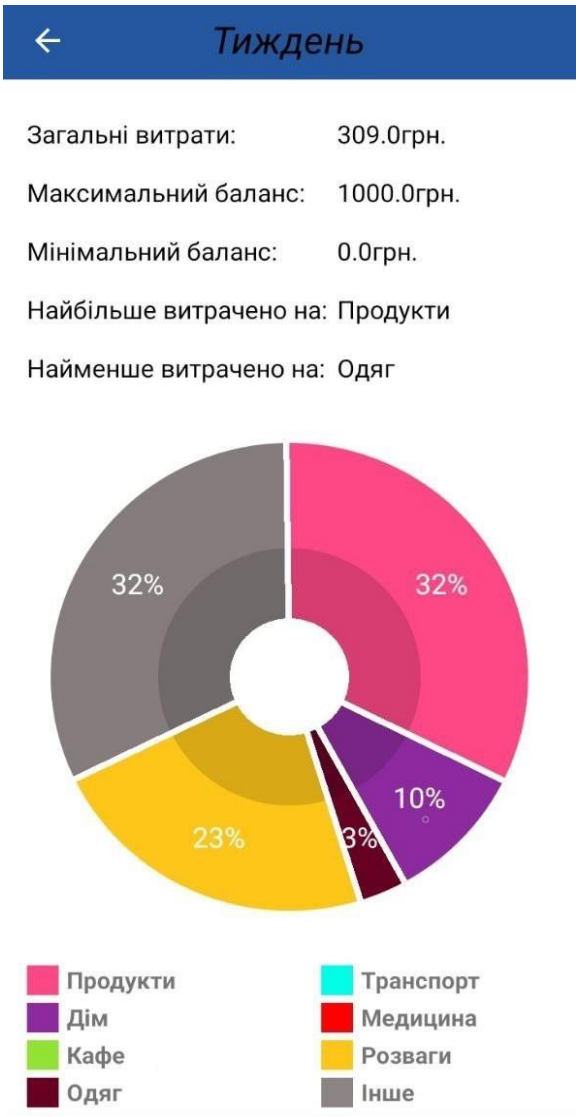


Рисунок 2.42 - Статистика за тиждень

Окремий розділ програми відведений для корегування поточного балансу, що дозволяє побачити картину наявної мінімальної та максимальної суми коштів у певний проміжок часу і приймати рішення з питань фінансового планування. Користування додатком можливе лише на смартфонах з операційною системою Android версії 6.0 і вище.

ВИСНОВКИ

У результаті виконаної роботи було створено мобільний застосунок для контролю персональних фінансів.

У ході написання кваліфікаційної роботи було виконано такі завдання:

- Проведено аналіз трьох мобільних додатків, доступних у Google Play, для контролю персональних фінансів та досліджено їх переваги і недоліки.
- Проаналізовано існуючі інструменти для розробки мобільних додатків та використані засоби розробки.
- Описано роботу, можливості та переваги середовища програмування Android Studio, бібліотеки Room, мови програмування Kotlin і бази даних SQLite.
- Описано процес розробки, логіку і надано фрагменти коду програми.
- Проведено тестування застосунку, показано роботу програми і надані рекомендації із її використання.

На майбутнє можна розширити функціонал програми, додавши можливість зберігання даних у хмарному сховищі та дозволити користувачам створювати власні категорії витрат.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мобільні додатки для управління фінансами URL: https://dou.ua/forums/topic/41422/.

2. Мобільні додатки для управління фінансами URL: https://www.epravda.com.ua/publications/2023/02/22/697326/.

3. Android Studio URL: https://en.wikipedia.org/wiki/Android_Studio.

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

5. Android Studio URL: https://qagroup.com.ua/publications/android-studio- perevagy-ta-osoblyvosti/.

6. Мова програмування Kotlin URL: https://uk.wikipedia.org/wiki/Kotlin.

7. Мова програмування Kotlin URL: https://developer.android.com/kotlin.

8. Мова програмування Kotlin URL: https://kotlinlang.org/docs/android- overview.html.

9. База даних SQLite URL: https://uk.wikipedia.org/wiki/SQLite.

10. База даних SQLite URL: https://www.sqlite.org/index.html.

11. Бібліотека Room URL: https://developer.android.com/training/ datastorage/room.

12. Водоспадна модель розробки програмного забезпечення URL: https://uk.wikipedia.org/ wiki/Водоспадна_модель.

13. Водоспадна модель розробки програмного забезпечення URL: https://stud.com.ua.102291/informatika/kaskadni_modeli.

14. Водоспадна модель розробки програмного забезпечення URL: https://government.com.ua /nashi-hroshi/kaskadna-model-zhittevogo-tsiklu-perevagi-inedoliki.html.

16. Водоспадна модель розробки програмного забезпечення URL: https://evergreens.com.ua/ua/articles/software-development-metodologies.html.

17. Шаблон проєктування архітектури програми Model - View - ViewModel URL: https://uk.wikipedia.org/wiki/Model-View-ViewModel.

18. Класс ViewModel URL: https://developer.android.com/topic/libraries /architecture/viewmodel.

19. Шаблон проєктування Model - View - ViewModel URL: https://www.geeksforgeeks.org/mvvm-model-view-viewmodel-architecture-pattern-inandroid/.

20. Мова програмування Kotlin URL: https://brander.ua/technologies/kotlin.

21. Мова програмування Kotlin URL: https://uk.education-wiki.com /1625670-what-is-kotlin..

22. База даних SQLite URL: http://yoip.com.ua/vvedennya-v-bazu-danihsqlite/ .

23. Бібліотека для побудови діаграми URL: https://github.com /mahozad/android-pie-chart.

24. Тестування програм та систем URL: https://pidru4niki.com/1628011847733/informatika/testuvannya_program_sim.

25. Тестування програмного забезпечення URL: https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення.