МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ

Кафедра комп'ютерних наук та кібербезпеки

РАСІК МИКОЛА РОМАНОВИЧ

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ КОМПЛЕКСНОГО УПРАВЛІННЯ ОСОБИСТИМИ ФІНАНСАМИ

Спеціальність: 122 «Комп'ютерні науки»

Освітньо-професійна програма Комп'ютерні науки та інформаційні технології

Робота на здобуття першого рівня «бакалавр»

Науковий керівник:
ГЛИНЧУК ЛЮДМИЛА ЯРОСЛАВІВНА,
кандидат фізико-математичних наук,
доцент кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ

Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 2024 р.

Завідувач кафедри
(_____) Гришанович Т. О.
(підпис)

Луцьк – 2024

Зміст

ВСТУП

Актуальність теми. У сучасному світі, де фінансові ринки стають дедалі складнішими, а економічні умови швидко змінюються, ефективне управління особистими фінансами стає не просто бажаним, а необхідним навиком. Кожна людина, незалежно від рівня доходів чи фінансової освіченості, стикається з викликами заощадження, інвестування, планування бюджету та контролю витрат. У цьому контексті використання технологій, зокрема мобільних застосунків, відкриває нові можливості для комплексного та зручного обліку фінансів.

В умовах диджиталізації суспільства та стрімкого розвитку мобільних технологій, смартфони стали невід'ємною частиною життя людей. За даними Державної служби статистики України, станом на 2022 рік понад 70% населення країни користуються смартфонами, і ця цифра продовжує зростати [1]. Мобільні пристрої вже давно вийшли за межі своєї первинної функції зв'язку і перетворилися на багатофункціональні інструменти, що допомагають у вирішенні широкого спектру завдань. Однією з таких сфер, де мобільні технології мають значний потенціал, є управління особистими фінансами.

Наукова новизна. Дослідження має на меті створення нового програмного продукту для менеджменту персональних фінансів, який відрізняється від існуючих рішень покращеною функціональністю та зручністю користування.

Мета роботи. Розробити мобільний застосунок для комплексного управління особистими фінансами.

Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз існуючих програмних продуктів, визначити їхні переваги та недоліки;
- розробити технічне завдання для мобільного застосунку з врахуванням вимог користувачів;
- вибрати модель розробки програмного продукту та обґрунтувати її вибір;

- розробити архітектуру програмного продукту, включаючи проєктування бази даних та інтерфейсу користувача;
- реалізувати мобільний застосунок, включаючи написання програмного коду та проведення тестування;
- провести налагодження програмного продукту, усунути виявлені недоліки та підготувати рекомендації щодо використання застосунку.

Об’єкт дослідження. Процес управління особистими фінансами за допомогою мобільних застосунків.

Предмет дослідження. Методи та засоби розробки мобільних застосунків для комплексного управління особистими фінансами, включаючи аналіз існуючих рішень, вибір інструментів розробки та архітектурних підходів.

Матеріал дослідження. Для дослідження були використані джерела, які містять інформацію про сучасні технології розробки мобільних додатків, зокрема, документи та керівництва з використання Android Studio, Java, SQLite. Крім того, було проведено аналіз трьох популярних програмних продуктів для контролю фінансів, доступних на платформі Google Play.

Практичне значення отриманих результатів. Враховуючи актуальність та важливість теми, дана робота має хороший потенціал для подальших досліджень та вдосконалення, а також може бути корисною для широкого кола користувачів, які прагнуть більш ефективно керувати своїми фінансовими витратами за допомогою сучасних мобільних технологій.

Апробація результатів роботи. Результати цієї роботи представлено на двох науково-практичних конференціях, присвячених питанням інформаційних технологій. Зокрема, на XVIII Міжнародній науково-практичній конференції студентів і аспірантів «Молода наука Волині: пріоритети та перспективи досліджень» 14 – 15 травня 2024 року у Волинському національному університеті імені Лесі Українки в м. Луцьк, Україна. Тези доповіді «Мобільний фінансовий асистент: розробка застосунку для менеджменту особистих фінансів» опубліковано в матеріалах конференції [2].

А також на І Міжнародній науково-практичній конференції «Проблеми комп’ютерних наук, програмного моделювання та безпеки цифрових систем» 13 – 16 червня 2024 року на базі практик табору «Гарт» Волинського національного університету імені Лесі Українки в с. Світязь, Україна. Тези доповіді «Розробка мобільного рішення для аналізу особистих фінансових витрат» опубліковано в матеріалах конференції [3].

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження мобільних застосунків для менеджменту фінансів

Мобільні застосунки для менеджменту фінансів стали невід'ємною частиною сучасного життя. З появою смартфонів, що надають зручний доступ до Інтернету та різноманітних сервісів, з'явилася можливість використовувати мобільні додатки для контролю та управління особистими фінансами. Перші мобільні фінансові застосунки були досить простими і мали обмежений функціонал, але з часом вони еволюціонували до повноцінних інструментів для фінансового планування, аналізу витрат і доходів, а також надання рекомендацій користувачам.

До появи мобільних застосунків для менеджменту фінансів, люди користувалися традиційними методами. Найпоширенішими були:

- ведення записів у блокнотах, де фіксувалися всі доходи та витрати. Це був трудомісткий процес, який вимагав багато часу та зусиль для аналізу даних;
- вже в 1990-х роках з’явилися перші програми для настільних комп’ютерів, які спрощували облік фінансів, але потребували доступу до комп’ютера.

Перше програмне забезпечення для фінансового менеджменту з'явилося на початку 2000-х років. Програми були орієнтовані на просте відстеження витрат і доходів, дозволяючи користувачам вручну вводити фінансові дані. Однією з таких програм був Quicken, який спочатку розроблявся для настільних комп'ютерів, а згодом був адаптований для мобільних пристроїв. Quicken надавав користувачам можливість відстежувати свої витрати, створювати бюджети і генерувати фінансові звіти [4].

Іншим популярним застосунком був Mint, який з'явився в 2006 році. Mint відрізнявся тим, що міг автоматично завантажувати фінансові дані з банківських рахунків користувачів, що значно спрощувало процес відстеження витрат і доходів. Застосунок надавав користувачам можливість бачити свої фінансові

операції в реальному часі, аналізувати витрати за категоріями та отримувати рекомендації щодо управління фінансами [5].

З розвитком смартфонів та появою операційних систем, таких як iOS та Android, можливості мобільних застосунків значно розширилися. Фінансові застосунки мали простий інтерфейс і базові функції для обліку витрат та доходів, але вони вже дозволяли користувачам зручно вести особистий фінансовий облік без використання паперових нотаток чи настільних комп'ютерів.

Мобільні застосунки для фінансового менеджменту працюють на основі кількох ключових принципів:

- збирання даних: застосунки можуть автоматично завантажувати фінансові дані з банківських рахунків, кредитних карток та інших фінансових інструментів, або ж дозволяти користувачам вручну вводити інформацію про свої доходи і витрати;
- категоризація витрат і доходів: для зручності аналізу всі фінансові операції поділяються на категорії (наприклад, харчування, транспорт, розваги, комунальні послуги тощо);
- фінансове планування: багато застосунків надають можливість створення бюджетів, встановлення фінансових цілей і планування витрат;
- аналіз даних: застосунки використовують інструменти аналізу для надання користувачам звітів про їх фінансовий стан, аналізу тенденцій витрат і доходів, а також надання рекомендацій щодо оптимізації витрат;
- безпека: оскільки мобільні застосунки працюють з конфіденційною фінансовою інформацією, важливою є безпека даних. Застосовуються різні методи захисту, такі як шифрування даних, використання паролів та біометричних даних для авторизації.

У контексті розробки мобільного застосунку для комплексного управління особистими фінансами важливо розуміти поточний стан ринку, особливо в умовах геополітичних викликів. Нещодавнє дослідження, проведене фінтех-стартапом Saldo Apps у співпраці з аналітичним сервісом ASObot,

проливає світло на тривожну тенденцію: значна частина популярних в Україні фінансових застосунків має російське походження.

В магазинах застосунків App Store та Google Play налічуються тисячі застосунків для менеджменту фінансів, з яких близько 100 активно використовуються в Україні. Дослідження виявило, що попри повномасштабне вторгнення росії в Україну, розпочате в лютому 2022 року, близько 50% нових завантажень фінансових застосунків в Україні припадає на продукти, розроблені в країні-агресорі. За оцінками спеціалістів, українські користувачі за півтора роки після початку вторгнення заплатили понад $250 000 за преміум-функції та підписки в цих застосунках, фактично спонсоруючи економіку цієї країни [6].

З технічної точки зору, причина такої популярності російських застосунків криється у їхній локалізації та інтеграції. Більшість цих продуктів пропонують або повністю україномовний інтерфейс, або тісну інтеграцію з українськими банками. Це створює ілюзію «місцевого» продукту, приховуючи його справжнє походження. Серед 16 найпопулярніших в Україні додатків російського походження – Манібокс, Money manager, CoinKeeper, Zenmoney, Журнал витрат та інші (Рис. 1.1).

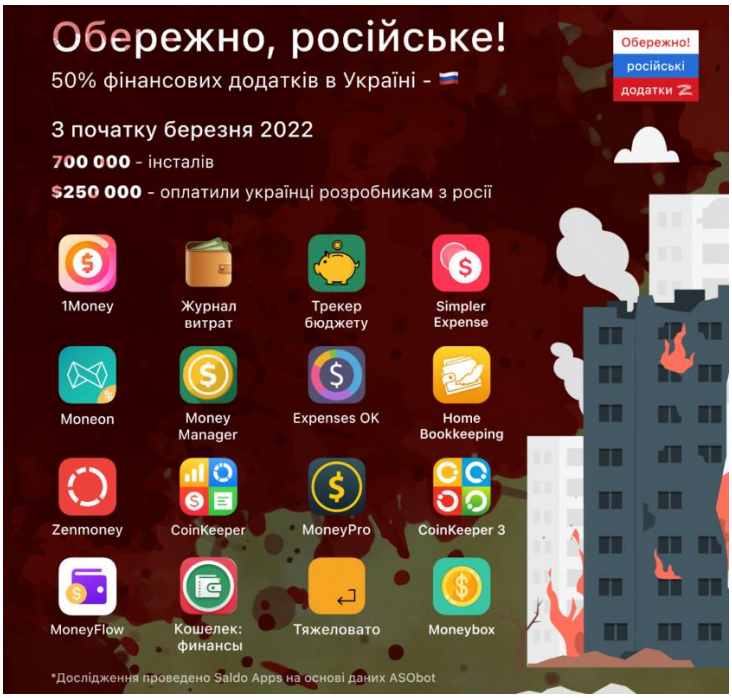


Рисунок 1.1. Популярні в Україні російські додатки з фінансового менеджменту

1.2 Огляд середовища розробки Android Studio

При створенні мобільного застосунку вибір правильного середовища розробки є важливим кроком. Він впливає не лише на ефективність процесу розробки, але й на продуктивність, безпеку та зручність використання кінцевого продукту.

Android Studio – офіційне інтегроване середовище розробки (IDE) для створення програмних продуктів на платформі Android. Воно було розроблене компанією Google та базується на популярному та потужному IntelliJ IDEA від JetBrains. Android Studio надає розробникам інструменти для розробки, тестування та налагодження мобільних застосунків для пристроїв, що працюють на операційній системі Android [7].

Основні можливості Android Studio:

- редактор коду: Android Studio містить потужний редактор коду з підтримкою підсвічування синтаксису, автозаповнення, рефакторингу та інших функцій, які сприяють зручності написання коду. Інтеграція з IntelliJ IDEA надає додаткові можливості для ефективної розробки;
- інтерфейс користувача: візуальний редактор макетів дозволяє розробникам створювати інтерфейси додатків за допомогою простого перетягування елементів. Також є можливість редагування XML-файлів для більш точного контролю;
- віртуальні пристрої: Android Studio містить потужний емулятор, який дозволяє тестувати додатки на різних версіях Android та різних пристроях без необхідності мати фізичні пристрої;
- інструменти для налагодження та аналізу: інтегровані інструменти для налагодження, включаючи логер Logcat, профайлери продуктивності, аналізатор пам'яті та інші інструменти для забезпечення високої якості та продуктивності додатків;

- система збірки Gradle: Android Studio використовує систему збірки Gradle, яка забезпечує гнучкість та автоматизацію процесу збірки додатків. Gradle дозволяє налаштовувати різні конфігурації збірки для різних середовищ;
- інтеграція з системами контролю версій: Android Studio підтримує інтеграцію з популярними системами контролю версій, такими як Git, що дозволяє ефективно керувати версіями проекту та співпрацювати з іншими розробниками [8].

Переваги використання Android Studio:

- підтримка новітніх технологій: завдяки тісній інтеграції з Google, Android Studio швидко отримує підтримку нових функцій та технологій платформи Android;
- підтримка: широка спільнота розробників та велика кількість ресурсів, таких як документація, навчальні матеріали та форуми, роблять Android Studio доступним та зручним для новачків та професіоналів;
- інструменти для всього циклу розробки: від проектування до тестування та випуску продукту – Android Studio надає всі необхідні інструменти для повного циклу розробки додатків [9].

Однак, є й недоліки. Android Studio споживає багато ресурсів, що може сповільнити процес розробки на менш потужних машинах. Крім того, воно зосереджено лише на Android, тоді як для максимального охоплення аудиторії потрібна також iOS-версія.

1.3 Огляд бази даних SQLite та мови програмування Java

Для реалізації зручного застосунку необхідна база даних, здатна ефективно зберігати та обробляти фінансові дані. Популярний вибір для застосунків Android – SQLite, мала та проста у використанні, але водночас потужна та безпечна база даних з відкритим кодом [10].

SQLite використовує мінімум пам'яті та місця на диску завдяки унікальному механізму, де вся інформація зберігається у файловій базі даних

безпосередньо на пристрої. Це забезпечує високу продуктивність для простих запитів, але може сповільнитись за більш складних. Проте, внутрішня структура SQLite дозволяє зберігати великі проміжні результати у оперативній пам'яті, що компенсує недолік продуктивності.

Основні особливості SQLite:

- вбудованість: SQLite не потребує окремого серверного процесу для роботи. Всі дані зберігаються у одному файлі бази даних, що робить її легкою для розгортання та використання;
- малий розмір: база даних займає дуже мало місця і не вимагає багато системних ресурсів, що ідеально підходить для мобільних застосунків;
- підтримка стандарту SQL: SQLite підтримує більшість команд SQL, що робить її зручною для використання розробниками, знайомими зі стандартами реляційних баз даних;
- транзакції: підтримка атомарних транзакцій гарантує, що всі операції з даними виконуються повністю або не виконуються взагалі, що забезпечує цілісність даних;
- багатоплатформність: SQLite працює на багатьох платформах, включаючи Android, iOS, Windows, macOS та інші [11].

Простота інтеграції та використання SQLite робить її популярним вибором для мобільних додатків, які потребують локального зберігання даних.

Для розробки мобільних застосунків на Android найбільш поширеною і рекомендованою мовою програмування є Java. Незважаючи на недавню появу альтернативних варіантів, таких як Kotlin, Java залишається провідним вибором для створення надійних і продуктивних застосунків.

Java – одна з найпопулярніших мов програмування, яка активно використовується для розробки мобільних застосунків під платформу Android. Це об'єктно-орієнтована мова, розроблена компанією Sun Microsystems (тепер частина Oracle Corporation), і вперше випущена у 1995 році [12].

Основні особливості Java:

- багатоплатформність: Java кодується один раз і може виконуватися на будь-якій платформі, що має JVM (Java Virtual Machine). Це дозволяє розробникам створювати крос-платформні рішення;
- об'єктно-орієнтоване програмування (ООП): Java підтримує принципи ООП, що сприяє організації коду у вигляді об'єктів та класів, забезпечуючи модульність та повторне використання коду;
- безпека: Java має вбудовані механізми безпеки, що робить її безпечною мовою для розробки мережевих та мобільних програмних продуктів;
- велика стандартна бібліотека: Java має велику стандартну бібліотеку класів (Java API), що містить безліч готових рішень для виконання різних задач, включаючи роботу з файлами, мережевими з'єднаннями, графічним інтерфейсом тощо;
- підтримка: велика спільнота розробників, численні ресурси для навчання та підтримка від компанії Oracle роблять Java одним з найпопулярніших виборів для розробки продуктів [13].

1.4 Огляд та аналіз аналогічних програмних розробок

У світі мобільних технологій існує велика кількість застосунків, створених для управління фінансами. Проаналізувавши деякі з них, ми зможемо краще зрозуміти потреби користувачів, тенденції ринку та сильні й слабкі сторони наявних рішень. Це допоможе при проектуванні власного застосунку, щоб він максимально задовольняв вимоги користувачів.

У цьому підрозділі проведемо огляд та аналіз трьох програмних продуктів: Monefy, Wallet та My Finances, щоб визначити їх переваги та недоліки.

Monefy (Рис. 1.2 - 1.3) – простий та інтуїтивний мобільний застосунок для управління фінансами, розроблений компанією Reflective Technologies. Основна мета Monefy – забезпечити користувачам легкий спосіб відстеження своїх витрат і доходів. Додаток підтримує різні валюти, автоматичну синхронізацію між пристроями, а також надає зручну візуалізацію фінансових даних у вигляді

графіків та діаграм. Відзначається зручним інтерфейсом, однак у безкоштовній версії функціонал дещо обмежений [14].

Рисунок 1.2. Застосунок Monefy

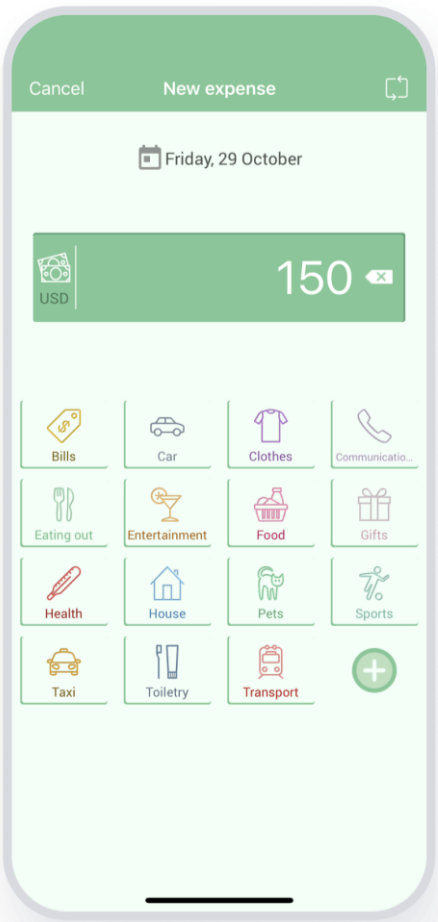


Рисунок 1.3. Сторінка обліку

Назва: Monefy

Розробник: Reflective Technologies

Архітектура: Web Application

Перелік функцій та характеристик:

- можливість відстежувати доходи та витрати за допомогою категорій;
- підтримка різних валют та обмінних курсів;
- автоматична синхронізація даних між різними пристроями;
- відображення статистики в зручному графічному вигляді;
- розширені можливості для користувачів платної версії.

Аналіз переваг та недоліків:

Переваги:

- інтуїтивно зрозумілий інтерфейс;
- можливість додавати власні категорії;
- автоматична синхронізація між пристроями.

Недоліки:

- обмежений функціонал у безкоштовній версії;
- відсутність додаткових функцій (наприклад, запис витрат постфактум).

Wallet (Рис. 1.4 - 1.5) – це багатофункціональний програмний продукт для фінансового менеджменту, розроблений Budget Bakers. Wallet дозволяє створювати бюджети, планувати витрати, аналізувати фінансові дані, додавати фотографії чеків, а також захищати дані за допомогою PIN-коду. Також підтримує синхронізацію з обліковими записами Google та Dropbox. Деякі функції доступні лише у преміум-версії, що може обмежувати користувачів [15].

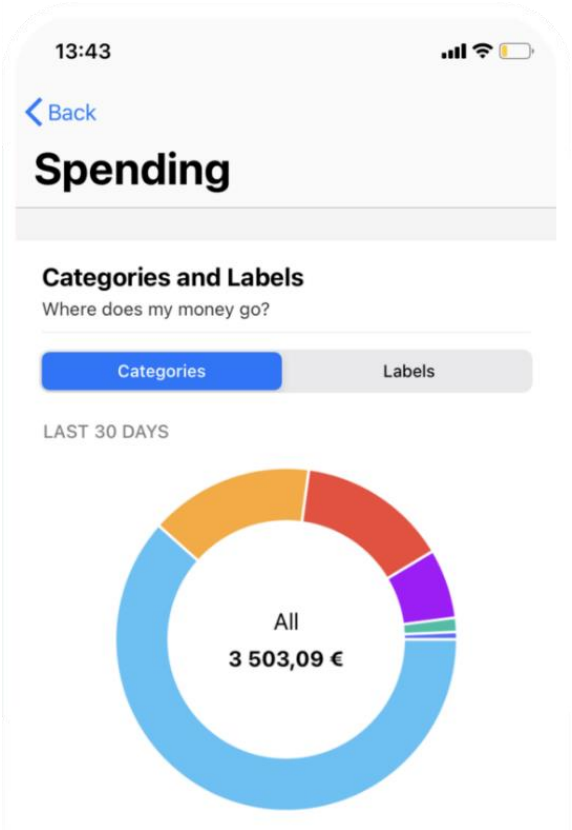


Рисунок 1.4. Застосунок Wallet

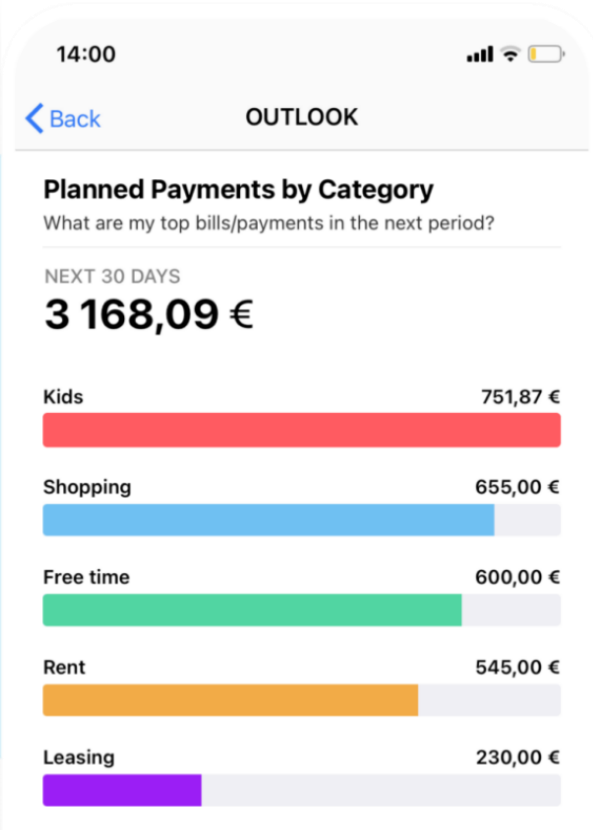


Рисунок 1.5. Заплановані платежі

Назва: Wallet

Розробник: Budget Bakers

Архітектура: Web Application

Перелік функцій та характеристик:

- створення бюджетів та планування витрат;
- аналіз витрат та доходів у різних категоріях;
- можливість додавання фотографій чеків та квитанцій;
- захист даних за допомогою PIN-коду або відбитка пальця;
- синхронізація з обліковими записами Google та Dropbox;
- розширені можливості для користувачів платної версії.

Аналіз переваг та недоліків:

Переваги:

- розширений функціонал;
- можливість додавати фотографії чеків;
- безпека даних.

Недоліки:

- обмежений функціонал у безкоштовній версії.

My Finances (Рис. 1.6 - 1.7) – мобільний застосунок для управління фінансами, розроблений 7Csolutions. Додаток дозволяє створювати різні бюджети, візуалізувати фінансові дані у вигляді діаграм та графіків, а також встановлювати фінансові цілі та відстежувати їх досягнення. Деякі функції доступні лише у преміум-версії [16].

Назва: My Finances

Розробник: 7csolutions

Архітектура: Web Application

Перелік функцій та характеристик:

- можливість створення різних бюджетів для різних потреб;
- зручна візуалізація статистики у вигляді діаграм та графіків;
- можливість встановлення мети заощаджень;
- розширені можливості для користувачів платної версії.

Аналіз переваг та недоліків:

Переваги:

- інтуїтивно зрозумілий інтерфейс;
- візуалізація статистики;
- безпека даних.

Недоліки:

- обмежений функціонал у безкоштовній версії.

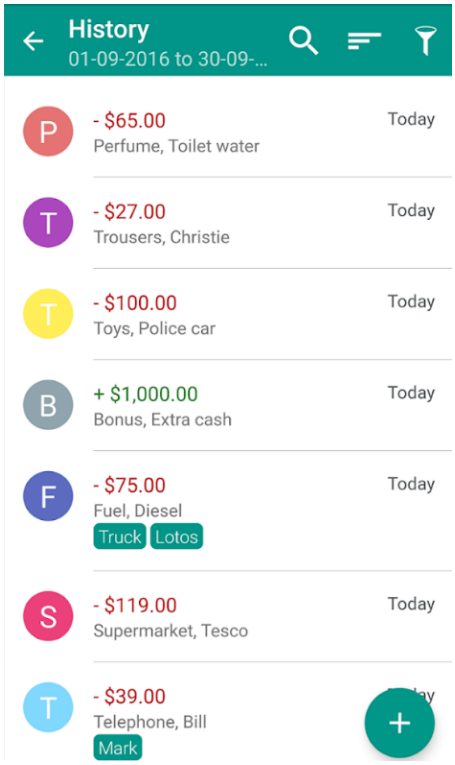


Рисунок 1.6. Історія витрат

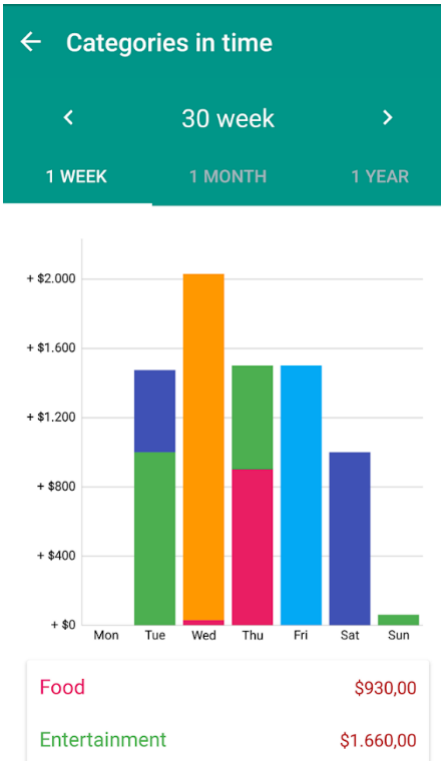


Рисунок 1.7. Діаграма витрат

На основі проведеного аналізу існуючих мобільних застосунків для управління фінансами можна зробити висновок, що найбільш затребуваними функціями є можливість відстежувати доходи та витрати за категоріями, візуалізація фінансових даних у вигляді графіків та діаграм, додавання витрат за минулі дні та робота застосунку без обов'язкового підключення до мережі. Також важливо приділити увагу зручності користувацького інтерфейсу.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

2.1 Постановка задачі, призначення та вимоги до мобільного застосунку «FinanceTracker»

Метою розробки мобільного застосунку «FinanceTracker» є створення зручного та ефективного інструменту для менеджменту особистих фінансових витрат. Застосунок призначений для індивідуального використання, не потребуючи постійного підключення до мережі. Ця особливість робить його зручним для користувачів, які прагнуть вести записи та відстежувати дані в будь-якому місці та в будь-який час, незалежно від доступу до Інтернету.

Основними завданнями, які має вирішувати застосунок, є:

- додавання та зберігання витрат в базі даних;
- надання статистики за різні періоди часу (сьогодні, тиждень, місяць, рік);
- побудова діаграми витрат.

Вимоги до інтерфейсу користувача:

- прості та зрозумілі елементи управління;
- логічна структура меню та навігації;
- дизайн з використанням приємних кольорів;
- хороша якість графічних елементів;
- оптимізація для різних розмірів екрану.

2.2 Вибір моделі розробки програмного засобу «FinanceTracker»

Процес розробки програмного забезпечення – це комплексний і структурований підхід до створення програмного продукту. Він включає в себе ряд послідовних етапів, починаючи від формулювання цілей проекту і закінчуючи підтримкою готового продукту.

Для розробки мобільного застосунку була обрана каскадна модель розробки «Waterfall» (Рис. 2.1). Дана модель передбачає послідовне

проходження через всі етапи життєвого циклу програмного забезпечення, причому кожен етап повинен бути завершений перед переходом до наступного. Каскадна модель забезпечує чітку структуру проєкту та контроль над виконанням кожного етапу [17].

Рисунок 2.1. Каскадна модель розробки

Основні етапи каскадної моделі:

- на першому етапі визначаються потреби користувачів та формуються вимоги до програмного забезпечення;
- після визначення вимог проводиться проєктування системи. Розробляються архітектура програмного забезпечення, проєктування бази даних та інтерфейсу користувача;
- на третьому етапі здійснюється написання коду відповідно до технічної документації. Розробка ведеться послідовно, модуль за модулем;
- на четвертому етапі всі модулі об'єднуються в єдину систему, яка проходить комплексне тестування. Перевіряється відповідність

програмного забезпечення вимогам, виявляються та виправляються помилки;

- після проходження тестування програмний продукт готовий до впровадження;
- на останньому етапі забезпечується підтримка користувачів, виправлення виявлених помилок та внесення необхідних змін і оновлень.

Каскадна модель має свої переваги та недоліки. До переваг можна віднести чітку структуру процесу розробки, легкість управління проектом та можливість ретельного планування кожного етапу. Однак ця модель може бути недостатньо гнучкою при необхідності внесення змін на пізніх етапах розробки.

2.3 Загальний опис проєкту

2.3.1 Архітектура

Архітектура мобільного застосунку «FinanceTracker» побудована на основі локального зберігання даних, що забезпечує автономність та швидкодію системи. Основні компоненти архітектури включають:

- клієнтська частина реалізована як мобільний застосунок для платформи Android, розроблений за допомогою Android Studio. Застосунок включає в себе інтерфейс користувача, який розроблений з використанням XML для опису макетів і мови програмування Java для реалізації логіки програми;
- логіка програми охоплює функціональні можливості для введення, обробки та відображення фінансових записів. Основні компоненти включають модулі для введення записів витрат, створення діаграми та взаємодії з базою даних;
- локальне зберігання даних реалізоване з використанням бази даних SQLite, що дозволяє зберігати фінансові дані безпосередньо на пристрої користувача. Це забезпечує доступність даних без необхідності підключення до інтернету.

2.3.2 База даних

База даних мобільного застосунку «FinanceTracker» побудована з використанням SQLite, яка є легкою реляційною базою даних, для використання на мобільних пристроях. SQLite обрана через її хорошу продуктивність, мінімальні вимоги до ресурсів та інтеграцію з платформою Android. Для візуалізації структури бази даних використано ER-діаграму (Entity-relationship diagram). Це потужний інструмент, що дозволяє ясно представити основні об’єкти бази даних [18].

Для підключення та побудови бази даних був створений клас DatabaseHelper, який виконує основні функції роботи з базою даних, такі як створення таблиць та управління ними.

База даних мобільного застосунку складається з двох основних таблиць user та expense (Рис. 2.2).

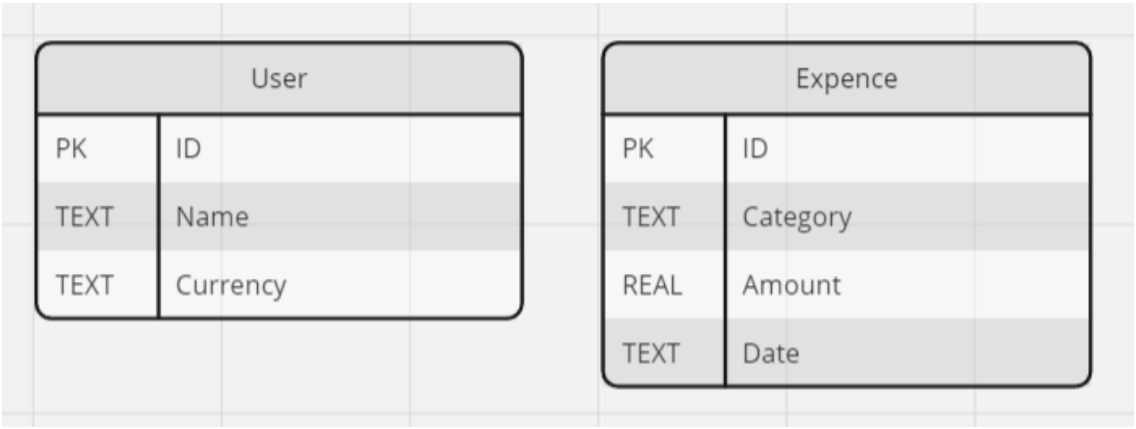


Рисунок 2.2. ER-діаграма бази даних

Таблиця user зберігає інформацію про користувача. Структура таблиці наступна:

- ID (Integer): унікальний ідентифікатор користувача;
- NAME (Text): ім'я користувача;
- CURRENCY (Text): основна валюта користувача.

Таблиця expense зберігає інформацію про витрати користувачів. Структура таблиці наступна:

- ID (Integer): унікальний ідентифікатор витрати.
  CATEGORY (Text): категорія витрати.
  AMOUNT (Real): сума витрати.
  DATE (Text): дата витрати.

2.4 Обґрунтування вибору інструментальних засобів розробки «FinanceTracker»

Для розробки мобільного застосунку було обрано кілька ключових інструментальних засобів: середовище розробки Android Studio, мову програмування Java та систему управління базами даних SQLite. Вибір цих інструментів був обґрунтований після детального аналізу існуючих технологій та їх відповідності вимогам проєкту.

Android Studio – офіційне інтегроване середовище розробки (IDE) для створення застосунків на платформі Android. Розроблене компанією Google, воно базується на потужному IntelliJ IDEA від JetBrains і пропонує широкий спектр інструментів для підвищення продуктивності. Використання Android Studio забезпечить тісну інтеграцію з усіма Android SDK та бібліотеками, що є важливим для створення якісного продукту. Крім того, воно включає додаткові функції, які ідеально підходять для розробки нашого фінансового додатку, такі як: потужний емулятор для тестування різних сценаріїв використання, інтелектуальний редактор коду з підказками та миттєвим виправленням помилок, візуальний редактор інтерфейсів.

Мова програмування Java була обрана для реалізації основної логіки застосунку з кількох причин. Java є однією з найпоширеніших мов програмування для розробки програмних продуктів, завдяки своїй стабільності, безпеці та великій спільноті розробників. Це дозволяє швидко знаходити вирішення виникаючих проблем та використовувати численні бібліотеки і

фреймворки, що спрощують розробку. Крім того, Java забезпечує платформну незалежність завдяки використанню віртуальної машини Java (JVM), що гарантує роботу застосунку на різних пристроях з операційною системою Android.

Для зберігання даних у застосунку було обрано SQLite – легку, вбудовану систему управління базами даних. SQLite відома своєю простотою інтеграції в мобільні застосунки, високою продуктивністю та надійністю [19]. Використання SQLite дозволяє зберігати дані локально на пристрої користувача, що забезпечує швидкий доступ до них без необхідності постійного підключення до інтернету.

Таким чином, вибір Android Studio, Java та SQLite як основних інструментальних засобів для розробки мобільного застосунку «FinanceTracker» є оптимальним з точки зору продуктивності, надійності та зручності використання, що дозволяє забезпечити високоякісну реалізацію поставленої задачі.

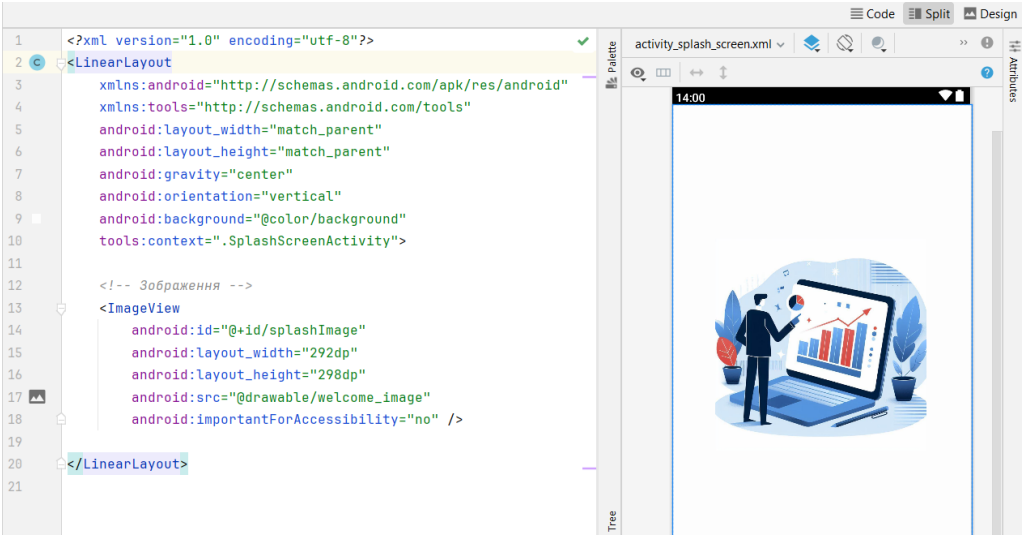
2.5 Особливості програмної реалізації та основні режими роботи «FinanceTracker»

Розробка мобільного застосунку «FinanceTracker» включає реалізацію низки важливих алгоритмів та компонентів, які забезпечують його функціональність та зручність використання. Інтерфейс користувача є важливою складовою будь-якого мобільного застосунку, оскільки він забезпечує взаємодію користувача з програмою. У цьому підрозділі розглядаються особливості програмної реалізації, включаючи опис основних модулів і функцій застосунку.

Спочатку створюємо стартовий екран завантаження, призначений для відображення логотипу з анімацією під час запуску застосунку.

Створений XML-файл (activity_splash_screen) визначає вигляд стартового екрану (Рис. 2.3). Він використовує контейнер LinearLayout для центрованого розміщення елементів. В якості логотипу використовується зображення ImageView, яке знаходиться у ресурсах програми.

Розроблено клас SplashScreenActivity, що відповідає за функціональність стартового екрану. У методі onCreate задається контент вікна з файлу activity_splash_screen.xml, після чого ініціалізується ImageView та застосовується анімація збільшення «zoom_in». Анімація дозволяє надати додаткову привабливість стартовому екрану (Рис 2.4).




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:background="@color/background"
    tools:context=".SplashScreenActivity">

    <!-- Зображення -->
    <ImageView
        android:id="@+id/splashImage"
        android:layout_width="292dp"
        android:layout_height="298dp"
        android:src="@drawable/welcome_image"
        android:importantForAccessibility="no" />

</LinearLayout>
```

Рисунок 2.3. Макет стартового екрану

```java
public class SplashScreenActivity extends AppCompatActivity {
    private static final int SPLASH_TIME_OUT = 500;
    private DatabaseHelper databaseHelper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash_screen);

        // Застосування анімації до ImageView
        ImageView imageView = findViewById(R.id.splashImage);
        Animation zoomIn = AnimationUtils.loadAnimation(this, R.anim.zoom_in);
        imageView.startAnimation(zoomIn);

        databaseHelper = new DatabaseHelper(this);
        // Перевірка чи існує користувач в базі та перехід до відповідного Activity
        new Handler().postDelayed(() -> {
            if (databaseHelper.isUserExist()) {
                Intent intent = new Intent(SplashScreenActivity.this, MainScreen.class);
                startActivity(intent);
            } else {
                Intent intent = new Intent(SplashScreenActivity.this, MainActivity.class);
                startActivity(intent);
            }
            overridePendingTransition(0, 0);
            finish();
        }, SPLASH_TIME_OUT);
    }
}
```

Рисунок 2.4. Реалізація класу SplashScreenActivity

Для зберігання даних використовуємо базу даних SQLite. Всі операції з базою даних здійснюються через створений клас DatabaseHelper, який є похідним від SQLiteOpenHelper (Рис. 2.5).

Метод onCreate класу DatabaseHelper викликається при першому створенні бази даних і створює необхідні таблиці. Використовуються SQL-запити для створення таблиць user та expense.

```
14  public class DatabaseHelper extends SQLiteOpenHelper {
15      private static final String DATABASE_NAME = "finance_tracker.db";
16      private static final int DATABASE_VERSION = 1;
17      private static final String TABLE_USER = "user";
18      private static final String COLUMN_ID = "id";
19      private static final String COLUMN_NAME = "name";
20      private static final String COLUMN_CURRENCY = "currency";
21      private static final String TABLE_EXPENSES = "expense";
22      private static final String COLUMN_EXPENSE_ID = "id";
23      private static final String COLUMN_EXPENSE_CATEGORY = "category";
24      private static final String COLUMN_EXPENSE_AMOUNT = "amount";
25      private static final String COLUMN_EXPENSE_DATE = "date";
26      public DatabaseHelper(Context context) {
27          super(context, DATABASE_NAME, null, DATABASE_VERSION);
28      }
29      @Override // Метод викликається при першому створенні бази даних, та створює таблиці
30  @   public void onCreate(SQLiteDatabase db) {
31          String createUserTable = "CREATE TABLE " + TABLE_USER + "("
32                  + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
33                  + COLUMN_NAME + " TEXT, "
34                  + COLUMN_CURRENCY + " TEXT)";
35          db.execSQL(createUserTable);
36          String CREATE_EXPENSE_TABLE = "CREATE TABLE " + TABLE_EXPENSES + "("
37                  + COLUMN_EXPENSE_ID + " INTEGER PRIMARY KEY AUTOINCREMENT,"
38                  + COLUMN_EXPENSE_CATEGORY + " TEXT,"
39                  + COLUMN_EXPENSE_AMOUNT + " REAL,"
40                  + COLUMN_EXPENSE_DATE + " TEXT" + ")";
41          db.execSQL(CREATE_EXPENSE_TABLE);
42      }
```

Рисунок 2.5. Реалізація класу DatabaseHelper

Метод addUser додає нового користувача разом з обраною валютою до бази даних. Метод isUserExist перевіряє, чи є інформація про користувача в базі даних (Рис. 2.6).

```
52      // Метод додає користувача та обрану валюту до бази даних
53      public boolean addUser(String name, String currency) {
54          SQLiteDatabase db = this.getWritableDatabase();
55          ContentValues values = new ContentValues();
56          values.put(COLUMN_NAME, name);
57          values.put(COLUMN_CURRENCY, currency);
58
59          long result = db.insert(TABLE_USER, null, values);
60          return result != -1; // Якщо результат -1, вставка не вдалася
61      }
62
63      // Метод перевіряє, чи існує інформація про користувача в базі даних
64      public boolean isUserExist() {
65          SQLiteDatabase db = this.getReadableDatabase();
66          Cursor cursor = db.rawQuery("SELECT * FROM " + TABLE_USER, null);
67          boolean exists = cursor.getCount() > 0;
68          cursor.close();
69          return exists;
70      }
```

Рисунок 2.6. Реалізація методів addUser і isUserExist

Після стартового екрану програма переходить до екрану привітання, який містить елементи тексту, зображення та кнопки для переходу до наступної сторінки (Рис. 2.7).

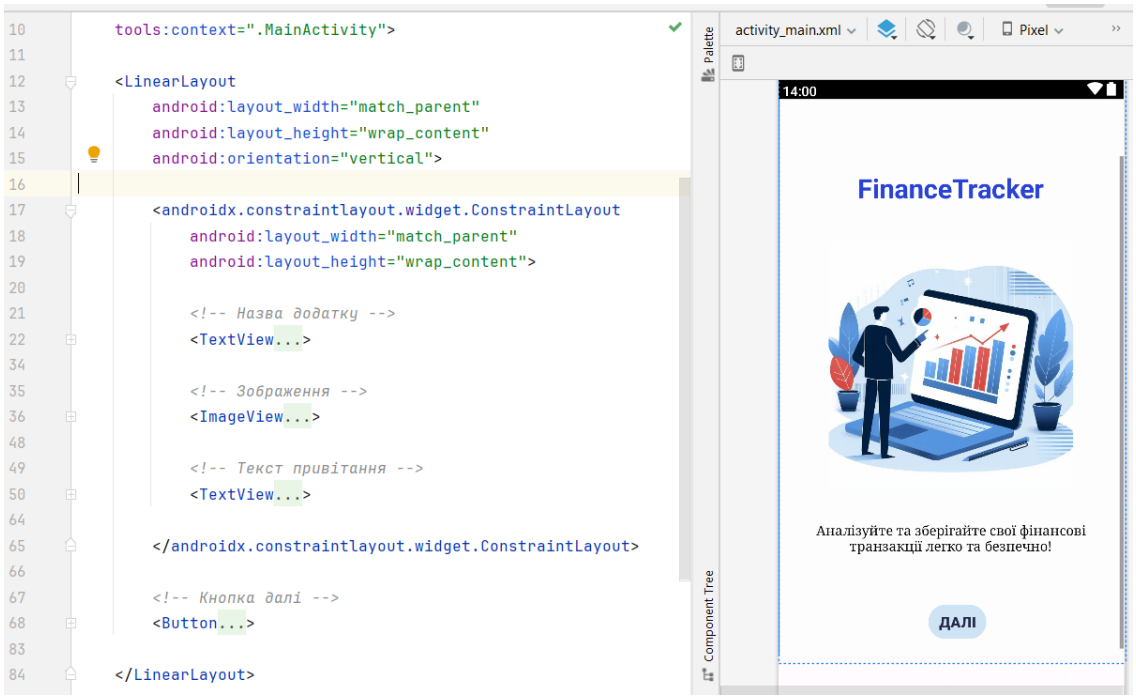


Рисунок 2.7. Макет екрану привітання

У класі MainActivity реалізується функціональність цієї сторінки:

- ініціалізація активності: метод onCreate встановлює макет з файлу activity_main.xml;
- обробка натискання на кнопку: знаходиться кнопка за допомогою методу findViewById, після чого додається слухач натискань OnClickListener. При натисканні кнопки здійснюється перехід до наступної активності NameInputActivity за допомогою Intent. Крім того, встановлюється нульова анімація переходу між екранами та завершується поточна активність (Рис. 2.8).

```
1  package com.example.financetracker;
2  import android.content.Intent;
3  import android.os.Bundle;
4  import android.widget.Button;
5  import androidx.appcompat.app.AppCompatActivity;
6
7  public class MainActivity extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_main);
13
14         Button nextButton = findViewById(R.id.nextButton);
15         nextButton.setOnClickListener(v -> {
16             Intent intent = new Intent(MainActivity.this, NameInputActivity.class);
17             startActivity(intent);
18             overridePendingTransition(0, 0);
19             finish();
20         });
21     }
22 }
```

Рисунок 2.8. Реалізація класу MainActivity

Після екрану привітання, користувач переходить до екрану введення імені та вибору основної валюти (Рис. 2.9).

На екрані розміщені:

- TextView для введення імені, який запрошує користувача ввести своє ім'я;
- EditText для введення імені користувача;

- TextView для вибору валюти, який запрошує користувача вибрати основну валюту;
- RadioGroup з трьома RadioButton для вибору основної валюти: гривня (UAH), долар США (USD) або євро (EUR);
- TextView підказка, яка пояснює що змінити валюту потім не можна;
- кнопка для переходу до наступного екрану.

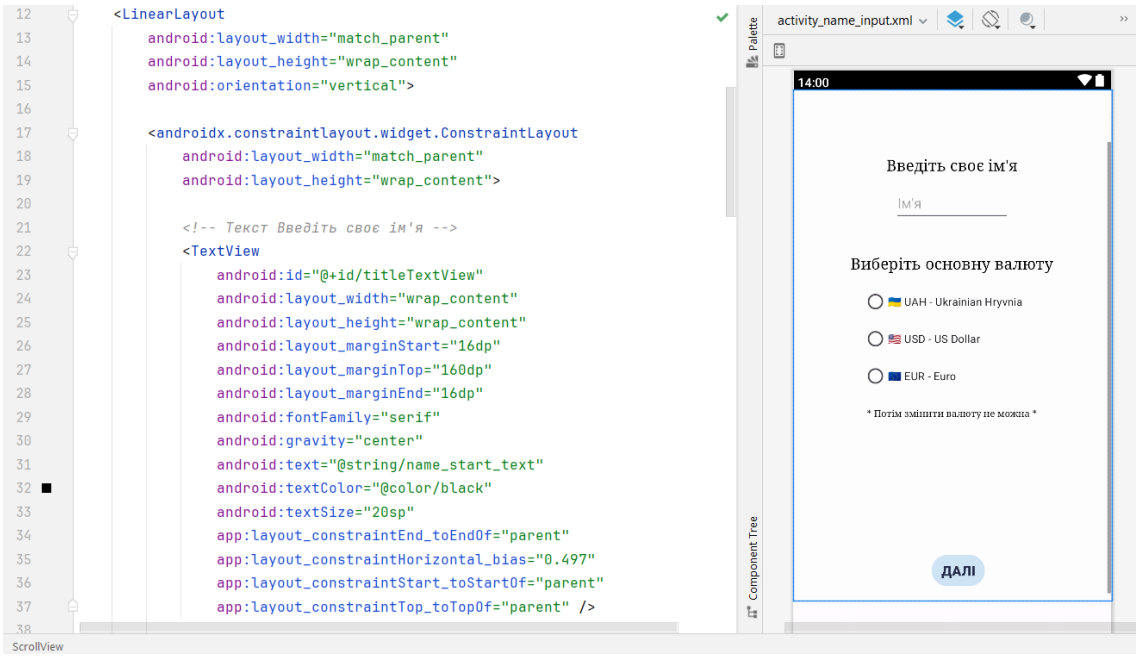


Рисунок 2.9. Макет екрану введення імені

У класі NameInputActivity реалізовано логіку обробки введених даних. У методі onCreate відбувається ініціалізація компонентів екрану, таких як EditText, RadioGroup та Button. При натисканні на кнопку «Далі» виконується перевірка, чи ввів користувач своє ім'я та вибрав валюту. Якщо одне з полів не заповнене, користувач отримує повідомлення про необхідність введення цих даних.

Якщо всі дані введені коректно, відбувається запис імені та валюти користувача в базу даних за допомогою класу DatabaseHelper та його методу addUser. Після успішного запису даних, додаток переходить до головного екрану MainScreen. У разі помилки збереження даних, користувач отримує відповідне повідомлення (Рис. 2.10).

```
nextButton.setOnClickListener(v -> {
    String name = nameEditText.getText().toString().trim();
    int selectedCurrencyId = currencyRadioGroup.getCheckedRadioButtonId();
    // Перевірка чи користувач ввід ім'я та вибрав валюту
    if (name.isEmpty()) {
        Toast.makeText(NameInputActivity.this, "Будь ласка, введіть своє ім'я", Toast.LENGTH_SHORT).show();
    } else if (selectedCurrencyId == -1) {
        Toast.makeText(NameInputActivity.this, "Будь ласка, виберіть валюту", Toast.LENGTH_SHORT).show();
    } else {
        // Отримання коду валюти на основі вибраного RadioButton
        String currencyCode = "";
        if (selectedCurrencyId == R.id.uahRadioButton) {
            currencyCode = "UAH";
        } else if (selectedCurrencyId == R.id.usdRadioButton) {
            currencyCode = "USD";
        } else if (selectedCurrencyId == R.id.eurRadioButton) {
            currencyCode = "EUR";
        }
        // Якщо користував ввів ім'я і вибрав валюту то при натисканні кнопки далі, дані запишуться в базу даних
        // Та відкриється наступна сторінка
        boolean isInserted = databaseHelper.addUser(name, currencyCode);
        if (isInserted) {
            Intent intent = new Intent(NameInputActivity.this, MainScreen.class);
            startActivity(intent);
            overridePendingTransition(0, 0);
            finish();
        } else {
            Toast.makeText(NameInputActivity.this, "Error saving data", Toast.LENGTH_SHORT).show();
        }
    }
```

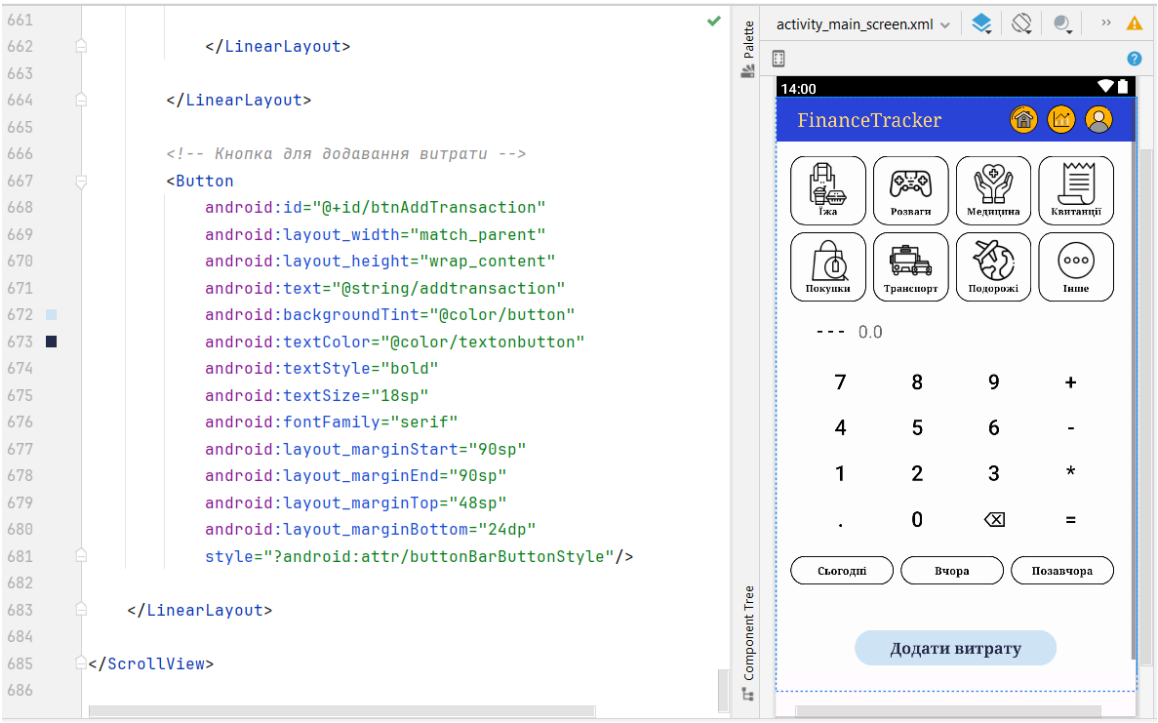
Рисунок 2.10. Реалізація класу NameInputActivity

Після введення імені та вибору валюти, користувач попадає на основну сторінку з додавання витрат (Рис. 2.11).

На екрані розміщені такі елементи:

- основний контейнер: використовується ScrollView, що дозволяє прокручувати вміст, якщо він перевищує висоту екрана. Внутрішній контейнер – LinearLayout з вертикальною орієнтацією для організації елементів в стовпець;
- меню сторінки: LinearLayout з горизонтальною орієнтацією, що містить назву додатка та три іконки (дім, статистика, профіль);
- ряди категорій: LinearLayout з горизонтальною орієнтацією, що містить по чотири LinearLayout з категоріями витрат. Кожна категорія складається з ImageView (іконка категорії) та TextView (назва категорії);
- поля валюти та суми: LinearLayout з горизонтальною орієнтацією, що містить два TextView: один для відображення вибраної валюти, інший для введення суми витрат;

- клавіатура: складена з чотирьох LinearLayout, кожен з яких містить ряд кнопок (цифри 0-9, арифметичні оператори, крапка, кнопки видалення та обчислення результату);
- вибір дати витрати: LinearLayout з горизонтальною орієнтацією, що містить три LinearLayout для вибору дати витрати (сьогодні, вчора, позавчора);
- кнопка додавання витрати: Button для підтвердження і додавання витрати.




```
661
662            </LinearLayout>
663
664        </LinearLayout>
665
666        <!-- Кнопка для додавання витрати -->
667        <Button
668            android:id="@+id/btnAddTransaction"
669            android:layout_width="match_parent"
670            android:layout_height="wrap_content"
671            android:text="@string/addtransaction"
672            android:backgroundTint="@color/button"
673            android:textColor="@color/textonbutton"
674            android:textStyle="bold"
675            android:textSize="18sp"
676            android:fontFamily="serif"
677            android:layout_marginStart="90sp"
678            android:layout_marginEnd="90sp"
679            android:layout_marginTop="48sp"
680            android:layout_marginBottom="24dp"
681            style="?android:attr/buttonBarButtonStyle"/>
682
683    </LinearLayout>
684
685</ScrollView>
686
```

Рисунок 2.11. Макет екрану додавання витрат

Клас MainScreen відповідає за функціонал основної сторінки. Ось деякі ключові аспекти цього класу:

- метод onCreate (Рис. 2.12): встановлює макет активності, ініціалізує змінні. Налаштовує обробники для кнопок меню, кнопок чисел, операцій, крапки, видалення та обчислення. Налаштовує обробники для вибору категорій та днів. Відображає валюту користувача. Налаштовує обробник для додавання витрати;

- метод addExpense (Рис. 2.13): перевіряє наявність всіх потрібних даних (категорія, день, сума). Отримує категорію та дату. Перевіряє коректність введеної суми. Додає витрату в базу даних. Очищує поля після додавання витрати;
- метод calculateResult (Рис. 2.14): обчислює введений вираз за допомогою примітивного інтерпретатора. Форматує та відображає результат.
- примітивний інтерпретатор eval (Рис. 2.15): реалізує рекурсивний інтерпретатор для обчислення математичних виразів. Класи всередині eval обробляють символи виразу та повертають результат обчислення.

Принцип роботи:

- користувач обирає категорію витрат та день;
- користувач вводить суму витрати за допомогою створеної клавіатури;
- користувач може ввести вираз, який буде обчислений при натисканні на кнопку результату;
- після введення всіх даних, користувач натискає кнопку додати витрату, програма перевіряє чи користувач заповнив всі поля, якщо так, то дані зберігаються в базі даних, якщо ні, користувач отримає відповідне повідомлення.

```
26      @Override
27      protected void onCreate(Bundle savedInstanceState) {
28
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.activity_main_screen);
31
32          enterSumMain = findViewById(R.id.enterSumMain);
33          inputExpression = new StringBuilder();
34          isResultDisplayed = false;
35          selectedCategory = null;
36          selectedDay = null;
37          databaseHelper = new DatabaseHelper(this);
38
39          // Обробники кнопок меню
40          ImageView btnHome = findViewById(R.id.btnHome);
41          btnHome.setOnClickListener(v -> {
42              Intent intent = new Intent(MainScreen.this, MainScreen.class);
43              startActivity(intent);
44              overridePendingTransition(0, 0);
45          });
46
```

Рисунок 2.12. Метод onCreate у класі MainScreen

```java
// Метод для додавання витрати в базу даних
private void addExpense() {
    if (selectedCategory == null || selectedDay == null || enterSumMain.getText().toString().isEmpty()) {
        Toast.makeText(this, "Вкажіть усі потрібні дані", Toast.LENGTH_SHORT).show();
        return;
    }
    String category = getCategory(selectedCategory.getId());
    String date = getDate(selectedDay.getId());
    // Перевірка чи введені дані є дійсним числом
    double amount;
    try {
        amount = Double.parseDouble(enterSumMain.getText().toString());
    } catch (NumberFormatException e) {
        Toast.makeText(this, "Натисніть '=' щоб порахувати приклад!", Toast.LENGTH_SHORT).show();
        return;
    }
    // Перевірка чи сума додатна, більша за 0.5 і менша за 100 000
    if (amount < 0.5 || amount > 100000) {
        Toast.makeText(this, "Витрата повинна бути не менше 0.5 і не більше 100 000", Toast.LENGTH_SHORT).show();
        return;
    }
    databaseHelper.addExpense(category, amount, date);   // Додавання витрати в базу даних
    enterSumMain.setText("0.0");     // Очищення полів після додавання витрати
    inputExpression.setLength(0);    // Очистити inputExpression
    isResultDisplayed = false;       // Скидання прапорця після очищення поля
    if (selectedCategory != null) {
        selectedCategory.setBackgroundResource(R.drawable.category_border);
        selectedCategory = null;
    }
```

Рисунок 2.13. Метод addExpense у класі MainScreen

```java
// Метод обчислює вираз, введений у inputExpression
// Використовує рекурсивний інтерпретатор для оцінки виразів, який реалізований в класі Parser
private void calculateResult() {
    try {
        double result = eval(inputExpression.toString());
        DecimalFormat df = new DecimalFormat("0.#", DecimalFormatSymbols.getInstance(Locale.US));
        String formattedResult = df.format(result);
        enterSumMain.setText(formattedResult);
        inputExpression.setLength(0);
        inputExpression.append(formattedResult);
        isResultDisplayed = true;
    }
    catch (Exception e) {
        enterSumMain.setText("Error");
        inputExpression.setLength(0);
    }
}
```

Рисунок 2.14. Метод calculateResult у класі MainScreen

```
296
297         // Примітивний інтерпретатор для обчислення виразів
298         private double eval(final String str) {
299             class Parser {
300                 int pos = -1, c; // Змінні використовується для відстеження позиції поточного символу з виразу
301
302                 // Метод отримує наступний символ з виразу
303                 void eatChar() { c = (++pos < str.length()) ? str.charAt(pos) : -1; }
306
307                 // Метод для того, щоб ігнорувати зайві пробіли у виразі.
308                 void eatSpace() { while (Character.isWhitespace(c)) eatChar(); }
311
312                 // Метод викликає методи для обробки виразу та повертає результат
313                 double parse() {
314                     eatChar();
315                     double v = parseExpression();
316                     if (c != -1) throw new RuntimeException("Unexpected: " + (char) c);
317                     return v;
318                 }
319
320                 // Метод обробляє вираз, який може містити додавання та віднімання
321                 double parseExpression() {
322                     double v = parseTerm();
323                     while (true) {
324                         eatSpace();
325                         if (c == '+') {
326                             eatChar();
327                             v += parseTerm();
328                         } else if (c == '-') {
```

Рисунок 2.15. Примітивний інтерпретатор eval у класі MainScreen

Також в класі DatabaseHelper створено два методи отримання валюти користувача та додавання витрат в базу даних (Рис. 2.16).

Метод getUserCurrency використовується для отримання валюти користувача з бази даних. Спочатку відкривається база даних у режимі читання, використовуючи getReadableDatabase. Потім виконується запит до таблиці користувачів USER з метою отримати значення колонки валюти CURRENCY. Якщо запит повертає результати (курсором не є null і він містить принаймні один запис), з першого рядка вибирається значення валюти, курсор закривається, і значення валюти повертається. Якщо жодних результатів не знайдено, метод повертає null.

Метод addExpense використовується для додавання витрат до бази даних. Спочатку відкривається база даних у режимі запису, використовуючи getWritableDatabase. Потім створюється об'єкт ContentValues, в який додаються значення категорії витрати, суми витрати та дати витрати. За допомогою методу insert ці дані вставляються у таблицю витрат EXPENSE. Після цього база даних закривається.

```
82      // Метод отримує валюту користувача з бази даних
83      public String getUserCurrency() {
84          SQLiteDatabase db = this.getReadableDatabase();
85          Cursor cursor = db.query(TABLE_USER, new String[]{COLUMN_CURRENCY}, null, null, null, null, null);
86          if (cursor != null && cursor.moveToFirst()) {
87              String currency = cursor.getString(0);
88              cursor.close();
89              return currency;
90          } else {
91              return null;
92          }
93      }
94
95      // Метод додає нову витрату до бази даних
96      public void addExpense(String category, double amount, String date) {
97          SQLiteDatabase db = this.getWritableDatabase();
98          ContentValues values = new ContentValues();
99          values.put(COLUMN_EXPENSE_CATEGORY, category);
100         values.put(COLUMN_EXPENSE_AMOUNT, amount);
101         values.put(COLUMN_EXPENSE_DATE, date);
102         db.insert(TABLE_EXPENSES, null, values);
103         db.close();
104     }
```

Рисунок 2.16. Метод getUserCurrency та addExpense

Меню застосунку знаходиться у верхній частині екрану. Воно містить три іконки, при натисканні яких відкривається потрібна сторінка (Рис. 2.17).

Перша (Дім) оновлює основну сторінку. Друга (Статистика) відкриє сторінку для огляду статистики витрат. Третя (Профіль) відкриє сторінку редагування профілю. Всі три іконки так само доступні на цих сторінках.

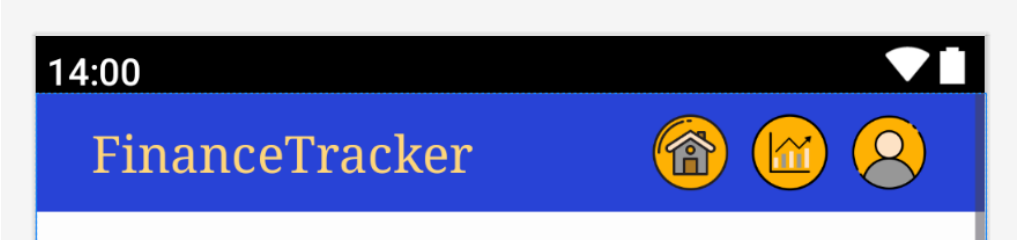


Рисунок 2.17. Меню застосунку

Після натискання іконки Статистика, відкривається сторінка аналізу витрат (Рис 2.18).

Верхня частина інтерфейсу займає горизонтальне меню з кнопками для переходу на головну сторінку, сторінку статистики та профіль користувача. Далі йдуть текстовий заголовок і кнопки для вибору періоду (сьогодні, тиждень, місяць, рік), які дозволяють користувачу вибрати період для відображення

діаграми витрат. Нижче розташовані текстове поле, яке відображає загальну суму витрат за обраний період, і кругова діаграма (PieChart). Під діаграмою знаходиться список категорій витрат з відповідними кольорами.

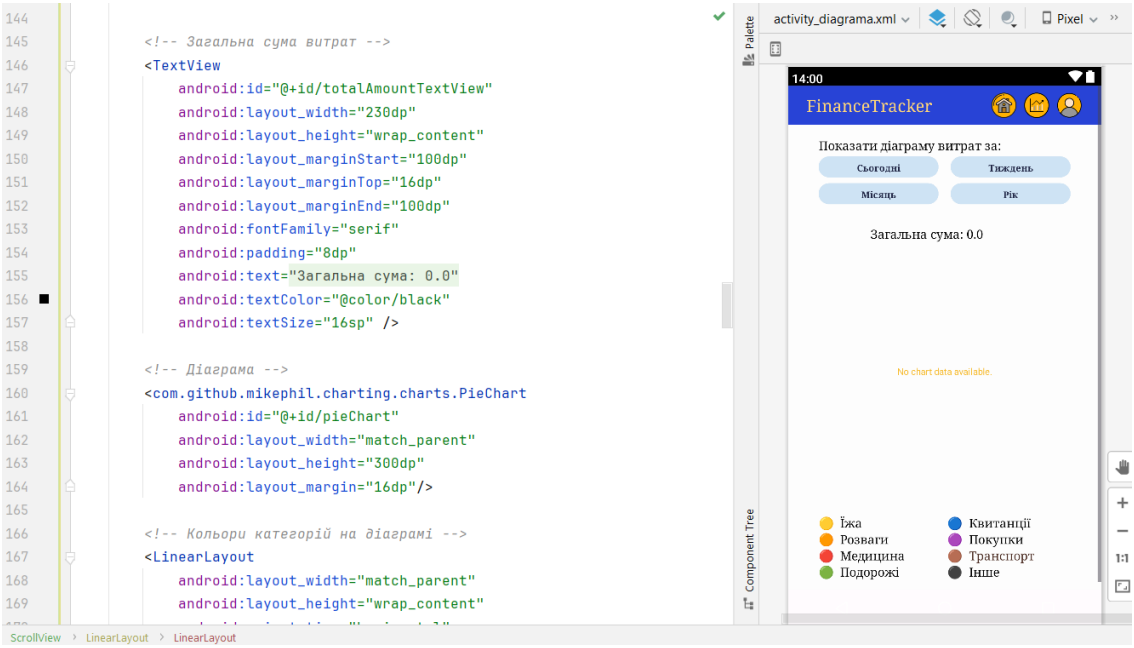


Рисунок 2.18. Макет екрану відображення статистики

Для реалізації діаграми витрат була підключена бібліотека MPAndroidChart з репозиторію GitHub [20]. Ця бібліотека надає можливість побудови різноманітних типів діаграм, у тому числі кругових. Підключення бібліотеки здійснюється у файлі build.gradle шляхом додавання відповідного запису в секцію dependencies (Рис. 2.19).

Рисунок 2.19. Підключення бібліотеки MPAndroidChart

Розроблено Java-клас MainDiagrama, який відповідає за функціональність сторінки статистики витрат.

Метод onCreate викликається при створенні активності. Він налаштовує зовнішній вигляд сторінки, прив’язуючи XML-розмітку до коду та ініціалізуючи ключові компоненти, такі як PieChart, TextView для загальної суми та DatabaseHelper для роботи з базою даних. Також додаються обробники натискань для кнопок меню та періодів, які викликають відповідні методи для завантаження даних діаграми.

Метод setupPieChart налаштовує зовнішній вигляд діаграми, включаючи відображення значень у відсотках, вимкнення опису діаграми, налаштування кольору та вигляду.

Метод loadEmptyPieChart завантажує пусту діаграму з повідомленням «Дані відсутні», коли немає даних для відображення (Рис. 2.20).

```
74      // Метод налаштовує зовнішній вигляд діаграми.
75      private void setupPieChart() {
76          pieChart.setUsePercentValues(true);           // Включає відображення значень витрат у відсотках.
77          pieChart.getDescription().setEnabled(false);  // Вимкнення опису діаграми.
78          pieChart.setDrawHoleEnabled(true);            // Залишає пусте місце в центрі діаграми.
79          pieChart.setHoleColor(android.R.color.white); // Встановлює колір центру білим.
80          pieChart.setTransparentCircleRadius(60f);     // Встановлює розмір прозорого кола навколо діаграми.
81          pieChart.getLegend().setEnabled(false);       // Вимкнення легенди (списку категорій з бази даних)
82      }
83
84      // Метод завантажує пусту діаграму
85      private void loadEmptyPieChart() {
86          List<PieEntry> entries = new ArrayList<>();
87          entries.add(new PieEntry(1f, "Дані відсутні"));
88          PieDataSet dataSet = new PieDataSet(entries, "");
89          dataSet.setColors(android.graphics.Color.LTGRAY);
90          PieData data = new PieData(dataSet);
91          data.setValueTextSize(13f);
92          data.setValueTextColor(android.graphics.Color.BLACK);
93
94          pieChart.setData(data);
95          pieChart.invalidate();
96      }
```

Рисунок 2.20. Методи setupPieChart та loadEmptyPieChart

Метод loadPieChartData завантажує дані про витрати з бази даних на основі обраного періоду (сьогодні, тиждень, місяць, рік), створює список об’єктів PieEntry для діаграми, обчислює загальну суму витрат та налаштовує дані для

відображення на діаграмі. Якщо даних немає, метод відображає порожню діаграму та повідомлення про відсутність даних (Рис. 2.21).

```
100     // Метод завантажує та відображає дані про витрати
101     private void loadPieChartData(String period) {
102         List<PieEntry> entries = new ArrayList<>();
103         List<Expense> expenses = databaseHelper.getExpensesByPeriod(period);
104         // Якщо список витрат пустий, відображає порожню діаграму та повідомлення про відсутність даних.
105         if (expenses.isEmpty()) {
106             loadEmptyPieChart();
107             Toast.makeText(this, "Дані для обраного періоду відсутні", Toast.LENGTH_SHORT).show();
108             return;
109         }
110         double totalAmount = 0;
111         for (Expense expense : expenses) {  // обчислюємо загальну суму витрат та формуємо список об'єктів
112             entries.add(new PieEntry((float) expense.getAmount(), ""));
113             totalAmount += expense.getAmount();
114         }
115         // Створює набір даних для діаграми PieData з отриманим списком PieEntry та налаштовує його формат.
116         PieDataSet dataSet = new PieDataSet(entries, "");
117         List<Integer> colors = new ArrayList<>();
118         for (Expense expense : expenses) {
119             colors.add(categoryColors.getOrDefault(expense.getCategory(), ColorTemplate.COLORFUL_COLORS[0]));
120         }
121         dataSet.setColors(colors);
122         PieData data = new PieData(dataSet);
123         data.setValueTextSize(13f);
124         data.setValueTextColor(Color.WHITE);
125         data.setValueFormatter(new PercentFormatter(pieChart));
126         pieChart.setData(data);
127         pieChart.invalidate();
128         totalAmountTextView.setText("Загальна сума: " + String.format(Locale.getDefault(), "%.1f", totalAmount));
```

Рисунок 2.21. Метод loadPieChartData

Метод initializeCategoryColors визначає кольори для кожної категорії витрат для відображення на діаграмі, що дозволяє легко ідентифікувати категорії за кольорами (Рис. 2.22).

```
131     // Метод співвідносить категорії з кольорами для відображення на діаграмі
132     private void initializeCategoryColors() {
133         categoryColors = new HashMap<>();
134         categoryColors.put("Food", Color.parseColor("#FECB32"));
135         categoryColors.put("Game", Color.parseColor("#FE9700"));
136         categoryColors.put("Health", Color.parseColor("#CA3025"));
137         categoryColors.put("Receipt", Color.parseColor("#3CAC54"));
138         categoryColors.put("Store", Color.parseColor("#12249E"));
139         categoryColors.put("Transport", Color.parseColor("#7A21A1"));
140         categoryColors.put("Travelling", Color.parseColor("#603B2E"));
141         categoryColors.put("Others", Color.parseColor("#373535"));
142     }
```

Рисунок 2.22. Метод initializeCategoryColors

Для відображення статистики витрат були розроблені ключові методи в класі DatabaseHelper. Ці методи забезпечують отримання та обробку даних про витрати з бази даних, а також визначають дати початку та закінчення періоду для аналізу витрат.

Метод getExpensesByPeriod призначений для отримання списку витрат за певний період, таких як сьогодні, тиждень (мається на увазі поточний тиждень), місяць (поточний місяць) або рік (поточний рік). Він групує витрати за категоріями та повертає загальну суму по кожній категорії (Рис. 2.23).

Опис реалізації:

- ініціалізація: створюється порожній список expenses, в якому будуть зберігатися результати. Відкривається база даних для читання;
- формування запиту: виконується SQL-запит, який вибирає категорії витрат та сумує витрати по кожній категорії за обраний період. Дати початку та закінчення періоду отримуються за допомогою методів getStartDate і getEndDate;
- виконання запиту: запит виконується, і результати зберігаються у курсорі;
- обробка результатів: якщо курсор не порожній, він проходиться по всім записам, і для кожного запису створюється об'єкт Expense з відповідною категорією та сумою витрат. Ці об'єкти додаються до списку expenses;
- закриття курсора: після обробки даних курсор закривається;
- повернення результату: список expenses повертається як результат роботи методу.

Метод getStartDate визначає дату початку обраного періоду (сьогодні, тиждень, місяць або рік) для подальшого використання у SQL-запиті (Рис. 2.24).

Опис реалізації:

- ініціалізація: використовується об'єкт Calendar для отримання поточної дати та часу.
- форматування дати: створюється об'єкт SimpleDateFormat для форматування дати у формат yyyy-MM-dd;

- визначення дати початку: в залежності від обраного періоду змінюється дата початку. Наприклад (сьогодні) - поточна дата, (тиждень) – дата встановлюється на понеділок поточного тижня, (місяць) – дата встановлюється на перший день поточного місяця, (рік) – дата встановлюється на перший день поточного року;
- повернення дати: форматована дата повертається як результат роботи методу.

Метод getEndDate визначає поточну дату, яка використовується як кінцева дата для SQL-запиту.

```
106     // Метод отримує список витрат за певний період
107     // групує витрати за категоріями та повертає загальну суму по кожній категорії
108     public List<Expense> getExpensesByPeriod(String period) {
109         List<Expense> expenses = new ArrayList<>();
110         SQLiteDatabase db = this.getReadableDatabase();
111         Cursor cursor;
112
113         String startDate = getStartDate(period);
114         String endDate = getEndDate();
115         String query = "SELECT category, SUM(amount) as totalAmount FROM " + TABLE_EXPENSES +
116                 " WHERE date BETWEEN ? AND ? GROUP BY category";
117         cursor = db.rawQuery(query, new String[]{startDate, endDate});
118
119         if (cursor != null) {
120             if (cursor.moveToFirst()) {
121                 do {
122                     String category = cursor.getString(cursor.getColumnIndexOrThrow("category"));
123                     double amount = cursor.getDouble(cursor.getColumnIndexOrThrow("totalAmount"));
124                     expenses.add(new Expense(category, amount));
125                 } while (cursor.moveToNext());
126             }
127             cursor.close();
128         }
129         return expenses;
130     }
```

Рисунок 2.23. Метод getExpensesByPeriod

Остання створена сторінка activity_profile, призначена для редагування профілю. Зараз доступно для редагування тільки ім’я користувача (Рис. 2.25).

Меню сторінки представлено горизонтальним LinearLayout, який містить назву додатку та три кнопки у вигляді іконок для навігації: додому, статистика та профіль. Для кожної кнопки визначено відповідні зображення та описи. Після меню розташовано текстове поле TextView, яке відображає поточне ім'я

користувача, і поле для введення нового імені EditText. Для збереження змін присутня кнопка Button, яка ініціює процес оновлення імені.

```
132     // Метод для визначення дати початку обраного періоду
133 @   private String getStartDate(String period) {
134         Calendar calendar = Calendar.getInstance();
135         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd", Locale.getDefault());
136
137         switch (period) {
138             case "today":
139                 break;
140             case "week":
141                 calendar.set(Calendar.DAY_OF_WEEK, Calendar.MONDAY);
142                 break;
143             case "month":
144                 calendar.set(Calendar.DAY_OF_MONTH, 1);
145                 break;
146             case "year":
147                 calendar.set(Calendar.DAY_OF_YEAR, 1);
148                 break;
149         }
150         return sdf.format(calendar.getTime());
151     }
152
153     // Метод для визначення дати поточного дня
154 @   private String getEndDate() {
155         Calendar calendar = Calendar.getInstance();
156         SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd", Locale.getDefault());
157         return sdf.format(calendar.getTime());
158     }
```

Рисунок 2.24. Метод getStartDate та getEndDate

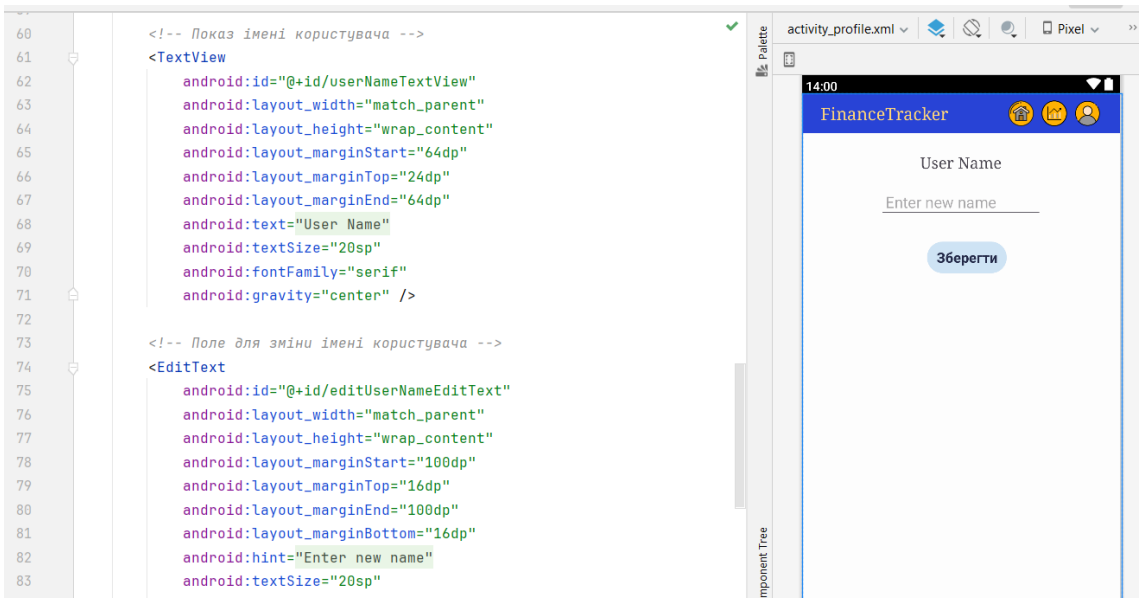


Рисунок 2.25. Макет екрану редагування профілю

Після завершення роботи над екранами, їх потрібно записати у головний файл проекту AndroidManifest.xml (Рис. 2.26).

```
18      <activity
19          android:name=".SplashScreenActivity"
20          android:exported="true"
21          android:theme="@style/Theme.Splash">
22          <intent-filter>
23              <action android:name="android.intent.action.MAIN" />
24              <category android:name="android.intent.category.LAUNCHER" />
25          </intent-filter>
26      </activity>
27
28      <activity
29          android:name=".MainActivity"
30          android:exported="false" />
31      <activity
32          android:name="NameInputActivity"
33          android:exported="false" />
34      <activity
35          android:name=".MainScreen"
36          android:exported="false" />
37      <activity
38          android:name=".MainDiagrama"
39          android:exported="false" />
40      <activity
41          android:name=".MainProfile"
42          android:exported="false" />
43
```

Рисунок 2.26. Ієрархія екранів у проекті

Також у файлі потрібно вказати назву та іконку застосунку (Рис. 2.27).

```
6       <application
7           android:allowBackup="true"
8           android:dataExtractionRules="@xml/data_extraction_rules"
9           android:fullBackupContent="@xml/backup_rules"
10          android:layout_width="292dp"
11          android:layout_height="298dp"
12          android:icon="@drawable/welcome_image"
13          android:label="FinanceTracker"
14          android:supportsRtl="true"
15          android:theme="@style/Theme.FinanceTracker"
16          tools:targetApi="31">
```

Рисунок 2.27. Опис назви та іконки застосунку

2.6 Організація тестування та налагодження програмного засобу «FinanceTracker»

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, оскільки воно забезпечує виявлення і виправлення помилок до моменту випуску продукту. Основними видами тестування є модульне, інтеграційне, системне та приймальне тестування. Модульне тестування перевіряє окремі компоненти або модулі програмного забезпечення. Інтеграційне тестування охоплює взаємодію між модулями, перевіряючи коректність їхньої спільної роботи. Системне тестування оцінює повний функціонал програми в цілому, а приймальне тестування перевіряє відповідність програмного забезпечення вимогам користувачів і бізнесу [21].

Під час розробки активно використовувалися можливості середовища програмування Android Studio для виявлення та виправлення синтаксичних помилок. Середовище в режимі реального часу підсвічувало проблемні ділянки коду, що значно прискорювало процес налагодження. Наприклад, було виправлено помилки, пов'язані з неправильним використанням типів даних, пропущеними крапками з комою та неправильними назвами змінних.

Важливим етапом було тестування на реальних пристроях. Застосунок було встановлено на кілька різних смартфонів з різними версіями Android для перевірки сумісності та коректності роботи на різних пристроях. Це дозволило виявити та усунути проблеми, пов'язані з різними розмірами екранів.

Такий підхід до тестування та налагодження дозволив створити стабільний та надійний мобільний застосунок, що відповідає поставленим вимогам.

2.7 Рекомендації по використанню та впровадженню програмного засобу «FinanceTracker»

Розроблений програмний продукт є ефективним інструментом для менеджменту особистих фінансів. Його можливості дозволяють користувачам

зручно та швидко записувати свої витрати та аналізувати їх. Застосунок має простий і зручний інтерфейс для запису витрат, аналізу фінансових даних та керування профілем користувача.

Додаток розроблений для мобільних пристроїв на базі Android, тому пристрій повинен мати операційну систему Android версії 5.0 або вище. Крім того, рекомендується мати не менше 1 ГБ оперативної пам'яті для стабільної роботи додатку та забезпечення плавності інтерфейсу.

ВИСНОВКИ

У результаті виконання роботи було розроблено мобільний застосунок для менеджменту особистих фінансів «FinanceTracker». Даний застосунок дозволяє користувачам вести облік витрат та аналізувати їх. Для реалізації проекту було використано мову програмування Java, платформу Android Studio та базу даних SQLite, що забезпечило стабільну та ефективну роботу додатку.

Основні результати роботи такі:

- проведено аналіз існуючих програмних продуктів, що дозволило визначити їхні переваги та недоліки, а також врахувати їх при розробці власного застосунку;
- розроблено архітектуру програмного продукту, включаючи проєктування бази даних та інтерфейсу користувача;
- описано процес розробки та логіки програмного продукту;
- проведено тестування та налагодження програмного продукту.

Можливе подальше дослідження в напрямку розширення функціональності застосунку, зокрема, додавання нових інструментів аналізу фінансових даних та підвищення його сумісності з різними операційними системами. Також важливо продовжувати вдосконалювати інтерфейс користувача для забезпечення максимальної зручності та ефективності використання програмного продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Статистичний щорічник України 2022. URL: https://ukrstat.gov.ua/druk/publicat/kat_u/2023/zb/11/year_22_u.pdf (дата звернення: 01.06.2024).

2. Молода наука Волині | Наука та інновації. Наука та інновації ВНУ імені Лесі Українки. URL: https://ra.vnu.edu.ua/naukove-tovarystvo/moloda-nauka-volyni (дата звернення: 01.06.2024).

3. МІЖНАРОДНА НАУКОВО-ПРАКТИЧНА КОНФЕРЕНЦІЯ «ПРОБЛЕМИ КОМП'ЮТЕРНИХ НАУК, ПРОГРАМНОГО МОДЕЛЮВАННЯ ТА БЕЗПЕКИ ЦИФРОВИХ СИСТЕМ». Прикладні проблеми комп'ютерних наук, безпеки та математики. URL: https://apcssm.vnu.edu.ua/index.php/conf/index (дата звернення: 01.06.2024).

4. Quicken - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Quicken (date of access: 01.06.2024).

5. Intuit Mint - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Intuit_Mint (date of access: 01.06.2024).

6. 50% фінансових застосунків в Україні є російськими, – дослідження - Bazilik Media. Bazilik Media. URL: https://bazilik.media/50-finansovykh-zastosunkiv-v-ukraini-ie-rosijskymy-doslidzhennia (дата звернення: 01.06.2024).

7. Android Studio - Wikipedia. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/Android_Studio (date of access: 01.06.2024).

8. Meet Android Studio | Android Developers. Android Developers. URL: https://developer.android.com/studio/intro (date of access: 01.06.2024).

9. Android Studio 2.0. QALight. URL: https://qalight.ua/baza-znaniy/android-studio-2-0/ (date of access: 01.06.2024).

10. SQLite – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/SQLite (дата звернення: 01.06.2024).

11. Навіщо потрібна SQLite: основні переваги та застосування. Pravo.home.cx.ua. URL: https://pravo.home.cx.ua/ukraincyam/navishho-potribna-sqlite-osnovni-perevagi-ta-zastosuvannya.html (дата звернення: 01.06.2024).

12. Учасники проектів Вікімедіа. Java – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 01.06.2024).

13. Що таке Java і де вона використовується – GoIT Global. GoIT Global. URL: https://goit.global/ua/articles/shcho-take-java-i-de-vona-vykorystovuietsia/ (дата звернення: 01.06.2024).

14. Monefy | Handy personal finance management tool for iOS and Android. Monefy. URL: https://monefy.me (date of access: 01.06.2024).

15. What is wallet. Wallet by BudgetBakers - Your New Personal Finance Manager. URL: https://budgetbakers.com/what-is-wallet (date of access: 01.06.2024).

16. 7cwebsite. URL: https://7csolutions.com (date of access: 01.06.2024).

17. Водоспадна модель – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Водоспадна_модель (дата звернення: 01.06.2024).

18. Бази даних. Поняття ER-моделі. Поняття сутності (entity). Атрибути. Види атрибутів | BestProg. BestProg | Програмування: теорія та практика. URL: https://www.bestprog.net/uk/2019/01/24/the-concept-of-er-model-the-concept-of-essence-and-communication-attributes-attribute-types-ua/ (дата звернення: 01.06.2024).

19. Програмування баз даних на SQLite: перші кроки в освоєнні. FoxmindEd. URL: https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/ (дата звернення: 01.06.2024).

20. GitHub - PhilJay/MPAndroidChart: A powerful Android chart view / graph view library. GitHub. URL: https://github.com/PhilJay/MPAndroidChart (date of access: 01.06.2024).

21. Тестування програмного забезпечення – Вікіпедія. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення (дата звернення: 01.06.2024).

ДОДАТКИ

Додаток А. Технічне завдання

1. Вступ

Мобільний застосунок «FinanceTracker» Призначений для персонального користування, надає користувачам можливість вести облік та аналіз власних витрат.

2. Підстави для розробки

Завдання наукового керівника на кваліфікаційну роботу за спеціальністю «Комп'ютерні науки» затверджено Волинським національним університетом імені Лесі Українки.

3. Призначення розробки

Функціональне призначення програми – забезпечення користувачів зручним інструментом для обліку та аналізу особистих фінансів.

Експлуатаційне призначення – використання на мобільних пристроях для введення, збереження та аналізу даних про витрати.

4. Вимоги до програми чи програмного продукту

4.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати введення, зберігання записів про витрати, групування витрат за категоріями, створення звітів за вибраний період. Вхідні дані включають категорію витрати, суму та дату. Вихідні дані – діаграми, що відображають статистику витрат.

4.2 Вимоги до надійності

Мобільний застосунок повинен відповідати призначенню та стабільно працювати.

4.3 Умови експлуатації

Щоб користуватися мобільним застосунком, на мобільний пристрій потрібно встановити apk-файл та запустити його.

4.4 Вимоги до складу і параметрів технічних засобів

Необхідний мобільний пристрій з операційною системою Android версії 5.0 або вище, процесором з тактовою частотою не менше 1.5 ГГц та оперативною пам'яттю не менше 1 ГБ.

4.5 Вимоги до інформаційної і програмної сумісності

Програма повинна забезпечувати сумісність з файловими системами Android, використовувати мову програмування Java, підтримувати інтеграцію з базою даних SQLite.

4.6 Вимоги до маркування і упаковки

Маркування додатку повинно включати назву програми та логотип. Упаковка – цифрова, у вигляді apk-файлу.

4.7 Вимоги до транспортування і збереження

Вимоги до транспортування і збереження відсутні

5. Вимоги до програмної документації

Документація повинна включати:

- технічне завдання;
- інструкція користувачу.

6. Техніко-економічні показники

Використання мобільного застосунку не передбачає з боку користувача будь-яких витратних операцій.

7. Стадії і етапи розробки

Етапи розробки включають:

- аналіз вимог користувачів;
- проєктування архітектури та інтерфейсу;
- розробка основної функціональності;
- тестування та налагодження.

8. Порядок контролю і приймання

Прийняття роботи включає функціональні випробування, тестування на стабільність та відповідність вимогам технічного завдання. Програма вважається прийнятою після успішного проходження всіх видів випробувань.

Додаток Б. Інструкція користувачу

1. **Загальні відомості**

Назва розробки: «FinanceTracker» – Мобільний застосунок для менеджменту особистих фінансів.

2. **Функціональне призначення**

Мобільний застосунок призначений для обліку та аналізу особистих фінансових витрат. Програма дозволяє користувачам вводити дані про витрати, їх категоризацію, а також створювати звіти для подальшого аналізу.

3. **Умови застосування програми**

Для коректної роботи застосунку необхідний мобільний пристрій з операційною системою Android версії 5.0 або вище. Пристрій повинен мати процесор з частотою не менше 1 ГГц. Не потребує підключення до інтернету.

4. **Повідомлення оператору**

Повідомлення та помилки виводяться в текстовому форматі у мобільному застосунку. Приклади повідомлень:

- «введіть своє ім'я»: з'являється, якщо поле для введення імені користувача залишене порожнім при спробі зберегти зміни;
- «успішно оновлено»: з'являється після успішного оновлення імені користувача в базі даних;
- «вкажіть усі потрібні дані»: з'являється, якщо під час додавання витрати користувач не ввів суму витрати або не обрав категорію і день.

5. **Опис роботи програми**

Робота з «FinanceTracker» починається зі встановлення застосунку на мобільний пристрій.

Якщо користувач відкриває застосунок вперше, він бачить екран привітання (Рис. Б.1) та екран де потрібно обов’язково ввести своє ім’я та обрати основну валюту (Рис. Б.2), якщо вже раніше вводив ці дані то програма відразу буде відкривати Головний екран.

Рисунок Б.1. Екран привітання

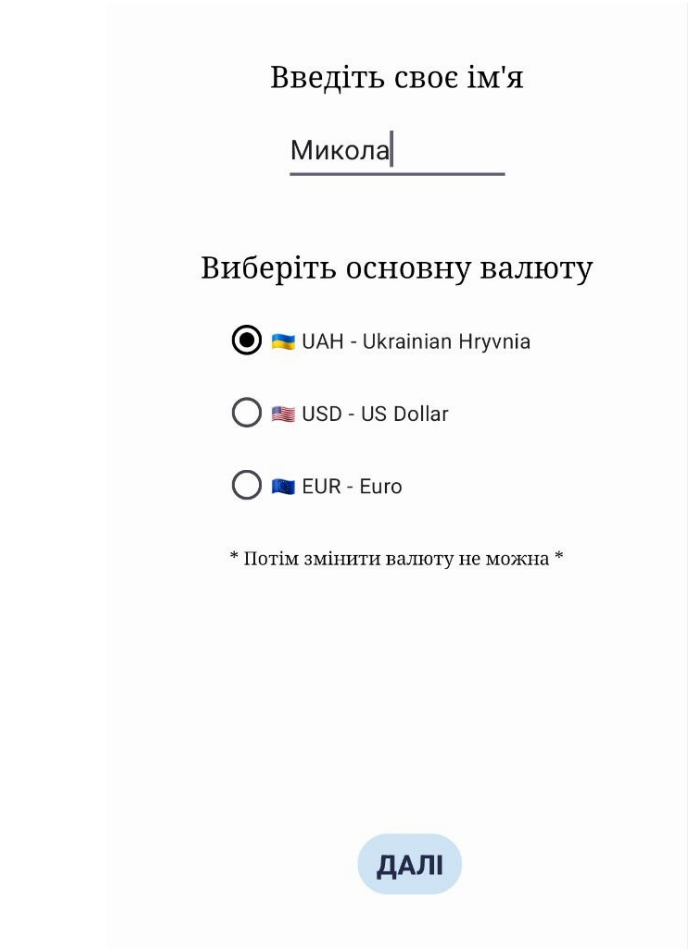


Рисунок Б.2. Екран введення даних

Після натискання кнопки ДАЛІ програма збереже інформацію в базі даних та відкриє головний екран (Рис. Б.3). Для того щоб добавити витрату потрібно обов’язково обрати категорію та день і ввести суму, потім натиснути кнопку Добавити витрату. Якщо користувач не виконав якусь дію програма покаже повідомлення що потрібно вказати усі дані витрати (Рис. Б.4). Також існує обмеження на суму витрати (Рис. Б.5).

У верхній частині застосунку присутнє меню для переходу на інші сторінки такі як: Головна сторінка, Статистика, Профіль. Якщо натиснути на середню іконку, відкриється сторінка статистики (Рис. Б.6).

Рисунок Б.3. Головний екран

Рисунок Б.4. Додавання витрати

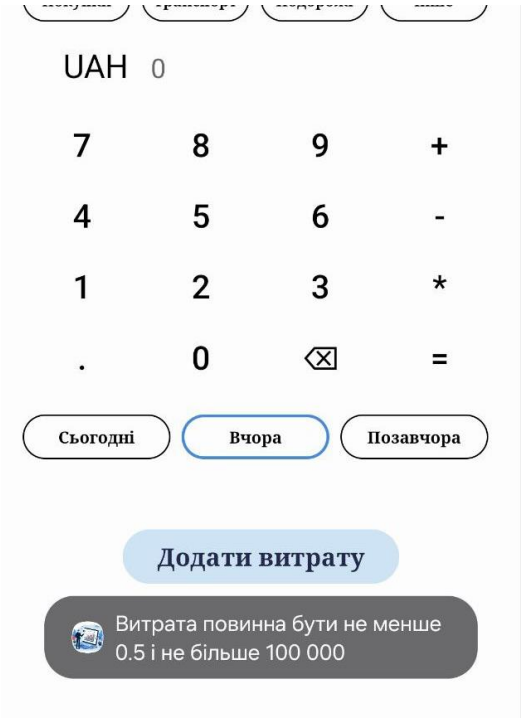


Рисунок Б.5. Ліміт витрати

Для перегляду статистики натискаємо одну з кнопок, наприклад Тиждень, бачимо оновлену діаграму, і загальну суму витрат за цей період (Рис. Б.7)

Рисунок Б.6. Екран статистики

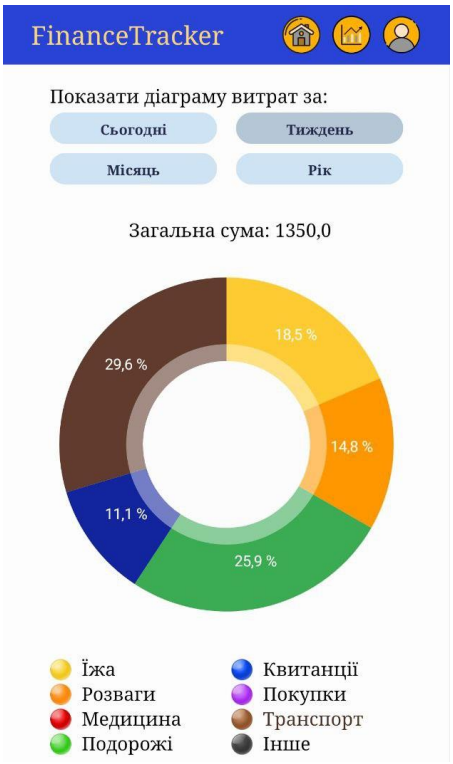


Рисунок Б.7. Діаграма витрат

У вкладці «Профіль» користувач може змінити своє ім'я (Рис. Б.8). Вводимо в текстове поле нове ім’я та натискаємо кнопку Зберегти, отримуємо повідомлення про успішну зміну (Рис. Б.9).

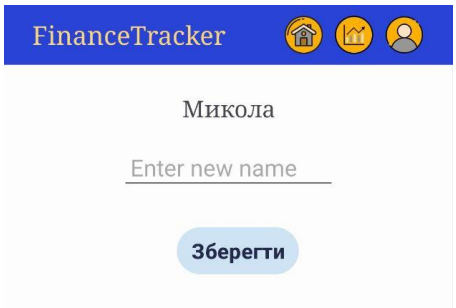


Рисунок Б.8. Екран профілю

Рисунок Б.9. Зміна імені

АНОТАЦІЯ

Расік М. Р. Розробка мобільного застосунку для комплексного управління особистими фінансами. Рукопис.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2024 р.

У роботі представлено розробку мобільного застосунку для обліку та аналізу особистих фінансових витрат користувачів. Проведено аналіз сучасних інструментів розробки мобільних застосунків, обґрунтовано вибір середовища розробки Android Studio, мови програмування Java та бази даних SQLite. Досліджено три аналогічних програмних продукти, визначено їхні переваги та недоліки, що стало основою для формулювання вимог до розроблюваного застосунку.

Розроблений мобільний застосунок надає користувачам можливість ефективно відслідковувати фінансові операції, додавати витрати за категоріями та аналізувати фінансові дані за допомогою діаграм. Детально описано архітектуру, процес розробки та логіку роботи програми, наведено ключові фрагменти коду.

У роботі також представлено рекомендації щодо подальшого вдосконалення програмного продукту, зокрема розширення функціональних можливостей та покращення користувацького досвіду. Для реалізації проєкту використано мову програмування Java, середовище розробки Android Studio, бібліотеку SQLite для роботи з базами даних та бібліотеку MPAndroidChart для візуалізації даних.

Ключові слова: мобільний застосунок, середовище програмування, мова програмування, Java, Android Studio, SQLite.