МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

ВИПУСКНА РОБОТА

на тему:

«Розробка мобільного додатку для планування бюджету»

Завідувач
випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Петров С.О.**

Студента групи ІН-73-9 **Коровай С.К**

СУМИ 2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

Кафедра комп'ютерних наук

Затверджую_____

Зав. кафедрою Довбиш А.С.

"_____" _____2021 г.

ЗАВДАННЯ
до випускної роботи

Студент четвертого курсу, групи ІН-73-9 спеціальності "Комп'ютерні науки" денної форми навчання Сергія Коровая Костянтиновича.

Тема: "**Розробка мобільного додатку для планування бюджету**"

Затверджена наказом по СумДУ

№_____ от _____2021 г.

Зміст пояснювальної записки: 1) огляд проблемної області;
2) аналіз існуючих продуктів аналогів; 3) постановка задачі; 5) вибір методів рішення задачі; 6) аналіз програмних засобів; 7) розробка моделі інформаційної системи; 8) програмна реалізація.

Дата видачі завдання "_____"_____2021 г.

Керівник випускної роботи _____ Петров С.О.

Завдання приняв до виконання _____ Коровай С.К.

РЕФЕРАТ

Записка: 45 стор., 23 рис., 1 табл., 4 додатки, 10 джерел.

Об’єкт дослідження — Розробка мобільного додатку планування бюджету.

Мета роботи — Розробка мобільного додатку, що дозволить записувати всі витрати та доходи користувача, тим самим фіксуючи їх, для подальшого аналізу та планування.

Результати — Розроблено додаток, з базовим функціоналом, що дозволяє виконувати облік персональних витрат користувача.

МОБІЛЬНИЙ ДОДАТОК, ANDROID, ФІНАНСОВІ ОПЕРАЦІЇ, ПЛАНУВАННЯ БЮДЖЕТУ, МОБІЛЬНА ПЛАТФОРМА.

ЗМІСТ

ВСТУП

Сучасні технології мають тенденцію до досить стрімкого прогресу, а з особливою швидкістю розвивається саме ІТ. У наш час вона впроваджується майже у всі сфери діяльності людини. В першу чергу будь які програми чи додатки направлені на спрощення життя людини.

Дуже важливим є використання інформаційних технологій у побуті. Майже кожна людина сьогодні використовує смартфон. Створення мобільних додатків, що допомагають у побуті, є досить важливим.

Одним із прикладів корисних побутових додатків є система обліку персональних доходів та витрат. Деякі люди ведуть облік власних фінансів у паперових носіях, що є неефективним у добу інформаційних технологій. Також, в наш час ми носимо потужні обчислювальні машини у нас в кишенях, кожного дня виходять програми, які покращують повсякденно якість життя. Сьогодні люди носять з собою в кишенях обчислювальні машини, фітнес трекери, часи, телефонні мережі та навіть пошти.

Гроші також є невід'ємною частиною людського життя, і кожного дня використовуються людьми, так як не у всіх людей є час та можливість рахувати скільки він витратив грошей, є спеціальні програми, які допомогають рахувати витрачені фінанси, вони називаються – фінансові додатки, або програми обліку витрат.

Саме в таких цілях і створювалась програма МММ "Magic Money Manager", за допомогою цієї програми користувач може, швидко та зручно рахувати свої щоденні витрати, та ефективно планувати свій бюджет.

Для досягнення мети проекту необхідно виконати наступні задачі:

- провести аналіз предметної області, визначити актуальність;
- ознайомитись із продуктами-аналогами;
- обрати технології для розробки програми;
- розробити модель та структуру програми;
- реалізувати дану структуру у вигляді програми;

- розробити функціонал даної програми;
- протестувати програму.

1 ІНФОРМАЦІЙНИЙ РОЗДІЛ

1.1 Огляд проблемної області

В житті сучасної людини її оточує багато гаджетів та девайсів, всі вони різні, та використовуються в різних цілях. Двадцять років тому, люди навіть уявити не могли, що девайси – потужністю з комп’ютер будуть у них в кишенях. Сьогодні ж люди можуть носити з собою бібліотеки книг, музикальні альбоми тисяч груп або ж тисячі фотографій та відео і все це в одному девайсі. Але з мобільний пристрій можна використовувати не тільки в цілях зберігання якоїсь інформації, на смартфоні може вміститись велика кількість програм.

Серед цих програм можна виділити розважальні, побутові та корпоративні, тобто ті, які використовуються для роботи. До розважальних можна віднести ігри та музичні сервіси. Побутові програми використовуються в людському щоденному житті, та спрощують повсякденне життя людини. Корпоративні програми в основному використовуються компаніями, для обміну інформацією або покращують процес роботи своїх працівників. Додаток який розробляється в даному дипломному проекті відноситься до побутових програм, тобто покращує повсякденне життя свого користувача.

Більшість людей в сучасному світі не слідкують за тим скільки грошей вони витрачають на ті чи інші речі, деякі все витрачають в день видачі зарплатні, та не відкладують гроші, даний мобільний додаток допоможе користувачу, вигідно планувати свої витрати, а найголовніше знати куди він їх витратив. [5]

1.2 Аналіз існуючих продуктів-аналогів

Звісно існують різні додатки для планування бюджету. Вони різноманітні інтерфейсом, і в кожного з них є якісь унікальні можливості. Одним з представників є додаток “Finkee”. [3]

Додаток оформлений, як класичний додаток для смартфону (Рис. 1.1).

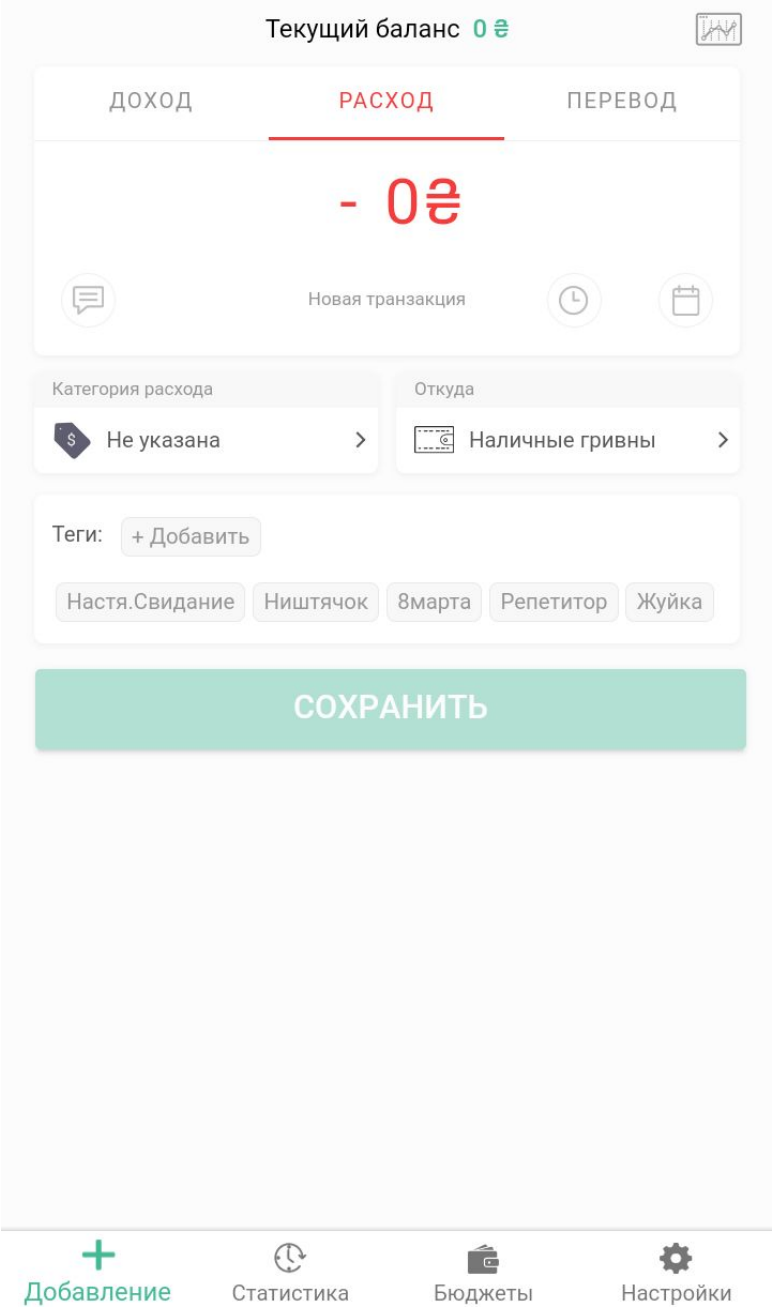

Рисунок 1.1 – Інтерфейс “Finkee”

Серед переваг додатку Finkee слід виділити наступні пункти:

- зручний інтерфейс;
- надійність зберігання даних;
- підтримка ПП розробником;
- можливість зберігати внесені дані в хмарному сховищі;
- підключення системи до банків.

Серед недоліків слід зазначити:

- Неможливість використання додатку без підключення до мережі Інтернет.

Наступним прикладом є додаток “Monefy” [7] (Рис. 1.2), що має менший функціонал, але є більш легким в використанні.

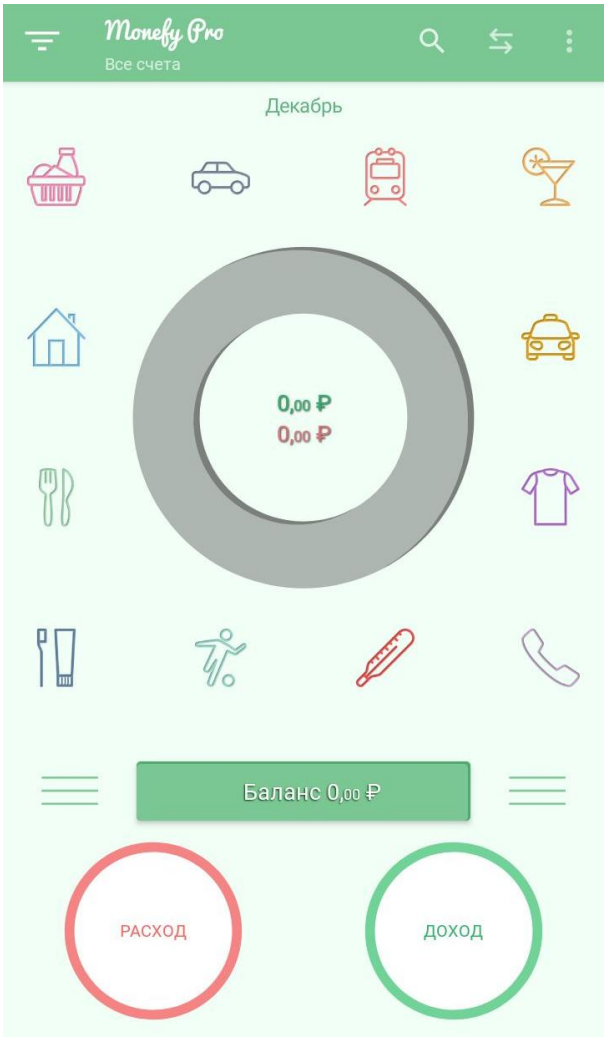

Рисунок 1.2 – Інтерфейс додатку “Monefy”

Переваги додатку “Monefy”:

- Додаток займає невелику кількість пам’яті;
- Можливість користуватися додатком без підключення до мережі інтернет.

Недоліками ж є:

- Не зручний інтерфейс;
- Відсутність можливості зберігання у хмарному сховищі.

Також подібним продуктом є додаток «Кошелек – учет расходов и доходов, бюджет» [2]. Застосунок виконано у простому стилі, додаток має інтуїтивно зрозумілий інтерфейс (Рис. 1.3).

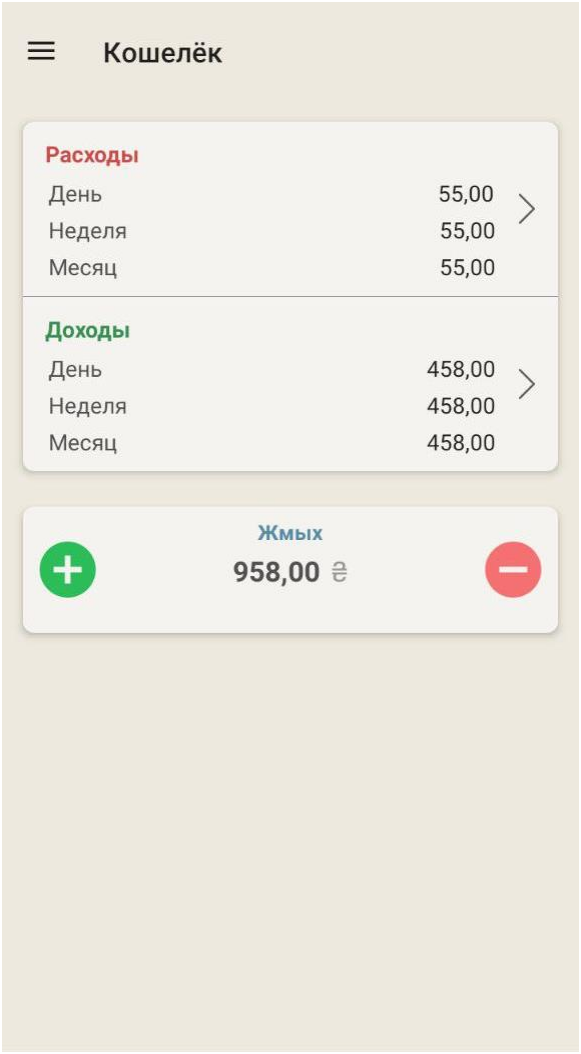

Рисунок 1.3 – Інтерфейс програми

Основною особливістю даного застосунку є мінімалістичність та ергономічність інтерфейсу, всі рахунки відображаються в відповідних для них полях, та з кожним рахунком можна взаємодіяти миттєво (Рис. 1.4).

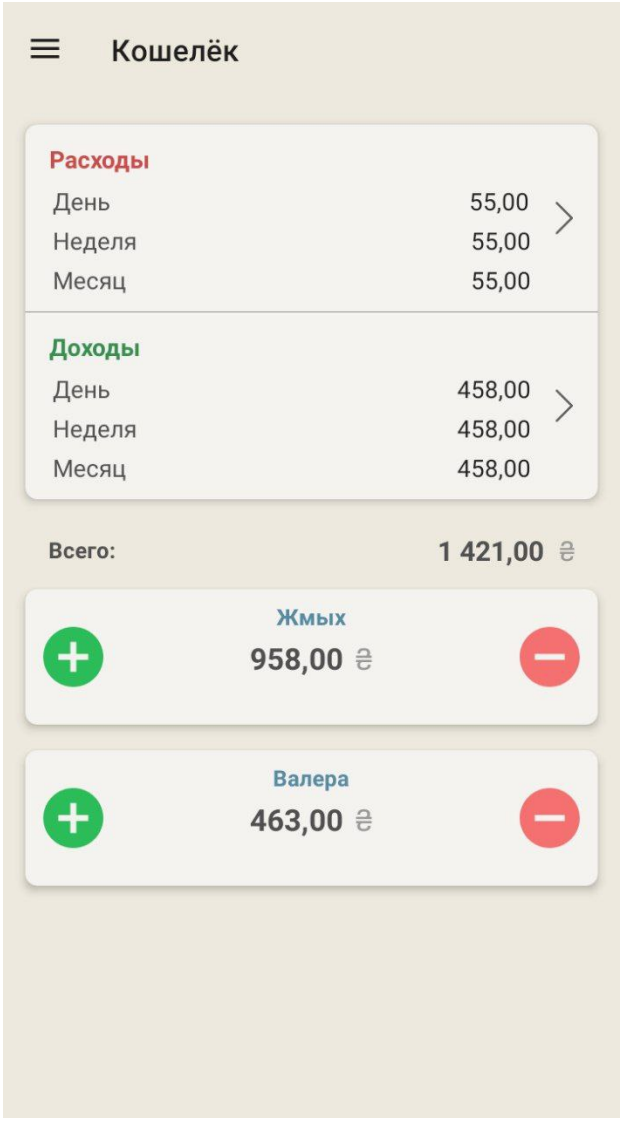

Рисунок 1.4 – Керування рахунками користувача

Головним недоліком даного додатку є незначна кількість категорій, створених розробником, але їх можна дадавати самостійно тому це не є значним недоліком (Рис. 1.5).

Рисунок 1.5 – Категорії створені за замовчуванням

Основними перевагами даного застосунку є:

- Мінімалізм;
- Низька потреба у внутрішній пам’яті;
- Ергономічність.

Недоліки:

- Відсутність зберігання у хмарному сховищі;
- Невелике початкове наповнення.

Тому виникає потреба у розробці власної додатку для планування бюджету, у якому визначені переваги будуть враховані, а недоліки подолані.

Таблиця 1.1 – Порівняльна характеристика додатків-аналогів

Характеристика	**Finkee**	**Monefy**	**Кошелек – учет расходов**
Зручний інтерфейс	+	-	+
Хмарне сховище	+	-	-
Сучасний дизайн	+	-	+
Ергономічність	+	-	+
Користування без інтернету	-	+	+
Можливість підключення до банків	+	-	-

Після порівняння дотатків аналогів, було прийняте рішення, щодо розробки мобільного додатку з наступними характеристиками:

- зручний інтерфейс;
- сучасний дизайн;
- ергономічність;
- можливість використання програми без інтернету.

1.3 Постановка задачі

Перед тим як розробляти мобільний додаток, потрібно визначитися з основними функціями майбутньої розробки, та як його використовувати.

Всього визначають чотири основні типи мобільних додатків:

- корпоративні;
- контентні;

- сервісні;
- ігрові.

Корпоративні додатки створюються великими компаніями, щоб спростити собі роботу або швидко та безпечно передавати інформацію між працівниками. Основною цільовою аудиторією таких програм є, насамперед, працівники компанії, клієнти або ж партнери. Контентні додатки використовуються для надавання різного сорту інформації, в основному такі додатки розробляються для ЗМІ (“засобів масової інформації”), радіо, телеканалів і порталів. Сервісні відповідають за певні сервісні послуги, тобто виконувати завдання, які користувач виставляє в режимі реального часу. Такі програми вважаються складними, адже вони повинні працювати в реальному часі як годинник, починаючи від калькулятора або будильника і закінчуючи додатками обробки великих обсязів тексту. Ігрові додатки відповідають за розважальну частину життя людини, але це не тільки основна їх ціль. В більшості додатків такого типу вдало розміщується реклама, або платний контент, який дозволяє розробникам заробляти гроші [1].

Із того часу як IT-сфера тісно пов’язалася з людським побутом, розробники почали створювати програмні продукти (ПП) різного призначення. Але потреби кожного користувача – індивідуальні і мають місце власні критерії ідеального ПП. Саме цим обумовлюється велика кількість аналогів різних програм. Найяскравішим прикладом є операційні системи. На сьогоднішній день є три найвідоміші операційні системи: Linux,

Windows та Mac OS. Кожна з них має власні особливості та переваги.

Із огляду на них, користувач може обрати найоптимальніший для себе варіант. Основною ціллю даного мобільного додатку є покращення умов життя та зберігання бюджету користувача. Користувач зможе в любий момент занести в додаток свій рахунок та записувати туди свої витрати. Додаток являється актуальним саме в даний період часу, адже більшість людей після покупок в різних торгових центрах не зберігають чеки, та не знають скільки вони витратили даний додаток допоможе їм

контролювати свої витрати, знати скільки вони витратили та скільки користувач витрачає за день.

Тому, можна зробити висновок, що розробка додатку контролю фінансами, є актуальною. Його використання надасть абсолютно кожному користувачу змогу курувати своїми фінансами.

2 ВИБІР МЕТОДІВ РІШЕННЯ ЗАДАЧІ

Метою даного дипломного проекту є розробка мобільного додатку планування бюджету, в якого буде гарний інтерфейс та зручність і легкість в використанні.

Для досягнення поставленої мети, необхідно наступні виконати задачі:

- аналіз предметної області;
- аналіз додатків аналогів;
- виділити їх плюси та мінуси додатків аналогів;
- обрати засоби реалізації;
- розробити програмний продукт;
- виконати тестування програмного продукту;
- оформити документацію.

Розробка програмного продукту – це складний та довготривалий процес, який потребує багато сил та уваги. Саме тому вибір середовища розробки та супутнього програмного забезпечення (ПЗ) є важливим.

2.1 Аналіз програмних засобів

Для кожної мови програмування існує власне IDE(Integrated development environment), яке ідеально підходить для створення програм на даній мові, враховуючи її особливості. Як приклад можна розглянути наступні середовища розробки:

- AndroidStudio;
- Borland C++ Builder;
- Clion;
- JetBrains Rider.

Звісно, існують додатки, в яких можна розробляти програмний продукт незалежно від мови програмування. Дуже яскравим прикладом такої IDE є Microsoft VisualStudio. Дане IDE має зручний інтерфейс та широкий функціонал, та є одним з найпопулярніших середовищ розробки для мов C++ та C#.

Мова програмування C# – це об’єктно орієнтована мова, яка на сьогоднішній день активно використовується для створення проектів різних типів.

Xamarin - це фреймворк для кроссплатформеної розробки мобільних додатків (iOS, Android, Windows Phone) з використанням мови C. Код пишеться з використанням всіх можливостей мови C#. При цьому ви маєте повний доступ до всіх можливостей SDK платформи і базового механізму створення UI, отримуючи на виході додаток, який нічим не відрізняється від нативних і не поступається їм у продуктивності. [10]

Фреймворк складається з декількох основних частин:

- Xamarin.IOS - бібліотека класів для C #, що надає розробнику доступ до iOS SDK;
- Xamarin.Android - бібліотека класів для C #, що надає розробнику доступ до Android SDK;
- Компілятори для iOS і Android;

Для написання скрипту на Xamarin краще всього використовувати інтегроване середовище розробки JetBrains Rider, адже дане IDE має зручний інтерфейс (Рис. 2.1), та має інтеграцію з Xamarin (для спрощення процесу написання коду).

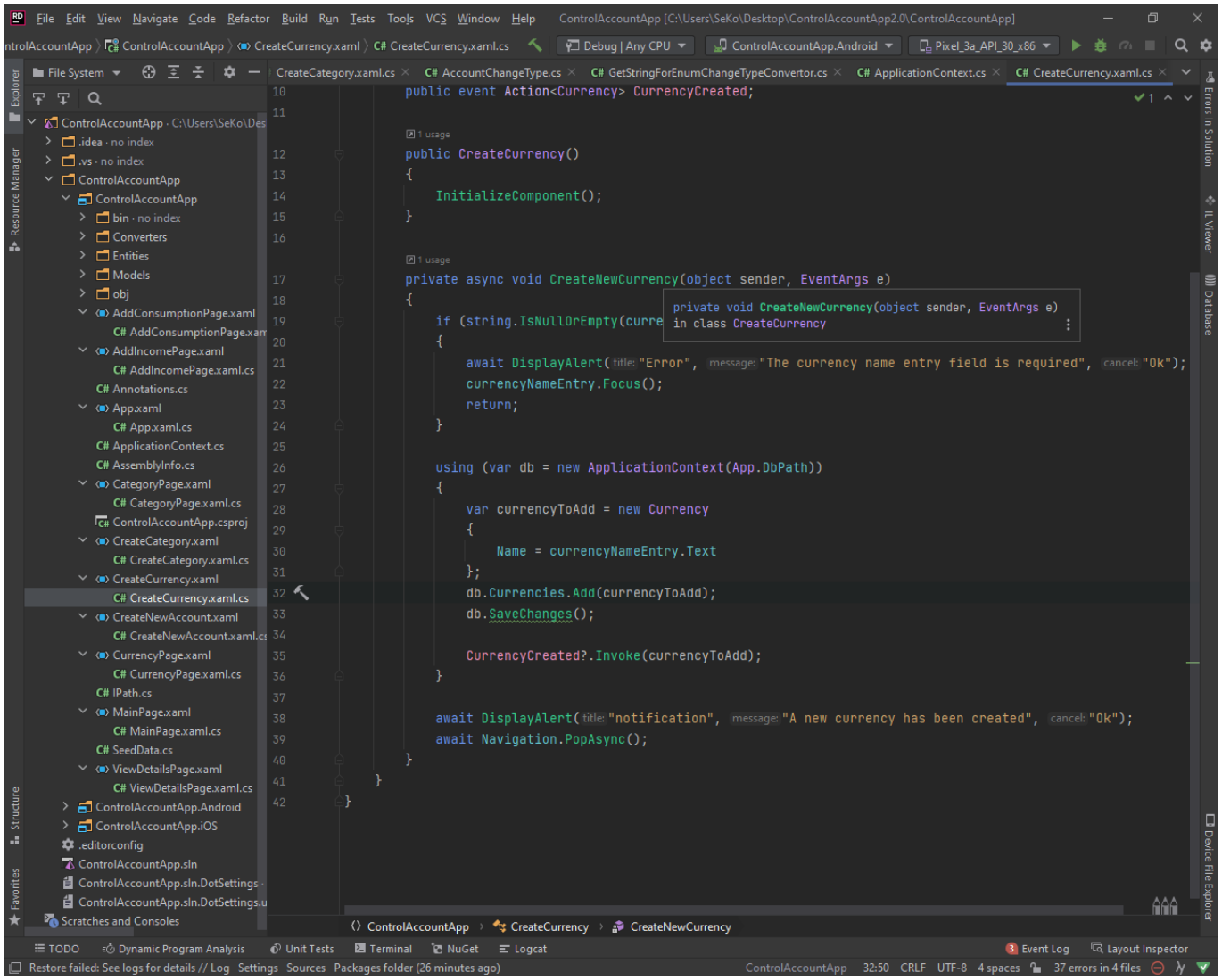

Рисунок 2.1 – Інтерфейс програми JetBrains Rider

Даний продукт орієнтований на .NET програмування [6]. Він є максимально ефективним у написанні коду на мові Java. Дане середовище має зручний та налаштовуваний інтерфейс, що підсвічує ключові слова та вказує на синтаксичні помилки.

Доволі важливим у процесі написання коду є вбудовані в IDE інструменти та відладки коду. Зважаючи на особливість розробки, дуже важливим є попереднє тестування функцій додатку, за допомогою вище вказаного IDE можна робити емуляцію смартфону на базі різних версій операційної системи Android (Рис. 2.2).

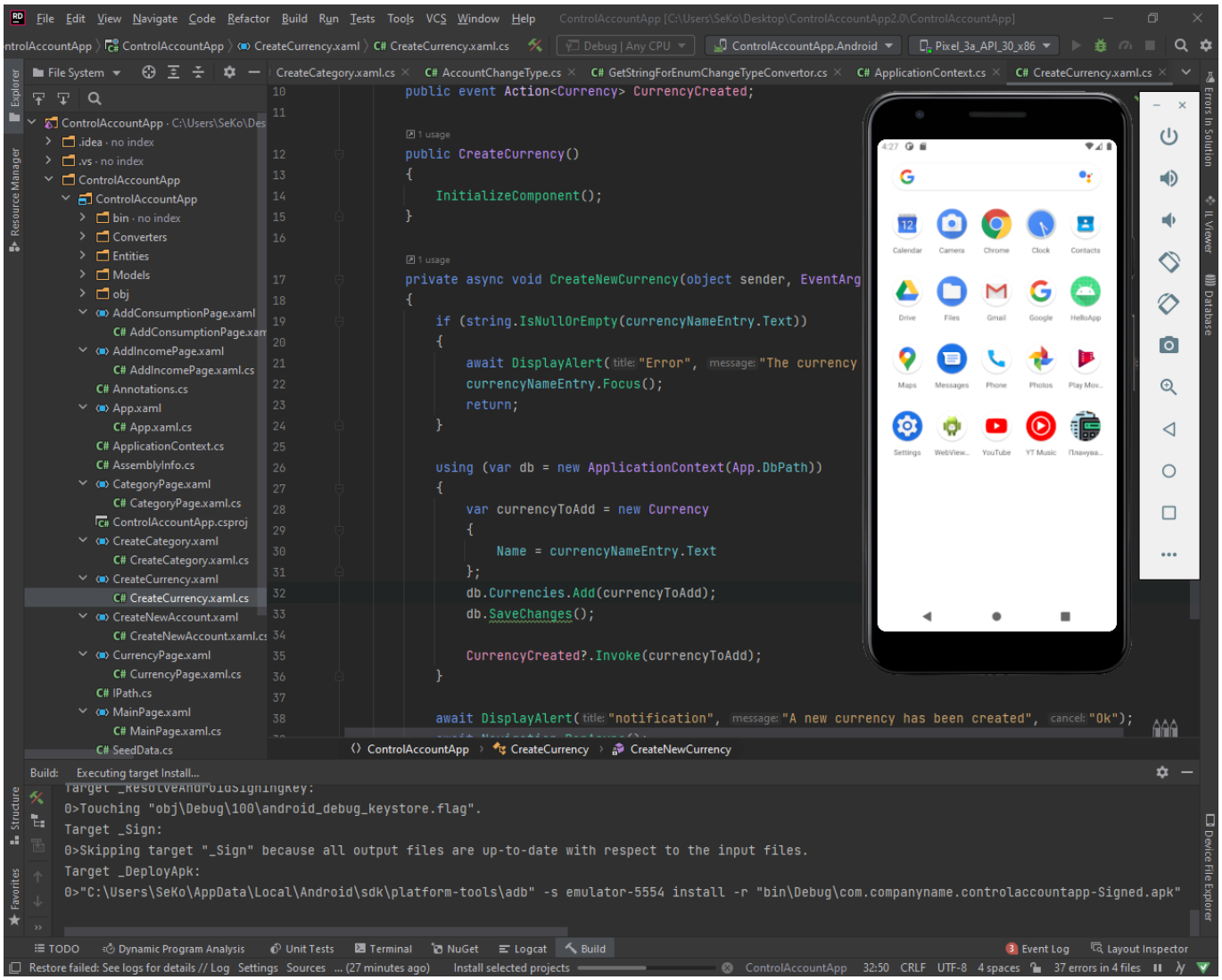

Рисунок 2.2 – Емуляція смартфону в JetBrains Rider

Процес розробки буде супроводжуватися використанням системи контролю версій GitHub. Завдяки цьому ПЗ фіксувати етапи розробки ПП.

У ході аналізу було виявлено що, представлені мови програмування та середовища розробки є найоптимальнішим варіантом для розробки даного ПП, оскільки серед усіх можливих варіантів вони є найзручнішими та найпростішими у вивченні.

В якості системи управління базою даних буде використано SQLite, що є стандартною СУБД для Android.

SQLite - компактна вбудована реляційна база даних. Вихідний код бібліотеки переданий в суспільне надбання. Є чисто реляційної базою даних.

Головною особливістю даної СУБД є те, що движок SQLite не є окремо працюючим процесом, з яким взаємодіє програма, а надає бібліотеку, з якої програма компонується і движок стає складовою частиною програми. Таким чином, в якості протоколу обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виповнюється програма. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції весь файл, який зберігає базу даних, блокується. [8]

Саме завдяки цим особливостям дана СУБД, є найбільш оптимальним варіантом для використання на мобільних пристроях, в тому числі і на Android.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Розробка моделі інформаційної системи

Метою бакалаврської роботи є створення мобільного додатку планування бюджету. Фундаментом даної системи є реляційна база даних, яка зберігатиме у собі всі необхідні дані. Саме тому розробка даного проекту почалася з визначення структури бази даних.

Проаналізувавши предметну область було визначено що додаток має зберігати дані про персональні фінансові операції та рахунки користувача.

Як результат була створена реляційна база даних, що складається з 4 пов’язаних між собою таблиці. Дані таблиці мають наступні назви:

- accounts;
- categories;
- currencies;
- history.

Зв’язки між таблицями реалізовано за допомогою зв’язаних між собою локальних та зовнішніх ключів (Рис. 3.1).

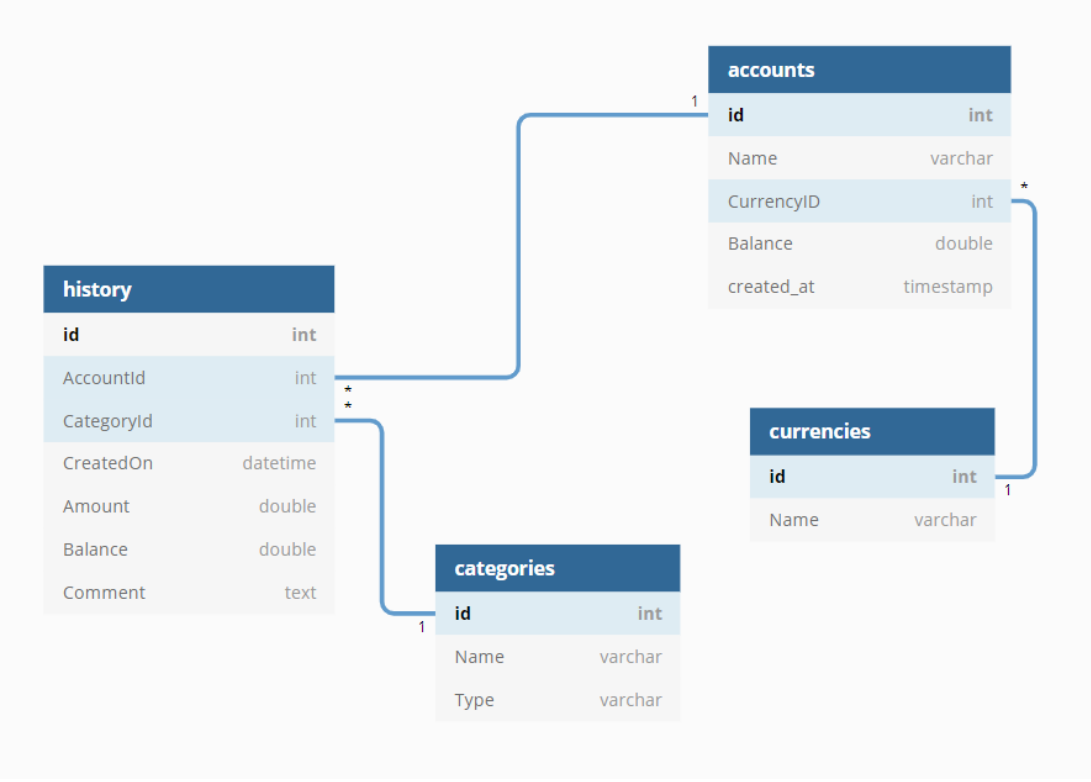

Рисунок 3.1 – ER-діаграма інформаційної системи

На даній діаграмі можна виділити декілька сутностей.

Сутність «accounts» містить записи про рахунки користувача (Рис. 3.2):

- id – унікальний ідентифікатор рахунку;
- name – назва рахунку;
- CurrencyID – ключове поле, унікальний ідентифікатор валюти;
- Balance – актуальний залишок на рахунку.

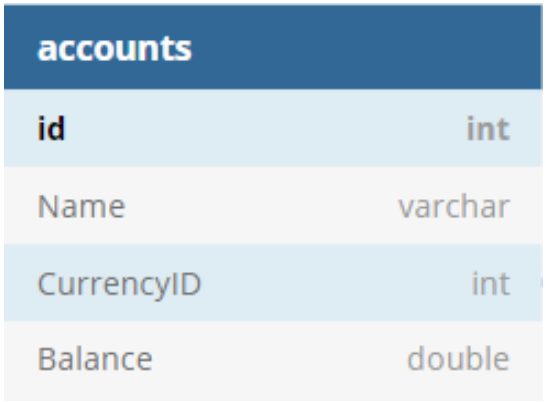

Рисунок 3.2 – Структура таблиці “accounts”

Сутність «currencies» зберігає суму яка є в користувача (Рис. 3.3):

- id – унікальний ідентифікатор рахунку;
- name – назва валюти.

Рисунок 3.3 – Структура таблиці “currencies”

Сутність «history» зберігає історію витрат (Рис. 3.4):

- id – унікальний ідентифікатор фінансової операції;
- AccountID – ключове поле, унікальний ідентифікатор рахунку;
- CurrencyID – ключове поле, унікальний ідентифікатор валюти;
- CreatedOn – дата та час в який була зроблена фінансова операція;

- Amount – сума операції;
- Balance – актуальний залишок на рахунку;
- Comment – коментар до операції.

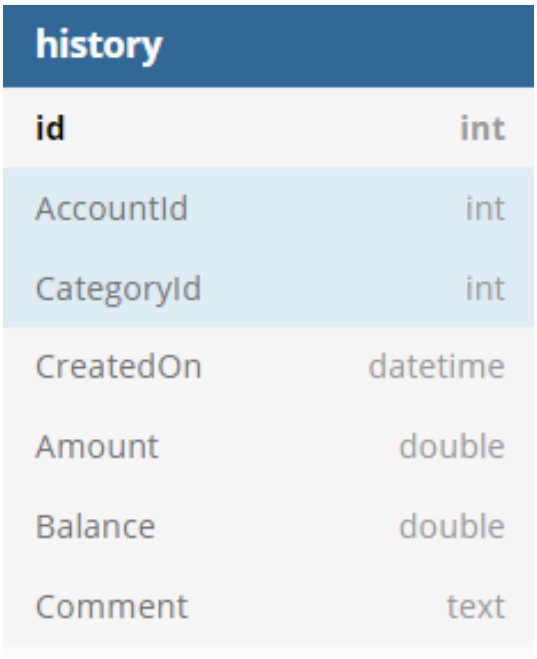

Рисунок 3.4 – Структура таблиці “history”

Сутність «categories» містить категорії доходів (Рис. 3.5):

- id – унікальний ідентифікатор категорії;
- name – назва валюти;
- type – тип трансакції.

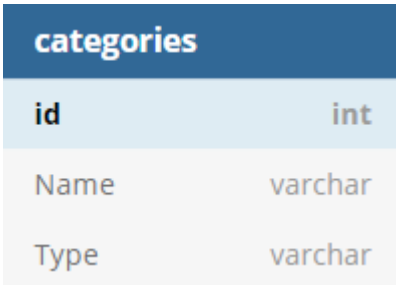

Рисунок 3.5 – Структура таблиці “categories”

База даних реалізується програмно за допомогою об’єктів сутностей, що пов’язані між собою.

Даний програмний продукт буде реалізовано на основі паттерну проектування MVVC (Model – View –ViewModel). Данний паттерн є адаптованою для мобільних

додатків версією паттерну MVC (Model – View – Controller), але зі змінами в структурі(Рис 3.6).

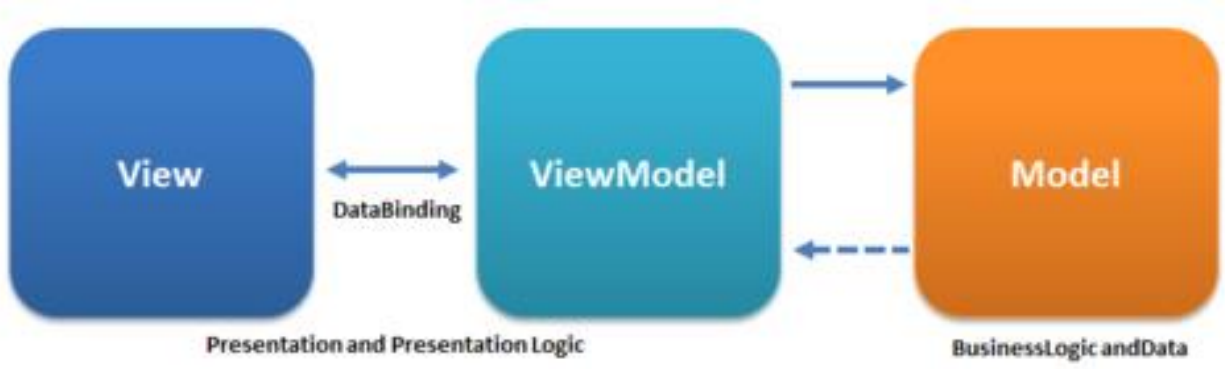

Рисунок 3.6 – Структура патерну MVVM

Згідно структури патерну можна виділити 3 складові:

- Model описує дані, що використовуються в додатку.
- View визначає візуальний інтерфейс, через який кристувач взаємодіє с додатком.
- ViewModel або модель уявлення пов'язує модель і уявлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення між собою не пов'язані.

Використання паттерну MVVM рекомендоване розробниками операційної системи Android. А також є базовим у розробці додатків за допомогою фреймворку Xamarin. [4]

Ключовим компонентом для створення візуального інтерфейсу в додатку Android є activity (активність). Нерідко activity асоціюється з окремим екраном або вікном додатка, а перемикання між вікнами буде відбуватися як переміщення від однієї activity до іншої. Додаток може мати одну або кілька activity.

Струкрура додатку – це логічна побудова всіх activity застосунку, схема за якою будується їх порядок, що забезпечує зручну та інтуїтивно зрозумілу навігацію в додатку (Рис. 3.7).

Рисунок 3.7 – Структура додатку «Magic Money Manager»

Кожна сторінка додатку створеня з урахуванням естетики дизайну та ергономічності використання. Головна сторінка додатку містить список рахунків користувача, та дає змогу з ними взаємодіяти (Рис. 3.8). Додаток А містить xaml код головної сторінки.

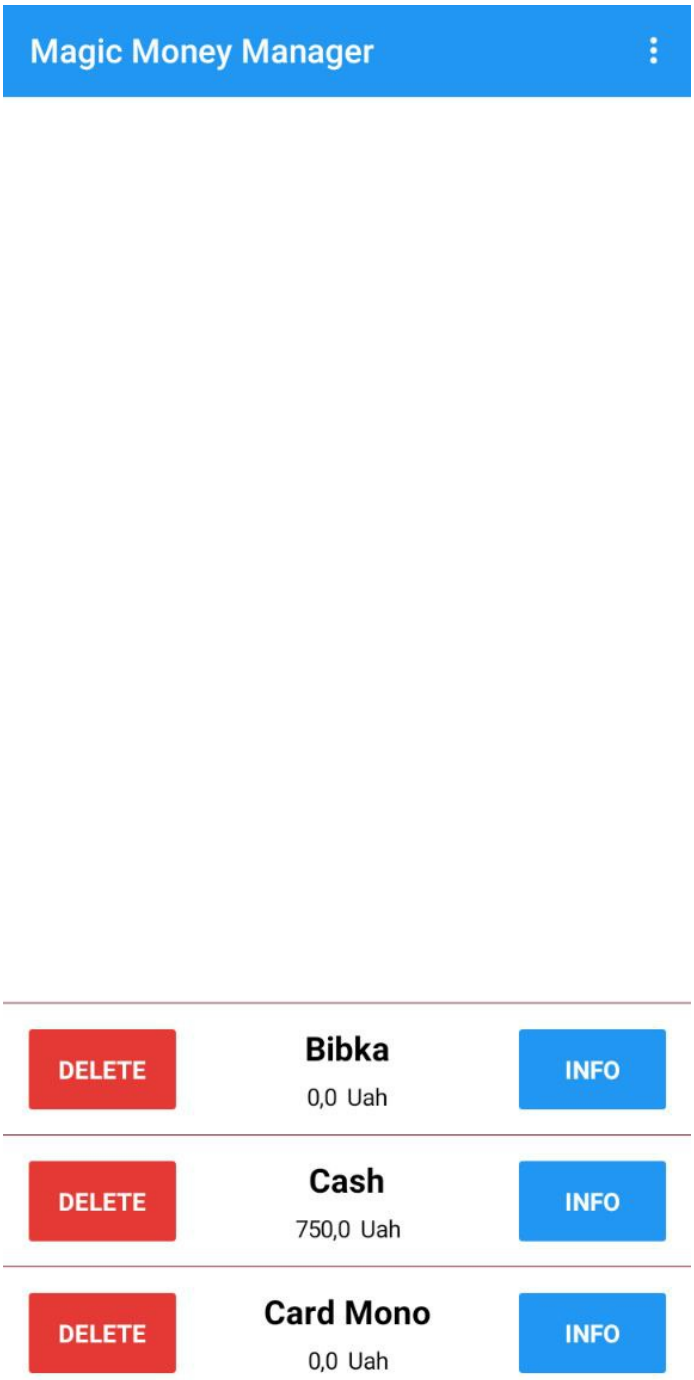

Рисунок 3.8 – Головна сторінка додатку «Magic Money Manager»

При натисканні на кнопку «Info», відобразиться історія операцій рахунку, та можливість додати новий запис про витрату абу дохід (Рис. 3.9). Додаток Б містить C# код функцій кнопок головної сторінки.

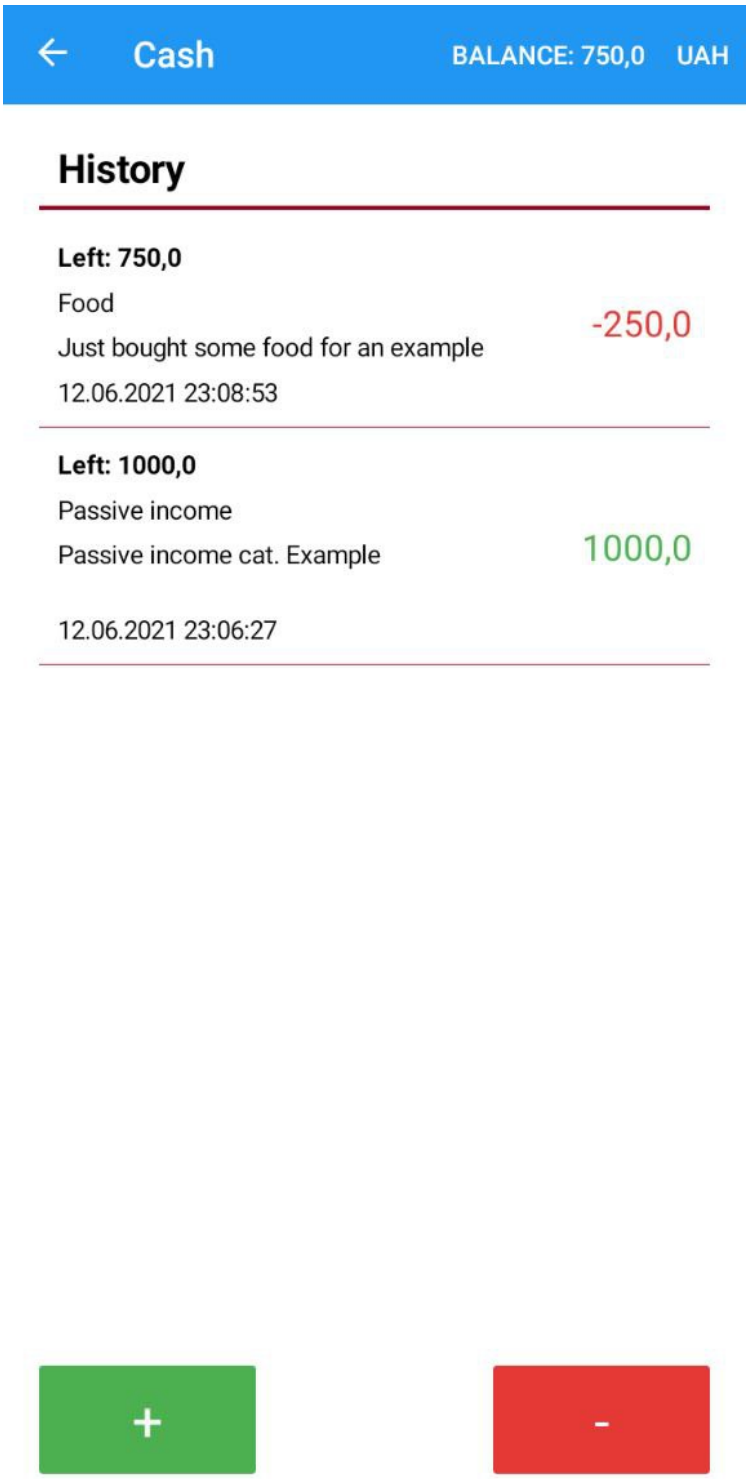

Рисунок 3.9 – Історія операцій рахунку

Для додавання витрати потрібно натиснути червону кнопку «-».Для додавання витрати потрібно натиснути зелену кнопку «+». Після натискання відобразиться форма додавання запису (Рис. 3.10).

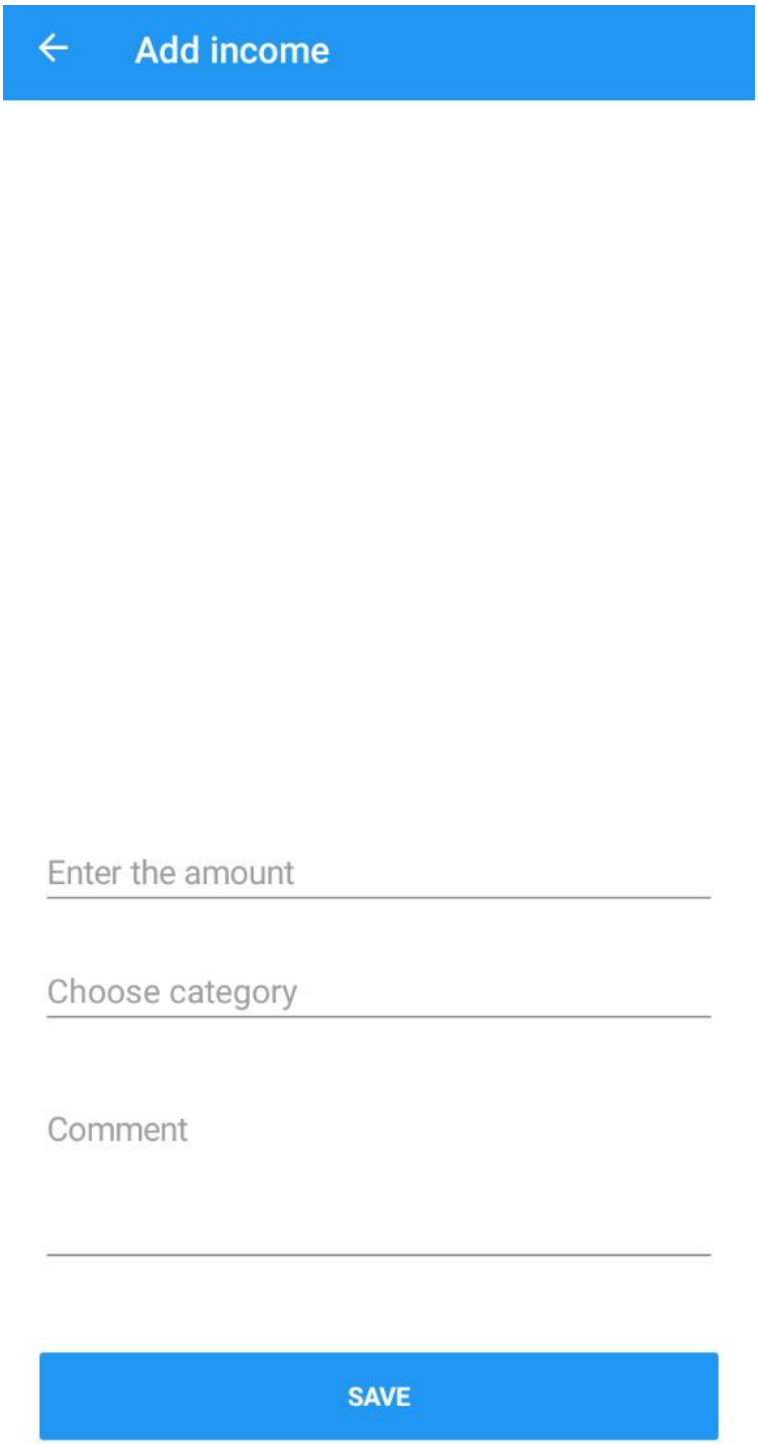

Рисунок 3.10 – Форма додавання запису

Також, верхня панель головної сторінки містить додаткове меню (Рис. 3.11), в якому є можливість перейти на сторінку створення нового рахунку (Рис. 3.12), переглянути наявні валюти (Рис. 3.13) та категорії (Рис. 3.15).

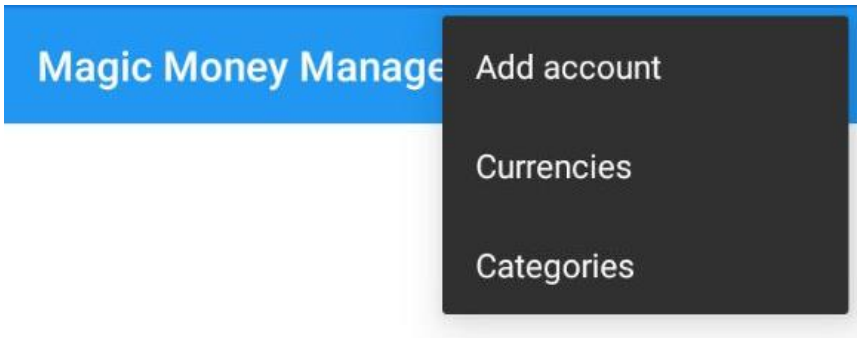

Рисунок 3.11 – Додаткове меню на головній сторінці

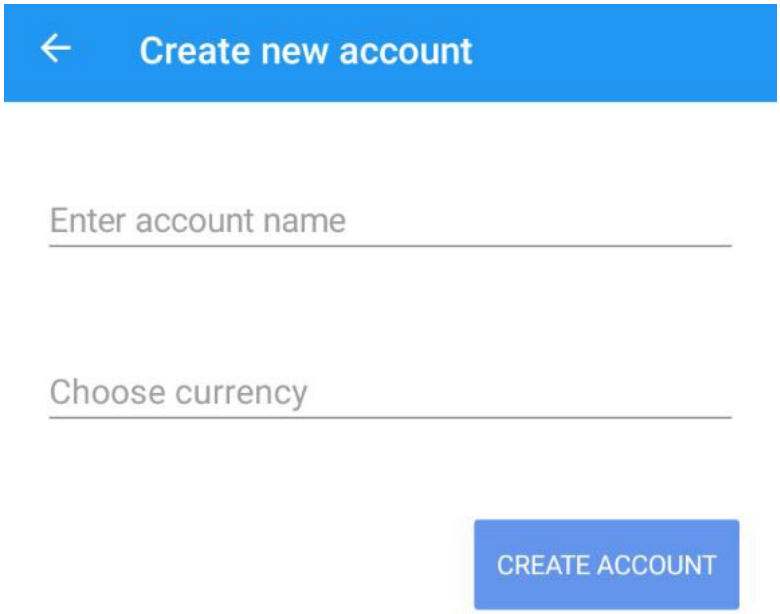

Рисунок 3.12 – Форма додавання нового рахунку

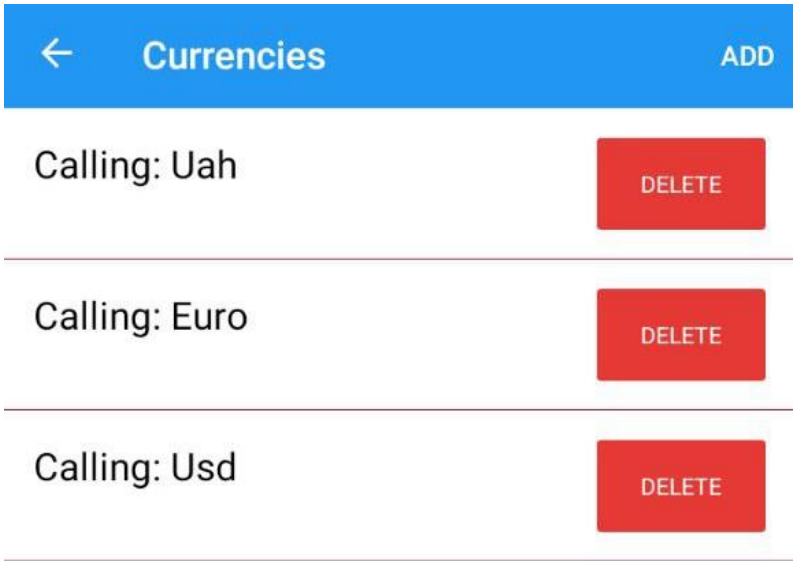

Рисунок 3.13 – Наявні валюти

Для відображення форми (Рис. 3.14) для додавання нової валюти потрібно натиснути кнопу «Add».

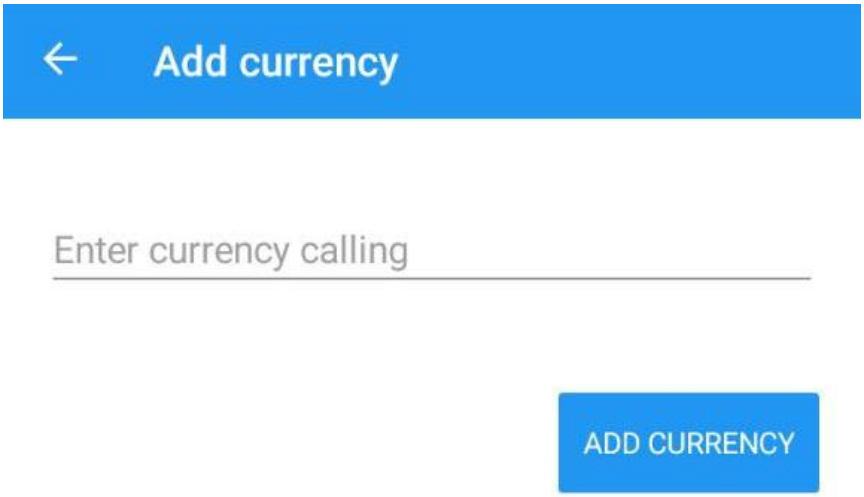

Рисунок 3.15 – Форма додавання валюти

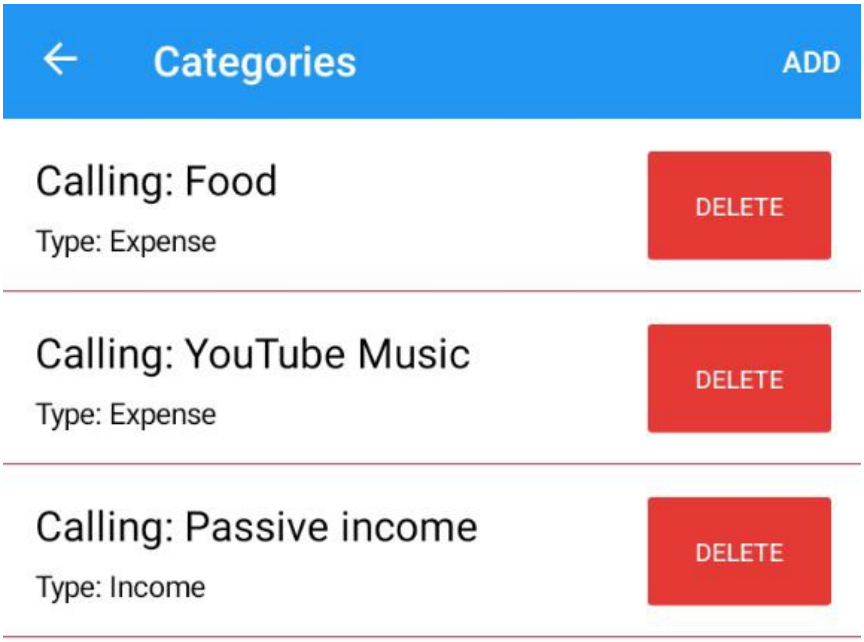

Рисунок 3.15 – Наявні категорії

Для додавання нової категорії потрібно натиснути «Add» та верхній панелі додатку, тоді відобразиться форма додавання категорії (Рис. 3.16).

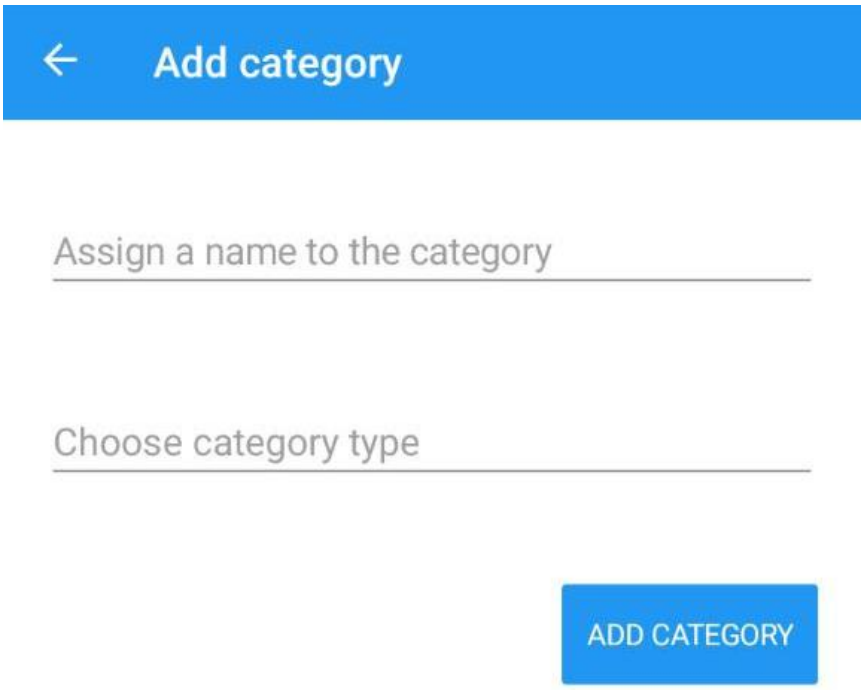

Рисунок 3.16 – Форма додавання категорії

3.2 Програмна реалізація

Структура проекту на створеного за допомогою фреймворку Xamarin передбачає розробку класів-сутностей (entities), що забезпечують інформаційний зв'язок між базою даних та функціями додатку. [9]

Об’єкти даних класів використовуються в функціях для додавання нових рядків до таблиць баз даних.

Всього можна виділити 4 сутності:

- Account.cs
- Category.cs
- Currency.cs
- History.cs

Додаток В містить лістинг коду всіх сутностей.

Кожна сутність складається з полів, що вітповідать колонкам таблиць в базі даних. А також із зв’язків між таблицями.

Ініціалізація классу:

```
[Table("Accounts")]
public class Account : IEntity, INotifyPropertyChanged
```

Оголошення полів класу сутності:

```
private decimal _balance;
[PrimaryKey, AutoIncrement, Column("_id")]
public int Id { get; set; }
public string Name { get; set; }

public int? CurrencyId { get; set; }
public Currency Currency { get; set; }
public decimal Balance
{
get => _balance;
set
{
if (_balance != value)
{
_balance = value;
OnPropertyChanged(nameof(Balance));
}}
```

В даному додатку є декілька основних функцій, що забезпечуюсь працездатність застосунку для виконання необхідних задач. А саме:

- CreateConsumptionRow;
- CreateIncomeRow;
- CreateAccount;
- CreateNewCategory;
- CreateNewCurrency.

Додаток Г містить лістинг коду всіх основних функцій.

Функція CreateConsumptionRow відповідає за додавання запису про витрату, з урахуванням появи модального вікна при помилці, та успішного додавання запису.

Блок коду, що перевіряє заповненість полів:

```
if (string.IsNullOrEmpty(consumptionAmountEntry.Text))
{
await DisplayAlert("Error", "Amount field is required", "Ok");
consumptionAmountEntry.Focus();
return;
}
if (categoryPicker.SelectedItem == null)
{
await DisplayAlert("Error", "Cetegory field is required", "Ok");
return;
}
if (string.IsNullOrEmpty(consumptionCommentEditor.Text))
{
await DisplayAlert("Error", "Comment field is required", "Ok");
consumptionCommentEditor.Focus();
return;
```

Код, що відповідає за додавання запису в таблицю локальної бази даних, та відображення модального вікна з повідомленням про успішне додавання запису:

```
var consumptionAmount = decimal.Parse(consumptionAmountEntry.Text);
var selectedCategory = (Category) categoryPicker.SelectedItem;
Account.Balance -= consumptionAmount;
db.Accounts.Update(Account);
var historyToAdd = new History{
AccountId = Account.Id,
Balance = Account.Balance,
CreatedOn = DateTime.Now,
Comment = consumptionCommentEditor.Text,
Amount = consumptionAmount,
CategoryId = selectedCategory.Id};
db.Histories.Add(historyToAdd);
db.SaveChanges();
historyToAdd.Category = selectedCategory;
ConsumptionCreated?.Invoke(historyToAdd);}
await DisplayAlert("Notification", "New expense was added", "Ok");
await Navigation.PopAsync();}
```

Функція передбачає додавання даних за допомогою об’єкта сутності History, та запису в його поля даних з форми.

Після додавання запису з'являється модальне вікно про успішне виконання функції.

У разі спроби залишити одне з полів порожнім відобразиться модальне вікно за повідомленням про помилку (Рис 3.17).

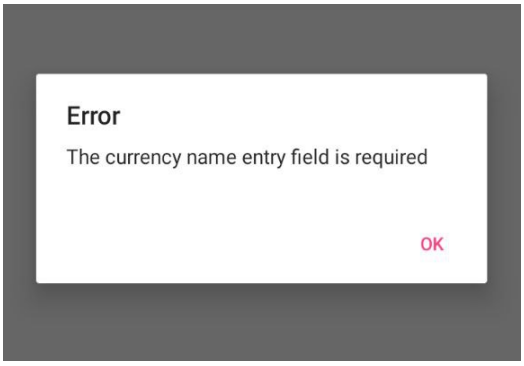

Рисунок 3.17 – Модальне вікно з повідомленням про помилку

ВИСНОВОК

Дана робота присвячена розробці мобільного додатку планування бюджету. Додатки даного типу є доволі корисними в побуті, та спрощують планувальні аспекти життя людини. В даному випадку користувач планує свої фінанси, та в перспективі витрачає їх менше.

В ході виконання було проаналізовано аналоги подібних додатків, визначені їх недоліки та переваги, всі ці пункти будуть враховані в ході розробки даного програмного продукту. В розробку майбутнього функціоналу був включений такий функціонал:

- зручний інтерфейс;
- сучасний дизайн;
- ергономічність;
- можливість використання програми без інтернету.

Також були обрані засоби реалізації та визначена структура майбутнього додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Актуальність розробки мобільних додатків на Android [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://htmler.ru/2020/12/29/aktualnost-razrabotki-mobilnyh-prilozhenij-pod-android/.

2 Додаток "Кошелёк" [Електронний ресурс] – Режим доступу до ресурсу: https://play.google.com/store/apps/details?id=com.vitvov.jc&hl=ru&gl=US.

3 Мобільний додаток Finkee [Електронний ресурс] – Режим доступу до ресурсу: https://finkee.org/.

4 Паттерн MVVM [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/wpf/22.1.php.

5 Фінансова грамотність як більше не втрачати свою зарплатню [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://trends.rbc.ru/trends/education/60abb47c9a79470e6482d2b7.

6 Швидка кросс-платформенна IDE для .NET [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/rider/.

7 Monefy –Personal finance application that makes money management easy [Електронний ресурс] – Режим доступу до ресурсу: https://monefy.me.

8 SQLite Home Page [Електронний ресурс] – Режим доступу до ресурсу: https://sqlite.org/index.html.

9 Xamarin та крос-платформерна розробка [Електронний ресурс]. – 701. – Режим доступу до ресурсу: https://metanit.com/sharp/xamarin/1.1.php.

10 Xamarin Forms documentation [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/xamarin/xamarin-forms/.

ДОДАТОК А

ЛІСТИНГ КОДУ РОЗМІТКИ ГОЛОВНОЇ СТОРІНКИ

XAML MainPage.xaml

```
<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="ControlAccountApp.MainPage"
Title="Magic Money Manager" BackgroundColor="{StaticResource PageBackgroundColor}">
<ContentPage.ToolbarItems>
<ToolbarItem
Text="Add account"
Order="Secondary"
Priority="1"
Clicked="OpenCreateAccountPage" />
<ToolbarItem
Text="Currencies"
Order="Secondary"
Priority="1"
Clicked="OpenCurrenciesPage" />
<ToolbarItem
Text="Categories"
Order="Secondary"
Priority="1"
Clicked="OpenCategoriesPage" />
</ContentPage.ToolbarItems>
<StackLayout>
<ListView
Rotation="180"
x:Name="AccountsList"
IsVisible="True" HasUnevenRows="True"
ItemsSource="{Binding Accounts}"
ItemSelected="AccountSelected"
SeparatorColor="{StaticResource ListSeparator}">
<ListView.ItemTemplate>
<DataTemplate>
<ViewCell>
<ViewCell.View>
<StackLayout Rotation="180" FlowDirection="RightToLeft">
<FlexLayout Direction="Row" JustifyContent="SpaceBetween">
<StackLayout VerticalOptions="Center" Margin="15">
<Button FontSize="15" BackgroundColor="#e53935" Text="Delete"
FontAttributes="Bold"
BorderWidth="1"
CommandParameter="{Binding .}" TextColor="White"
VerticalOptions="CenterAndExpand" Clicked="RemoveAccount" />
</StackLayout>
<StackLayout HorizontalOptions="Center" VerticalOptions="CenterAndExpand">
<StackLayout VerticalOptions="CenterAndExpand">
<Label FontAttributes="Bold" HorizontalOptions="Center"
TextColor="{StaticResource AppFont}"
Text="{Binding Path=Name,
StringFormat='{0}'}"
FontSize="20" />
<StackLayout HorizontalOptions="Center" Orientation="Horizontal">
<Label HorizontalOptions="Center" TextColor="{StaticResource AppFont}"
Text="{Binding Path=Currency.Name,
StringFormat='{0}'}" />
<Label HorizontalOptions="Center" TextColor="{StaticResource AppFont}"
```

```
Text="{Binding Path=Balance,
i
FontSize="15"
BackgroundColor="#2196f3"
Text="Info"
FontAttributes="Bold"
CommandParameter="{Binding .}"
TextColor="White"
VerticalOptions="CenterAndExpand" Clicked="ViewDetails" />
</StackLayout>
</FlexLayout>
</StackLayout>
</ViewCell.View>
</ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
<Label TextColor="{StaticResource ListSeparator}"
Text="No data" IsVisible="False" x:Name="NoData" FontSize="20"
VerticalOptions="CenterAndExpand"
HorizontalOptions="Center" />
</StackLayout>
</ContentPage>
```

ДОДАТОК Б

C# MainPage.xaml.cs

```
using System;
using System.Collections.ObjectModel;
using System.Linq;
using ControlAccountApp.Entities;
using Microsoft.EntityFrameworkCore;
using Xamarin.Forms;
namespace ControlAccountApp
{
    public partial class MainPage
    {
        public Account AccountToView { get; set; }
        public ObservableCollection<History> AccountHistory { get; set; }
         private async void CreateIncome(object sender, EventArgs e)
                {
                    var addIncomePage = new AddIncomePage(AccountToView);
                    addIncomePage.IncomeCreated += (history) =>
                    {
                        AccountHistory.Insert(0, history);
                        CalculateControlsVisibility();
                    };
                    await Navigation.PushAsync(addIncomePage);
                }
        public ObservableCollection<Account> Accounts { get; set; }

        public MainPage()
        {
            InitializeComponent();
        }
        protected override void OnAppearing()
        {
            base.OnAppearing();
            using (var db = new ApplicationContext(App.DbPath))
            {
                Accounts = new ObservableCollection<Account>(db.Accounts.Include(a =>
a.Currency).ToList());
                CalculateControlsVisibility();
            }
            BindingContext = this;
        }
        private async void OpenCreateAccountPage(object sender, EventArgs e)
        {
            var createAccountPage = new CreateNewAccount();
            createAccountPage.AccountCreated += (account) =>
            {
                Accounts.Add(account);
                CalculateControlsVisibility();
            };
            await Navigation.PushAsync(createAccountPage);
        }
        private async void OpenCategoriesPage(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new CategoryPage());
        }
        private async void OpenCurrenciesPage(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new CurrencyPage());
        }
        private void AccountSelected(object sender, SelectedItemChangedEventArgs e)
        {
            ((ListView) sender).SelectedItem = null;
        }
        private async void ViewDetails(object sender, EventArgs e)
```

```
        {
            var accountViewDetailsButton = (Button) sender;
            var accountToView = (Account) accountViewDetailsButton?.CommandParameter;
            var viewDetailsPage = new ViewDetailsPage(accountToView);
            await Navigation.PushAsync(viewDetailsPage);
        }
        private async void RemoveAccount(object sender, EventArgs e)
        {
            var result = await DisplayAlert("Notification", "Delete selected
account?", "Yes", "No");
            if (result)
            {
                var accountDeleteButton = (Button) sender;
                var accountToDelete = (Account)
accountDeleteButton?.CommandParameter;
                using (var db = new ApplicationContext(App.DbPath))
                {
                    var historyToDelete = db.Histories.Where(h => h.AccountId ==
accountToDelete.Id);
                    db.Histories.RemoveRange(historyToDelete);
                    db.Accounts.Remove(accountToDelete);
                    Accounts.Remove(accountToDelete);
                    db.SaveChanges();
                }
                CalculateControlsVisibility();
                await DisplayAlert("Notification", "Account deleted successfully",
"Ok");
            }
        }
        private void CalculateControlsVisibility()
        {
            NoData.IsVisible = !Accounts.Any();
            AccountsList.IsVisible = Accounts.Any();
        }
        private void AccountsList_OnItemTapped(object sender, ItemTappedEventArgs e)
        {
            throw new NotImplementedException();
        }
    }
}
```

ДОДАТОК В

ЛІСТИНГ СУТНОСТЕЙ РОЗРОБЛЕНОГО ANDROID-ДОДАТКУ

C# Account.cs

```
using System.ComponentModel;
using System.Runtime.CompilerServices;
using ControlAccountApp.Annotations;
using ControlAccountApp.Entities.interfaces;
using SQLite;
namespace ControlAccountApp.Entities
{
    [Table("Accounts")]
    public class Account : IEntity, INotifyPropertyChanged
    {
        private decimal _balance;

        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }

        public string Name { get; set; }

        public int? CurrencyId { get; set; }

        public Currency Currency { get; set; }

        public decimal Balance
        {
            get => _balance;
            set
            {
                if (_balance != value)
                {
                    _balance = value;
                    OnPropertyChanged(nameof(Balance));
                }
            }
        }
        public event PropertyChangedEventHandler PropertyChanged;
        [NotifyPropertyChangedInvocator]
        protected virtual void OnPropertyChanged([CallerMemberName] string
propertyName = null)
        {
            PropertyChanged?.Invoke(this, new
PropertyChangedEventArgs(propertyName));
        }
    }
}
```

C# Category.cs

```
using ControlAccountApp.Entities.interfaces;
using ControlAccountApp.Models;
using SQLite;
namespace ControlAccountApp.Entities
{
    [Table("Categories")]
    public class Category : IEntity
    {
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }
        public string Name { get; set; }
        public AccountChangeType Type { get; set; }
    }
}
```

C# Currency.cs

```
using ControlAccountApp.Entities.interfaces;
using SQLite;

namespace ControlAccountApp.Entities
{
    [Table("Currencies")]
    public class Currency : IEntity
    {
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }

        public string Name { get; set; }
    }
}
```

C# History.cs

```
using System;
using ControlAccountApp.Entities.interfaces;
using SQLite;
namespace ControlAccountApp.Entities
{
    [Table("History")]
    public class History : IEntity
    {
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }
        public int? AccountId { get; set; }
        public Account Account { get; set; }
        public int? CategoryId { get; set; }
        public Category Category { get; set; }
        public DateTime CreatedOn { get; set; }
        public decimal Amount { get; set; }
        public decimal Balance { get; set; }
        public string Comment { get; set; }
 }}
```

ДОДАТОК Г

ЛІСТИНГ ОСНОВНИХ ФУНКЦІЙ ДОДАТКУ

C# AddConsumptionPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using ControlAccountApp.Entities;
using ControlAccountApp.Models;
using Xamarin.Forms.Xaml;
```

```
namespace ControlAccountApp
{
    [XamlCompilation(XamlCompilationOptions.Compile)]
    public partial class AddConsumptionPage
    {
        public event Action<History> ConsumptionCreated;

        public Account Account { get; set; }

        public List<Category> Categories { get; set; }

        public AddConsumptionPage(Account account)
        {
            InitializeComponent();
            Account = account;
        }

        protected override void OnAppearing()
        {
            base.OnAppearing();
            using (var db = new ApplicationContext(App.DbPath))
            {
                Categories = db.Categories.Where(c => c.Type ==
AccountChangeType.Expense).ToList();
            }

            BindingContext = this;
        }

        private async void CreateConsumptionRow(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(consumptionAmountEntry.Text))
            {
                await DisplayAlert("Error", "Amount field is required", "Ok");
                consumptionAmountEntry.Focus();
                return;
            }

            if (categoryPicker.SelectedItem == null)
            {
                await DisplayAlert("Error", "Cetegory field is required", "Ok");
                return;
            }

            if (string.IsNullOrEmpty(consumptionCommentEditor.Text))
            {
                await DisplayAlert("Error", "Comment field is required", "Ok");
                consumptionCommentEditor.Focus();
                return;
            }

            using (var db = new ApplicationContext(App.DbPath))
            {
                var consumptionAmount = decimal.Parse(consumptionAmountEntry.Text);
                var selectedCategory = (Category) categoryPicker.SelectedItem;
                Account.Balance -= consumptionAmount;
                db.Accounts.Update(Account);
                var historyToAdd = new History
                {
                    AccountId = Account.Id,
                    Balance = Account.Balance,
                    CreatedOn = DateTime.Now,
```

```
                    Comment = consumptionCommentEditor.Text,
                    Amount = consumptionAmount,
                    CategoryId = selectedCategory.Id
                };
                db.Histories.Add(historyToAdd);
                db.SaveChanges();

                historyToAdd.Category = selectedCategory;
                ConsumptionCreated?.Invoke(historyToAdd);
            }

            await DisplayAlert("Notification", "New expense was added", "Ok");
            await Navigation.PopAsync();
        }
    }
}
```

C# AddIncomePage.xaml.cs

```
private async void CreateIncomeRow(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(incomeAmountEntry.Text))
            {
                await DisplayAlert("Error", "Amount field is required", "Ok");
                incomeAmountEntry.Focus();
                return;
            }

            if (categoryPicker.SelectedItem == null)
            {
                await DisplayAlert("Error", "Cetegory field is required", "Ok");
                return;
            }

            if (string.IsNullOrEmpty(incomeCommentEditor.Text))
            {
                await DisplayAlert("Error", "Comment field is required", "Ok");
                incomeCommentEditor.Focus();
                return;
            }

            using (var db = new ApplicationContext(App.DbPath))
            {
                var consumptionAmount = decimal.Parse(incomeAmountEntry.Text);
                var selectedCategory = (Category) categoryPicker.SelectedItem;
                Account.Balance += consumptionAmount;
                db.Accounts.Update(Account);
                var historyToAdd = new History
                {
                    AccountId = Account.Id,
                    Balance = Account.Balance,
                    CreatedOn = DateTime.Now,
                    Comment = incomeCommentEditor.Text,
                    Amount = consumptionAmount,
                    CategoryId = selectedCategory.Id
                };
                db.Histories.Add(historyToAdd);
                db.SaveChanges();

                historyToAdd.Category = selectedCategory;
                IncomeCreated?.Invoke(historyToAdd);
            }
```

```
            await DisplayAlert("Notification", "New income was added", "Ok");
            await Navigation.PopAsync();
        }
    }
```

C# CreateNewAccount.xaml.cs

```
private async void CreateAccount(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(accountNameEntry.Text))
            {
                await DisplayAlert("Error", "The field for entering the account name
is required", "Зрозуміло");
                accountNameEntry.Focus();
                return;
            }

            if (currencyPicker.SelectedItem == null)
            {
                await DisplayAlert("Error", "The currency selection field is
required", "Ok");
                return;
            }

            using (var db = new ApplicationContext(App.DbPath))
            {
                var selectedCurrency = (Currency) currencyPicker.SelectedItem;
                var accountToAdd = new Account
                {
                    CurrencyId = selectedCurrency.Id,
                    Name = accountNameEntry.Text,
                    Balance = 0
                };
                db.Accounts.Add(accountToAdd);
                db.SaveChanges();

                accountToAdd.Currency = selectedCurrency;
                AccountCreated?.Invoke(accountToAdd);
            }

            await DisplayAlert("Notification", "A new account has been created",
"Ok");
            await Navigation.PopAsync();
        }
    }
```

C# CreateCurrency.xaml.cs

```
  private async void CreateNewCurrency(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(currencyNameEntry.Text))
            {
                await DisplayAlert("Error", "The currency name entry field is
required", "Ok");
                currencyNameEntry.Focus();
                return;
            }

            using (var db = new ApplicationContext(App.DbPath))
            {
                var currencyToAdd = new Currency
                {
                    Name = currencyNameEntry.Text
```

```
                };
                db.Currencies.Add(currencyToAdd);
                db.SaveChanges();

                CurrencyCreated?.Invoke(currencyToAdd);
            }

            await DisplayAlert("notification", "A new currency has been created",
"Ok");
            await Navigation.PopAsync();
        }
    }
```

C# CreateCategory.xaml.cs

```
private async void CreateNewCategory(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(categoryNameEntry.Text))
            {
                await DisplayAlert("Error", "Category field required", "Ok");
                categoryNameEntry.Focus();
                return;
            }

            if (categoryTypePicker.SelectedItem == null)
            {
                await DisplayAlert("Error", "Category type field required", "Ok");
                return;
            }

            using (var db = new ApplicationContext(App.DbPath))
            {
                var categoryToAdd = new Category
                {
                    Type = categoryTypePicker.SelectedItem.ToString() == "Income"
                        ? AccountChangeType.Income
                        : AccountChangeType.Expense,
                    Name = categoryNameEntry.Text
                };
                db.Categories.Add(categoryToAdd);
                db.SaveChanges();

                CategoryCreated?.Invoke(categoryToAdd);
            }

            await DisplayAlert("Notification", "New category was create", "Ok");
            await Navigation.PopAsync();
        }
    }
```