Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

Пояснювальна записка

До бакалаврської дипломної роботи

на тему:« Програмне забезпечення для контролю персональних фінансів фізичних осіб для мобільної платформи»

Виконав:студент 4курсу, групи 1КІ-18б
напряму підготовки (спеціальності)
123 – Комп'ютерна інженерія
Дідур І. В._____

Керівник
ст., викл. Муращенко О. Г._____

Рецензент Зав.Кафедри ЗІ,д.т.н., проф.
Лужецький В.А_____

Вінниця 2022

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 123 – «Комп'ютерна інженерія»
Спеціальність 123 – «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Завідувач кафедри
обчислювальної техніки
д.т.н., проф. Азаров О. Д.
"21" січня 2022року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Дідуру Ігорю Вячеславовичу

1 Тема роботи «Програмне забезпечення для контролю персональних фінансів фізичних осіб для мобільної платформи», керівник роботи Муращенко Олександр Геннадійович , к.т.н., доц., затверджені наказом вищого навчального закладу від "09"березня2022року №66

2 Строк подання студентом роботи 20.06.22.

3 Вихідні дані до роботи: технічний опис програмного застосунку, мова програмування C#, мобільний додаток, середовище розробки Microsoft Visual Studio.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):вступ, аналіз технологій об'єктно–орієнтованого програмування для розробки мобільних додатків, розробка структури та алгоритмів роботи програми для проектування опаленння, програмна реалізація та тестування програми.

5 Графічний матеріал — діаграма класів.

6 Консультанти розділів роботи наведені в таблиці 1.

Таблиця 1 — Консультанти роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Муращенко О.Г., ст,.вкл.		

7 Дата видачі завдання 09.03.2022р.

8 Календарний план наведено в таблиці 2.

Таблиця 2— Календарний план

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів проекту (роботи)	Приміт-ка
1	Постановка задачі роботи	09.03.22	вик.
2	Пошук матеріалів по технологіям розробки віконних додатків	10.03.22 —25.03.22	вик.
3	Структурне проектування додатку організації та оптимізації часу та робочого процессу	26.03.22 — 28.03.22	вик.
4	Обгрунтування та вибір засобів реалізації системи	29.03.22 — 04.04.22	вик.
5	Розробка головного вікна, методів створення календарю та подій	05.04.22 —29.04.22	вик.
6	Підготовка матеріалів та розробка алгоритму збереження та виведення списку подій	30.04.22 —27.05.22	вик.
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.05.22 — 06.06.22	вик.
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	07.06.22	вик.

Студент _____ Дідур І.В.

Керівник роботи _____ Муращенко О.Г.

АНОТАЦІЯ

Дідур І.В. Бакалаврський дипломний проект зі спеціальності 123 — Комп’ютерна Інженерія, Вінниця: ВНТУ, 2022.

Пояснювальна записка містить 85 сторінки,15 рисунків,5 таблиць та 20 посилань.

Дану бакалаврську дипломну роботу присвячено створенню додатку контролю персональних фінансів фіз осіб.

В роботі був виконаний аналіз найбільш сучасних методів розробки мобільних додатків.

Перевірка працездатності системи протестована шляхом практичного тестування системи.

Ключові слова: облік фінансів, мобільна розробка, android, фінансова грамотність.

ABSTRACT

This bachelor's thesis is devoted to the creation of an application for the control of personal finances of individuals.

The analysis of the most modern methods of mobile application development was performed in the work.

The system is tested by practical testing of the system.

Key words: finance accounting, mobile development, android, financial literacy.

ЗМІСТ

ВСТУП

В сучасному світі у нас стає все більше коштів і дуже важливо грамотно ними розпоряджатися. Для цього можна вести зошит з доходами та витратами, найняти особистого бухгалтера чи встановити додаток на смартфон і покласти всі турботи на нього.

На даний час існують сотні подібних програм, проте частина з них платна, а інша частина потребує значного рівня фінансової та/або технічної грамотності. Лише дуже незначна кількість додатків є зручними для користувача і не потребують суттєвих знань. Проте вони зазвичай не мають української локалізації чи призначені для роботи в іноземній валюті і цим суттєво втрачають цінність для українського користувача.

Тому **актуальність** даної роботи в тому, що розроблена програма буде спеціалізована для українського користувача, не потребуватиме суттєвих технічних навиків та не потребуватиме складних інструкцій для початку роботи.

Об’єкт дослідження — процес створення програмного забезпечення для контролю персональних фінансів фіз осіб для мобільних платформ.

Предмет дослідження — програмного забезпечення для контролю персональних фінансів фіз осіб для мобільних платформи

Метою роботи є розробка програмного забезпечення для контролю персональних фінансів фіз осіб для Android.

Задачі дослідження бакалаврської роботи:

— проаналізувати методи та технології розробки додатків під мобільні;
— проаналізувати існуючі аналоги;
— проаналізувати та обрати засоби реалізації системи;
— виконати моделювання;
— розробити мобільний додаток за допомогою мови програмування C#;
— протестувати розроблений додаток.

Практичне значення отриманих результатів полягає в можливості використовування розробленого програмного забезпечення для керування власними фінансами.

Методи дослідження бакалаврської роботи: у роботі використано принципи об’єктно-орієнтованого програмування мовою C# для реалізації мобільного додатку.

Апробація результатів бакалаврської роботи: зроблено доповідь на L науково–технічній конференції підрозділів Вінницького національного технічного університету.

1 АНАЛІЗ ТЕХНОЛОГІЙ ОБ'ЄКТНО–ОРІЄНТОВАНОГО ПРОГРАМУВАННЯ ДЛЯ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ

1.1 Особливості створення мобільних додатків

Насамперед важливо визначитися з технологією розробки. Існує кілька видів технологій, проте найчастіше вибір стоїть між нативними та гібридними програмами.

Нативні програми розробляють під конкретну ОС. Наприклад, додаток для Android пишуть мовами C# або Java, а для iOS - на Swift і Objective C. Такі програми відрізняються швидкою роботою та інтерфейсом. Проте їхня розробка коштує вдвічі дорожче, ніж розробка гібридних рішень.

Гібридні або кросплатформні програми містять риси нативного та веб-додатків. Для розробки використовують фреймворки, засновані на HTML5/JavaScript (React Native, Ionic). Гібридні програми працюють на будь-яких платформах, але мають більше обмежень [1].

Вибір технології залежить від того, чи потрібна вам програма iOS і Android або на перший час достатньо однієї ОС. У першому випадку варто вибрати нативний варіант. У другому – кросплатформовий [2].

Dажливо знати кілька моментів при написанні програми під Android. По-перше, це велика кількість пристроїв. Це ускладнює розробку програми.

Наприклад апаратно у девайса може бути передня камера, а може і ні. Сімкарт може бути будь-яку кількість. Фізичні кнопки можуть бути присутніми чи ні. Екрана може бути два: додатковий із тильного боку або на чохлі.

Існуючі елементи мають різні параметри. Наприклад, датчик акселерометра у всіх мобільних може бути встановлений у кількох варіантах [3].

Розмір екрану та роздільна здатність – окрема проблема. Наприклад, якщо ви хочете розмістити зображення на повний екран iOS, ви можете використовувати кілька зображень типових розмірів для iPhone 6 і вище, iPhone

6 Plus і вище, iPhone X і iPhone X Max. У випадку з Android екрани мають різну роздільну здатність, співвідношення сторін і щільність [4].

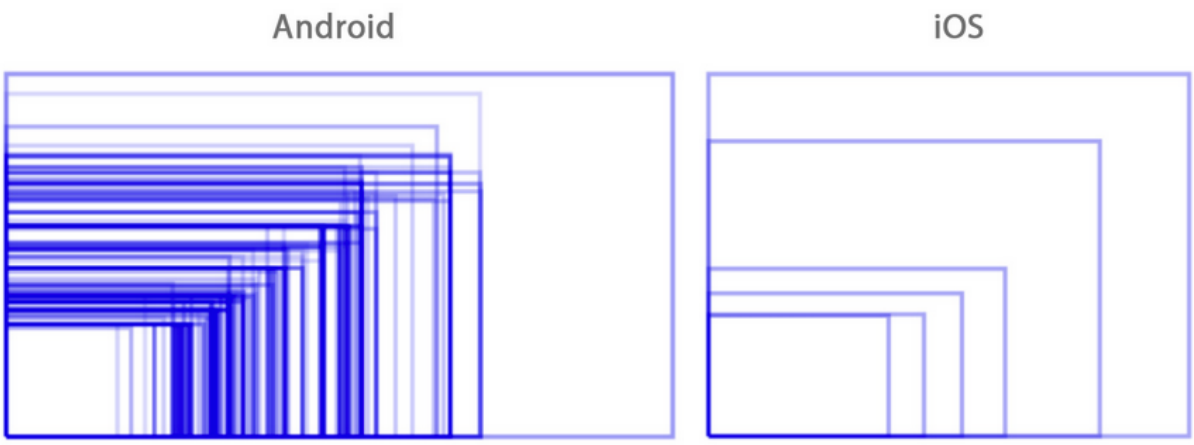

Рисунок 1.1 — Схема основних розмірів екранів для смартфонів

У зв'язку з цим для Android розробників існують різні інструменти, наприклад 9 Patch — схема розмітки, що дозволяє задати правила розтягування зображення при зміні його розміру. Без неї складно розробити програми під android і коректно відобразити, у тому числі, і фонові зображення у зв'язку з розкидом у розмірах екранів.(див. рисунок 1.2)

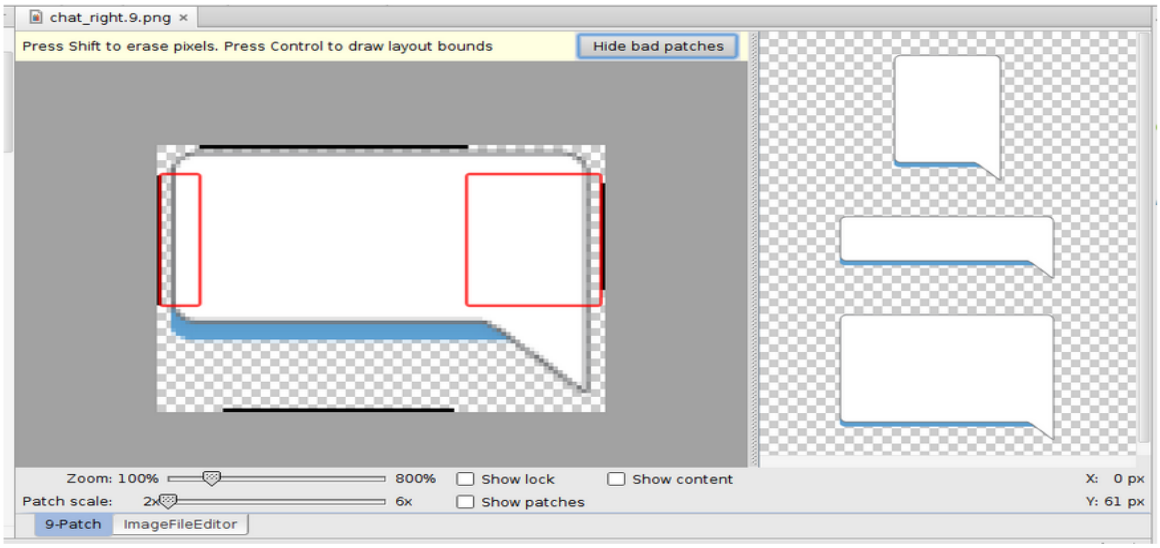

Рисунок 1.2 — 9-patch

По-друге, це великий розкид операційної системи Android, встановлених у користувачів. Це породжує безліч проблем при створенні мобільних додатків для Android з нуля:

У процесі розробки необхідно враховувати особливості відображення інтерфейсу різних версіях ОС і оболонках. Так, системні елементи управління можуть виглядати не однаково на кількох версіях Android і оболонках однієї і тієї ж версії Android [5].

Різні версії у ряді моментів мають відмінну одна від одної логіку роботи. Наприклад, до версії 6.0 програми не повинні були вимагати будь-яку роздільну здатність окремо (доступ до камери, мікрофону і так далі), вони вказувалися списком в Google Play і, мав на увазі, що користувач ознайомлюється з ними до моменту завантаження. Починаючи з 6.0, кожен дозвіл має бути запитаний окремо вже в момент роботи програми. Відповідно, якщо ви не реалізуєте обидва варіанти логіки при розробці мобільного додатка андроїд, воно не буде працювати або до версії 6.0, або в більш пізніх.

Програмні методи та бібліотеки змінюються: якісь із них визнаються застарілими та їх потрібно замінювати на новіші. Таким чином завжди встає вибір: або підтримувати останні функції ОС, або дозволити якомога більшій кількості користувачів встановити мобільний додаток.

В останніх видах ОС додалася багатозадачність робочої області. Користувач може відобразити на робочій області одночасно кілька додатків і вашому може бути виділена абсолютно довільна за розміром область. Це також треба враховувати під час створення програм.

Третє, що потрібно враховувати при розробці додатків андроїд з нуля, це архітектура самого додатка. На відміну від iOS, де програми архітектурно є чимось єдиним цілим, в Android вони збираються з логічно самостійних і відокремлених частин — активіті і фрагментів, див. рисунок 1.3.

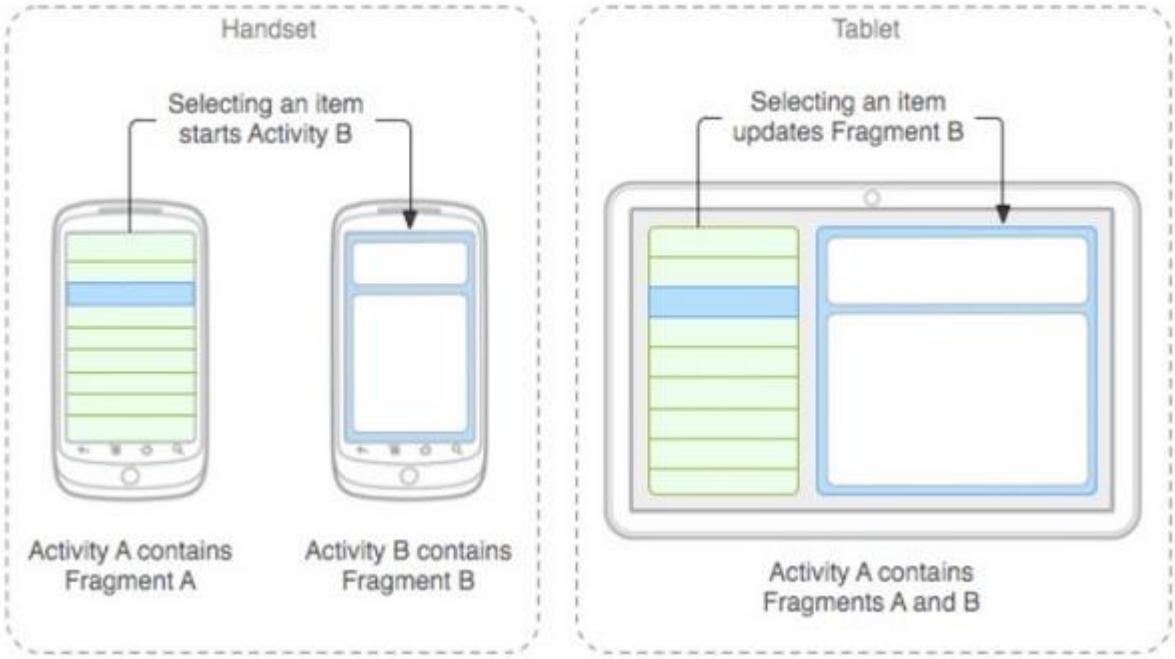

Рисунок 1.3 — Активіти і фрагменти

Такий підхід був створений саме для того, щоб забезпечити роботу додатків на будь-яких мобільних, у тому числі з дуже малим обсягом оперативної пам'яті та дуже слабкими процесорами. Якщо частини програми незалежні, будь-яку з них можна в потрібний момент викинути з пам'яті і не витрачати на підтримку її життєвого циклу дорогоцінні ресурси [6].

Наприклад, якщо на екрані набір гаманців, користувач обирає гаманець і провалюється в нього. Другий екран, картка гаманця, нічого не повинна знати про попередній список, тому що в будь-який момент часу, зокрема відразу після переходу в картку, він може бути вивантажений з оперативної пам'яті та знищений. Це станеться, наприклад, якщо у фоні запущено багато програм або в картці ви починаєте програвати відео у високій якості.

Щоб програма працювала коректно і без збоїв, екран картки не повинен звертатися до будь-якої інформації попереднього екрана, приймаючи на вхід лише певні дані. Якщо, наприклад, користувач має можливість перейти на наступний гаманець, не повертаючись до списку, то картка повинна самостійно отримати

необхідну інформацію. У той самий час екран списку нічого не повинні знати про саму картку, т.к. після повернення з неї вона також може бути знищена.

Цей аспект архітектури додатків звучить занадто технічно, але він дає зрозуміти чому, наприклад, далеко не для всіх типів додатків можлива кроссплатформенна розробка: якщо це щось об'ємне по функціоналу, воно повністю вивантажується з пам'яті при нестачі місця і на слабких пристроях можливість роботи з ними просто відсутня.

У процесі розробки додатків для Android є ряд особливостей [7].

На відміну від iOS, додатки для Android є взаємозв'язком окремих, логічно відокремлених елементів, як про це говорилося вище. Тобто не можна просто взяти та імпортувати програму на іншу операційну систему, переписавши код з однієї мови програмування на іншу. Тобто при створенні програм під android потрібно закладати зовсім іншу архітектуру.

Інший підхід спостерігається та в інших аспектах.

Наприклад, сучасна іконка програм може мати різну форму залежно від налаштувань операційної системи. Дизайнер повинен це враховувати та переконатися, що логотип виглядає чудово та гармонійно у всіх варіантах.

При розробці мобільних додатків для Android важливо спиратися на Material Design. Це ціла філософія створення інтерфейсу користувача. Офіційна документація з цього підходу включає сотні документів, що докладно описують як його принципи, так і конкретні приклади правильного і неправильного використання правил для кожного елемента інтерфейсу.

Кнопка, панель навігації, іконка та всі інші елементи повинні дотримуватися цих правил, якщо потрібно сконструювати гарний матеріал інтерфейс і потім не було проблем з просуванням програми в Google Play.

Рекомендованим Google мовою програмування, при створенні програм під android, в даний час є Kotlin, не Java. Відмінність між ними істотно менша, ніж між Objective-C і Swift для iOS, але все ж таки це трохи відрізняються підходи до розробки. Див рисунок 1.4

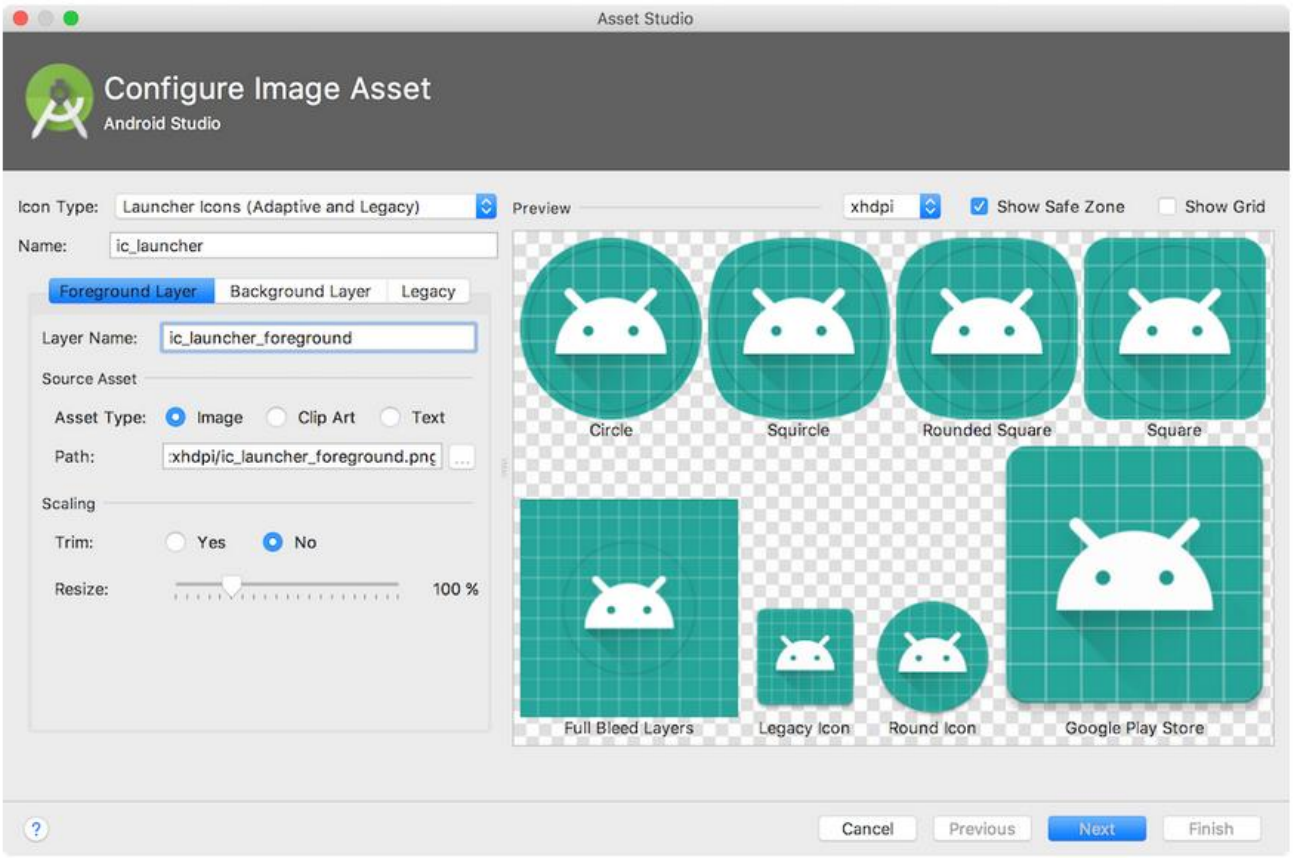

Рисунок 1.4 — іконка для різних пристроїв

Тестування на великій кількості фізичних пристроїв (не емуляторів) при цьому має надзвичайно важливе значення при створенні додатків андроїд. Навіть воно, через величезну кількість телефонів на ринку, не забезпечує безпроблемне функціонування на всіх доступних моделях, але принаймні знижує ймовірність проблем на найбільш популярних девайсах.

1.2 Огляд популярних способів локального збереження даних

Існує три способи збереження даних мобільного додатка: Shared preferences/User defaults, файли і база даних. Вибір того чи іншого способу залежить від обсягу даних, з якими має справу додаток, їхнього типу і того, що потрібно буде з цими даними робити. Традиційно розробники мобільних додатків використовували SQLite, проте також можна підключити і інші СУБД, наприклад Realm.

SharedPreferences — постійне сховище на платформі Android, яке використовується програмами для збереження своїх налаштувань, наприклад. Це сховище є відносно постійним, користувач може зайти в налаштування програми та очистити дані програми, тим самим очистивши всі дані у сховищі.

Для роботи з даними постійного сховища нам знадобиться екземпляр класу SharedPreferences, який можна отримати у будь-якого об'єкта, успадкованого від класу android.content.Context (наприклад, Activity або Service). Об'єкти цих класів (успадковані від Context) мають метод getSharedPreferences, який приймає таких два параметри.

Name — вибране налаштування файлу. Якщо файл налаштувань з таким ім'ям не існує, він буде створений під час виклику методу edit() та фіксування змін за допомогою методу commit().

Mode — режим роботи, можливі значення:

— MODE_PRIVATE — використовується в більшості випадків для приватного доступу до даної програми-власника;

—MODE_WORLD_READABLE – тільки для читання;

—MODE_WORLD_WRITEABLE - тільки записи;

—MODE_MULTI_PROCESS – кілька процесів спільно використовують один файл SharedPreferences.

Щоб отримати значення необхідної змінної, існують такі методи об'єкта SharedPreferences:

— getBoolean(String key, boolean defValue);
— getFloat(String key, float defValue);
— getInt(String key, int defValue);
— getLong(String key, long defValue);
— getString(String key, String defValue);
— getStringSet(String key, Set defValues).

А для запису значень – такі:

— putBoolean(String key, boolean value);

— putFloat(String key, float value);

— putInt(String key, int value);

— putLong(String key, long value);

— putString(String key, String value);

— putStringSet(String key, Set values).

На лістингу 1.1. показано код для запису змінної типу рядка в сховище.

Лістинг 1.1. Запис string в сховище

```
SharedPreferences settings =
context.getSharedPreferences(PERSISTANT_STORAGE_NAME,
Context.MODE_PRIVATE);
SharedPreferences.Editor editor = settings.edit();
editor.putString( "name", "John" );
editor.commit();
```

User defaults — це аналогічний підхід для iOS додатків.

Також можна використовувати файли для збереження даних додатку. Вони можуть бути у форматі json, xml чи csv. Розглянемо детальніше ці формати.

JSON (англ. JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється дже́йсон) — це текстовий формат обміну даними між комп'ютерами. JSON базується на людинозрозумілому тексті, який може бути зрозумілий для людини. Формат дає змогу описувати різні об'єкти та структури даних. Цей формат використовується в основному для передавання структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє основне призначення в написанні вебпрограм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає як заміна XML (використовується в AJAX) під час асинхронного пересилання структурованої інформації між клієнтом та сервером. При цьому перевагою JSON

перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти [8].

XML – це розши́рювана мо́ва розмі́тки (англ. Extensible Markup Language, скорочено XML) — запропонований консорціумом World Wide Web Consortium (W3C) стандарт побудови мов розмітки ієрархічно структурованих даних для обміну між різними програмами, зокрема, через Інтернет. Є спрощеною підмножиною мови розмітки SGML. XML-документ складається із текстових символів, і придатний до читання людиною.

Стандарт XML визначає набір базових лексичних та синтаксичних правил для побудови мови описування інформації шляхом застосування простих тегів. Цей формат достатньо гнучкий для того, аби бути придатним для застосування в різних галузях [2]. Іншими словами, даний стандарт визначає метамову, на основі якої шляхом запровадження обмежень на структуру та вміст документів визначаються специфічні, предметно-орієнтовані мови розмітки даних. Ці обмеження описуються мовами схем (англ. Schema), такими як XML Schema (XSD), DTD або RELAX NG. Прикладами мов, заснованих на XML, є: XSLT, XAML, XUL, RSS, MathML, GraphML, XHTML, SVG, а також XML Schema [9].

CSV — файловий формат, котрий є відмежовувальним форматом для представлення табличних даних, у якому поля відокремлюються символами коми та переходу на новий рядок. Поля, що містять коми, декілька рядків, або лапки (позначаються подвійними лапками), мають обмежуватися з обох боків лапками.

Формат CSV використовується для перенесення даних між базами даних та програмами — редакторами електронних таблиць.

Якщо порівнювати ці три формати, то Json найкраще підходить для збереження різнопланової інформації без суттєвої надлишковості, xml – людинозрозумілий формат, а csv найкраще підходить для збереження великих обсягів табличних даних з можливістю легкого редагування звичайним блокнотом.

І третій та найпопулярніший метод – бази даних. Як вже було сказано SQLite одна з найпопулярніших СУБД.

СУБД – це систе́ма управління ба́зами да́них — набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних [10].

Основні характеристики СУБД:

— контроль за надлишковістю даних;

— несуперечливість даних;

— підтримка цілісності бази даних (коректність та несуперечливість);

— цілісність описується за допомогою обмежень;

— незалежність прикладних програм від даних;

— спільне використання даних;

— підвищений рівень безпеки.

Можливості СУБД:

— дозволяється створювати БД (здійснюється за допомогою мови визначення даних DDL (Data Definition Language));

— дозволяється додавання, оновлення, видалення та читання інформації з БД (за допомогою мови маніпулювання даними DML, яку часто називають мовою запитів);

— можна надавати контрольований доступ до БД за допомогою:

— системи забезпечення захисту, яка запобігає несанкціонованому доступу до БД;

— системи керування паралельною роботою прикладних програм, яка контролює процеси спільного доступу до БД;

— система відновлення — дозволяє відновлювати БД до попереднього несуперечливого стану, що був порушений в результаті збою апаратного або програмного забезпечення.

Використання СКБД в мобільних пристроях:

— апаратне забезпечення;

— програмне забезпечення;

— дані;

— процедури — інструкції та правила, які повинні враховуватись при проєктуванні та використанні БД;

— користувачі

— адміністратори даних (керування даними, проєктування БД, розробка алгоритмів, процедур) та БД (фізичне проєктування, відповідальність за безпеку та цілісність даних);

— розробники БД;

— прикладні програмісти;

— кінцеві користувачі.

Загальноприйнятою є трирівнева система організації СУБД ANSI-SPARC, при якій існує незалежний рівень для ізоляції програми від особливостей представлення даних на нижчому рівні.

Рівні:

— зовнішній — представлення БД з точки зору користувача.

— концептуальний — узагальнене представлення БД, описує які дані зберігаються в БД і зв'язки між ними, підтримує зовнішні представлення, підтримується внутрішнім рівнем;

— внутрішній — фізичне представлення БД в комп'ютері.

Логічна незалежність — повна захищеність зовнішніх моделей від змін, що вносяться в концептуальну модель.

Фізична незалежність — захищеність концептуальної моделі від змін, які вносяться у внутрішню модель.

SQLite — це легка СУБД. Поширюється як суспільне надбання. Там реалізовано багато зі стандарту SQL-92.

Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє програма, а надає

бібліотеку, з якою програма компілюється і двигун стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує додаткові витрати, час відгуку і суттєво спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, таблиці, індекси та дані) в єдиному стандартизованому файлі на тому комп'ютері, на якому виконується додаток. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції весь файл, що зберігає базу даних, блокується; ACID-функції досягаються зокрема за рахунок створення файлу-журналу.

Кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. Запис в базу можна здійснити лише в тому випадку, коли жодних інших запитів у цей час не обробляється; інакше спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоповторення спроб запису протягом деякого інтервалу часу.

У комплекті постачання йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3. З допомогою даного файлу демонструється реалізація основних функцій цієї бібліотеки. Клієнтська частина працює з командного рядка, і дозволяє звертатися до файлу БД на основі типових функцій ОС.

Завдяки архітектурі рушія можливо використовувати SQlite як на вбудовуваних (embedded) системах, так і на виділених машинах з гігабайтними масивами даних.

Realm — СУБД з відкритим вихідним кодом. Realm забезпечує в 2,3 рази вищу швидкодію порівняно з SQLite.

Realm підтримує денормалізацію — зберігання моделей, які мають зв’язки, на місці.

Припустимо, що у нас є дві моделі: користувач та адреса електронної пошти. Користувач може мати кілька поштових скриньок. Щоразу, коли нам потрібна модель користувача, нам також доступні адреси усіх його електронних скриньок.

Нам не потрібно виконувати додаткові запити на отримання списку усіх його адрес. Ми можемо зберегти адреси його поштових скриньок у вигляді списку моделей в моделі користувача.

Також Realm простіший у впровадженні порівняно з SQLite, проте це суб’єктивна думка.

1.3 Аналіз існуючих додатків контролю фінансів.

Існує безліч додатків для контролю фінансів, нижче наведено 3 з них які користуються значною популярністю:

— Money Manager Expense & Budget;

— ClevMoney — Personal Finance;

— Money Lover — Менеджер расходов.

У додатку Money Manager Expense & Budget можна вести витрати, доходи та загальний баланс за календарем: день, тиждень, місяць. Кожній витраті можна присвоїти назву рахунку (наприклад, «Гаманець» або «Карта»), категорію («Харчування», «Одяг», «Транспорт» тощо), коментар та фотографію (чека або, наприклад, самої покупки). Є статистика у вигляді графіків та діаграм.

Фішки програми: основну інформацію з програми можна вивести на екран як віджет.

Отримати доступ до програми можна і з комп'ютера - зручно для тих, хто звик аналізувати витрати в Excel. Часто використовувані операції можна додати до «Вибраного» та проводити автоматично. Для кредитних карток і боргів можна налаштувати «нагадування» та відслідковувати суму майбутніх оплат.

ClevMoney — Personal Finance має лише англійський інтерфейс — від іноземних розробників. Хороший спосіб підтягнути мову, тим більше, що у ClevMoney немає складних функцій. Користувачам доступні витрати з розбивкою за категоріями, різнокольорові діаграми, графіки зі стовпцями витрат (можна побачити, в які дні місяця ви витрачаєте більше і менше грошей). Мінімалістичний дизайн порадує любителів мінімалізму.

Фішки програми: програма може автоматично підтягувати в витрати і доходи інформацію з СМС від банків - щоправда, українським користувачам для цього, можливо, потрібно написати в техпідтримку. Дані можна зберігати у хмарі, захищати паролями та експортувати до файлу CSV та інше.

Money Lover — одна з найвідоміших і найтитулованіших додатків про бюджет. Витрати розносяться за категоріями, кожна з яких має свою красиву іконку. Усі витрати за вибраний період можна проаналізувати на кольоровій діаграмі.

Фішки програми: звіт за доходами і витратами за рік можна побачити на одному графіку відразу — і зробити висновки, наскільки ваші витрати залежать від надходжень.

З Money Lover зручно контролювати бюджет: кожній категорії витрат можна привласнити суму на місяць, і якщо ви виходитиме за межі, програма дасть сигнал.

Також сервіс нагадає про відповідні терміни оплати рахунків та боргів. Дані можна захистити паролем та зберігати у хмарі (в тому числі через Dropbox). Якщо вам потрібно підрахувати сімейні витрати, гаманець можна розділити серед кількох користувачів.

У платній версії: немає реклами, необмежені бюджети, є веб-версія, управління кредитними та дебетовими картами, експорт до друкованої версії. Користувачам доступні витрати з розбивкою за категоріями, різнокольорові діаграми, графіки зі стовпцями витрат (можна побачити, в які дні місяця ви витрачаєте більше і менше грошей).

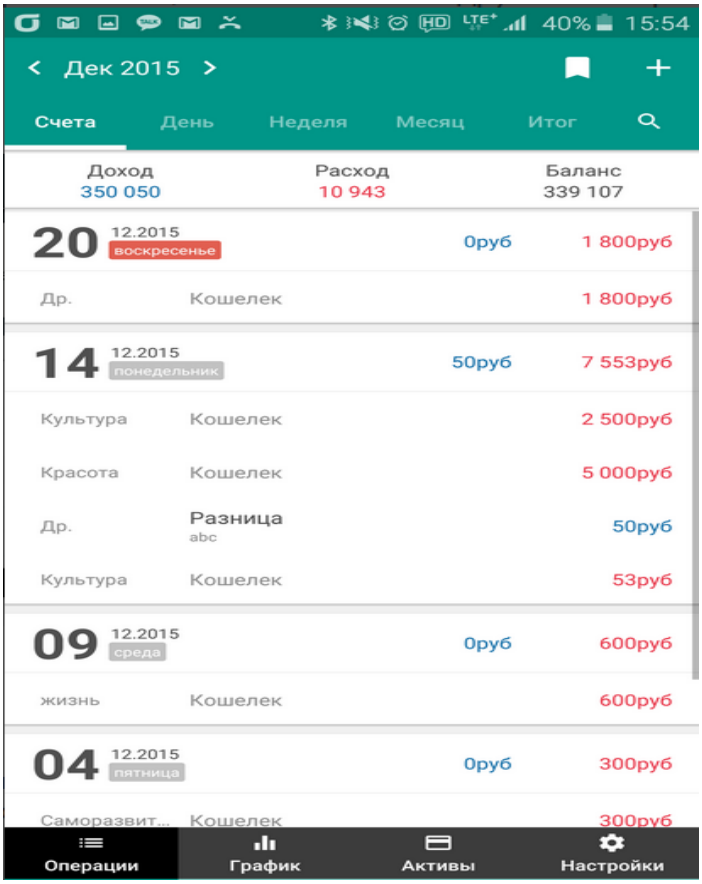

Рисунок 1.5 — Money Manager Expense & Budget

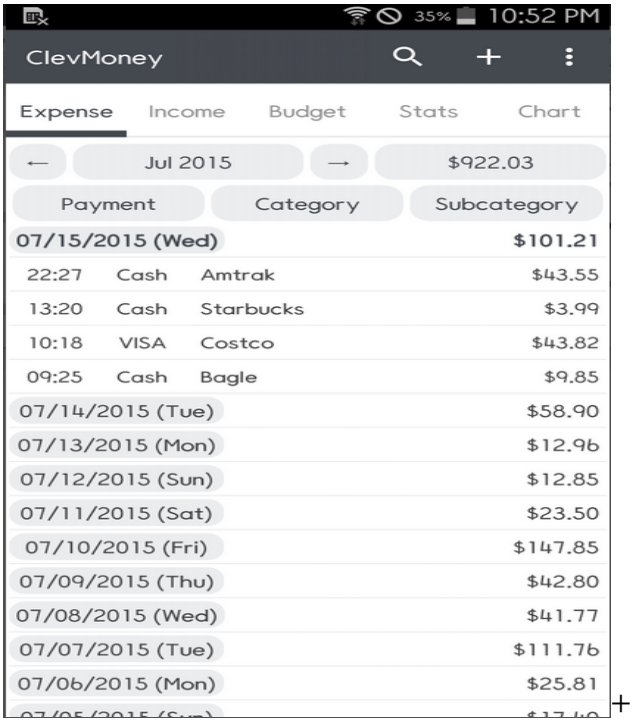

Рисунок 1.6 — ClevMoney — Personal Finance

Мінімалістичний дизайн порадує любителів мінімалізму. Витрати розносяться за категоріями, кожна з яких має свою красиву іконку. Усі витрати за вибраний період можна проаналізувати на кольоровій діаграмі.

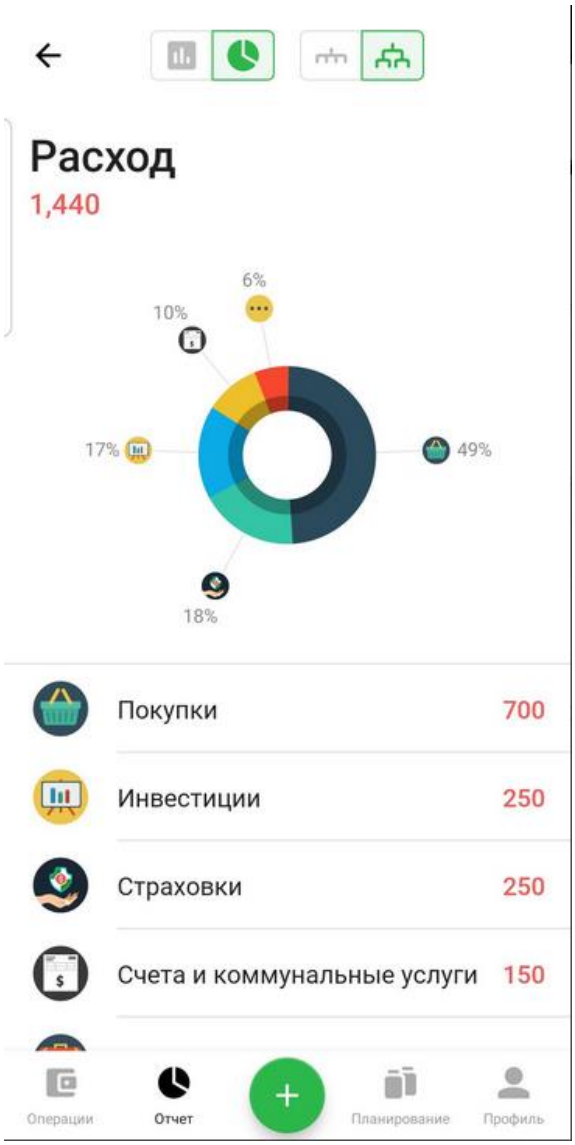

Рисунок 1.7 — Money Lover

2 РОЗРОБКА СТРУКТУРИ ТА АЛГОРИТМІВ РОБОТИ ПРОГРАМИ ДЛЯ КОНТРОЛЮ ФІНАНСІВ

2.1 Розробка інтерфейсу користувача

При розробці інтерфейсу для мобільних пристроїв дуже важливим є врахування зручності використання та повнота функціоналу. Також з врахуванням того, що мова йде про мобільний додаток, потрібно рахуватися з обмеженим простором для інтерфейсу. Всі мобільні додатки розробляються по шаблону single page, тобто одна сторінка чи в нашому випадку екран. На жаль, чи на щастя, мобільний не дозволяє компонувати кілька форм на одному екрані як планшет чи монітор комп’ютера.

Отже спершу розроблялися мокапи інтерфейсу. Мокап це схематична будова користувацьких елементів. Мокапи бувають інтерактивними, проте в даному випадку розроблялися класичні. Їх основна ціль зрозуміти які користувацькі елементи потрібна на екранах і як їх грамотно розташувати.

Почнемо з авторизації. Сценарій даного екрану зі сторони користувача простий як 2 копійки. Користувачу потрібно ввести пароль і натиснути кнопку ОК.

Рисунок 2.1 — Модель екрану авторизації

Далі йде екран пошуку. Його сценарій для користувача наступний:

— вибрати категорію пошуку;

— ввести шукане слово;

— натиснути кнопку «Шукати».

Рисунок 2.2 — Модель екрану пошуку

Також потрібен екран створення звіту. Його логіка роботи наступна:

— вибрати період звіту;

— вибрати вид звіту;

— вибрати форму відображення звіту.

Рисунок 2.3 — Модель екрану створення звіту

Моделі для створення доходу та розходу приблизно однакові (див рис 2.4, рис 2.5). їх логіка наступна:

— вибрати закладку «Дохід» / «Витрата»;
— вибрати «Рахунок»;
— вибрати «Категорію»;
— внести суму;
— натиснути кнопку «Записати».

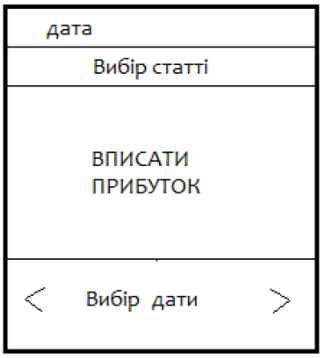

Рисунок 2.4 — Модель екрану внесення прибутку

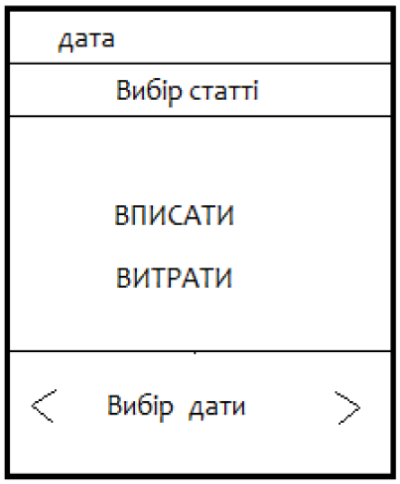

Рисунок 2.5 — Модель екрану внесення витрати

І також не варто забувати про екран для створення категорії. Її сценарій для користувача наступний:

— вибрати закладку «Витрати» або «Доходи»;

— внести відсутню категорію;

— натиснути кнопку зберегти.

Даний клас відповідає за завантаження та збереження бази даних, створення бекапів та додаванняя записів про витрати та доходу. Також даний клас перевіряє правомірність доступу до даних.

Рисунок 2.6 — Модель екрану створення нової категорії

2.2 Проектування класів та алгоритмів обробки.

Було спроектовано 8 класів.

Найперша варто розглянути клас Core.

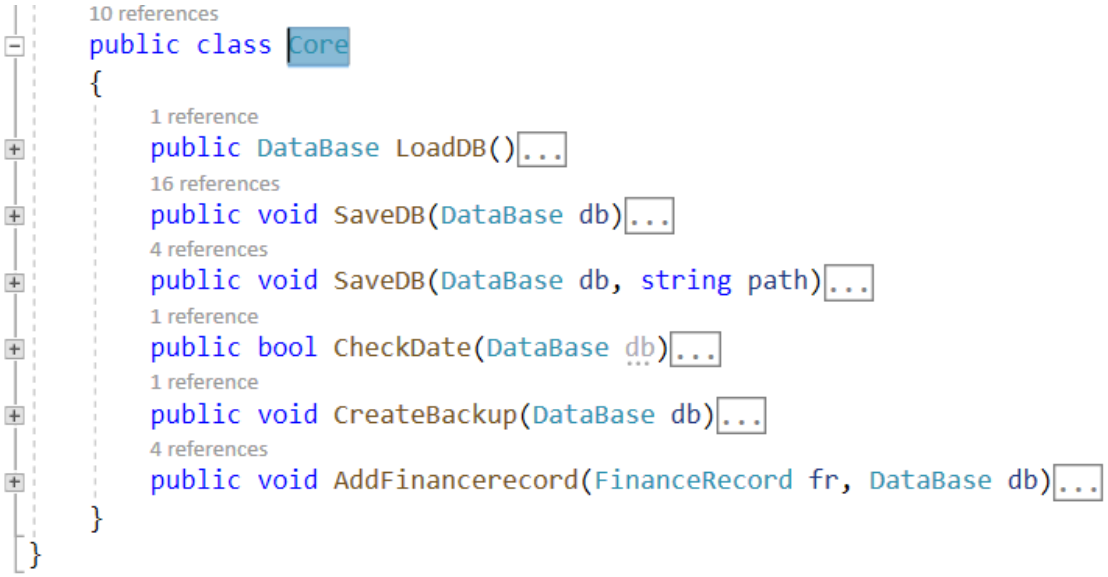

Рисунок 2.7 — Клас Core

Даний клас відповідає за завантаження та збереження бази даних, створення бекапів та додаванняя записів про витрати та доходу. Також даний клас перевіряє правомірність доступу до даних.

Другим по важливості є клас DataBase

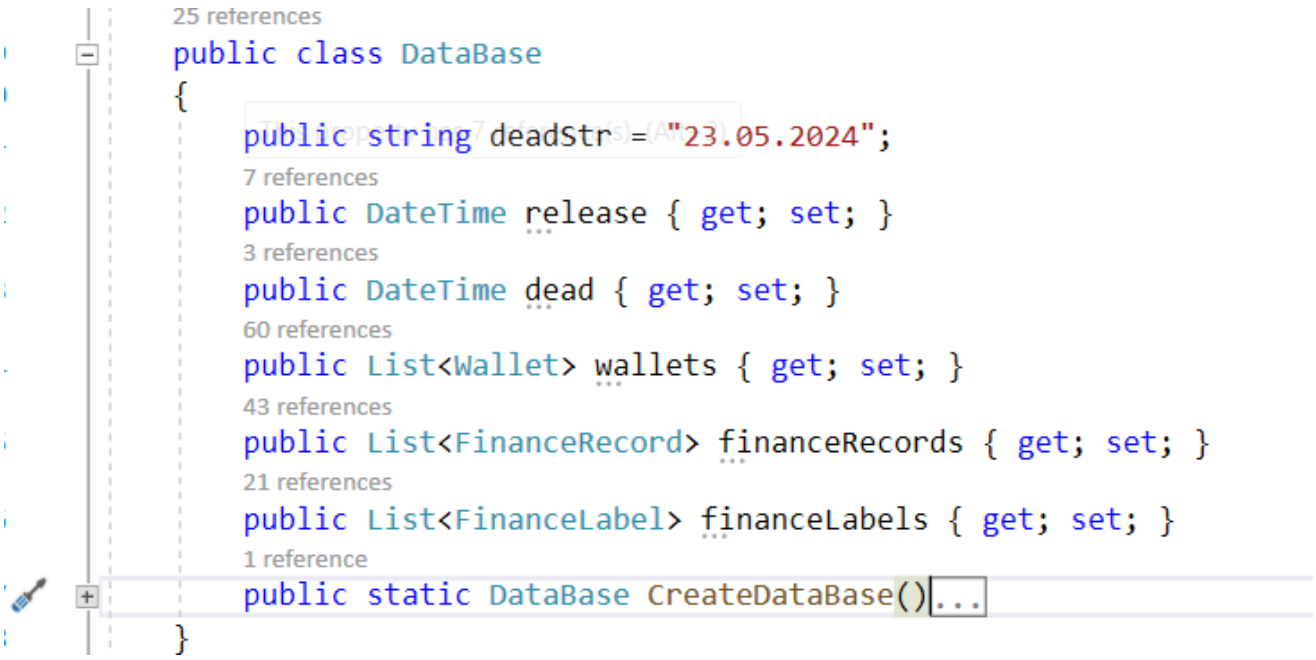

Рисунок 2.8 — Клас DataBase

Даний клас відповідає за збереження даних, а також містить функцію створення пустої бази даних якщо не вдалося завантажити дані чи користувач запускає додаток вперше.

Також важливо згадати про клас гаманця (див рис. 2.9.) даний клас містить інформацію про назву та суму коштів на гаманці, його валюту, опис та пінкод (опціонально).

Даний клас відповідає за завантаження та збереження бази даних, створення бекапів та додаванняя записів про витрати та доходу. Також даний клас перевіряє правомірність доступу до даних.

Також, одним з центральних класів є FinanceRecord. Даний клас відповідає за фінансовий запис, який являється або доходом або витратою. Об’єкти даного класу створюються найчастіше і їх найбільша кількість в базі даних. Кожен об’єкт даного класу містить ідентифікатор, суму, ідентифікатор гаманця якого стосується транзакція, опис транзакції та список фінансових міток. Про них буде згадано пізніше.Див. рисунок 2.10

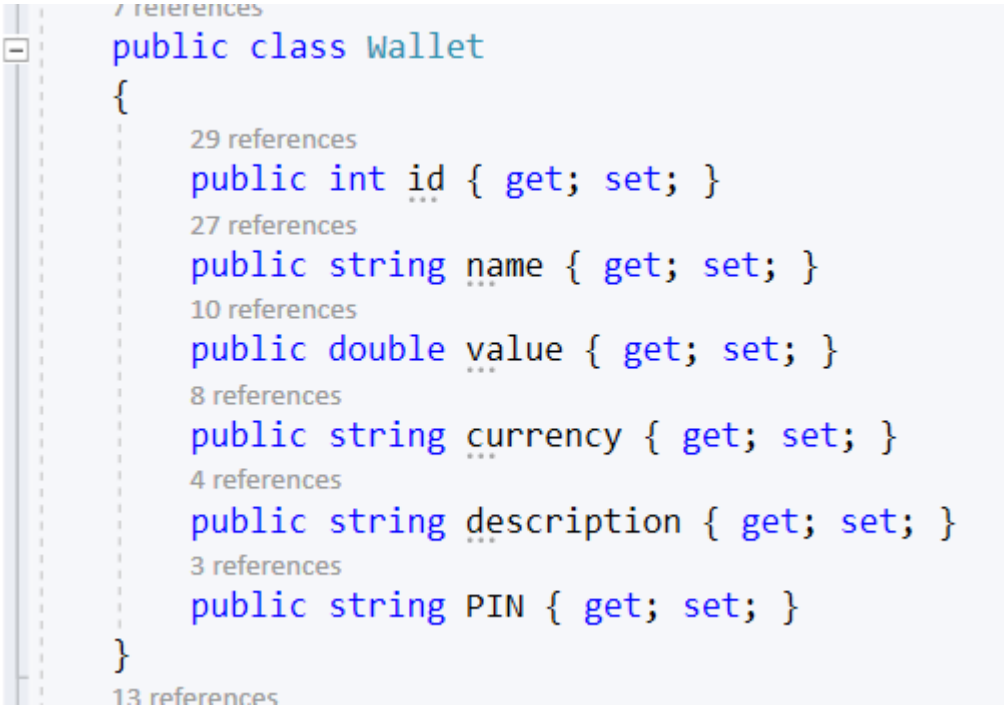

Рисунок 2.9 — Клас Wallet

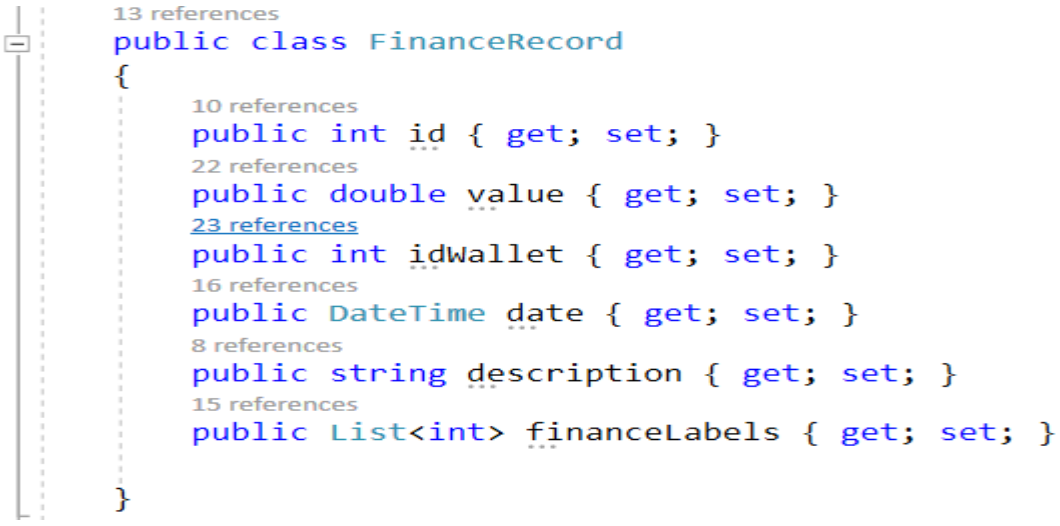

Рисунок 2.10 — Клас FinanceRecord

Клас FinanceLabel, або фінансова мітка. Цей клас містить інформацію про фінансову мітку, її ім’я, бажаний колір та чи є вона наразі активна (неактивні фінансові мітки не можна обрати при створення транзакції).

```
7 references
public class FinanceLabel
{
    10 references
    public int id { get; set; }
    6 references
    public string name { get; set; }
    4 references
    public System.Windows.Media.Color color { get; set; }
    4 references
    public bool isActive { get; set; }
}
```

Рисунок 2.11 — Клас FinanceLabel

При створенні транзакції можна обрати одну, всі чи кілька фінансових міток які її стосуються. Наприклад при покупці в магазині, можна створити транзакцію та відмітити її мітками “Харчі”, “Дім” та “Вкусняшки”, проте ці мітки потрібно попередньо створити. При грамотному виділенні фінансових міток можна суттєво спростити собі фінансову звітність.

Також є два класи які допоміжні і більше стосуються графічного інтерфейсу. Вони показані на рис. 2.12.

```
{
    4 references
    public class HelpFinanceRecordUI...
    5 references
    public class HelpSummary...
}
```

Рисунок 2.12 — Допоміжні класи UI

Перший приводить компактний формат класу FinanceRecord в зручний для читання фомат. Це суттєво економить пам’ять необхідну для збереження даних.

Другий клас, допомагає виводити статистику.

Також варто представити клас StringCipher, який відповідає за шифрування та дешифрування даних.

Даний клас був запозичений з відкритого доступу, проте він дозволяє в один рядочок коду шифрувати та розшифровувати дані. Це свого роду фасад для більших структур коду, які використовують вбудовані класи RijndaelManaged, RNGCryptoServiceProvider та CryptoStream. Перший відповідає за шифрування алгоритмом AES, другий – за криптографічну генерацію чисел, а третій – за те, щоб дані розшифровувалися потоком, хоча алгоритм AES і є блоковим.

```
3 references
public static class StringCipher
{
    // This constant is used to determine the keysize of the encryption algorithm in bits.
    // We divide this by 8 within the code below to get the equivalent number of bytes.
    private const int Keysize = 256;

    // This constant determines the number of iterations for the password bytes generation function.
    private const int DerivationIterations = 1000;

    2 references
    public static string Encrypt(string plainText, string passPhrase)...

    1 reference
    public static string Decrypt(string cipherText, string passPhrase)...

    2 references
    private static byte[] Generate256BitsOfRandomEntropy()...
}
```

Рисунок 2.13 — Клас StringCipher

2.3 Огляд структури збережених даних

Методом збереження даних було обрано збереження у файли структури JSON. Даний метод є найкомпактніший. А з врахуванням того, що дані шифруються, зберігати їх читабельність немає сенсу.

Так як дані шифруються, їх вихідний стан буде суцільним рядком з символами що кодуються UTF8 кодуванням. Проте перед кодуванням ці дані мають більш зрозумілий формат, його і розглянемо.

Спершу йде рядок на кшталт "23.05.2024" та дві дати. Це контрольна дата та дати створення бази та закінчення терміну підписки.

```
"deadStr": "23.05.2022",
"release": "2022-06-10T19:21:18.1453936+03:00",
"dead": "2022-05-23T13:19:23.4681986+03:00",
```


Рисунок 2.14 — Дати

Далі йде список гаманців та їх дані. Див рис 2.15


```
"wallets": [
  {
    "id": 2,
    "name": "ПриватБанк Visa для виплат",
    "value": 7207.0,
    "currency": "UAH",
    "description": "4149 4991 1576 1361",
    "PIN": "[redacted]"
  },
  {
    "id": 3,
    "name": "ПриватБанк MasterCard універсальна",
    "value": 9.0,
    "currency": "UAH",
    "description": "5168 7554 1919 1560",
    "PIN": "[redacted]"
  },
  {
    "id": 8,
    "name": "A-Bank MasterCard для виплат",
    "value": 438.0,
    "currency": "UAH",
    "description": "5375 2352 0941 2061",
    "PIN": "[redacted]"
  },
```


Рисунок 2.15 — Гаманці

Після гаманців йдуть фінансові записи у порядку спадання дати. Це зроблено для того, щоб не потрібно було сортувати дані щоразу при виведенні, адже простіше змінити порядок запису, ніж виконувати алгоритм зі складністю $O(n*\log_2 N)$ щоразу при відкриття екрану з витратами та доходами.

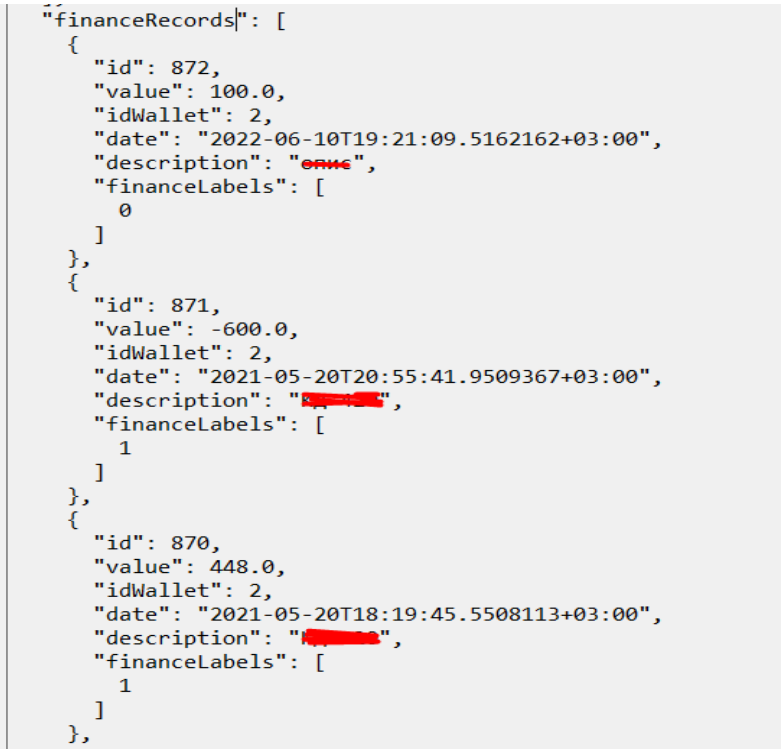

Рисунок 2.16 — Фінансові записи

В самому кінці розташовано масив фінансових міток див. рис. 2.17.

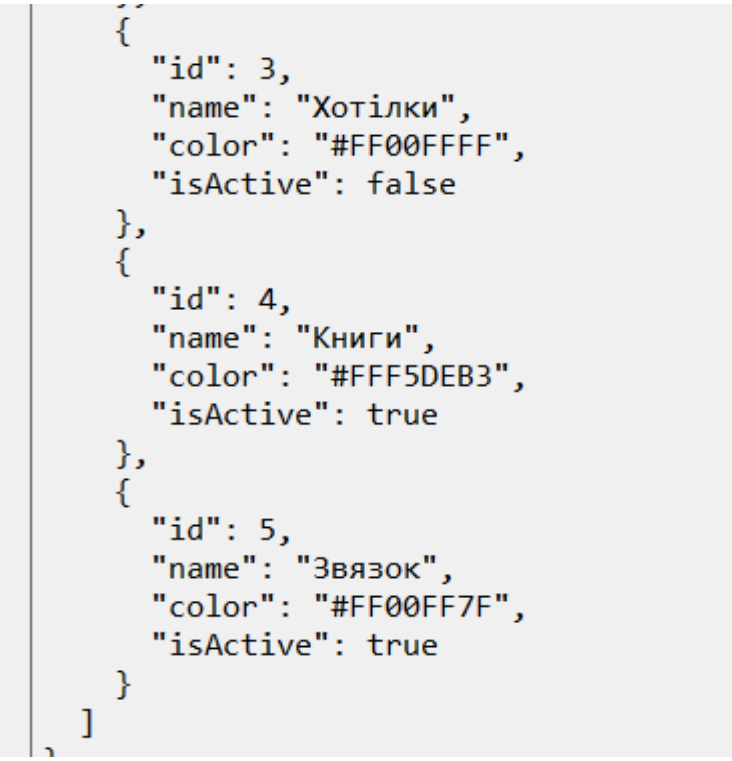

Рисунок 2.17 — Фінансові мітки

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

3.1 Огляд існуючих мов програмування

На даний час існує кілька популярних мов для програмування під мобільні пристрої. До цього списку можна віднести:

— Swift;

— Kotlin;

— Java;

— C#.

Swift — відкрита мультипарадигмальна мова програмування загального призначення. Дана мова компілюється. Вона розроблена і підтримується компанією Apple. Перша версія була представлена у 2014 році.

Найчастіше Swift використовується в розробці програм для macOS, iOS, iPadOS, watchOS та tvOS, однак мова також доступна для Windows та Linux. На платформах Apple мова працює з фреймворками Cocoa та Cocoa Touch та сумісна з основною кодовою базою Apple, написаною більш старою мовою Objective-C. Swift замислювався як більш легка для читання і стійка до помилок зі сторони програміста мова: істотна частина проблем, які у випадку з Objective-C виявляються тільки при виконанні програм і призводять до помилок виконання, у Swift не дозволять скомпілювати код.

Компілятор Swift працює на базі LLVM, внаслідок чого один і той самий код може бути скомпільований для різних платформ: x86, ARM, WASM та інших. Набір інструментів (toolchain) для роботи з мовою вбудований в інтегроване середовище Xcode 6 і вище. Swift може використовувати рантайм Objective-C, що уможливлює використання обох мов (а також мови Ci) в рамках однієї програми.

Apple розділила код Swift на кілька відкритих репозиторіїв:

— компілятор та стандартна бібліотека;

— swift — основний Swift репозиторій, який містить вихідний код компілятора Swift, стандартні бібліотеки і SourceKit;

— swift-Evolution — документація, що відносяться до розвитку Swift, що продовжується, включаючи цілі для майбутніх випусків, пропозиції для змін та розширень Swift;

— бібліотеки ядра;

— swift corelibs-foundation— вихідний код Foundation, який надає загальну функціональність для всіх додатків;

— swift corelibs-libdispatch — вихідний код для libdispatch, який надає примітив паралелізму для роботи з багатоядерномим апаратномим забезпеченням;

— swift corelibs-xctest: вихідний код для XCTest, який забезпечує фундаментальну інфраструктуру для тестування Swift-додатків та бібліотек;

— менеджер пакетів:

— swift package-manager: вихідний код менеджера пакетів Swift;

— swift llbuild: вихідний код для llbuild, системи низького рівня, яка використовує Swift package-manager;

— клоновані репозиторії:

— swift спирається на кілька інших проектів із відкритим кодом, особливо на компілятор LLVM;

— swift llvm: вихідний код LLVM, зі шматочками Swift-додатків;

— swift clang: вихідний код для Clang, зі шматочками Swift доповнень;

— swift lldb: вихідний код Swift-версії LLDB для налагодження Swift програм.

Kotlin (Ко́тлін) — статично типізована, об'єктно-орієнтована мова програмування, що працює поверх Java Virtual Machine і розробляється компанією JetBrains. Також код написаний даною мовою може компілюватися в JavaScript і в код ряду платформ через інфраструктуру LLVM. Мова названа на честь острова Котлін у Фінській затоці, на якій розташоване місто Кронштадт.

Автори ставили собі за мету створити мову більш лаконічну та типобезпечну, ніж Java, і простішу, ніж Scala. Наслідком спрощення порівняно зі Scala стали також швидша компіляція та краща підтримка мови у середовищі

розробки. Мова повністю сумісна з Java, що дозволяє Java-розробникам поступово перейти до її використання без суттєвого навчання; зокрема, мова також вбудовується в Android, що дозволяє для існуючої android-програми впроваджувати нові функції на Kotlin без повного переписування програми.

Kotlin є мовою яку рекомендує Android для написання додатків.

Синтаксис мови використовує елементи JavaScript, Паскаля, TypeScript, Haxe, PL/SQL, F#, Go і Scala, C++, Java, C# та Rust. При оголошенні змінних параметрів типи даних вказуються після назви (розділювач — двокрапка). Крапка з комою, як роздільник операторів, також необов'язковий (як у Scala, Groovy та JavaScript); у більшості випадків переведення рядка достатньо, щоб компілятор зрозумів, що вираз закінчено. Крім ООП Kotlin також підтримує процедурний стиль з використанням функцій. Як і Ci, C++ та D, точка входу в програму — головна функція main, що приймає масив параметрів командного рядка. Програми Kotlin також підтримують perl- та shell-стиль інтерполяції рядків (змінні, включені в рядок, замінюються на свій вміст). Також підтримується виведення типів [11].

Java — строготипізована мова програмування з підтримкою ООП загального призначення. Дана мова розроблена компанією Sun Microsystems (надалі придбаною компанією Oracle). Розробка ведеться спільнотою ентузіастів, організованою через Java Community Process; мова та основні технології, поширюються за ліцензією GPL. Права на торгову марку належать корпорації Oracle.

Програми Java зазвичай компілюються у спеціальний байт-код, тому вони можуть працювати на будь-якій комп'ютерній архітектурі, для якої існує реалізація JVM. Дата офіційного випуску – 23 травня 1995 року. Займає високі місця у рейтингах популярності мов програмування (2-е місце у рейтингах IEEE Spectrum (2020) та TIOBE (2021).

Програми на Java транслюються в байт-код Java, що виконується віртуальною машиною Java (JVM) — програмою, що обробляє байтовий код і передає інструкції обладнання як інтерпретатор [12].

Перевагою такого способу виконання програм є практично повна незалежність байт-коду від операційної системи та обладнання, що дозволяє виконувати Java-програми на будь-якому пристрої, для якого існує відповідна реалізація віртуальної машини. Іншою важливою особливістю мови Java є гнучка система безпеки, у рамках якої виконання програми повністю контролюється віртуальною машиною [13]. Будь-які операції, що перевищують встановлені повноваження програми (наприклад, спроба несанкціонованого доступу до даних або з'єднання з іншим комп'ютером), викликають негайне переривання програми[14].

Часто до вад концепції віртуальної машини відносять зниження продуктивності. Ряд удосконалень дещо збільшив швидкість виконання програм на Java:

— застосування технології трансляції байт-коду в машинний код безпосередньо під час роботи програми (JIT-технологія) з можливістю збереження версій класу в машинному коді;

— широке використання платформно-орієнтованого коду (native-код) у стандартних бібліотеках;

— апаратні засоби, що забезпечують прискорену обробку байт-коду (наприклад, технологія Jazelle, яка підтримується деякими процесорами архітектури ARM).

За даними сайту shootout.alioth.debian.org, для семи різних завдань час виконання на Java становить у середньому у півтора-два рази більше, ніж для C/C++, у деяких випадках Java швидше, а в окремих випадках – у 7 разів повільніше. З іншого боку, обсяг споживання пам'яті Java-машиной був у 10—30 разів більшим, ніж програмою на C/C++. Також деяке дослідження, проведене

компанією Google, згідно з яким зазначається істотно нижча продуктивність і більший обсяг споживання оперативної пам'яті в тестових прикладах на Java в порівнянні з аналогічними програмами написаними на C++ [15].

Ідеї, закладені в концепцію та різні реалізації середовища віртуальної машини Java, надихнули велику кількість ентузіастів на збільшення переліку мов, які могли б використовуватися для створення програм, що виконуються на віртуальній машині. Ці ідеї знайшли отримали друге життя в реалізації загальномовної інфраструктури CLI, закладеної в основу платформи .NET компанією Microsoft.

На даний час Java це єдина мультисередовищна мова програмування. Програми написані з використанням даної мови можуть виконуватися в різних середовищах не залежно від оточення. Єдиною вимогою є встановлена JVM.

Обробка помилок Java схожа на обробку помилок C++ за винятком необхідності в блоці finally. Ця відмінність обумовлена тим, що Java не може дотримуватися концепції RAII через наявність збирача сміття, а автоматичне звільнення ресурсів в деструкторі може йти в непередбачуваному порядку через довільні проміжки часу. Хоча це все звучить дивно, оскільки у C# є та ж проблема зі збирачем сміття.

Здійснюється обробка помилок за допомогою операторів try, catch та finally. Викидається помилка описується об'єктом певного класу, що успадковується від Throwable [12] та відповідного типу помилки. Всередину блоку try міститься код, який може викинути виняток, а блок catch який відловлює задані програмістом типи помилок. При цьому можна вказувати більше одного блоку catch для обробки різних класів помилок або multi-catch для обробки декількох помилок. Блок finally не є обов'язковим, але за наявності виконується незалежно від виникнення помилки та призначений для звільнення виділених під час роботи блоку try ресурсів.

Починаючи з Java 7 підтримується інтерфейс AutoCloseable [13], який дозволяє реалізовувати класи з автоматичним звільнненням ресурсів. Об'єкти

подібних класів потрібно створювати у круглих дужках перед блоком try. Іншими словами автоматичне звільнення ресурсів може бути таким:

Лістинг 3.1. Читання вмісту файла

```
import java.io.*;

public class Main {
   public static void main(String[] args) throws IOException {
      if (args.length < 2) {
         System.err.println("Не вказано ім’я файлу.");
         return;
      }
      String filename = args[1];
      // відкритий файл буде автоматично закритий через помилку
      try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {
         String line;
         for (int n = 1; (line = reader.readLine()) != null; ++n) {
            System.out.println(n + ": " + line);
         }
      } catch (FileNotFoundException e) {
         System.err.println("вказаний файл не знайдено.");
      }
         }
}
```

C# — це універсальна, багатопарадигмальна мова програмування. C# охоплює дисципліни програмування зі статичною типізацією, жорсткою типізацією, лексичною, імперативною, декларативною, функціональною, загальною, об'єктно-орієнтованою і компонентно-орієнтованою.

Мова програмування C# — була розроблена Андерсом Хейлсбергом з Microsoft у 2000 році і пізніше була затверджена як міжнародний стандарт Ecma (ECMA-334) у 2002 році та ISO/IEC (ISO/IEC 23270) у 2003 році. Microsoft представила C# разом із .NET Framework і Visual Studio, обидва продукти мали закритий код. На той час у Microsoft не було продуктів з відкритим кодом взагалі. Через чотири роки, у 2004 році, запустився безкоштовний проект з відкритим вихідним кодом під назвою Mono, який забезпечував кросплатформний компілятор і середовище виконання для мови програмування C#. Через десять років Microsoft випустила Visual Studio Code (редактор коду), Roslyn (компілятор) і уніфіковану платформу .NET (фреймворк програмного забезпечення), усі з яких підтримують C# і є безкоштовними, з відкритим вихідним кодом і підтримкою кросплатформенності. Проект Mono також приєднався до Microsoft, але не був об’єднаний з .NET.

Хоча визначення мови C# і CLI стандартизовані ISO та Ecma, і це забезпечує розумний та недискримінаційний ліцензійний захист (RAND) від патентних позовів, Microsoft використовує C# і CLI у своїй бібліотеці Base Class Library (BCL), яка є основою їхньої власної платформи .NET framework, і яка надає низку нестандартизованих класів (розширений I/O, GUI Windows Forms, вебслужби тощо).

В деяких випадках, де патенти Microsoft відносяться до стандартів, використаних у .NET framework, документовані Microsoft, і застосовані патенти доступні через інші RAND умови або через Обіцянку Відкритої Специфікації Microsoft (Microsoft's Open Specification Promise, OSP), які випускають патентні права публічно. Але є деякі застереження і обговорення про те, що існують додаткові аспекти, патентовані Microsoft, що не покриті патентами, які можуть утримувати незалежних реалізаторів певного фреймворку.

Microsoft погодився не позиватися проти розробників open source програмного забезпечення щодо порушення прав у неприбуткових проектах для частини свого фреймворку, покритого OSP. Microsoft погодився не позиватися з

приводу патентних суперечок що стосуються продуктів Novell проти платних клієнтів Novell за невеликим винятком продуктів, що явно згадують C#, .NET чи реалізацію .NET від Novell (проєкт Mono). Проте Novell дотримується точки зору, що Mono не порушує жодного патенту Microsoft. Microsoft також уклав спеціальну угоду, за умовами якої він не позивається проти браузерного плагіну Moonlight, який спирається на Mono, отриманного від Novell.

У зауваженні, що було опубліковано на новинному сайті Free Software Foundation у червні 2009 Річард Столлман попереджає, що він вважає, що «Microsoft можливо планує одного дня оголосити всі вільні реалізації C# такими, що використовують програмні патенти» і рекомендував розробникам уникати того, що він називає «безвідплатним ризиком», пов'язаним із «залежністю вільних реалізацій C#». Free Software Foundation згодом повторили свої попередження, стверджуючи, що розширення Microsoft Community Promise на специфікації ECMA C# і CLI можуть не вберігти від шкоди Microsoft відкритим реалізаціям C#, оскільки багато специфічних для Windows бібліотек, включених у .NET та Mono, не покриті даними домовленостями. Тому більшість провідних дистрибутивів Лінукс, за винятком Novell SUSE Linux, не включають Mono в інсталяцію за замовчуванням.

3.2 Вибір IDE для розробки

Для розробки було обрано IDE Xamarin, який вбудований у Microsoft Visual Studio.

Середовище програмування Microsoft Visual Studio 2019 — це один з найбільш відомих інструментів для програмування на різних мовах програмування таких як C#/C++/F# та інших. Microsoft Visual Studio являється найбільш використовуваним середовищем для програмуваня і його використовують майже у всіх великих компаніях.

Microsoft Visual Studio один з найпотужніших інструментів програмування з використанням мови програмування C# доступних для операційної системи

Windows. Microsoft Visual Studio дозволяє розробляти як програми з графічним інтерфейсом, так і консольні програми, а також з підтримкою технології Windows Forms також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows CE,.NET Framework,.NET Compact Framework та Microsoft Silverlight. Microsoft Visual Studio обєднує потужність висококласних інструментів програмування для мов високого рівня, таких як C++ або C# з простотою роботи та практично. Всі компоненти Microsoft Visual Studio якісно задокументовані та мають вбудовану систему підказок. Система інтиративної допомоги доступна на німецькій, англійській, французькій і російській мовах. Microsoft Visual Studio має ряд особливостей, що виділяють його серед конкуруючих систем. Функції автоматичного оголошення і форматування, адаптивний асистент введення максимально спрощують роботу. Всі команди мають можливість управління мишею і швидкого введення з клавіатури. Це робить роботу програміста комфортною та ефективною.

Вбудований компілятор генерує швидкий машинний код. Це забезпечує максимально високу швидкодію прикладних проектів. Сучасні інтелектуальні технології, включаючи «інкрементальний компілятор», дозволяють обробляти проекти, що містять тисячі змінних і сотні програмних компонентів вкрай швидко. Microsoft Visual Studio забезпечує розробника ПЗ набором високоефективних інструментів, включаючи повноцінну емуляцію ПЛК, відладку по крокам, точки зупинки, візуалізацію обєкта управління, трасування значень змінних та «гаряче» коригування коду.

Xamarin була заснована в травні 2011 Міґелем де Ікаса і Нетом Фрідманом після розподілу активів компанії Novell [16].

Станом на 2013 рік в компанії працювало близько 76 осіб. У лютому 2016 року куплена компанією Microsoft за неназваною ціною. У складі Microsoft робота Xamarin буде зосереджена на розвитку платформи для розробки мобільних застосунків на мові C# з використанням технологій .NET. Поєднання напрацювань

Xamarin з продуктами Visual Studio, Visual Studio Team Services і Azure утворює рішення, яке охопить всі аспекти, необхідні для розробки, тестування і поширення мобільних додатків для будь-яких категорій пристроїв, включаючи пристрої на платформах Android та iOS [17].

Xamarin стверджує, що єдиною IDE, яка дозволяє розробляти додатки для Android, iOS і Windows у Microsoft Visual Studio. Компанія Xamarin постачає додаткові модулі для Microsoft Visual Studio, що дозволяє розробникам створювати програми для Android, iOS та Windows у середовищі IDE за допомогою завершення коду та IntelliSense. Xamarin для Visual Studio також має одноіменне розширення у Microsoft Visual Studio, яке надає підтримку для створення, розгортання та налагодження програм на емуляторі або пристрої. Наприкінці 2013 року компанія Xamarin і Microsoft оголосили про партнерство, яке включало подальшу технічну інтеграцію та програми для клієнтів, щоб клієнти могли створювати спільні бази розробників для всіх мобільних платформ. Крім того, Xamarin тепер включає підтримку Microsoft Portable Class Libraries і більшість функцій C# 5.0, таких як async / await. Генеральний директор і співзасновник компанії Xamarin, Nat Friedman, оголосив альянс під час запуску Visual Studio 2013 у Нью-Йорку [18].

31 березня 2016 року Microsoft оголосила про те, що вони об'єднають все програмне забезпечення Xamarin з кожною версією Microsoft Visual Studio, включаючи Visual Studio Community, і це додасть різноманітні можливості Xamarin, які були попередньо встановлені в Visual Studio, наприклад, емулятор iOS [19].

Xamarin — зручний набір інструментів для розробки крос-платформних мобільних додатків на C# з використанням .NET. Він підтримує iOS, Android та Windows Phone.

Для розробки програми на основі Xamarin розробнику не потрібно досконально знати специфічні мови окремих платформ. Крім того, при роботі з

будь-якою платформою у роробника буде повний доступ до можливостей її пакета SDK і вбудованих механізмів створення інтерфейсів користувача [20].

Таким чином, Xamarin дозволяє створювати додатки, які майже не відрізняються від нативних аналогів, а отже, цілком підходять для розповсюдження через офіційні магазини (наприклад, Google Play та App Store).

Крім того, за словами розробників Xamarin, готове рішення не суттєво поступатиметься і в плані продуктивності.

Одна з основних причин, через які розробники уникають інструментів крос-платформної розробки, полягає в тому, що такі засоби не дозволяють користуватися всім спектром можливостей конкретних середовищ. В першу чергу це відноситься до дизайну (Flat Design в iOS, Material Design в Android) та інтелектуальних можливостей користувацьких пристроїв (доступ програми до контактів, камери, даних GPS тощо).

Кінцеве крос-платформне рішення не матиме високу продуктивність (ця ознака притаманна практично всім програмам, створеним за допомогою крос-платформних веб-інструментів) і не зможе користуватися перевагами конкретних платформ (а це означає, що деякі можливості, спочатку заплановані розробником проекту, не можуть бути реалізовані в повному обсязі).

Однак при використанні інструментів Xamarin для кросс-платформної розробки «рідних» програм все інакше. У цьому випадку розробник не зіштовхнеться з проблемами які описані вище. Розробникам доступні не тільки стандартні класи .NET: вони можуть легко підключити і класи, що підтримуються конкретною мобільною платформою (вони містяться в бібліотеках C# для Xamarin.Android і Xamarin.iOS відповідно). Це означає, що при використанні Xamarin для розробки програм під iOS та Android у вашому розпорядженні буде весь набір можливостей цих платформ. При цьому не потрібно використовувати сторонні (і зазвичай платні) інструменти.

3.3 Тестування програми

Тестування — це один з найважливіших етапів розробки, адже на ньому визначається кінцева якість продукту. Згідно з більшістю моделей життєвого циклу програмного забезпечення тестування продовжується разом з впровадженням ПЗ та закінчується лише з його завершенням підтримки.

Розробка через тестування — методологія розробки програмного забезпечення, яка ґрунтується на повторенні дуже коротких циклів розробки: спочатку пишеться тест, що покриває певну зміну, потім пишеться код, який дозволить пройти тест, і під кінець проводиться рефакторинг нового коду до відповідних стандартів. Кент Бек, який вважається винахідником цієї техніки, стверджував у 2003 році, що технологія через тестування заохочує простий дизайн і вселяє впевненість в проекті.

У 1999 року у своїй появі технологія через тестування була пов'язані з концепцією «спочатку тест», застосовуваної в екстремальному програмуванні, проте пізніше виділилася як самостійна методологія.

Тест — це процедура, яка дозволяє або підтвердити або спростувати коректність коду. Коли програміст перевіряє працездатність розробленого ним коду, він виконує тестування вручну або ж проводить модульне тестування.

Дослідження 2005 року показало, що використання розробки через тестування передбачає написання більшої кількості тестів, у свою чергу програмісти, що пишуть більше тестів, схильні бути більш продуктивними. Гіпотези, що пов'язують якість коду з TDD, були непереконливими.

Програмісти, які використовують TDD на нових проектах, відзначають, що вони рідше відчувають необхідність використовувати налагоджувач. Якщо деякі тести несподівано перестають проходити, відкат до останньої версії, яка проходить всі тести, може бути більш продуктивним, ніж відлагодження.

Розробка через тестування пропонує більше ніж просто перевірку коректності, вона також впливає на проектування програми. Спочатку

сфокусувавшись на тестах, простіше уявити, який функціонал необхідний користувачу. Отже, розробник продумує деталі інтерфейсу до реалізації. Тести змушують робити свій код більш пристосованим для тестування. Наприклад, відмовлятися від глобальних змінних, шаблонів одинакі (singleton), робити класи менш пов'язаними та легкими для використання. Сильно пов'язаний код або код, який вимагає складної ініціалізації, буде значно складніше протестувати. Модульне тестування сприяє формуванню чітких та невеликих інтерфейсів. Кожен клас виконуватиме певну роль, як правило, невелику. Як наслідок, взаємозв’язки між класами знижуватимуться, а зв'язковість підвищуватиметься. Контрактне програмування доповнює тестування, формуючи необхідні вимоги через затвердження (англ. assertions).

Незважаючи на те, що при розробці через тестування потрібно написати більше коду, загально часу, витраченого на розробку, зазвичай йде менше. Тести захищають від помилок. Тому час, що витрачається на налагодження, знижується багаторазово. Велика кількість тестів допомагає зменшити кількість помилок у коді. Усунення дефектів на більш ранньому етапі розробки, перешкоджає появі хронічних і “дорогих” помилок, які призводять до тривалого та виснажливого відлагодження в майбутньому.

Тести дозволяють робити рефакторинг коду без ризику його зіпсувати. При внесенні змін до добре протестованого коду ризик появи нових помилок значно нижчий. Якщо нова фіча призводить до помилок, тести, якщо вони, звичайно, є, одразу ж це покажуть. При роботі з кодом, на який немає тестів, помилку можна виявити через значний час, коли з кодом працювати буде набагато складніше. Добре протестований код легко переносить рефакторинг. Впевненість у тому, що зміни не порушать існуючу функціональність, надає впевненості розробникам та збільшує ефективність їхньої роботи. Якщо існуючий код добре покритий тестами, розробники будуть почуватися набагато вільнішими при внесенні архітектурних рішень, які покликані покращити дизайн коду.

Розробка через тестування сприяє більш модульному, гнучкому та розширюваному коду. Це пов'язано з тим, що при даній методології розробнику необхідно думати про програму як про безліч незначних модулів, написаних і протестованих незалежно і лише потім з'єднаних разом. Це призводить до менших, більш спеціалізованих класів, зменшення пов'язаності та чистіших інтерфейсів. Використання mock-об'єктів також робить внесок у модульність коду, оскільки вимагає наявності простого механізму для перемикання між mock- та звичайними класами.

Оскільки пишеться лише код, необхідний для проходження тесту, автоматизовані тести покривають всі шляхи виконання. Наприклад, перед додаванням нового умовного оператора розробник повинен написати тест, що перевіряє додавання цього умовного оператора. У результаті отримані в результаті розробки тестування тести досить повні: вони виявляють будь-які ненавмисні зміни поведінки коду.

Тести можуть використовуватись як документація. Хороший код розповість про те, як він працює, краще за будь-яку документацію. Адже документація та коментарі в коді можуть застаріти. Це може збивати з пантелику розробників, які вивчають код. Оскільки документація, на відміну від тестів, не може сказати, що вона застаріла, такі ситуації, коли документація не відповідає дійсності справді не рідкість.

Модульне тестування відноситься до тестів, які перевіряють функціональність певного куска коду, зазвичай на функціональному рівні. В об'єктно-орієнтованому середовищі, це, як правило, тестування на рівні класу чи збірки, а мінімальні модульні тести містять у собі конструктори та деструктори.

Такі типи тестів зазвичай пишуться розробниками під час роботи над кодом (стиль «білої скриньки»), щоб впевнитись, що дана функція працює так, як очікувалося. Одна функція може мати кілька тестів, щоб переглянути всі випадки використання коду. Модульне тестування саме по собі не може перевірити

функціонування частини ПЗ, а використовується щоб гарантувати, що основні блоки ПЗ працюють незалежно один від одного і працюють коректно.

Модульне тестування — це процес розробки ПЗ, що включає в себе синхронізовані застосування широкого спектра для виялення дефектів та для виявлення стратегій із метою зниження ризиків розробки ПЗ, часу та витрат. Воно виконується розробником ПЗ або інженером, під час фази розробки проекту життєвого циклу розробки ПЗ. Модульне тестування спрямоване на усунення помилок проектування. Ця стратегія спрямована на підвищення якості програмного забезпечення, до такого рівня, як вимагає процес контролю якості чи вище.

Залежно від очікуваної організації розробки ПЗ, модульне тестування може включати статичний аналіз коду, аналіз потоку даних аналізу метрик, експертні оцінки коду, аналізу покриття коду та інші методи перевірки ПЗ.

Модульне тестування зазвичай проводиться програмістами без залучення відділу якості. Це найперше тестування яке повинна пройти програма аби перейти до наступного етапу.

Покриття коду, по своїй суті, є тестуванням білого ящика. ПЗ що тестується збирається зі специфічними налаштуваннями або бібліотеками й/або запускається в особливому середовищі, в результаті чого для кожної використовуваної (виконуваної) функції програми визначається місцезнаходження даної функції у вихідному коді. Цей процес дозволяє розробникам та співробітникам з відділу контролю якості визначити частини системи, які, при нормальній роботі, використовуються дуже рідко або ніколи не використовуються (такі як код обробки помилок тощо). Це дозволяє зорієнтувати тестувальників на тестуванні найважливіших режимів та вразливих місць.

Приймальне тестування — це формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою визначення чи задовольняє система приймальним критеріям; винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні.

Приймальне тестування виконується на основі набору типових тестових випадків та сценаріїв, розроблених на основі вимог до даного ПП. Рішення про проведення приймального тестування приймається в тому разі, якщо продукт досяг необхідного рівня якості та замовник ознайомлений із Планом приймальних Робіт (Product Acceptance Plan) чи іншим документом, де описано набір дій, пов'язаних із проведенням приймального тестування, дата його проведення, відповідальні за тестування та інше.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або реліз програмного продукту.

Як правило, інструменти та бібліотеки, які використовуються для отримання покриття коду, вимагають суттєвих витрат продуктивності та/або пам'яті, недопустимих при нормальному функціонуванні ПЗ. Тому вони можуть використовуватися лише в лабораторних умовах.

Ручне тестування — це процес ручної перевірки програмного забезпечення на помилки.

Тестувальник відіграє роль користувача програми й використовує властивості програми для знаходження помилок у роботі додатка. Для професійного тестування тестувальник зазвичай користується написаним планом тестування з тесткейсами.

Як правило ручне тестування проводиться двома людьми. Перший розробляє тесткейси, а другий проводить саме тестування. Ролі тестерів змінюються з кожним проектом. Такий підхід є найефективніший оскільки з одного боку дозволяє виконувати однотипну роботу, що підвищує швидкість виконання, а з іншого дає поглянути на процес тестування з іншого боку та не втратити пов’язані навики.

Таблиця 3.1— Ручне тестування програми

№	Дія	Результат	Пройшло/не пройшло
1	Запустити програму	З’являється стартове вікно програми	Пройдено
2	Ввести логін та пароль та натиснути “Вхід”	Перехід до діалогового вікна пошуку інформації	Пройдено
3	Вибрати параметри пошуку та натиснути на кнопку “Пошук”	Появиться інформація про записи з необхідною інформацією	Пройдено
4	У вікні створенн звіту вибрати аналіз витрат, період, категорії витрат та рахунки	Поява звіту у вибраній формі	Пройдено
5	У вікні внесення доходу обрати рахунок, категорію та дату та натиснути на кнопку “+”	Інформація записується до файлу з даними	Пройдено
6	У вікні внесення витрат обрати рахунок, категорію та дату та натиснути на кнопку “записати”	Інформація записується до файлу з даними	Пройдено
7	У вікні внесення нової категорії внести назву категорії та натиснути кнопку «Зберегти»	Інформація записується до файлу з даними	Пройдено
8	У вікні внесення нової категорії витрам внести назву одиниці виміру та натиснути кнопку «Зберегти»	Інформація записується до файлу з даними	Пройдено

ВИСНОВКИ

Виконуючи дану роботу було розроблено програмне забезпечення для контролю фінансів фізичних осіб.

Також було розроблено класи програми та описано їх поля та методи, а також призначення в даному програмному середовищі та логіку взаємодії.

Було спроектовано та розроблено інтерфейс користувача з врахуванням користувацького досвіду та сценаріїв використання.

Розглянуто основні особливості мобільної розробки та її відмінності від розробки десктопних додатків. Оглянуто популярні способи локального збереження даних та обрано спосіб збереження у json файлі. Це пов’язано з його компактністю та простотою реалізації, а ще високою щільністю даних.

Також було проаналізованоі програми аналоги та виділено їх основні особливості. На їх основі було розроблено основні вимоги до продукту та розроблено основні сценарії використання.

Розглянуто основні мови програмування та обрано C# для реалізації даного проекту. Також було оглянуто середовища розробки та обрано Xamarin + MS Visual Studio.

Було проведено ручне тестування програми та виправлено всі знайдені баги.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Drayton, Peter; Albahari, Ben; Neward, Ted (2002). C# Language Pocket Reference. O'Reilly. ISBN 0-596-00429-X.

2. Petzold, Charles (2002). Programming Microsoft Windows with C#. Microsoft Press. ISBN 0-7356-1370-2.

3. Коматинэни С., Маклин Д., Хэшими С. Google Android: программирование для мобильных устройств = Pro Android 2. — 1-е изд. — СПб.: Питер, 2011. — 736 с. — ISBN 978-5-459-00530-1.

4. Сатия Коматинени, Дэйв Маклин. Android 4 для профессионалов. Создание приложений для планшетных компьютеров и смартфонов = Pro Android 4. — М.: Вильямс. — 880 с. — ISBN 978-5-8459-1801-7.

5. Роджерс Р., Ломбардо Д. Android. Разработка приложений. — М.: ЭКОМ Паблишерз, 2010. — 400 с. — ISBN 978-5-9790-0113-5.

6. Донн Фелкер. Android: разработка приложений для чайников = Android Application Development For Dummies. — М.: Диалектика, 2011. — 336 с. — ISBN 978-5-8459-1748-5.

7. Edelman, Jason; Lowe, Scott; Oswalt, Matt. Network Programmability and Automation. O'Reilly Media. "for data representation you can pick one of the following: YAML, YAMLEX, JSON, JSON5, HJSON, or even pure Python"

8. Роберт Тейбор. Реалізація XML Web-служб на платформі Microsoft .NET = Реализация XML Web-служб на платформе Microsoft .NET. — М. : «Вільямс», 2002. — С. 464. — ISBN 0-6723-2088-6.

9. Silberschatz, Abraham; Sudarshan, S. (2011). Database system concepts (вид. 6). New York: McGraw-Hill. ISBN 9780073523323. OCLC 436031093.

10. Janice J. Heiss (April 2013). The Advent of Kotlin: A Conversation with JetBrains' Andrey Breslav

11. Кей С. Хорстманн. Java SE 8. Вводный курс = Java SE 8 for the Really Impatient. — М.: «Вильямс», 2014. — 208 с. — ISBN 978-5-8459-1900-7.

12. Фрэд Лонг, Дхрув Мохиндра, Роберт С. Сикорд, Дин Ф. Сазерленд, Дэвид Свобода. Руководство для программиста на Java: 75 рекомендаций по написанию надежных и защищённых программ = Java Coding Guidelines: 75 Recommendations for Reliable and Secure Programs. — М.: «Вильямс», 2014. — 256 с. — ISBN 978-5-8459-1897-0.
13. Кей С. Хорстманн. Java. Библиотека профессионала, том 1. Основы. 10-е издание = Core Java. Volume I - Fundamentals (Tenth Edition). — М.: «Вильямс», 2017. — 864 с. — ISBN 978-5-8459-2084-3.
14. Кей С. Хорстманн. Java. Библиотека профессионала, том 2. Расширенные средства программирования. 10-е издание = Core Java. Volume II - Advanced Feature (Tenth Edition). — М.: «Вильямс», 2017. — 976 с. — ISBN 978-5-9909445-0-3.
15. Ed Snider. Mastering Xamarin.Forms - Third Edition, 2019 - ISBN 978-1839213380
16. Steven F. Daniel. Mastering Xamarin UI Development - Second Edition, 2018 - ISBN 978-1786462008
17. Касаткин, А. И. Профессиональное программирование на языке си. Управление ресурсами / А.И. Касаткин. - М.: Высшая школа, 2012. - 432 с.
18. Лотка, Рокфорд C# и CSLA .NET Framework. Разработка бизнес-объектов / Рокфорд Лотка. - М.: Вильямс, 2010. - 816 с.
19. Мак-Дональд, Мэтью Silverlight 5 с примерами на C# для профессионалов / Мэтью Мак-Дональд. - М.: Вильямс, 2013. - 848 с.
20. Марченко, А. Л. Основы программирования на C# 2.0 / А.Л. Марченко. - М.: Интернет-университет информационных технологий, Бином. Лаборатория знаний, 2011. - 552 с.

ДОДАТОК А

Технічне завдання

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ ВНТУ
д.т.н., проф.
_____ О. Д. Азаров
«20» квітня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської дипломної роботи
«Програмне забезпечення для контролю персональних фінансів фізичних осіб для мобільної платформи»
08–23.БДР.021.00.000 ПЗ

Науковий керівник ст.,вкл.
_____ Муращенко О.Г.
Студент групи 1КІ–18б
_____ Дідур І.В.

Вінниця 2022

1 Підстави для виконання бакалаврської дипломної роботи (БДР)наступні:

— актуальність розробки, яка полягає у необхідності вирішення проблеми обліку фінансових доходів та витрат;

— наказ про затвердження теми бакалаврської дипломної роботи.

2 Мета і призначення БДР наступні:

— метою БДР є розробка програмного забезпечення для контролю персональних фінансів фіз осіб для моб платформи, яке дозволить облікувати доходи та витрати та переглядати статистику про зміну персонального балансу;

— призначення розробки полягає у виконанні бакалаврської дипломної роботи із подальшим впровадженням та розвитком.

3 Вихідні дані для виконання БДР наступні:

— технічний опис програмного застосунку;

— мова програмування C#;

— віконний додаток;

— середовище розробки Xamarin+Microsoft Visual Studio.

4 Вимога до виконання БДР наступна:

— створення мобільного додатку;

— можливість входу по паролю;

— можливість створення фінансових міток;

— можливість створення записів про доходи;

— можливість створення записів про витрати;

— можливість створення нових гаманців;

— можливість створення звітів про зміну балансу та категоріях.

5 Етапи БДР та очікувані результати приведені в таблиці А.1.

Робота виконується за вісім етапів.

6 Матеріали, що подаються до захисту БДР, наступні:

пояснювальна записка БДР, графічні і ілюстративні матеріали, протокол попереднього захисту БДР на кафедрі, відгук наукового керівника, відгук опонента, протоколи складання державних екзаменів, анотації до БДР українською та іноземною мовами, нормоконтроль про відповідність оформлення ДР діючим вимогам.

Таблиця А.1 — Етапи виконання роботи

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Постановка задачі роботи	09.03.22	09.03.22	Вступ
2	Пошук матеріалів по технологіям розробки додатків під андроїд	10.03.22	25.03.22	Розділ 1
3	Структурне проектування додатку проектування опалення	26.03.22	28.03.22	Розділ 2
4	Обгрунтування та вибір засобів реалізації системи	29.03.22	04.04.22	Розділ 2
5	Розробка логіки взаємодії додатку	05.04.22	29.04.22	Розділ 3
6	Підготовка матеріалів та розробка алгоритму створення звітів	30.04.22	27.05.22	Розділ 3, Працююча система
7	Оформлення пояснювальної записки та ілюстративного матеріалу	28.05.22	06.06.22	Пояснювальна записка
8	Перевірка якості виконання бакалаврської роботи та усунення недоліків	07.06.22	07.06.22	Презентація

7 Порядок контролю виконання та захисту БДР:
виконання етапів графічної та розрахункової документації ДР контролюється науковим керівником згідно зі встановленими термінами. Захист ДР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 При оформлюванні БДР використовуються:
— ДСТУ 3008 : 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302 : 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

Технічне завдання до виконання отримав _____ Дідур І.В.

ДОДАТОК Б

Код програми

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Windows;
using Newtonsoft.Json;

namespace WalletsManager
{
    public class Core
    {
        public DataBase LoadDB()
        {
            string path = Environment.CurrentDirectory + "\\data";
            string json = "";
            string[] SB;
            try
            {
                SB = File.ReadAllLines(path);
                DataBase tmpDB = new DataBase();
                for (int i = 0; i < SB.Length; i++)
                    json += SB[i];
                string decryptJson = StringCipher.Decrypt(json, "5d#a)s4f%4");
                DataBase db = JsonConvert.DeserializeObject<DataBase>(decryptJson);
                if (!CheckDate(db))
```

```
            return null;
        return db;
    }
    catch (Exception ex)
    {
        DataBase db = DataBase.CreateDataBase();
        return db;
    }
}
public void SaveDB(DataBase db)
{
    string path = Environment.CurrentDirectory + "\\data";
    if (db.release <= DateTime.Now)
        db.release = DateTime.Now;
    else
        db.release = db.dead;
    string json = JsonConvert.SerializeObject(db, Formatting.Indented);
    string encryptJson = StringCipher.Encrypt(json, "5d#a)s4f%4");
    File.WriteAllText(path, encryptJson);
    CreateBackup(db);
}
public void SaveDB(DataBase db, string path)
{
    if (db.release <= DateTime.Now)
        db.release = DateTime.Now;
    else
        db.release = db.dead;
    string json = JsonConvert.SerializeObject(db, Formatting.Indented);
    string encryptJson = StringCipher.Encrypt(json, "5d#a)s4f%4");
```

```
            File.WriteAllText(path, encryptJson);
        }
        public bool CheckDate(DataBase db)
        {
            //db.dead = DateTime.Now.AddYears(1);
            //db.deadStr = "23.05.2022";
            DateTime today = DateTime.Now;

            /*if (today > db.dead)
            {
                new Core().SaveDB(db);
                MessageBox.Show("Термін роботи програми закінчився." +
Environment.NewLine +
                    "Звяжіться з EtelionMelgan@gmail.com для продовження підписки." +
Environment.NewLine +
                    "Якщо дата ще не " + db.deadStr + " це буде безкоштовно");
                return false;
            }
            if (today < db.release)
            {
                TimeSpan timespan;
                timespan = db.release.Date - today.Date;
                db.release -= timespan;
                db.dead -= timespan;
                new Core().SaveDB(db);
                return true;
            }
            if (today > db.release && today <= db.dead)
            {
```

```
            db.release = today;
            new Core().SaveDB(db);
            return true;
        }
        return false;*/
        return true;
    }
    public void CreateBackup(DataBase db)
    {
        if (db == null)
            return;
        String dir = Environment.CurrentDirectory+"\\Backups";
        if (!Directory.Exists(dir))
        {
            Directory.CreateDirectory(dir);
        }

        if (!File.Exists(dir + "\\data1"))
        {
            SaveDB(db, dir + "\\data1");
            return;
        }
        else if (!File.Exists(dir + "\\data2"))
        {
            SaveDB(db, dir + "\\data2");
            return;
        }
        else if (!File.Exists(dir + "\\data3"))
        {
```

```
            SaveDB(db, dir + "\\data3");
            return;
        }

        DateTime dt1 = File.GetLastWriteTime(dir + "\\data1");
        DateTime dt2 = File.GetLastWriteTime(dir + "\\data2");
        DateTime dt3 = File.GetLastWriteTime(dir + "\\data3");
        string older="";
        string younger="";

        if (dt1 > dt2 && dt1 > dt3)
            younger = dir + "\\data1";
        if (dt2 > dt1 && dt2 > dt3)
            younger = dir + "\\data2";
        if (dt3 > dt2 && dt3 > dt1)
            younger = dir + "\\data3";

        if (dt1 < dt2 && dt1 < dt3)
            older = dir + "\\data1";
        if (dt2 < dt1 && dt2 < dt3)
            older = dir + "\\data2";
        if (dt3 < dt2 && dt3 < dt1)
            older = dir + "\\data3";

        TimeSpan ts = File.GetLastWriteTime(younger) - DateTime.Now;
        if (Math.Abs(ts.TotalDays) >= 1)
            SaveDB(db, older);
    }
    public void AddFinancerecord(FinanceRecord fr, DataBase db)
```

```
        {
            if (db.financeRecords.Count == 0)
            {
                fr.id = 0;
                db.financeRecords.Add(fr);
            }
            fr.id = db.financeRecords[0].id + 1;
            for (int i = 0; i < db.financeRecords.Count; i++)
            {
                if (fr.date > db.financeRecords[i].date)
                {
                    db.financeRecords.Insert(i, fr);
                    break;
                }
            }
        }
    }
}
```

ДОДАТОК В

Код програми

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;

namespace WalletsManager
{
   public class DataBase
   {
      public string deadStr = "23.05.2024";
      public DateTime release { get; set; }
      public DateTime dead { get; set; }
      public List<Wallet> wallets { get; set; }
      public List<FinanceRecord> financeRecords { get; set; }
      public List<FinanceLabel> financeLabels { get; set; }
      public static DataBase CreateDataBase()
      {
         DataBase db = new DataBase();
         db.release = new DateTime(2020, 8, 9);
         db.dead = new DateTime(2024, 5, 23);
         db.wallets = new List<Wallet>();
         db.financeRecords = new List<FinanceRecord>();
         Wallet w = new Wallet() { id = 0, name = "Гаманець", currency="USD", description = "Гаманець по замовчуванню у USD", value = 0 };
         db.wallets.Add(w);
```

```
        return db;
    }
}
public class Wallet
{
    public int id { get; set; }
    public string name { get; set; }
    public double value { get; set; }
    public string currency { get; set; }
    public string description { get; set; }
    public string PIN { get; set; }
}
public class FinanceRecord
{
    public int id { get; set; }
    public double value { get; set; }
    public int idWallet { get; set; }
    public DateTime date { get; set; }
    public string description { get; set; }
    public List<int> financeLabels { get; set; }

}
public class FinanceLabel
{
    public int id { get; set; }
    public string name { get; set; }
    public System.Windows.Media.Color color { get; set; }
    public bool isActive { get; set; }
}}
```

ДОДАТОК Г

Код програми

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WalletsManager
{
   public class Helper
   {
      public static string LenRightAlligment(string str, int n)
      {
         if (str.Length > n)
         {
            return str;
         }
         else
         {
            int count = n - str.Length;
            for(int i=0;i<count;i++)
            {
               str = " " + str;
            }
            return str;
         }
      }
   }}
```

ДОДАТОК Д

Код програми

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace WalletsManager
{
   public class HelpFinanceRecordUI
   {
      public int id { get; set; }
      public string value { get; set; }
      public string nameWallet { get; set; }
      public String date { get; set; }
      public String time { get; set; }
      public string description { get; set; }
      public HelpFinanceRecordUI( FinanceRecord fr, DataBase db)
      {
         id = fr.id;
         date = fr.date.ToShortDateString();
         time = fr.date.ToShortTimeString();
         try
         {
            nameWallet = db.wallets.Where(w => w.id == fr.idWallet).First().name;
            value = fr.value.ToString("F2") + " " + db.wallets.Where(w => w.id == fr.idWallet).First().currency;
         }
```

```
            catch
            {
                nameWallet = "Не знайдено";
                value = fr.value.ToString("F2")+" ???";
            }
            description = fr.description;
        }
    }
    public class HelpSummary
    {
        public string date { get; set; }
        public string change { get; set; }
        public string[] value { get; set; }
        public string description { get; set; }
        public HelpSummary(int n)
        {
            value = new String[n];
        }
    }
}
```

ДОДАТОК Е

Код програми

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WalletsManager
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>

    public partial class MainWindow : Window
    {
        DataBase db;
        Core core = new Core();
        public MainWindow()
        {
```

```
            InitializeComponent();
            ToolTip tt = new ToolTip();
            tt.Content = "Created by Etelion" + Environment.NewLine + "Date: 26.05.2022"
+ Environment.NewLine + "Email: EtelionMelgan@gmail.com";
            LogoDev.ToolTip = tt;



        }

        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            WindowState = WindowState.Maximized;
            db = core.LoadDB();

            // зміна порядку гаманців
            /*Wallet w = db.wallets[8];
            db.wallets[8] = db.wallets[6];
            db.wallets[6] = w;
            w = db.wallets[9];
            db.wallets[9] = db.wallets[7];
            db.wallets[7] = w;
            */
            //core.SaveDB(db);

            /*
            for (int i = 0; i < db.financeRecords.Count; i++)
            {
                db.financeRecords[i].id = i;
            }
```

```
    */
    //core.SaveDB(db);
    ToFormW();


    /*          db.financeRecords[0].financeLabels = new List<int>();
                db.financeRecords[0].financeLabels.Add(1);
                db.financeRecords[0].financeLabels.Add(3);*/
}

public void ToFormW()
{
    DataGridWallets.ItemsSource = null;
    DataGridWallets.ItemsSource = db.wallets;
}

private void AddWallet_Click(object sender, RoutedEventArgs e)
{
    WindowWalletUpd windowWalletUpd = new WindowWalletUpd();
    windowWalletUpd.db = db;
    windowWalletUpd.isNew = true;
    windowWalletUpd.ShowDialog();
    ToFormW();
    core.SaveDB(db);
}

private void UpdateWallet_Click(object sender, RoutedEventArgs e)
{
    try
```

```
        {
            WindowWalletUpd windowWalletUpd = new WindowWalletUpd();
            windowWalletUpd.db = db;
            windowWalletUpd.isNew = false;
            int index = DataGridWallets.SelectedIndex;
            try
            {
                int val = Convert.ToInt32(DataGridHelper.GetCell(DataGridWallets, index, 0).ToString().Substring(38));
                windowWalletUpd.wallet = db.wallets.Where(w => w.id == val).ToList()[0];
            }
            catch
            { }
            if(windowWalletUpd.wallet!= null)
                windowWalletUpd.ShowDialog();
            ToFormW();
            core.SaveDB(db);
        }
        catch { }
    }

    private void DeleteWallet_Click(object sender, RoutedEventArgs e)
    {
        if (MessageBox.Show("Видалити гаманець? Це призведе до видалення всіх пов\'язаних фінансових операцій", "Видалити", MessageBoxButton.YesNo, MessageBoxImage.Warning) == MessageBoxResult.Yes)
        {
            int index = DataGridWallets.SelectedIndex;
```

```
            try
            {
                int val = Convert.ToInt32(DataGridHelper.GetCell(DataGridWallets,
index, 0).ToString().Substring(38));
                Wallet wallet = db.wallets.Where(w => w.id == val).ToList()[0];

                db.wallets.Remove(wallet);
                for (int i = 0; i < db.financeRecords.Count; i++)
                {
                    if (db.financeRecords[i].idWallet == val)
                    {
                        db.financeRecords.RemoveAt(i);
                        i--;
                    }
                }
            }
            catch
            {
                MessageBox.Show("Оберіть гаманець в таблиці гаманців");
            }
            ToFormW();
            core.SaveDB(db);
        }
        else
        {

        }
    }
```

```
private void FinanceRecords_Click(object sender, RoutedEventArgs e)
{
    WindowFinanceRecords wfr = new WindowFinanceRecords();
    wfr.db = db;
    wfr.ShowDialog();
    ToFormW();
}

private void AddFROut_Click(object sender, RoutedEventArgs e)
{
    WindowFinanceRecord wfr = new WindowFinanceRecord();
    wfr.db = db;
    wfr.isNew = true;
    wfr.inOut = false;
    wfr.ShowDialog();
    core.SaveDB(db);
    ToFormW();
}

private void AddFR_Click(object sender, RoutedEventArgs e)
{
    WindowFinanceRecord wfr = new WindowFinanceRecord();
    wfr.db = db;
    wfr.isNew = true;
    wfr.inOut = true;
    wfr.ShowDialog();
    core.SaveDB(db);
    ToFormW();
}
```

```
private void Summary_Click(object sender, RoutedEventArgs e)
{
    WindowSummary ws = new WindowSummary();
    ws.db = db;
    ws.Show();
}

private void Statictic_Click(object sender, RoutedEventArgs e)
{
    WindowStatistic ws = new WindowStatistic();
    ws.db = db;
    if (db.financeRecords.Count > 0)
        ws.Show();
}

private void TransferBtwWallets_Click(object sender, RoutedEventArgs e)
{
    WindowTransferBtwWallets wtw = new WindowTransferBtwWallets();
    wtw.db = db;
    wtw.ShowDialog();
    ToFormW();
    core.SaveDB(db);
}

private void FinanceLabels_Click(object sender, RoutedEventArgs e)
{
    WindowFinanceLabels wfl = new WindowFinanceLabels();
    wfl.db = db;
```

```
            wfl.ShowDialog();
            core.SaveDB(db);
        }

        private void DataGridWallets_MouseDoubleClick(object sender,
MouseButtonEventArgs e)
        {
            try
            {
                WindowWalletUpd windowWalletUpd = new WindowWalletUpd();
                windowWalletUpd.db = db;
                windowWalletUpd.isNew = false;
                int index = DataGridWallets.SelectedIndex;
                try
                {
                    int val = Convert.ToInt32(DataGridHelper.GetCell(DataGridWallets,
index, 0).ToString().Substring(38));
                    windowWalletUpd.wallet = db.wallets.Where(w => w.id ==
val).ToList()[0];
                }
                catch
                { }
                windowWalletUpd.ShowDialog();
                ToFormW();
                core.SaveDB(db);
            }
            catch { }
        }
    }}
```

ДОДАТОК Ж

Код програми

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace WalletsManager
{
    /// <summary>
    /// Interaction logic for WindowTransferBtwWallets.xaml
    /// </summary>
    public partial class WindowTransferBtwWallets : Window
    {
        public DataBase db;
        ObservableCollection<string> wallets = new ObservableCollection<string>();
        bool isLoaded = false;
        public WindowTransferBtwWallets()
```

```
        {
            InitializeComponent();
        }
        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            for (int i = 0; i < db.wallets.Count; i++)
                wallets.Add(db.wallets[i].name);

            WalletFrom.ItemsSource = wallets;
            WalletTo.ItemsSource = wallets;

            WalletFrom.SelectedIndex = 0;
            WalletTo.SelectedIndex = 0;

            isLoaded = true;

            Date.Value = DateTime.Now;

            Description.Text = "Переказ коштів";

        }
        private void btn1_Click(object sender, RoutedEventArgs e)
        {
            if (WalletFrom.SelectedIndex == WalletTo.SelectedIndex)
            {
                MessageBox.Show("Переказ між одним гаманцем", "Помилка",
MessageBoxButton.OK, MessageBoxImage.Error);
                return;
            }
```

```
        double valueFrom;
        double valueTo;

        valueFrom = -Convert.ToDouble(ValueFrom.Text);
        valueTo = Convert.ToDouble(ValueTo.Text);
        if (valueFrom == 0 || valueTo == 0)
        {
            MessageBox.Show("Нульовий переказ", "Увага", MessageBoxButton.OK,
MessageBoxImage.Warning);
            return;
        }

        FinanceRecord frFrom = new FinanceRecord();
        FinanceRecord frTo = new FinanceRecord();

        frFrom.value = valueFrom;
        frTo.value = valueTo;
        frTo.date = frFrom.date = (DateTime)Date.Value;

        frTo.description = frFrom.description = Description.Text;

        try
        {
            frFrom.idWallet = db.wallets.Where(w => w.name ==
wallets[WalletFrom.SelectedIndex]).First().id;
            db.wallets.Where(w => w.id == frFrom.idWallet).First().value +=
frFrom.value;
            frTo.idWallet = db.wallets.Where(w => w.name ==
wallets[WalletTo.SelectedIndex]).First().id;
```

```
            db.wallets.Where(w => w.id == frTo.idWallet).First().value += frTo.value;
        }
        catch
        {
            frFrom.idWallet = 0;
            frTo.idWallet = 0;
            MessageBox.Show("Щось не так");
            return;
        }
        new Core().AddFinancerecord(frTo, db);
        new Core().AddFinancerecord(frFrom, db);
        Close();
    }

    private void WalletFrom_SelectionChanged(object sender,
SelectionChangedEventArgs e)
    {
        if (isLoaded)
        {
            Description.Text = "Переказ коштів між " +
                db.wallets.Where(w => w.name ==
wallets[WalletFrom.SelectedIndex]).ToList()[0].name +
                " та " +
                db.wallets.Where(w => w.name ==
wallets[WalletTo.SelectedIndex]).ToList()[0].name;
        }
    }
```

```
        private void WalletTo_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            if (isLoaded)
            {
                Description.Text = "Переказ коштів між " +
                    db.wallets.Where(w => w.name ==
wallets[WalletFrom.SelectedIndex]).ToList()[0].name +
                    " та " +
                    db.wallets.Where(w => w.name ==
wallets[WalletTo.SelectedIndex]).ToList()[0].name;
            }
        }
    }
}
```

ДОДАТОК И

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ
ЗАПОЗИЧЕНЬ

Назва роботи: «Програмне забезпечення для контролю персональних фінансів фізичних осіб для мобільної платформи»

Тип роботи: бакалаврська дипломна робота

Підрозділ кафедра обчислювальної техніки

Показники звіту подібності Unicheck
Оригінальність 81,7% Схожість 18,3%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Дідур І.В.
Керівник роботи _____ Муращенко О.Г.