МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА МОБІЛЬНОГО ДОДАТКА ДЛЯ ВІДСТЕЖЕННЯ ФІНАНСОВИХ ВИТРАТ КОРИСТУВАЧА»**

Виконав: студент 4 курсу, групи 6.1219-1пі

спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

освітньої програми програмна інженерія
(назва освітньої програми)

І.В. Шведов
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Мухін В.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

« 07 » 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Шведову Іллі Владиславовичу

(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка мобільного додатка для відстеження фінансових витрат користувача

керівник роботи Мухін Віталій Вікторович, к.т.н., доцент

(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Основні теоретичні відомості.
3. Розробка мобільного додатка для відстеження фінансових витрат користувача.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
презентація за темою докладу

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1			
2			
3			

7. Дата видачі завдання 07.02.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.02.2023	
2.	Збір вихідних даних.	22.02.2023	
3.	Обробка методичних та теоретичних	14.03.2023	
	джерел.		
4.	Розробка першого розділу.	29.03.2023	
5.	Розробка другого розділу.	20.04.2023	
6.	Розробка третього розділу.	15.05.2023	
7.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи.		
8.	Захист кваліфікаційної роботи.	22.06.2023	

Студент _____ І.В. Шведов
(підпис) (ініціали та прізвище)

Керівник роботи _____ В.В. Мухін
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка мобільного додатка для відстеження фінансових витрат користувача»: 55 с., 21 рис., 18 джерел, 3 додатки.

АНДРОЇД, АНДРОЇД СТУДІО, ДАРТ, МОБІЛЬНИЙ ДОДАТОК, СОЦІАЛЬНИЙ ПРОЄКТ, ФАЕРБЕЙС.

Об’єкт дослідження: основні методи та засоби розробки мобільних застосунків з використанням фреймворку Flutter.

Мета роботи: розробити мобільний додаток для відстеження фінансових витрат користувача за допомогою Flutter.

Метод дослідження: аналітичний, порівняльний.

У кваліфікаційній роботі розглянуто проєктування програмного застосунку для відстеження фінансових витрат на ОС Android та ОС IOS, спроєктовано його інтерфейс. Розглянуто основні поняття «прикладний програмний інтерфейс», «інтегроване середовище розробки». Розглянуто існуючі аналоги Android-застосунків для відстеження фінансових витрат. На основі цих застосунків було визначено можливості розширення функціоналу. Розроблений застосунок є одним з найбільш інформативних у своїй сфері.

SUMMARY

Bachelor's qualification work «Development of a Mobile Application for Tracking the User's Financial Expenses»: 55 pages, 21 figures, 18 references, 3 supplements.

ANDROID, ANDROID STUDIO, DART, MOBILE APP, SOCIAL PROJECT, FIREBASE.

Research object is basic methods and tools for developing mobile applications using the Flutter framework.

The object of the study is develop a mobile application for tracking the user's financial expenses using Flutter.

The methods of research are analytical, comparative.

In the qualification work, the design of a software application for tracking financial expenses on the Android OS and OC IOS was considered, and its interface was designed. The basic concepts of "application programming interface" and "integrated development environment" are considered. Existing analogues of Android applications for tracking financial expenses are considered. On the basis of these applications, the possibilities of expanding the functionality were determined. The developed application is one of the most informative in its field.

ЗМІСТ

ВСТУП

Мобільні телефони увійшли в усі сфери життя, багато людей практично не можуть без них обходитися, це призвело до появи фундаментально нової платформи для розробки застосунків, відмінної від класичних платформ персональних комп'ютерів. Телефон з камерою, сенсорним екраном, навігатором і датчиками положення та інші став новим засобом сприйняття навколишнього світу [1].

Сучасний світ має дві популярні платформи для розробки мобільних застосунків – Android та IOS. У середньому користувач платформи Android встановлює і використовує близько 40 різних програм. Користувач IOS використовує приблизно таку ж кількість застосунків . З цього можно зробити висновок, про зміну способу взаємодії людей зі своїми телефонами.

У наш час вже давно є сенс перейти на використання електронних ресурсів для допомоги життя, тому доцільно розробити мобільний застосунок для відстежування трат користувача, оскільки мобільний телефон постійно знаходиться поруч з користувачем.

З цього можна виділити такі переваги:

- кількість користувачів допомагає масштабувати додаток;
- доступність – не потрібно думати де записати трату;
- актуальність інформації – трати синхронізуються між девайсами.

Мета дипломної роботи: розробка програмного застосунку для відстежування трат користувача зі зручним інтерфейсом, що дозволяє створити транзакцію та переглядати минулі транзакції, та містить синхронізацію.

Для досягнення мети були поставлені наступні завдання:

- зробити огляд подібних застосунків для відстежування витрат користувача;
- розробити дизайн та архітектуру застосунку;

– створити програмний застосунок.

Об’єкт дослідження – основні методи та засоби розробки застосунків за допомогою крос-платформного фреймворку Flutter.

Предмет дослідження – мобільній застосунок для відстежування трат користувача.

Дипломна робота складається зі вступу, трьох розділів, висновків, переліку посилань та додатків.

У першому розділі розглядаються існуючі застосунки для відстежування трат користувача, обґрунтовується вибір середовища програмування, фреймворку. У другому розділі представлено технічне завдання та діаграми проєктування. В межах третього розділу було описано розроблені класи та модулі, приведена методика роботи з програмою.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГИ

1.1 Загальні терміни

AS – Android Studio.

IDE – Інтегроване середовище розробки (англ. Integrated Development Environment).

ПЗ – Програмне забезпечення.

ОС – Операційна система.

API – Application Programming Interface.

1.2 Постановка задачі

У наш час користувачі роблять багато транзакцій зі своїми рахунками, особливо якщо користувачі мають власне підприємство або використовують мобільний телефон для оплати у різних місцях. Тому виникає необхідність розробки мобільного застосунку для допомоги відстежування, що дозволяє переглядати історію транзакцій у зручному форматі.

1.3 Огляд існуючих програмних застосунків

На сьогоднішній день існує кілька застосунків з подібною метою, доступних для мобільних пристроїв. Розглянемо такі застосунки, як Monefy: Money Tracker [2], CoinKeeper: budget planner [3].

1.3.1 Monefy: Money Tracker

Monefy: Money Tracker [2] – міжнародний онлайн додаток, який дає змогу створювати транзакції, акаунти та продивлятися історію транзакцій, які були додані користувачем у минулому. Система може генерувати великі дані та надавати можливість бухгалтерам та податковим працювати з ними. Доступний лише як мобільний додаток, відсутні аналоги в веб (див. рис. 1.1).

Рисунок 1.1 – Стартовий екран додатку Monefy: Money Tracker [2]

На стартовому екрані, вгорі, відображається кнопка меню та кнопка налаштувань. Нижче міститься інформація о транзакціях за поточний місяць у вигляді діаграми. Ще нижче розташувався поточний баланс користувача.

У головному блоці міститься кнопки для перенаправлення на екрани створення транзакцій є можливість переглядати транзакції за категоріями.

Цей застосунок також допомагає отримувати файл з усіма вашими транзакціями. Ви можете створити профіль та налаштувати синхронізацію

даних між девайсами.

Основні переваги:

– дані зберігаються тільки локальні та не передаються на інші сервіси;
– можливість отримувати файл з транзакціями.

Основні недоліки:

– доступний лише як мобільний застосунок;
– додаток не вистачає синхронізації з банками.

1.3.2 CoinKeeper: budget planner

Мобільний застосунок націлений на ринок США. Допомагає відстежувати та аналізувати витрати користувача. Застосунок має гарний інтерфейс та в ньому гарно реалізовано відстежування фінансового потоку користувача, також застосунок дозволяє встановлювати фінансові цілі.

На головному екрані зображена інформація про доступні акаунти, категорії, та звітність. Цей додаток має гарний дизайн. Але в цьому застосунку також відсутня можливість підключити банк для синхронізації транзакцій. В цьому застосунку є можливість створювати транзакції та відстеж витрати за окремими аккаунтами (див. рис. 1.2).

Основні переваги:

– гарний дизайн аналітичних елементів;
– підтримка усіх відомих валют.

Основні недоліки:

– доступний лише як мобільний застосунок;
– немає можливості під'єднання банка для синхронізації;
– застосунок спеціалізується на аналітиці витрат користувача.

На основі проведеного огляду існуючих програмних застосунків можна зробити висновок, що розробка мобільного застосунку для відстежування витрат користувачем є актуальним завданням.

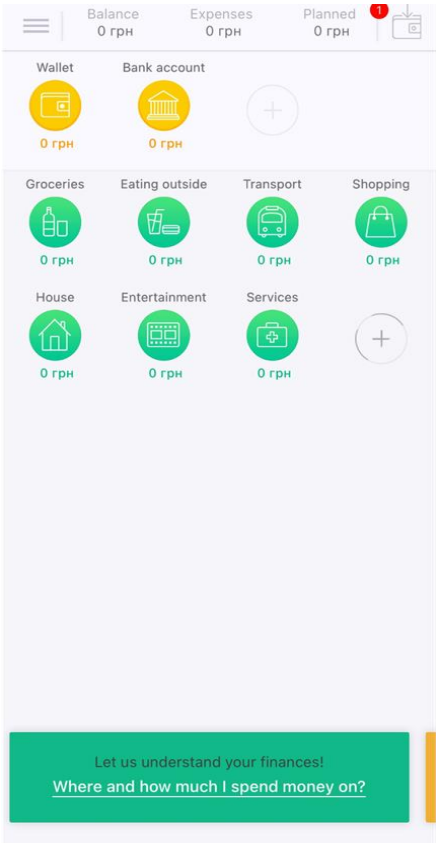

Рисунок 1.2 – Стартовий екран додатку CoinKeeper: Budget Planner [3]

1.4 Опис засобів розробки

1.4.1 Вибір мови програмування

Для розробки програмного застосунку була обрана об'єктно-орієнтована мова програмування Dart [4].

Dart – мова програмування, яку розробляє компанія Google, позиціонуючи як мову структурованого програмування для веб-застосунків. Розробники вважали, що в довгостроковій перспективі Dart може стати прогресивною заміною JavaScript, котрий потерпає від наявних в даний час проблем з розширюваністю, продуктивністю і підтримкою розробки складних застосунків. Мова має схожий на Java синтаксис, не вимагає явного визначення типів і її можна використовувати для створення серверних та клієнтських застосунків [4].

У березні 2015 компанія Google представила оновлену стратегію просування Dart, у котрій вирішено не прив'язувати Dart до браузеру і відмовитися від ідеї інтеграції віртуальної машини Dart у Chrome. Розробку буде зосереджено на застосуванні Dart як проміжної мови, скомпільованої в JavaScript. Розвиток Dart як окремої мови, альтернативної JavaScript і безпосередньо підтримуваної у браузерах, визнано недоцільним. Замість цього Dart рухатиметься у бік якісної інтеграції з JavaScript і генерації оптимального JavaScript-коду. При цьому розробку віртуальної машини Dart VM буде продовжено, але вона позиціонуватиметься в основному для створення серверних і мобільних застосунків [4].

Мова має схожий на Java синтаксис, не вимагає явного визначення типів і може використовуватися для створення серверних і клієнтських застосунків. Для запуску всередині браузера код мовою Dart може бути перетворений в JavaScript-подання або запущений безпосередньо під управлінням спеціального JavaScript-інтерпретатора Dartboard. Підтримується вбудовування коду мовою Dart в HTML-сторінки, використовуючи MIME тип "application/dart". На стороні сервера застосунок на мові Dart може бути виконаний всередині спеціальної віртуальної машини, яка забезпечує продуктивність виконання близьку до компільованих в машинний код мов. Віртуальну машину Dart планують інтегрувати в майбутні версії браузера Chrome, що дозволить виконувати застосунки мовою Dart без компіляції в JavaScript [4].

Мова підходить як для розробки одним програмістом невеликих скриптів без жорсткої структури, так і для створення високо маштабованих великих модульних проєктів, підтримуваних великим колективом з потребою більш явної типізації для того, щоб уникнути плутанини і помилок. При цьому явне завдання типів не обов'язкове, наприклад, можна почати розробку без вказання типів, а надалі при необхідності додати їх (наприклад, спочатку написати "var x", а потім замінити на "num x"). Код Dart завжди виконується тільки в рамках одного потоку, для організації паралельного виконання

пропонується використовувати класи з атрибутом isolate. У кожному скрипті використовується власний простір імен, для використання зовнішніх об'єктів, функцій або змінних слід їх явно імпортувати за допомогою конструкції "import". Всі змінні, початково, діють тільки в межах поточного скрипта і не експортуються глобально [4].

Особливості мови Dart:

- звичний і простий для вивчення синтаксис, природний для програмістів на JavaScript, C і Java [4];
- забезпечення швидкого запуску і високої продуктивності для всіх сучасних веб-браузерів і різних типів оточень, від портативних пристроїв до потужних серверів [4];
- можливість визначення класів і інтерфейсів, що дозволяють використовувати інкапсуляцію і повторно використовувати існуючі методи і дані [4];
- необов'язкове вказування типів, використовувати чи ні статичні типи вирішує розробник. Вказування типів дозволяє спростити відлагодження і виявлення помилок, робить код яснішим і читаним, спрощує його доопрацювання та аналіз сторонніми розробниками [4];
- підтримка різноманітних типів: різні види хешів, масивів і списків, черги, числові і рядкові типи, типи для визначення дати і часу, регулярні вирази (RegExp). Можливо створення своїх типів [4];
- організацію паралельного виконання пропонується реалізовувати з використанням класів з атрибутом isolate, код яких виконується повністю в ізольованому просторі в окремій області пам'яті, взаємодіючи з основним процесом через відправку повідомлень [4];
- підтримка використання бібліотек, що спрощують підтримку і відлагодження великих веб-проєктів. Сторонні реалізації функцій можуть підключатися у вигляді поділюваних бібліотек. Застосунки можна розбити на частини і доручити розробку кожної з частин

окремій команді програмістів [4];

- набір готових інструментів для підтримки розробки мовою Dart, включаючи реалізацію засобів динамічної розробки та відлагодження з виправленням коду на льоту ("edit-and-continue") [4];
- можливість створювати однорідні системи, що охоплюють як клієнтську, так і серверну частину. Використання однієї мови та інструментарію для клієнтських і серверних компонентів спрощує процес кодування і позбавляє від постійної зміни контексту [4].

1.4.2 Flutter

Для розробки програмного застосунку був обраний фреймворк Flutter, це програмний каркас із відкритим кодом, для створення застосунків для платформ Android та iOS, розроблений компанією Google. Він є основним способом створення застосунків для Google Fuchsia [5].

Flutter складається з:

- flutter рушій – програмний рушій для рендерингу, написаний в основному на C++ з використанням графічної бібліотеки Google Skia. Він також використовує SDK платформ Android або iOS [5];
- базової бібліотеки (Foundation library) – бібліотека складається з класів та функцій (написані на Dart), які використовують для побудови Flutter програм, для взаємодії із Flutter рушій [5];
- віджетів – дизайн інтерфейсу користувача у Flutter будують з віджетів. Віджет у Flutter являє собою незмінний об'єкт, який описує частину інтерфейсу користувача. Вся графіка, текст, фігури та анімації створюють за допомогою віджетів. Складні віджети створюють шляхом об'єднання простих [5].

На поточний час Flutter містить два набори віджетів, які відповідають

відповідним принципами побудови:

– віджети Material Design використовують дизайн Google;
– віджети Cupertino імітують дизайн Apple iOS.

Архітектура Flutter відрізняється від інших програмних каркасів (React, Apache Cordova) тим, що він не використовує для побудови інтерфейсу мови HTML, CSS та Javascript, відповідно і вбудований рушій WebView. Використовується власний рушій для рендерингу [5].

Flutter використовує тільки одну мову програмування Dart [4].

Цей фреймворк призначений для розробників, які хочуть швидше створити прекрасні мобільні застосунки, або спосіб за допомогою однієї інвестиції досягти більшої кількості користувачів. Flutter також призначений для інженерних менеджерів, яким потрібно очолити мобільні команди з розробки. Flutter дозволяє менеджерам з англомовних програм створити єдину команду розробників мобільних застосунків, об'єднавши інвестиції в розвиток, щоб швидше поставляти більше функцій, одночасно відправляти ту саму функцію, що встановлена для iOS та Android, і зменшувати витрати на обслуговування. Хоча це не початкова цільова аудиторія, Flutter також призначений для дизайнерів, які хочуть, щоб їхні оригінальні дизайнерські бачення послідовно, з високою вірністю пропонувались усім користувачам мобільних пристроїв. По суті, Flutter призначений для користувачів, які хочуть красивих додатків із чудовим рухом та анімацією, а також користувацькі інтерфейси з персонажем та особистістю [5].

Завдяки йому розробникам легко досягти постійних 60 кадрів в секунду. Програми Flutter запускаються за допомогою власне складеного коду. Це означає, що програми Flutter запускаються швидко [5].

Гаряче перезавантаження допомагає розробнику для швидкого бачення коду. На пристрої чи емуляторі/симуляторі ви можете очікувати час повторного завантаження на секунду. Спеціальне завантаження Flutter є надзвичайним, що означає, що стан застосунку зберігається після перезавантаження. Це означає, що ви можете швидко повторювати екран,

глибоко вкладений у застосунок, не починаючи з головного екрану після кожного перезавантаження [6].

Функція гарячого перезавантаження працює за допомогою введення оновлених файлів вихідного коду в запущений Dart VM (Virtual Machine). Це включає не тільки додавання нових класів, а й додавання методів та полів до існуючих класів та зміну існуючих функцій [6,7].

Кілька типів змін коду неможливо перезавантажити:

- глобальні змінні ініціалізатори;
- ініціалізатори статичного поля;
- основний метод програми;
- код Flutter компілюється для iOS та Android.

Операційні системи для мобільних пристроїв: Android Jelly Bean, v16, 4.1.x або новішої версії, і iOS 8 або новіші.

Flutter добре працює на планшетах. Наразі не реалізовані всі адаптації для планшетів, рекомендовані Material Design, хоча розробники планують подальші інвестиції в цю сферу. Можна перепакувати існуючий код Flutter для роботи в веб-перегляді, але є деякі застереження. Також, можливо використовувати Flutter для створення настільних застосунків.

1.4.3 Особливості Visual Studio Code

Visual Studio Code – засіб для створення, редагування та відлагодження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux та OS X [4].

Компанія Microsoft представила Visual Studio Code у квітні 2015 на конференції Build 2015. Це середовище розробки стало першим кросплатформним продуктом у лінійці Visual Studio [4].

За основу для Visual Studio Code використовуються напрацювання

вільного проєкту Atom, що розвивається компанією GitHub. Зокрема, Visual Studio Code є надбудовою над Atom Shell, що використовують браузерний рушій Chromium і Node.js. Примітно, що про використання напрацювань вільного проєкту Atom на сайті Visual Studio Code і в прес-релізі і в офіційному блозі не згадується [8].

Продукт підтримує розробку для платформ ASP.NET і Node.js, і позиціонується як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки.

Технічними характеристиками Visual Studio Code є:

- IntelliSense – виходять за межі підсвічування синтаксису та автозаповнення з IntelliSense, який забезпечує розумні доповнення на основі змінних типів, значень функцій та імпортних модулів;
- робота з Git та іншими постачальниками SCM ніколи не була простішою.

Особливості Visual Studio Code для Flutter:

- підсвічування синтаксису;
- доповнення коду на основі аналізу багатого типу;
- навігація до введення оголошень та пошуку типів використання;
- перегляд усіх поточних проблем з вихідним кодом.

1.4.4 Особливості Firebase

Firebase – це платформи розробки мобільних та веб- застосунків. Firebase розвивається з 2011 року компанією Firebase Inc., яку придбав Google у 2014 [9].

Він має Служби і рішення для розробки. Розглянемо їх нижче.

Firebase Analytics – безкоштовне рішення для оцінки застосунків, яке дає змогу ознайомитись із використанням застосунків та залученням користувачів [9].

Firebase Auth – це служба, яка може аутентифікувати користувачів, використовуючи лише код на стороні клієнта. Він підтримує соціальні логін-провайдери Facebook, GitHub, Twitter і Google (і Google Play Games). Крім того, вона включає в себе систему управління користувачами, за допомогою якої розробники можуть увімкнути автентифікацію користувача за допомогою входу з електронної пошти та пароля, що зберігаються в Firebase [9].

Firebase Cloud Messaging – раніше відомий як Google Cloud Messaging (GCM), Firebase Cloud Messaging (FCM) — це крос-платформове рішення для повідомлень і нотифікацій для Android, iOS та веб-застосунків, які наразі можна використовувати безкоштовно [10].

Firebase Storage – забезпечує надійне завантаження та вивантаження файлів для застосунків Firebase незалежно від якості мережі. Розробник може використовувати його для зберігання зображень, аудіо-, відео- чи іншого вмісту, створеного користувачами. Зберігання Firebase підтримується Google Cloud Storage [11].

1.5 Висновки до першого розділу

В даному розділі було виконано огляд існуючих програмних застосунків, наведено опис засобів розробки. Для розробки мобільного застосунку була обрана мова програмування Dart та фреймворк Flutter.

2 ПРОЄКТУВАННЯ ЗАСТОСУНКУ ДЛЯ ВІДСТЕЖУВАННЯ ФІНАНСОВИХ ВИТРАТ КОРИСТУВАЧА

2.1 Технічне завдання

2.1.1 Введення

Призначення застосунку для відстежування фінансових витрат за допомогою Flutter полягає в створенні зручного та ефективного інструменту для користувачів, що дозволить їм відстежувати свої фінансові витрати.

Розглянемо детально цілі системи.

Зберігання та відстеження витрат: додаток дозволить користувачам вносити та зберігати дані про свої фінансові витрати. Він буде забезпечувати зручні інтерфейси для додавання витрат, вказування категорій витрат, дати та суми.

Генерація звітів та статистики: система надасть можливість користувачам переглядати звіти та статистику про їх фінансові витрати. Це може включати загальну суму витрат за певний період, розподіл витрат за категоріями, статистику витрат за дні, тижні, місяці тощо.

Установлення та керування бюджетами: система дозволить користувачам встановлювати межі бюджетів для різних категорій витрат або загального бюджету. Користувачі отримають сповіщення, коли досягаються або перевищуються встановлені межі бюджетів.

Інтеграція зі сторонніми джерелами даних: система може надавати можливість інтегруватися зі сторонніми API для отримання актуальних даних про валютні курси або інші фінансові показники, що дозволяють користувачам отримувати додаткову інформацію про свої витрати.

Забезпечення безпеки даних: система повинна забезпечувати захист користувацьких даних, використовуючи захисні механізми, такі як

шифрування та автентифікація, щоб убезпечити від несанкціонованого доступу до особистої фінансової інформації.

Призначення системи полягає у полегшенні та покращенні процесу відстеження фінансових витрат для користувачів, надаючи їм зручний, ефективний та безпечний спосіб контролювати свої фінанси.

Для реалізації цієї мети застосунок повинен виконувати певні завдання. Розглянемо їх більш детально.

Реєстрація та аутентифікація користувачів: забезпечення можливості реєстрації нових користувачів та аутентифікації вже зареєстрованих користувачів, щоб забезпечити доступ до особистих фінансових даних.

Додавання витрат: можливість вводити дані про витрати, включаючи категорію витрат, дату та суму. Додаток повинен мати інтерфейс для введення цих даних та збереження їх у базі даних.

Перегляд та редагування витрат: користувачі повинні мати можливість переглядати свої попередні витрати, сортувати їх за датою чи категорією, а також редагувати або видаляти витрати за потреби.

Генерація звітів та статистики: надання користувачам звітів та статистичних даних про їх фінансові витрати. Це може включати загальну суму витрат за певний період, графіки розподілу витрат за категоріями, середні значення витрат тощо.

Інтеграція зі сторонніми джерелами даних: можливість підключення до сторонніх API для отримання актуальних даних, наприклад, валютних курсів або фінансових новин, які можуть бути корисними для користувачів при прийнятті фінансових рішень.

Забезпечення безпеки даних: захист особистих фінансових даних користувачів шляхом використання шифрування та механізмів автентифікації. Запобігання несанкціонованому доступу до даних та забезпечення конфіденційності.

Розгортання та підтримка: передбачення процесу розгортання додатку на платформах Android та iOS, а також забезпечення підтримки та можливостей майбутнього розширення, оновлення та вдосконалення додатку.

2.1.2 Найменування і область застосування

Дане технічне завдання поширюється на розробку програми призначеної для відстежування фінансових витрат користувача з використанням технології Flutter.

Програмний продукт, що розробляється, отримує найменування: «Money Tracker». Застосунок призначений для відстежування фінансових витрат користувача з можливістю синхронізації та повним функціоналом різних налаштувань.

2.1.3 Підстава для розробки

Програма розробляється на підставі наказу ЗНУ від 26 січня 2023 року № 102-с.

2.1.4 Призначення розробки

Дана програма призначена для забезпечення користувача основними можливостями додатків фінансових витрат. Розглянемо їх детальніше.

Введення фінансових витрат: користувач може вводити свої фінансові витрати, вказуючи суму, дату, категорію та опис. Це дозволяє зберігати інформацію про всі витрати.

Відстеження фінансових витрат: користувач може переглядати свої фінансові витрати за певний період, категорії витрат, а також загальну суму витрат. Це надасть зручну інформацію для аналізу та контролю витрат.

Встановлення бюджету: користувач може встановити максимальну суму для свого бюджету на певний період часу або для окремих категорій витрат. Додаток буде надсилати сповіщення, коли користувач наближатиметься до

ліміту бюджету.

Статистика та звіти: додаток надає користувачеві статистичні дані та звіти про його фінансові витрати. Це може включати графіки, діаграми та інші візуальні засоби для кращого розуміння фінансової ситуації.

Категоризація витрат: додаток дозволить користувачу створювати та налаштовувати категорії витрат, наприклад, їжа, транспорт, розваги, медицина тощо. Це спростить організацію та аналіз витрат за різними категоріями.

Нагадування про платежі: додаток може надсилати користувачеві нагадування про заплановані платежі, такі як рахунки, кредити або інші фінансові зобов'язання.

Збереження історії витрат: додаток буде зберігати історію фінансових витрат, що дозволить користувачу переглядати минулі записи та відстежувати зміни в часі.

Підтримка множини валют: додаток може підтримувати різні валюти, дозволяючи користувачу ввести витрати в потрібній йому валюті та отримати конвертовану інформацію.

2.1.5 Вимоги до функціональних характеристик

Програма повинна забезпечувати можливість виконання наступних функцій:

- реєстрація та вхід в систему;
- відстежування фінансових витрат;
- керування бюджетом;
- генерація звітів та статистики;
- нагадування про платежі;
- синхронізація та резервне копіювання даних;
- налаштування профілю користувача;
- безпеку та конфіденційність.

2.2 Діаграма класів

Діаграма класів – статичне представлення структури моделі. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів, також, може містити позначення для пакетів та може містити позначення для вкладених пакетів. Також, діаграма класів може містити позначення деяких елементів поведінки, однак їх динаміка розкривається в інших типах діаграм. Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно–орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини [8].

Всього було реалізовано 30 класів (див. рис. 2.1 та 2.2). На рисунку 2.1 бачимо їх взаємодію, з яких класів складається система та структуру кожного класу доменів Акаунти та Транзакції.

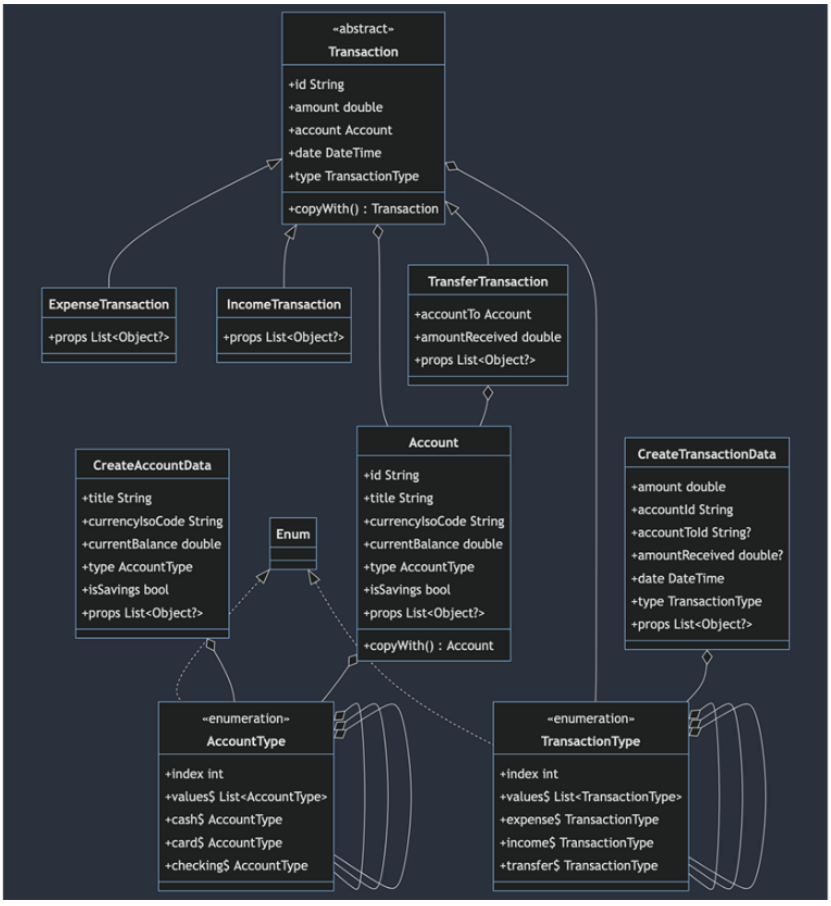

Рисунок 2.1 – Діаграма класів домену Аккаунти і Транзакції

На рисунку 2.2 продемонстровано, з класів складається домен Аутентифікація та структуру кожного класу.

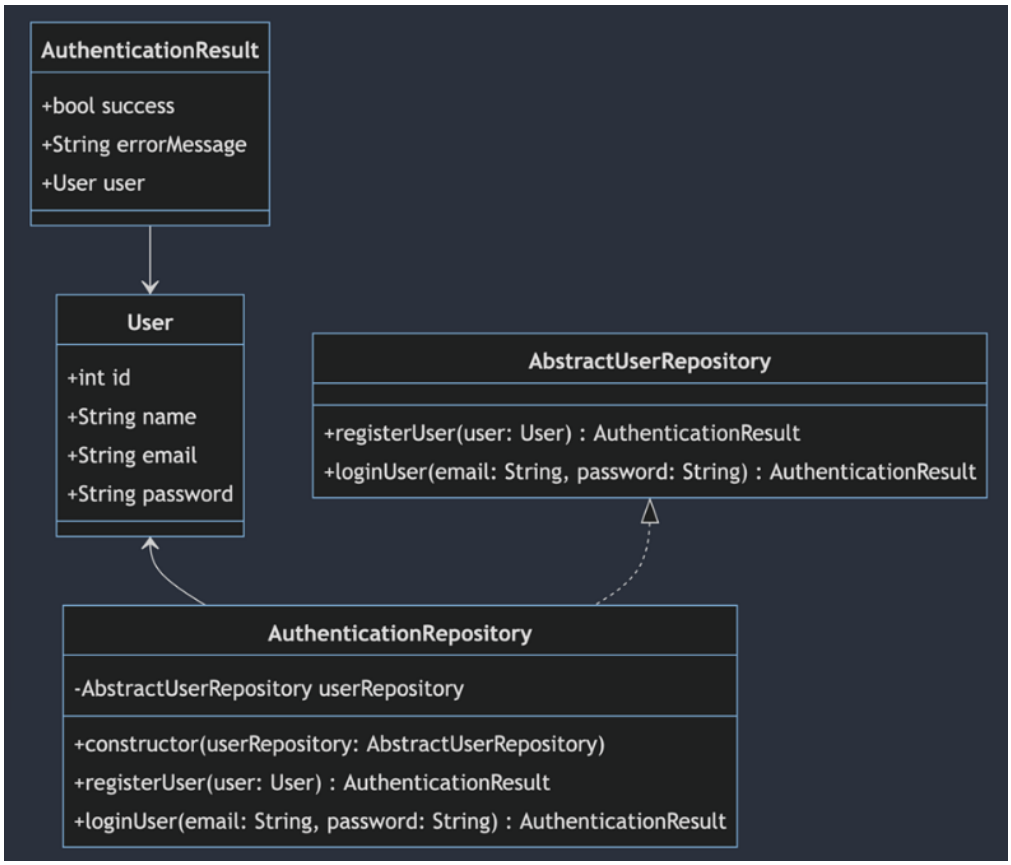

Рисунок 2.2 – Діаграма класів домену Аутентифікація

2.3 Діаграма прецендентів

Діаграма прецедентів – це діаграма, на якій зображено відношення між акторами і прецедентами в системі. Також, перекладається як діаграма варіантів використання (див. рис. 2.3).

Проєктована система представляється у вигляді безлічі сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання використовують для описання послуг, які система надає актору.

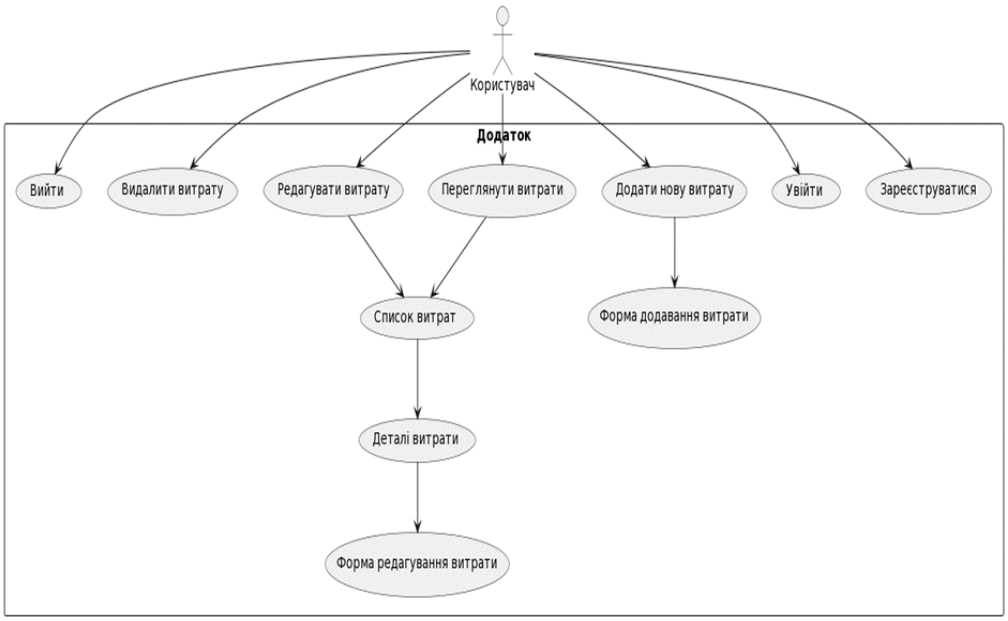

Рисунок 2.3 – Діаграма прецедентів

2.4 Діаграма взаємодії

На діаграмі взаємодії відображають обробку інформації в одному варіанті використання. Поведінка об'єкту представляється його операціями.

Діаграми взаємодії містять:

– об'єкти: можна використовувати імена як об'єктів, так і класів, або того і іншого;
– повідомлення: за допомогою повідомлення один об'єкт або клас запитує в іншого виконання якоїсь конкретної функції.

Діаграми послідовності впорядковані за часом. Вони корисні для того, хто хоче зрозуміти логічну послідовність подій в сценарії. Хоча інформація про послідовність входить і в кооперативні діаграми, вона краще сприймається на діаграмі послідовності. Діаграму послідовності слід читати зверху вниз. В кожного варіанту використання є велика кількість альтернативних потоків. Кожна діаграма послідовності описує один з потоків варіанту використання.

На рисунку 2.4 зображено процедуру створення транзакції.

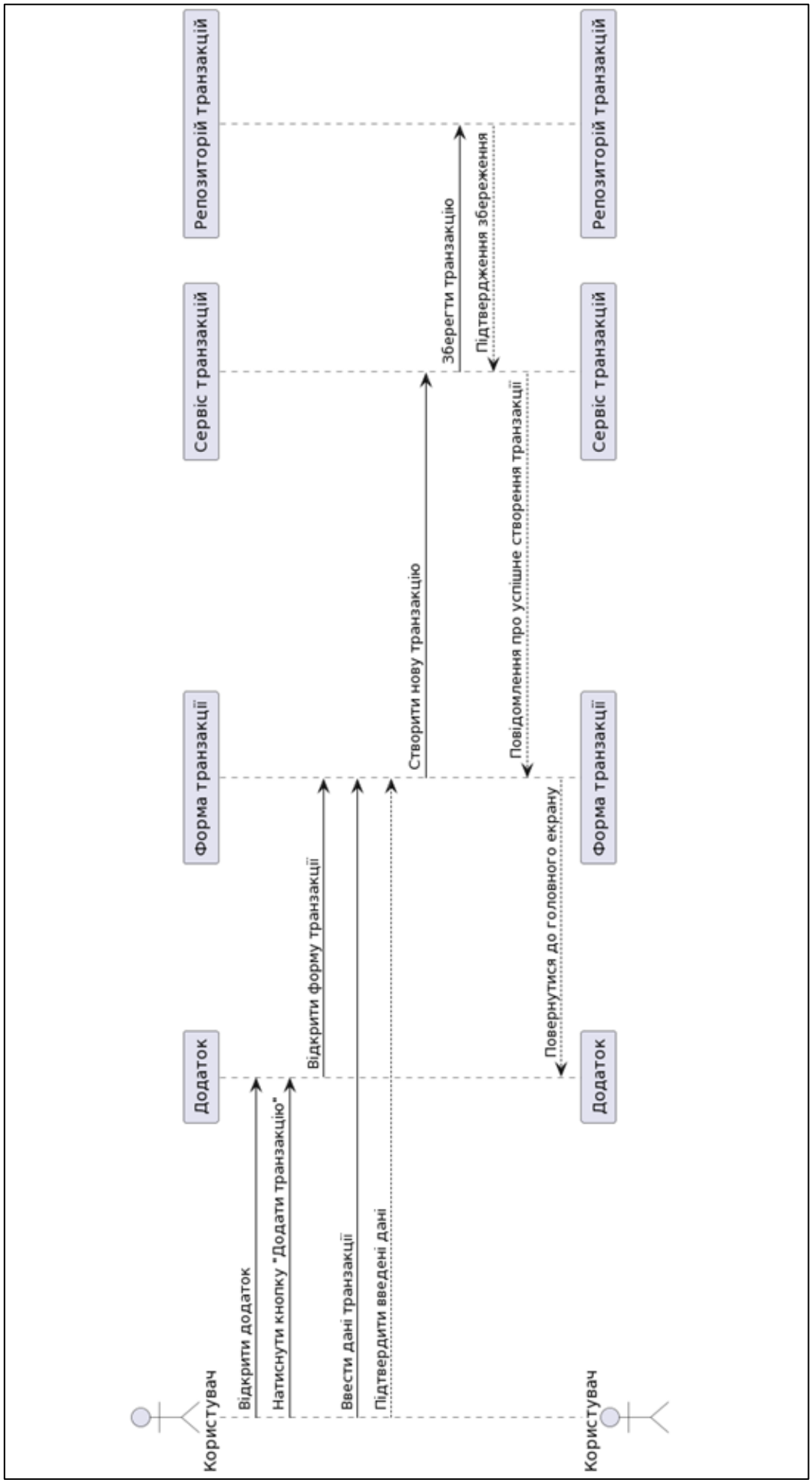

Рисунок 2.4 – Діаграма послідовності на створення транзакції

2.5 Методологія DDD

2.5.1 Технічне проєктування

Суть предметно-орієнтованого проєктування полягає в плануванні і створенні виключно виразних моделей. Крім того, DDD прагне до створення таких моделей, які зрозумілі практично всім IT-фахівцям, а не тільки розробникам, які пишуть код.

Оскільки такі моделі розраховані і на використання людьми без технічної освіти, було б зручно представляти їх різними способами. Як правило, модель предметної області може бути оформлена у вигляді UML-схеми, виражена у вигляді коду або описана на мові предметної області.

Патерни – це інструменти, а не правила. Їх можна вважати мовою для опису проєктування, для повідомлення ідей та подання моделей всередині команди. Не менш важливо і те, що предметно-орієнтоване проєктування націлене на прийняття прагматичних рішень.

Дана методологія допомагає розуміти кордону вирішуваної проблеми, а також справлятися з ще більш складним завданням – правильно окреслювати ці кордони при описі проблеми. Важливо і те, що предметно-орієнтоване проєктування привчає корисних практик, зокрема до розробки через тестування (TDD), використання патернів і безперервному рефакторингу.

Для розуміння і опису ключових деталей, специфічних для предметної області, абсолютно необхідно використовувати сумісний чіткий мову. У предметно-орієнтованому проєктуванні мова – це не іменники і дієслова, а концепції та їх зв'язку. Точніше, мова описує призначення концепцій. Важливо розуміти і доносити до слухача, наскільки важлива і цінна та чи інша інформація. Важливо і те, як саме реалізується певне призначення, але, як правило, в кожному конкретному випадку можливо кілька варіантів реалізації.

При роботі в предметної області все повинні максимально регулярно і при першій-ліпшій можливості користуватися такою мовою, щоб розуміти і

пояснювати концепції і призначення. При спілкуванні на універсальній мові більш творчий характер набуває співробітництво з експертами-предметниками, користь такої взаємодії також підвищується.

Необхідно відстежувати і усувати в предметному мовою такі технічні та ділові неясності, які затемнюють зміст важливих концепцій. З точки зору фахівця в предметній області такі концепції можуть бути «прихованими» або «якими мається на увазі». І найчастіше подібні ознаки характерні саме для варіантів реалізації, а не для самих концепцій предметної області. DDD не виключає опису реалізації, але інформація про призначення моделі в цій методології цінується набагато вище.

Не варто забувати, що мова використовується для представлення моделі предметної області. Цим вона схожа на код. При внесенні в код змін і доповнень проводиться його рефакторинг. Відповідно, необхідний і рефакторинг мови, якщо ви вводите в нього той чи інший новий термін. Переконайтеся, що концепція, що позначається терміном, чітко визначена і експерти-предметники згодні з таким варіантом і призначенням терміну.

При роботі з експертами-предметниками зазвичай зручно обговорювати проблему на конкретних прикладах. Найчастіше зручно описувати приклади з предметної області у вигляді історій «розробки через реалізацію поводжень» (BDD) і шаблонів сценаріїв.

2.5.2 Стратегічне проєктування

Стратегічне проєктування – це побудова найбільш загальної цілісної моделі. У центрі стратегічного проєктування лежить то безліч деталей, з яких складається загальна модель, а також на взаємозв'язках між цими частинами. Можно завчасно виконати певну частину проєктування, і цього достатньо для сталого розвитку моделі, а також для того, щоб модель не виходила «монолітної».

Такі деталі, які при предметно-орієнтованому проєктуванні можна вважати «малими моделями», існують в обмежених контекстах. Спосіб взаємозв'язку цих обмежених контекстів зазвичай називають «картою контекстів» (context mapping).

Для кожної малої моделі цілеспрямовано і явно будемо визначати контекст, в якому вона існує. Правил для створення контексту немає, але важливо, щоб всі фахівці, що працюють з ним, чітко розуміли умови, що обмежують цей контекст.

Контексти можна створювати (зокрема) за такими принципами:

– по організації команди;
– за структурою і компонуванні бази коду;
– по використанню в конкретній частині предметної області.

Складання карт контекстів – це частина процесу проєктування, в ході якої викреслюються точки контакту і переходи між обмеженими контекстами. Особлива увага приділяється «картографування» наявного предметного ландшафту, а робота з можливими трансформаціями здійснюється вже пізніше.

В окремо взятому обмеженому контексті доцільно застосовувати безперервну інтеграцію, щоб згладжувати «задирки», які можуть виникати через різне розуміння предмета. Часті коміти коду, автоматичні тести і застосування універсальної мови дозволяють швидко виявляти фрагментацію в рамках обмеженого контексту.

2.6 Висновки до другого розділу

Було виконано опис технічного завдання на розробку мобільного застосунку, були спроєктовані діаграми класів, прецедентів та взаємодії. Розглянуто основи методології DDD.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

3.1 Розробка серверної частини

Розглянемо структуру серверної частини. У проєкті використовується Cloud Firestore. Це гнучка, хмарна база даних від Firebase і Google Cloud Platform для мобільних платформ і серверних застосунків. А також використовується Firebase Auth для зручної авторизації.

Як можна побачити на рисунку 3.1, Firebase надає можливість реєструватися або авторизуватися. У застосунку реєстрація та авторизація було реалізовано за допомогою бібліотеки firebase_auth. Для зручної авторизації було вирішено використовувати авторизацію за допомогою Google або Apple.

Рисунок 3.1 – Список зареєстрованих користувачів

Головна частина проєкту – база даних, структуру якої можна побачити на рис. 3.2, рис. 3.3 та рис. 3.4. Тут створено три колекції: користувачі, транзакції та акаунти.

У першій зберігаються усі необхідні дані користувачів, які не змінюються впродовж усього часу. Це пов’язано з безпекою та коректною роботою застосунку. У таблиці транзакції зберігається інформація про транзакцію та зв’язок з аккаунтом, за допомогою цього зв’язку ми маємо змогу відновити усі інші зв’язки.

Слід звернути увагу, що база даних Firebase не дуже зручна та часто

використовується для демо версій. Також Firebase залежний від платформи, тому необхідно втручатися у нативний код та робити налаштовування для кожної платформи згідно з документацією Firebase. На приклад, для модулю Android потрібно додати google-servises.json, для коректної роботи застосунку. Саме тому майбутній застосунок повинен надавати змогу швидко змінювати логіку роботи з базою даних без втручання в код, який відповідальний за інтерфейс користувача [10].

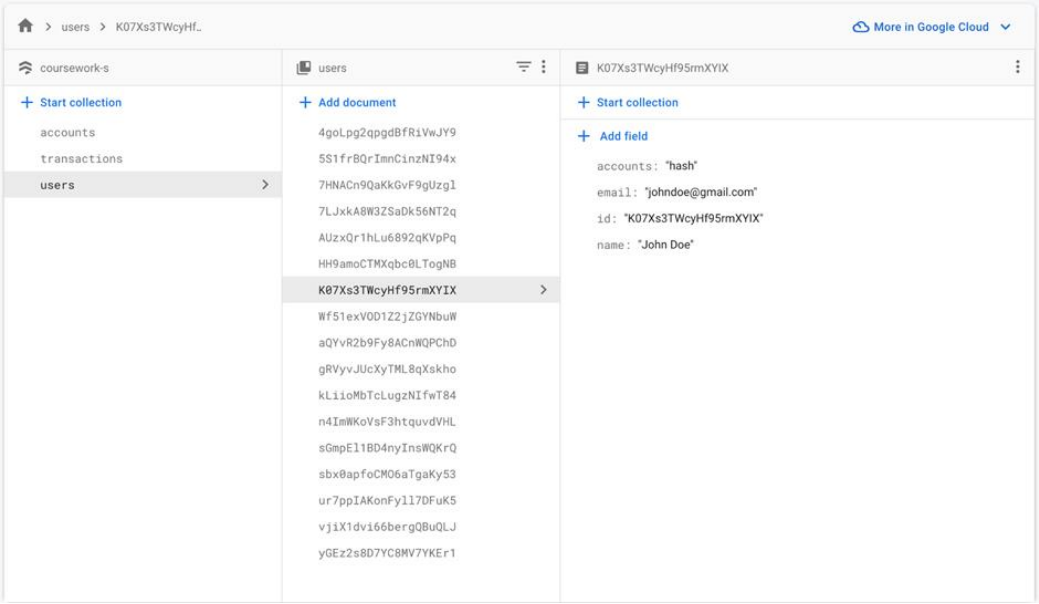

Рисунок 3.2 – Колекція користувачів у Cloud Firestore

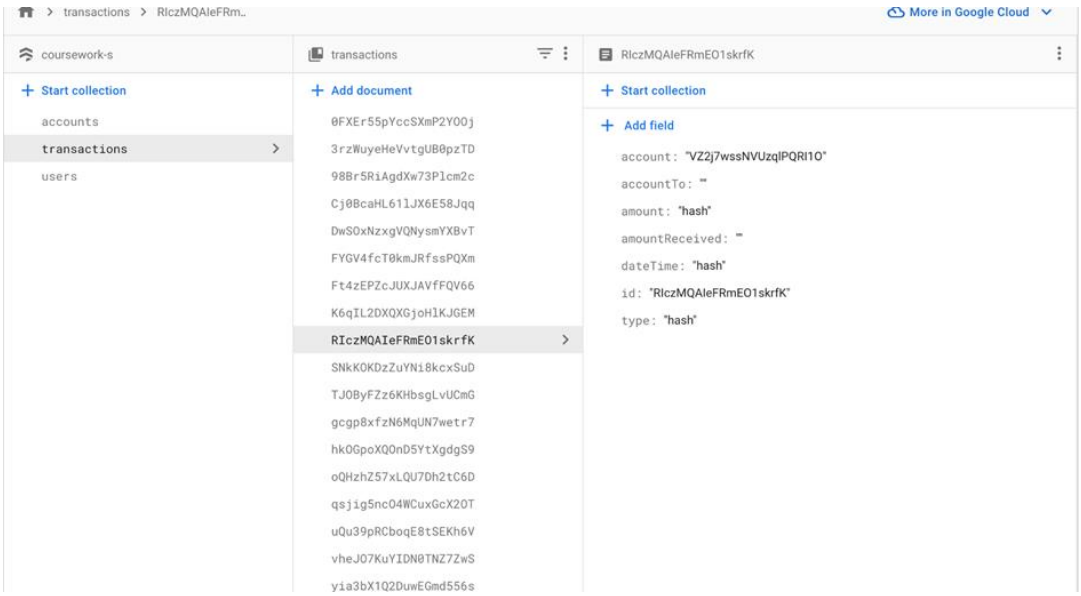

Рисунок 3.3 – Колекція транзакцій у Cloud Firestore

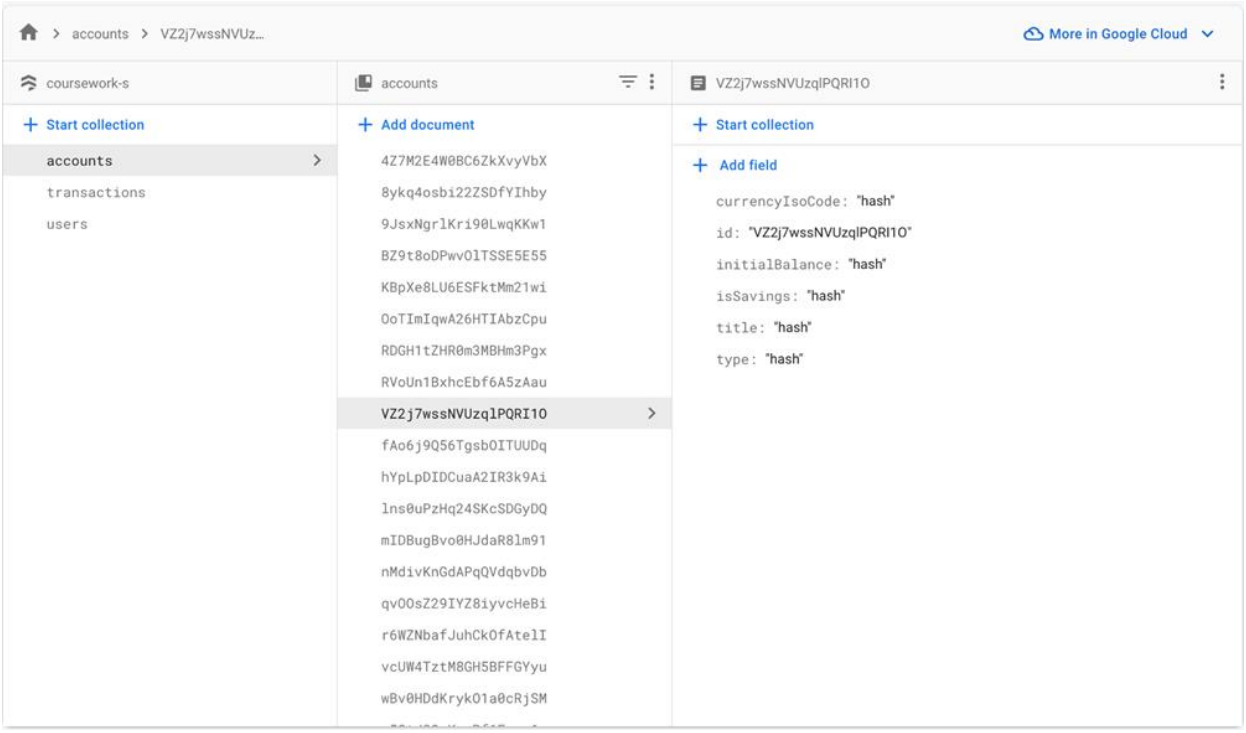

Рисунок 3.4 – Колекція аккаунтів у Cloud Firestore

3.2 Структура програми

Як було зазначено у попередньому підрозділу, застосунок повинен надавати змогу швидко змінювати логіку роботи з сервером, а також бути модульним для тестування. Саму тому частина, яка відповідальна за роботу з базою даних, повинна залежати від абстрактного класу або інтерфейсу. Нижче зображено інтерфейс AuthRepo (див. рис. 3.5).

```
abstract class AuthRepo {
Future<UserToken> signInViaEmailAndPassword(String password, String email);
Future<UserToken> signUpViaEmailAndPassword(String password, String email);
Future<UserToken> signInViaSocial(SocialAuthType socialAuthType);
Future<void> logOut();}
```

Рисунок 3.5 – Інтерфейс AuthRepo

AuthRepo має лише 4 методи які відповідають за авторизацію, реєстрацію та вихід. Різні методи використовуються в залежності від вибору способу авторизації. Як вже було визначено у попередньому підрозділі, застосунок буде використовувати авторизацію лише за допомогою Email. RemoteAuthRepo успадковується від AuthRepo. Весь проєкт має залежність тільки від AuthRepo, тому RemoteAuthRepo можна замінити на будь яку реалізацію AuthRepo.

Слід зауважити, що доступ до класів-репозиторіїв здійснюється за принципом Dependency Injection. Далі представлено код головного методу проєкту – main, з якого починається робота будь-якої програми на мові Dart (див. рис. 3.6).

Саме тут можна побачити єдину згадку про реалізовані від інтерфейсів класи які було описано вище. Напряму, в програмному коді, вони не використовуються.

```
void main() {
injector.setUpInjector();
runApp(App());
}
void setUpInjector() {
_register<AuthRepo>(RemoteAuthRepo());
_register<UserRepo>(RemoteUserRepo());

_register<TransactionRepo>(RemoteTransactionRepo());
_register<AccountRepo>(RemoteAccountRepo());
}
```

Рисунок 3.6 – Код головного методу main та setUpInjector

Класи відповідальні за інтерфейс користувача знаходяться у каталозі presentation. Застосунок використовує своє рідний архітектурний шаблон

похідний від MVP. Тобто класи з суфіксом cubit виконують функцію шару між юз кейсами та інтерфейсом користувача. Ці класи не знають про те, який клас, що відповідний за сторінку, їх використовує, але знають про юз кейси які постачають дані з репозиторію. Такі класи успадковуються від Cubit-у. Саме Cubit дозволяє сторінці відстежувати, коли слід оновлюватися, згідно стейту який отримав cubit за допомогою методу emit() та відправляти дані про дії користувача до серверу.

В цілому, проєкт містить багато класів сторінок та стільки ж класів кубітів, які використовують однаковий механізм роботи.

Застосунок має сторінку запуску або так званий Splash Screen. Сторінки запуску є нативними, тому необхідно втручатися у код Android та iOS. На приклад, в Android версії необхідно додати meta-data у AndroidManifest.xml як нижче в лістингу, та макет Splash Screen (див. рис. 3.7).

```
<meta-data
android:name="io.flutter.embedding.android.SplashScreenDrawable"
android:resource="@drawable/splash" />
    <?xml version="1.0" encoding="utf-8"?>

<layer-list xmlns:android="http://schemas.android.com/apk/res/android"> <item
android:drawable="@android:color/white" />
<item>

<bitmap
android:gravity="center"
android:src="@mipmap/launch_image" />
</item>
</layer-list>
```

Рисунок 3.7 – Код Splash Screen

Підводячи підсумок, розроблений застосунок є повністю реалізованим, усі зазначені у технічному завдані функції виконує, є модульним, легко

тестується та надає можливість легко змінювати будь-який модулі програмної реалізації.

3.3 Розроблені класи інтерфейсу користувача

Для широкого функціоналу застосунку були розроблені такі класи: App, TransactionsPage, AccountsPage, LoginPage, SettingsPage та інші.

Клас App – це "головний" клас застосунку. Створює головне вікно. За допомогою віджета MaterialApp створює середовище та генерує навігаційні шляхи. MaterialApp це стандартний віджет фреймворка Flutter. Він відповідає за поєднання всіх інших класів та головне меню. Надає доступ до інших вікон (HomePage, TransactionsPage, AccountsPage та іші).

Клас TransactionsPage містить у собі відображення всіх транзакцій користувача. Він відображає транзакції у вигляді листа за допомогою ListTransactionsView.

Клас SettingsPage відповідає за налаштування програми, там можливо перемикати мову застосунку та вибирати головну валюту.

Клас AccountsPage відповідає за відображення списку всіх акаунтів користувача та їх баланс, також на цій сторінки можно переглянути загальний баланс.

Клас LoginPage відповідає за можливість авторизації у застосунку. На цьому екрані відображаються кнопки для авторизації за допомогою соціальних мереж, також на цьому екрані відображається кнопка для переходу на інший екран пов’язаний з авторизацією за допомогою пошти.

Клас SignUpPage містить у собі форму для реєстрації користувача у системі. Також цей клас відображає повідомлення з помилок котрі ми можемо отримати у процесі реєстрації.

Клас TransactionCreateView містить у собі форму та кнопку за допомогою яких створюється Transaction. Цей клас використовує декілька

класів з логіки щоб отримати доступні акаунти користувача та тут використовується CreateTransactionUseCase для створення самої транзакції.

Клас AccountCreateView працює за схожим принципом, також бере декілька класів з логіки для отримання доступних даних, та використовує CreateAccountUseCase для створення аккаунта.

3.4 Опис процесу інсталяції програми

Встановлення програми на ОС Android здійснюється шляхом запуску .apk файлу або через завантаження через Play Маркет. Для iOS процес встановлення буде складніше тому, що розповсюдження застосунку здійснюється лише через AppStore або TestFlight, причому останній використовується тільки для тестування та потребує встановлення додаткового програмного забезпечення на пристрій. Обидва методи є платними так як розробник повинен бути зареєстрованим на ресурсі developer.apple.com та кожного року сплачувати близько 100$.

Після встановлення застосунку, при першому запуску програми користувач побачить сторінку авторизації де він може обрати чи авторизуватись або зареєструватись. При реєстрації застосунок відкриває сторінку авторизації.

Якщо користувач спробує створити профіль з поштою яка вже існує, застосунок повідомить про це користувача та повернеться до сторінки авторизації.

3.5 Тестування

Були написано unit-тести для впевненості у коректній роботи додатку та його модулів. Було створено json-файли які містять данні для тестування. Для

доступу до цих файлів використовується метод (див. рис. 3.8).

```
import 'dart:io';
File getFile(String nameFile, String $path) = File('$path/$nameFile.json);
```

Рисунок 3.8 – Код методу для доступу до файлів

Прикладом коду тестування наведеного нижче (див. рис. 3.9 та 3.10), за допомогою так званих моків, тестується робота методу fetchAccountById(String id) репозиторію AccountRepo.

```
void main() {
final expectedAccount = Account(
id: '36fc0178-1d73-469e-bb9c-2fabc0004364'
```

Рисунок 3.9 – Код тестування Account

```
AccountRepo accountRepo;

setUp(() {
accountRepo = MockAccountsRepo();
});
group(AccountRepo test', () {
test('fetch account by id', () async {
final account = await accountRepo
.fetchAccountById('36fc0178-1d73-469e-bb9c-2fabc0004364');
expect(account, equals(expectedAccount)); });
});
}
class MockAccountsRepo extends AccountsRepo {
List<dynamic> accountsJson;
```

```
MockAccountsRepo() {
accountsJson = json.decode(fixture('accounts'));
}
@override
Future<Account> fetchAccountById(String id) async { final accountJson =
accountssJson.firstWhere((obj) => obj['id'] == id, orElse: () => null);
final accountDto = AccountDto.fromJson(accountJson);
return accountDto.toModel();
}
}
```

Рисунок 3.10 – Код тестування fetchAccountById

Тестування класів-репозиторіїв є можливим лише завдяки Dependency Injection, про який було згадано вище.

Підводячи підсумок, розроблений застосунок є повністю реалізованим, усі зазначені у технічному завдані функції виконує, є модульним, легко тестується та надає можливість легко змінювати будь-який модулі програмної реалізації.

3.6 Підсумки до третього розділу

У даному розділі було описано створення програмної системи, яка складається з мобільного застосунку та Firebase, де Firebase – база даних, яка зберігає дані користувачів, дані та статистику, а застосунок – це клієнт-серверний застосунок який забезпечує зв’язок користувачів з даними серверу.

У підрозділі 3.2 було звернено увагу на важливі аспекти реалізації кросплатформного мобільного застосунку, а саме на реалізацію принципів SOLID: принцип інверсії залежності та принцип відкритості/закритості. Також було написано unit-тести для кожного модулю застосунку.

ВИСНОВКИ

Результатом виконання дипломної роботи є мобільний додаток для відстеження фінансових витрат користувача.

У ході роботи було вирішено наступні задачі:

– ознайомитися з предметною областю та проаналізувати її;
– обрати та обґрунтувати структуру системи, яка проєктується;
– розробити компоненти системи;
– провести тестування розробленої системи;
– зробити висновки.

Було виділено такі вимоги до розроблюваної системи:

– реєстрація та вхід в систему;
– відстежування фінансових витрат;
– керування бюджетом;
– генерація звітів та статистики;
– нагадування про платежі;
– синхронізація та резервне копіювання даних;
– налаштування профілю користувача;
– безпеку та конфіденційність.

Систему було реалізовано мовою програмування Dart з використанням фреймворку Flutter та IDE Android Studio. Для як базу даних було використано Firebase. На етапі вибору структури системи було побудовано структурну схему, виділено модулі, з яких складається програма. Для полегшення програмної реалізації системи, а також подальшого супроводження, було побудовано схему функціонування системи і наступні UML діаграми: діаграма класів та діаграма компонентів.

Далі було звернено увагу на важливі аспекти реалізації кросплатформного мобільного застосунку, а саме на реалізацію принципів

SOLID. Також було написано unit-тести для кожного модулю застосунку. Було описано як працює система, проведено модульне та ручне тестування. Отже, у даній роботі було вивчено принципи побудови програм з можливістю маштабування та змін шляхом використанням MVP архітектури та принципами SOLID.

ПЕРЕЛІК ПОСИЛАНЬ

1. Мобільний телефон. Official site. URL: https://goo.su/1FKV (дата звернення: 23.03.2023).
2. Monefy:Money tracker. Official site. URL: https://apps.apple.com/us/app/monefy-money-tracker/id1212024409 (дата звернення: 23.03.2023).
3. CoinKeeper: Budget planner. Official Site. URL: https://apps.apple.com/uz/app/coinkeeper-budget-planner/id849747345 (дата звернення: 23.03.2023).
4. Visual Studio Code. Official Site. URL: https://code.visualstudio.com/ (дата звернення: 23.03.2023).
5. FAQ Flutter. Official Site. URL: https://flutter.dev/docs/resources/faq (дата звернення: 03.04.2023).
6. Dart. Official Site. URL: https://dart.dev/ (дата звернення: 20.04.2023).
7. Buckett C. Dart in action. New York : Manning, 2013. 528 p.
8. Flutter release. Official Site. URL: https://github.com/flutter/flutter/releases (дата звернення: 20.04.2023).
9. Firebase Analytics. Official Site. URL: https://firebase.google.com/docs/analytics/ (дата звернення: 20.04.2023).
10. Firebase Auth. Official Site. URL: https://firebase.google.com/docs/auth/ (дата звернення: 20.04.2023).
11. Firebase Cloud Messaging. Official Site. URL: https://firebase.google.com/docs/cloud-messaging/ (дата звернення: 20.04.2023).
12. Firebase Storage. Official Site. URL: https://firebase.google.com/docs/storage/ (дата звернення: 20.04.2023).
13. Android Studio vs Visual Studio Code: What are the differences? Official Site. URL: https://stackshare.io/stackups/trending (дата звернення:

20.04.2023).

14. Relevant. Official Site. URL: https://relevant.software (дата звернення: 20.04.2023).
15. Leler W. What's Revolutionary about Flutter. Official Site. URL: Access mode: https://hackernoon.com/whats-revolutionary- about-flutter-946915b09514 (дата звернення: 20.04.2023).
16. Clow M. Learn Google Flutter Fast: 65 Example Apps. London: Independently Published, 2019. 507 p.
17. Дикяк В. Стоит ли инвестировать во Flutter. Сравнение Flutter и React Native. Official Site. URL: https://dou.ua/lenta/articles/flutter-for-mobile-apps/ (дата звернення: 20.04.2023).
18. Flutter. Official Site. URL: https://flutter.dev/docs/ (дата звернення: 20.04.2023).

ДОДАТОК А

Лістинг файлу main.dart

```
import 'dart:async';
import 'dart:developer';
import 'package:flutter/material.dart' hide RouterConfig;
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:get_it/get_it.dart';
import 'package:money_tracker/app.dart';
import 'package:money_tracker/core/app.env.dart';
import
'package:money_tracker/core/bloc/bloc.config.dart';
import
'package:money_tracker/core/di/injection_container.dart'
;
import
'package:money_tracker/core/initialization/initialization_
manager.dart';
import
'package:money_tracker/core/initialization/initializers/cu
stom_initializer.dart';
import
'package:money_tracker/core/initialization/initializers/da
ta_initializer.dart';
import
'package:money_tracker/core/initialization/initializers/de
vice_orientation_initializer.dart';
import
'package:money_tracker/core/initialization/initializers/eq
```

```
uatable_initializer.dart';
import
'package:money_tracker/core/initialization/initializers/lo
calization_initializer.dart';
import
'package:money_tracker/core/initialization/initializers/wi
dget_binding_initializer.dart';
import 'package:money_tracker/core/router.config.dart';
import
'package:money_tracker/domain/catcher/use_cases/handl
e_zoned_error.use_case.dart';
import
'package:money_tracker/domain/language/models/app_l
anguage.model.dart';
import
'package:money_tracker/domain/settings/use_cases/get_s
ettings.use_case.dart';
import
'package:money_tracker/presentation/routing/app.router.
dart';

const _devToolsKey = 'HAS_DEV_TOOLS';

final _getIt = GetIt.instance;

Future<void> main() async {
  runZonedGuarded(
      () async {
      final initializationManager = InitializationManager()
      ..addOrderedInitializer(WidgetBindingInitializer())
```

```
    ..addOrderedInitializer(CustomInitializer(() =>
configureDependencies(_getIt, AppEnv.name)))
    ..addOrderedInitializer(LocalizationInitializer())
    ..addOrderedInitializer(EquatableInitializer())
    ..addUnorderedInitializer(DataInitializer())

    ..addUnorderedInitializer(DeviceOrientationInitialize
r());

    await initializationManager.initialize();
    final blocConfig = _getIt.get<BlocConfig>();
    final routerConfig = _getIt.get<RouterConfig>();
    final appRouter = _getIt.get<AppRouter>();
    final appLanguage = _getIt.get<AppLanguage>();

    final settings = await
_getIt.get<GetSettingsUseCase>().call();

    Bloc.observer = blocConfig.observer;
    runApp(
    App.create(
    appRouter: appRouter,
    navigatorObserversBuilder:
routerConfig.navigatorObserversBuilder,
    settings: settings,
    appLanguage: appLanguage,
    ),
    );
    },
    (error, stackTrace) {
```

```
      if
(_getIt.isRegistered<HandleZonedErrorUseCase>()) {
      final          handleZonedErrorUseCase          =
_getIt.get<HandleZonedErrorUseCase>();
      handleZonedErrorUseCase(error, stackTrace);
      }          else          if          (const
bool.fromEnvironment(_devToolsKey)) {
      log('------------ZONED ERROR-------------');
      log('HandleZonedErrorUseCase not ready');
      log('ZONED ERROR -> $error -> $stackTrace');
      log('-----------------------------------');
      }
      },
  );
}
```

ДОДАТОК Б

Лістинг файлу account.dart

```
import 'package:equatable/equatable.dart';
import 'package:money_tracker/domain/account/enums/account_type.enum.dart';
class Account extends Equatable {
  final String id;
  final String title;
  final String currencyIsoCode;
  final double currentBalance;
  final AccountType type;
  final bool isSavings;
  const Account({
      required this.id,
      required this.title,
      required this.currencyIsoCode,
      required this.currentBalance,
      required this.type,
      required this.isSavings,
  });
  Account copyWith({
      String? id,
      String? title,
      String? currencyIsoCode,
      double? currentBalance,
      AccountType? type,
      bool? isSavings,
  }) {
      return Account(
```

```
      id: id ?? this.id,
      title: title ?? this.title,
      currencyIsoCode: currencyIsoCode ?? this.currencyIsoCode,
      currentBalance: currentBalance ?? this.currentBalance,
      type: type ?? this.type,
      isSavings: isSavings ?? this.isSavings,
      );
  }
  @override
  List<Object?> get props => [
      id,
      title,
      isSavings,
      ];
}
```

ДОДАТОК В

Вигляд додатку

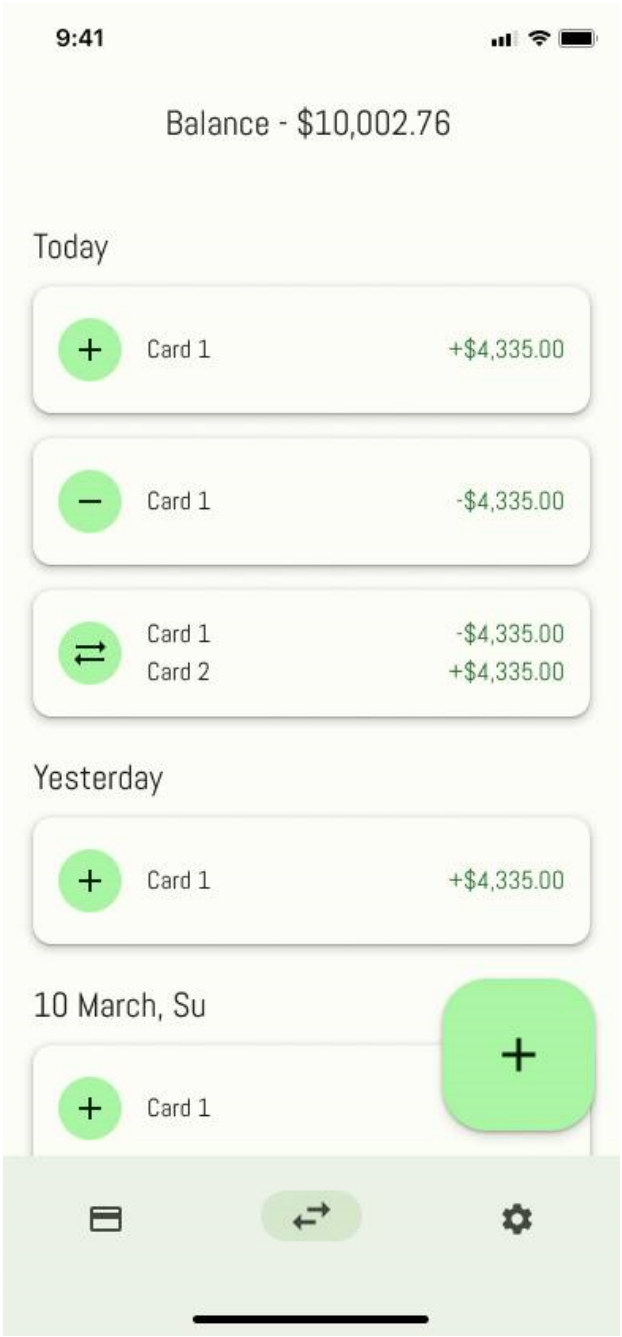

Рисунок В.1 – Головний екран

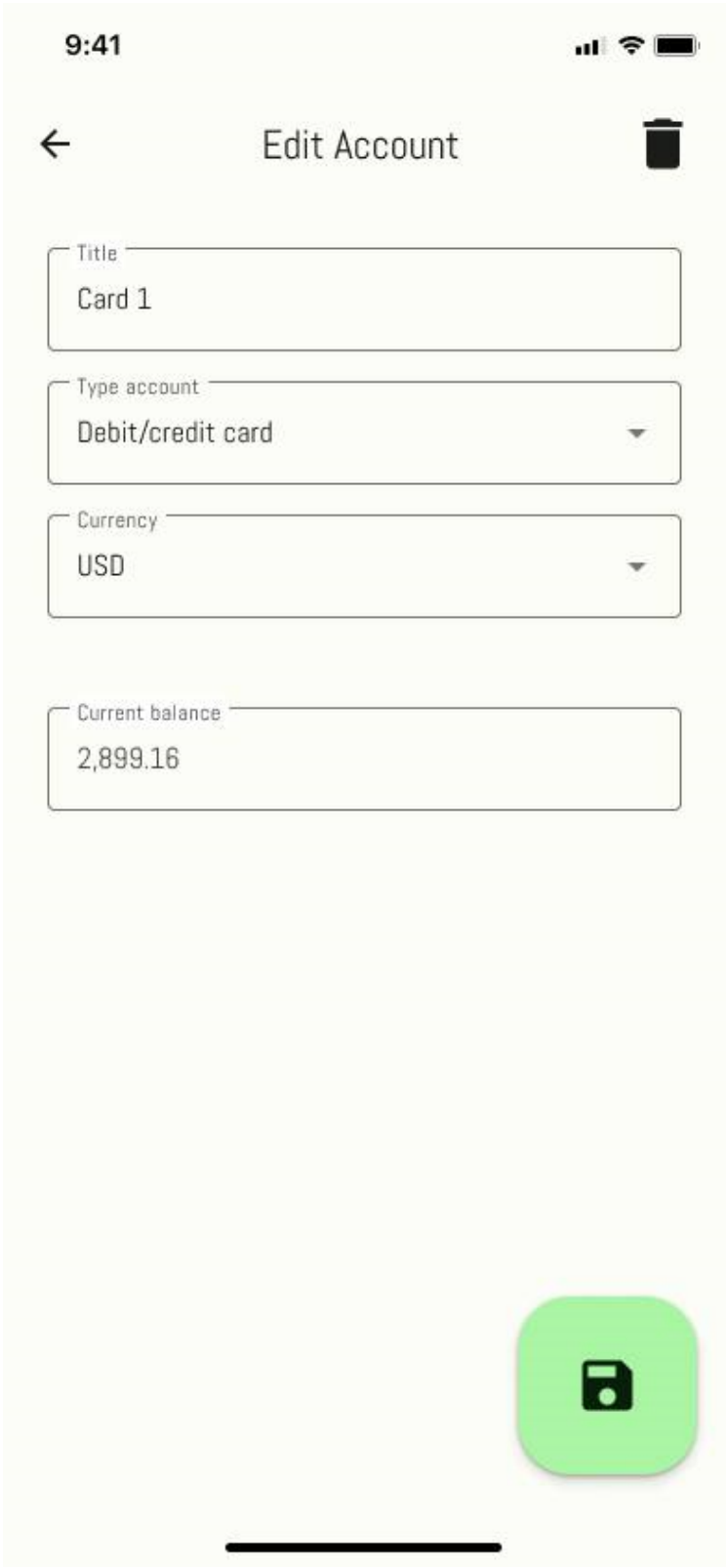

Рисунок В.2 – Вікно аккаунта

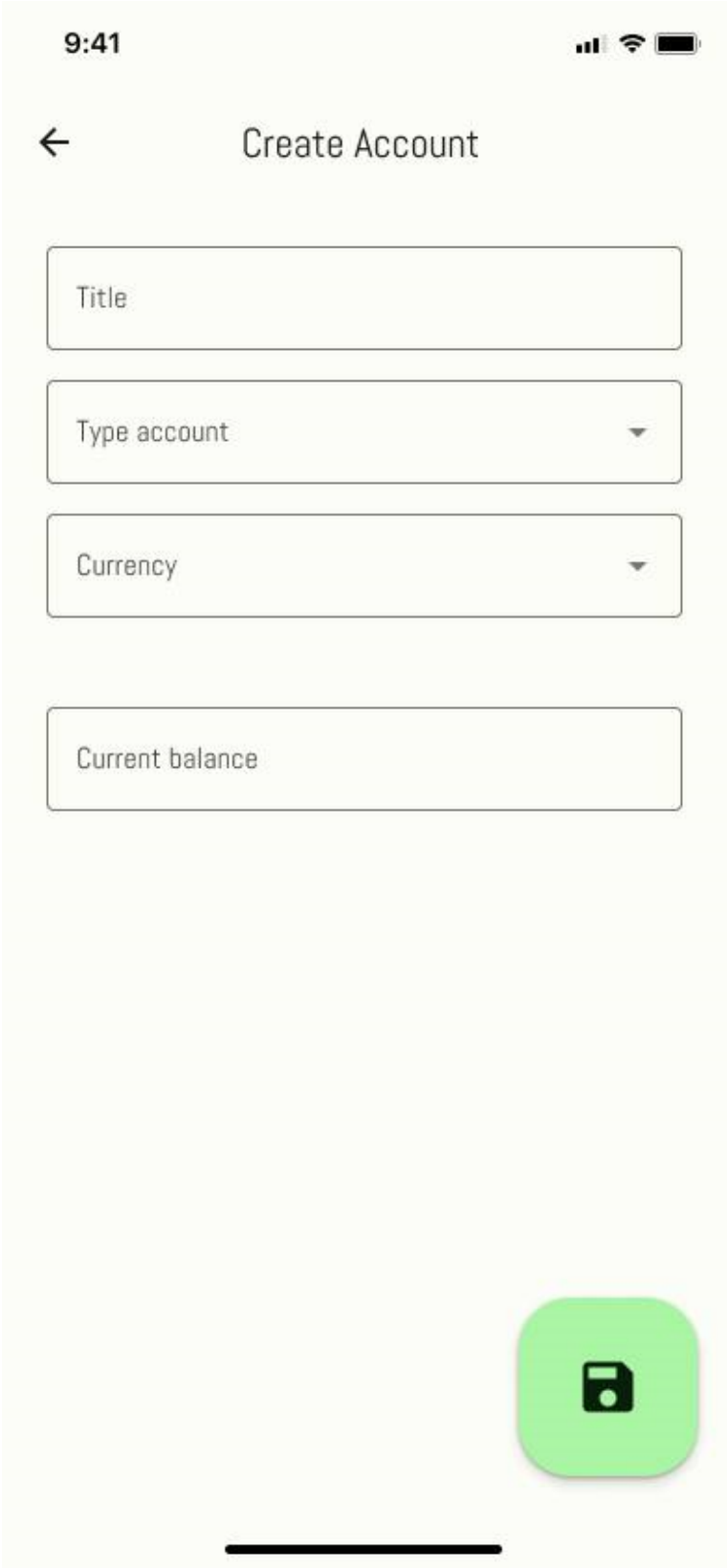

Рисунок В.3 – Форма створення аккаунта

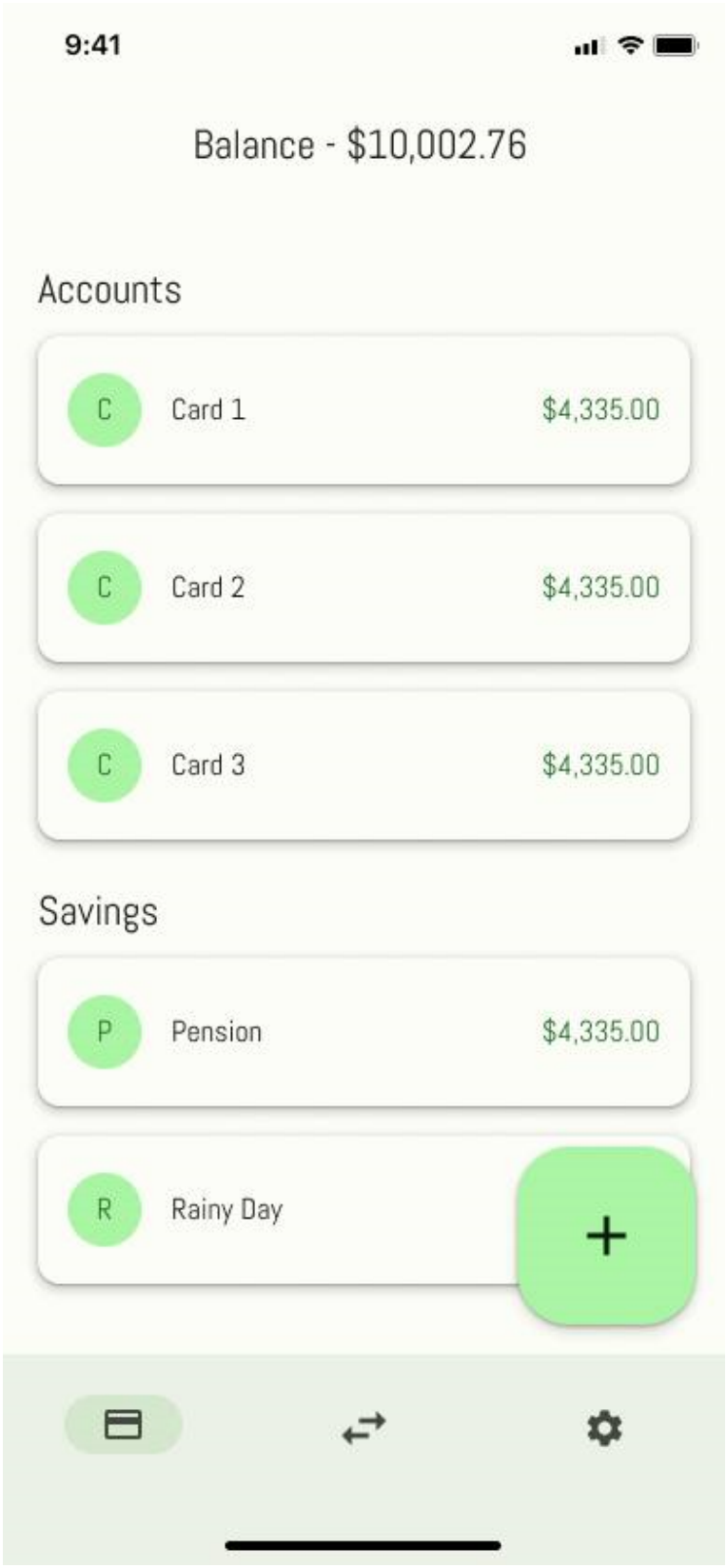

Рисунок В.4 – Меню застосунку

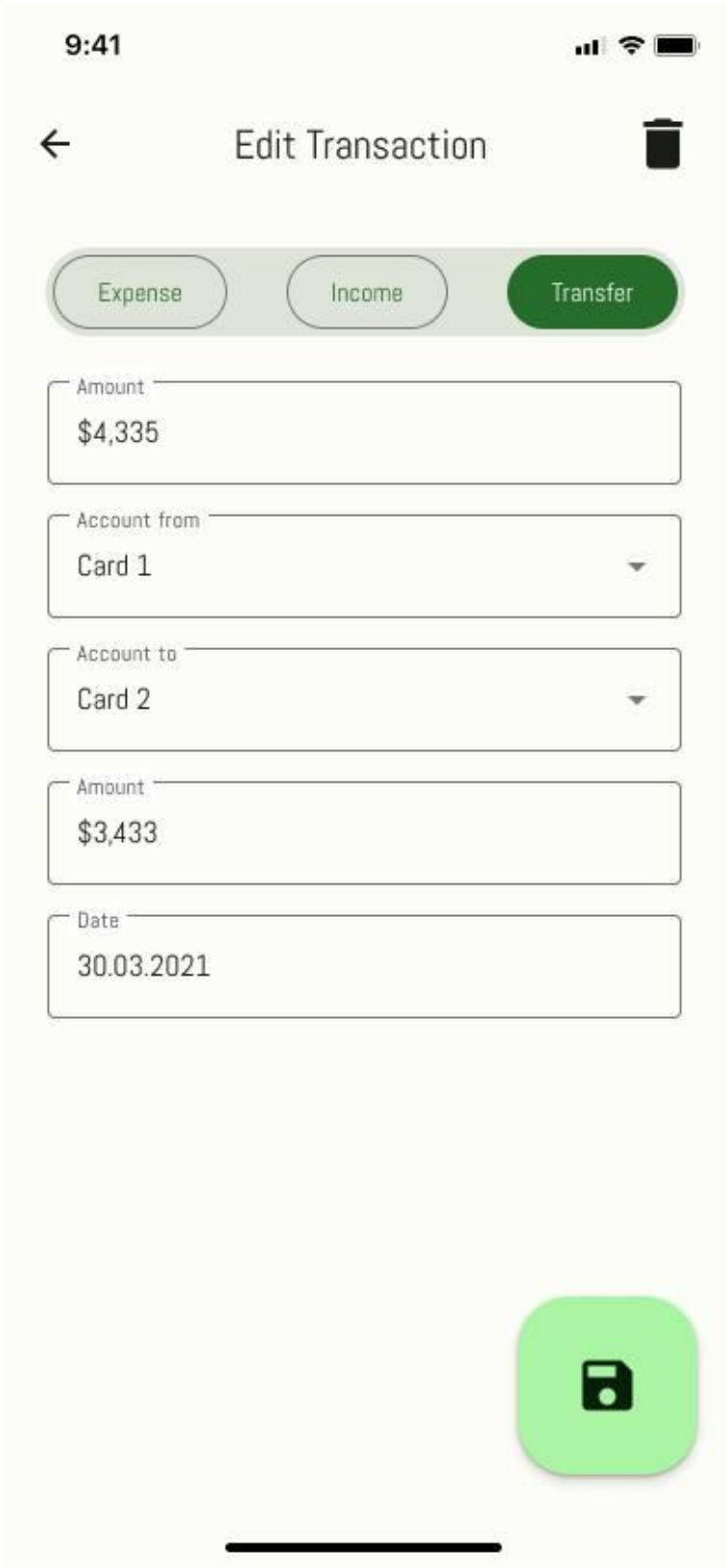

Рисунок В.5 – Налаштування застосунку