Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Василика Віктора Михайловича
Прізвище, ім'я, по батькові студента(ки)

на здобуття ступеня вищої освіти Бакалавра

Мобільний застосунок для ведення особистих витрат і доходів
Назва теми

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.200158.01.03.ПЗ

Виконав студент IV курсу, група ІПЗ-20-1 _____ Віктор ВАСИЛИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник старший викладач _____ Ганна БЕДРАТЮК
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер к.п.н., доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

6 червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ІПЗ
Л. П. Бедратюк
02 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Василика Віктора Михайловича
Прізвище, ім'я, по батькові студента

1. Тема роботи Мобільний застосунок для ведення особистих витрат і доходів

Керівник роботи Бедратюк Ганна Іванівна, старший викладач
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 08.01.2024 р. № 6
2. Строк подання студентом роботи на кафедру 01.06.2024 р.
3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, Проєктування мобільного застосунку, Програмна реалізація та тестування мобільного застосунку

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення:
1. UML-діаграма варіантів використання
2. Гібридна VIPER архітектура мобільного застосунку
3. UML-діаграма пакетів, що містить варіанти використання

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н. І., доцент	30.06.24	06.06.24
Антиплагіат	Форкун Ю. В., доцент	30.06.24	06.06.24

7. Дата видачі завдання «02» січня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проєктування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12 – 31.12.2023	
2 Збір матеріалу за темою кваліфікаційної роботи (КвР); дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2024	
3 Проєктування програмного забезпечення	21.02 – 20.03 2024	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2024	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно з вимогами	01.05 – 25.05.2024	
6 Попередній захист КвР	травень 2024	Згідно з графіком
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2024	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2024	

Студент _____ Віктор ВАСИЛИК
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Ганна БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Мобільний застосунок для ведення особистих витрат і доходів».

Автор роботи: Василик Віктор Михайлович.

Керівник роботи: Бедратюк Ганна Іванівна.

Пояснювальна записка: 83 с., 35 рис., 9 табл., 3 дод., 43 джерела.

Графічна частина: 3 креслення ф. А3.

ВЕДЕННЯ ВИТРАТ І ДОХОДІВ, МОБІЛЬНИЙ ЗАСТОСУНОК, СИСТЕМА ПЕРСОНАЛІЗАЦІЇ, ФІНАНСИ, JETPACK COMPOSE, KOTLIN.

Метою роботи є розроблення мобільного застосунку під Android ОС для ведення особистих витрат та доходів, яке допомагає користувачам економити свій час і бути спокійним за свої гроші, за допомогою представлення даних з різних джерел в єдиному програмному інтерфейсі, а також системи персоналізації робочого простору під індивідуальні потреби кожного клієнта.

У кваліфікаційній роботі проведено аналіз предметної області та його ринку програмного забезпечення, визначено особливості та специфіку процесу обліку доходів і витрат, визначені вимоги до мобільного застосунку, спроєктована архітектура та її артефакти. В пояснювальній записці зроблено акцент на системі персоналізації програмного продукту.

Для реалізації програмного продукту використано мову програмування Kotlin, фреймворк UI побудови Jetpack Compose, базу даних H2 та засоби проєктування UML, SysML.

Мобільний застосунок призначений для персонального користування на мобільних пристроях з операційною системою Android версії 8.1 та вище. Отримав кращі функціональні характеристики для персоналізації порівняно з наявними продуктами на ринку.

30.05.2024
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200158.01.03.ПЗ	Пояснювальна записка	83		
2	А4		Завдання на кваліфікаційну	1		
			роботу			
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.200158.01.03.Е1	UML-діаграма варіантів	1		
			використання			
5	А3	КвРІПЗ.200158.01.03.Е2	Гібридна VIPER архітек-	1		
			тура мобільного застосунку			
6	А3	КвРІПЗ.200158.01.03.Е3	UML-діаграма пакетів, що	1		
			містить варіанти			
			використання			

Змн.	Арк.	№ докум.	Підпис	Дат	КвРІПЗ.200158.01.03.ВД		
Виконав		Василик В. М.			Мобільний застосунок для ведення особистих витрат і доходів. Відомість документів	Літ.	Арк. 1 / Аркушів 1
Керівник		Бедратюк Г. І.					
Рецензент		Говорущенко Т. О.				ХНУ. ІПЗ-20-1	
Н. Контр.		Праворська Н. І.					
Зав. каф.		Бедратюк Л.П.					

ЗМІСТ

					КвРІПЗ.200158.01.03.ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дат			
Виконав		Василик В. М.		30.05.24	Мобільний застосунок для ведення особистих витрат і доходів. Пояснювальна записка	Літ.	Арк. 6 / Аркушів 83
Керівник		Бедратюк Г. І.		30.05.24			
Рецензент		Говорущенко Т. О.		6.06.		ХНУ. ІПЗ-20-1	
Н. Контр.		Праворська Н. І.		30.05.24			
Зав. каф.		Бедратюк Л.П.		30.05.24			

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
ГРДУ	–	розлад гіперактивності з дефіцитом уваги
ОС	–	операційна система
ПЗ	–	програмне забезпечення
ПП	–	програмний продукт
СДУГ	–	синдром дефіциту уваги й гіперактивності
СУБД	–	система управління базами даних
ADHD	–	attention deficit hyperactivity disorder
API	–	application programming interface
PWA	–	progressive web apps
SysML	–	systems modeling language
UI	–	user interface
UML	–	unified modeling language
UX	–	user experience
XML	–	extensible markup language

ВСТУП

Стан предметної області та ринку. Аналіз та спостереження застосунків протягом 2023-2024 років за темою кваліфікаційної роботи на платформі «Play Market» покази, що громадяни України є активними користувачами продуктів в категорії «Фінанси». Згідно зі відкритою статистикою сервісу «AppMagic» більшість компанії, що розробляють програмні рішення є іноземного походження, які не завжди добре працюють з українськими банками через API, частка українців серед активних користувачів може сягати до 10%. Серед тих компаній, які добре працюють з українськими фінансами є російського походження, частка українців серед активних користувачів таких застосунків може сягати до 15%. На другий рік повномасштабного вторгнення, українці ще досі продовжують користуватися платними послугами країни агресора. Програмних рішень українського виробництва досить мало, серед них «Saldo: Облік доходів і витрат» випущений у квітні 2022 року.

Основні тенденції розвитку предметної області.

Мобільні пристрої дуже часто використовуються для проведення платежів, проблема моніторингу за фінансами є вкрай важливою для розв’язання, і зручним місцем для цього є смартфон. Частка розрахунків смартфонами та гаджетами з NFC серед всіх безконтактних оплат з картками Mastercard у фізичному ритейлі в Україні перетнула рубіж у 60%, про це свідчить транзакційна статистика Mastercard за другий квартал 2023 року.

На ринку фінансів наявна велика кількість криптовалюти, для даного сегменту існує ряд наявних рішень з відстеження балансу в режимі реального часу. Спільних рішень, які б одночасно добре працювати з криптовалютними та банківськими рахунками українських банків аналіз предметної області не виявив. Станом на 2023 рік було понад 9000 цифрових монет, хоча в перші місяці 2022 року їх було набагато більше [1].

Фінансова грамотність є основою для ухвалення відповідальних фінансових рішень, забезпечення та поліпшення власного добробуту життя. Українці поступово

покращують свою фінансову грамотність, про що свідчать результати дослідження проєкту USAID "Трансформація фінансового сектору" в співпраці з Національним банком України [2]. Так за 2018-2021 роки індекс фінансової грамотності українців зріс з 11,6 до 12,3 бала. Також в січні 2024 року міжвідомча робоча група у складі фахівців Національного банку України, Фонду гарантування вкладів фізичних осіб, Міністерства освіти та науки України та Національної комісії з цінних паперів і фондового ринку підготувала Рамку фінансових компетентностей дітей та молоді України [3]. Також позитивно впливає на підвищення фінансових компетентностей застосування на практиці фінансових знань: користування фінансовими послугами, контроль доходів і витрат, фінансове планування, заощадження. До прикладу, з п’ятьма фінансовими продуктами обізнані близько 80% українців [2]. Щоб полегшити й пришвидшити цей процес потрібні програмні продукти, які зможуть задовольнити потреби різних цільових аудиторій, одне із таких буде розроблятися в ході виконання цієї кваліфікаційної роботи.

Передумови виникнення проблем. Вагома причина існування програмного забезпечення для ведення витрат та доходів полягає в тому, що на ринку фінансів відсутній монопольний представник, так, наприклад, популярний банк України «ПриватБанк» на лютий 2024, згідно з їхньої статистики, налічує 19 мільйонів активних клієнтів, цього не достатньо аби закрити потреби всіх учасників ринку, тому банківська установа не може надавати інформацію по транзакціях, які в неї відсутні. Потенційні клієнти ПЗ часто мають, одночасно кілька відкритих банківських рахунків в різних установах, де дані балансу та фінансових операцій не об’єднані в єдину базу даних. Також в багатьох людей є гаманці, які не мають функцій під’єднання до Інтернету та автоматичного розрахунку балансу.

Звідси виникають наступні проблеми на розв’язання якої спрямована КвР:

– дані фінансових транзакцій є розподіленими або частково відсутніми, що унеможливлює доступ до них через єдиний програмний інтерфейс;

– записи про доходи та витрати повинні бути представлені належним чином, в зрозумілій та легкій формі, до ознайомлення;

– при великій кількості фінансових потоків стає важко за ними спостерігати, потрібні інструменти моніторингу коштів;

– при великій кількості фінансових операцій, ручні розрахунки вимагають багато часу, потрібні автоматизовані рішення для формування звітів та представлення статистики.

Актуальність теми. Таким чином, цільова аудиторія програмного продукту має декілька фінансових потоків із різних джерел, дані яких є децентралізованими, український ринок програмних продуктів не може покрити запити клієнтів на розв'язування вищенаведених проблем, тому було прийнято рішення про необхідність розробки нового програмного забезпечення.

Мета кваліфікаційної роботи. Розробити мобільний застосунок під Android ОС для ведення особистих витрат та доходів, яке допомагає користувачам економити свій час і бути спокійним за свої гроші, за допомогою представлення даних з різних джерел в єдиному програмному інтерфейсу, а також системи персоналізації робочого простору під індивідуальні потреби кожного клієнта. Згідно з поставленою метою, було сформовано план та його завдання для виконання кваліфікаційної роботи:

– провести дослідження предметної області, проблем користувачів, ринку та основних тенденцій його розвитку;

– визначити особливості та специфіку процесу обліку доходів та витрат;

– виконати аналіз рішень, що існують;

– розробити технічне завдання;

– розробити архітектуру та функціональну структуру, алгоритм роботи та програмну логіку мобільного застосунку;

– здійснити аналіз та вибір технологій, модель життєвого циклу і методів реалізації програмного продукту;

– виконати програмну реалізацію мобільного застосунку;

– провести тестування готового мобільного застосунку.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Мобільний застосунок, який найчастіше називають програмою (англ. Application, App), – це тип прикладного програмного забезпечення, призначеного для роботи на мобільному пристрої, наприклад смартфоні або планшетному комп'ютері. Мобільні програми часто служать для надання користувачам послуг, подібних до тих, які доступні на ПК.

Розробка мобільного застосунку починається із дослідження предметної області [4], і першими запитаннями на які варто знайти відповіді є: «Хто наші користувачі?», «На яких пристроях вони працюють?», «Які операційні системи використовують?», «Які особливості ринку, їх обмеження?».

У четвертому кварталі 2023 року Android зберіг свою позицію провідної мобільної операційної системи в усьому світі з часткою ринку 70,1 відсотка. Найближчий конкурент Android, iOS від Apple, мав частку ринку у 29,2 відсотка протягом того ж періоду [5]. Таким чином операційні системи Android від Google та iOS від Apple разом складають 99% світового ринку мобільних ОС, тому для швидкого доступу до ринку мобільних застосунків потрібно використовувати одну із них. Згідно з інструментом статистики «Statcounter Global Stats» є незначна відмінність в даних використання мобільних операційних систем по всьому світу (таблиця 1) та в Україні (таблиця 2).

Таблиця 1 – Розподіл використання мобільних ОС у світі

Android	iOS	Samsung	Інші	KaiOS	Windows
69.94%	29.32%	0.36%	0.16%	0.16%	0.02%

Таблиця 2 – Розподіл використання мобільних ОС в Україні

Android	iOS	Samsung	Windows	Linux	Nokia та інші
70.49%	29.09%	0.28%	0.03%	0.03%	0.02%

Ключова відмінність між двома системами стосується апаратного забезпечення: iOS доступна лише на пристроях Apple, тоді як Android постачається з пристроями ряду виробників, таких як Samsung, Google та OnePlus. Крім того, Apple досягла набагато більшого успіху в оновленні своїх користувачів. Станом на лютий 2023 року 81 відсоток користувачів iOS мали встановлену найновішу версію iOS 16, тоді як у січні 2023 року останню версію використовували лише чверть користувачів Android.

Обоє операційних систем мають власні магазини за допомогою яких постачається користувачам нове програмне забезпечення. Магазин для Android називається «Google Play Store» і містить більше програм (2,56 мільйона), ніж «Apple App Store» (1,85 мільйона) від iOS [5]. За даними Tech Crunch, у 2020 році в Apple і Google App Store було завантажено понад 218 мільярдів мобільних застосунків. На телефонах Samsung Galaxy попередньо встановлено два магазини застосунків: Play Store і Galaxy Store. Samsung Galaxy Store, як правило, більше зосереджується на додатках і сервісах компанії Samsung.

Географія користувачів також важлива, щоб вибрати правильну операційну систему для ринку ПЗ. На таблиці 2 добре простежується розподіл операційних систем, важливо розуміти, що дані для окремо взятої країни можуть сильно відрізняються від загальної статистики в регіоні світу [6].

Таблиця 3 – Розподіл використання мобільних ОС за регіонами світу

	Європа	Південна Америка	Північна Америка	Африка	Азія	Австралія
Android	66.09%	83.86%	41.39%	84.63%	78.64%	69.94%
iOS	33.42%	15.88%	58.19%	13.05%	20.57%	29.32%

Існує три основні типи мобільних програм, якщо класифікувати їх за технологією, яка використовується для їх програмування.

Рідні мобільні застосунки або нативні (англ. Native apps) створюються для однієї конкретної платформи чи операційної системи. Як правило, вони ефективніше використовують ресурси пристрою, ніж інші типи мобільних програм.

Рідні програми підключаються безпосередньо до апаратного забезпечення пристрою, вони мають доступ до широкого вибору функцій пристрою, таких як NFC, Bluetooth, контакти телефонної книги тощо. Програмний код, який створюється для однієї платформи, не можна повторно використовувати на іншій, для цього доведеться дублювати зусилля для кожної несумісної платформи. Це збільшує витрати на підтримку та оновлення кодової бази для кожної версії.

Вебпрограми (англ. Web apps) – це адаптивні версії вебсайтів, які можуть працювати на будь-якому мобільному пристрої чи ОС, оскільки вони доступні за допомогою мобільного браузера. Такі програми адаптують інтерфейс користувача до пристрою, на якому перебуває користувач. Коли користувач натрапляє на опцію «встановити» вебпрограму, вона часто просто додає URL-адресу вебсайту в закладки на пристрої. Ці програми працюють через Інтернет, нема потреби налаштовувати її під платформу чи ОС, вони не займатимуть місце в пам’яті пристрою, як рідна програма, що спрощує обслуговування – достатньо опублікувати оновлення через Інтернет. Вебпрограми повністю залежать від браузера, який використовується на пристрої. Існуватимуть функції, доступні в одному вебпереглядачі й недоступні в іншому, що, можливо, забезпечить користувачам різний досвід. Одним із видів вебпрограми є прогресивна вебпрограма (PWA), яка в основному є рідною програмою, що працює в браузері.

Гібридні мобільні застосунки (англ. Hybrid apps) – це комбінації як нативних, так і вебпрограм, що включені в нативну програму, що дає їй можливість мати власну піктограму або завантажуватися з магазину програм. Вони також мають адаптивний дизайн і навіть можливість працювати в режимі офлайн. Створення гібридної програми набагато швидше та економніше, ніж нативної програми. Крім того, вони швидко завантажуються, ідеально підходять для використання в країнах із повільнішим Інтернет-з’єднанням і надають користувачам узгоджену взаємодію. Гібридним програмам може бракувати потужності та швидкості, які є характерними рисами нативних програм.

Найпростішим способом ведення обліку фінансів є звичайний зошит у клітинку. Сьогодні практично кожна людина володіє сучасним смартфоном, який

також зручно використовувати й для обліку фінансів. При достатньому розвитку продукту, він може бути поширений й на інші операційні системи, що збільшує цільову аудиторію та зменшує обмеження на несумісність пристрої сприяючи вільній та зручній взаємодії користувачів між собою.

Для ведення обліку фінансів існує кілька важливих причин:

– щоб уникнути ситуації, коли фінансові запаси закінчилися, а до наступної зарплати чекати ще довго;

– до кінця місяця ви будете точно знати, куди «пішли» гроші;

– це хороша можливість проаналізувати раціональність витрат і необхідність здійснюваних покупок;

– облік доходів і витрат дозволить вам здійснювати лише дійсно необхідні витрати та економити значну частину отриманих коштів;

– визначте свої фінансові обмеження і зможете від них позбутися.

Фінансова грамотність – знання та розуміння фінансових процесів, які впливають на життя індивіда. Ці процеси можуть бути пов’язані з особистими фінансами окремої людини, або з ширшими економічними зв’язками між компаніями, підприємствами, регіонами чи державами. Також існує глобальний рівень, на якому аналізують вплив макроекономіки на стан світової економіки. Фінансова грамотність передбачає вміння раціонально управляти своїми доходами й витратами, а також планувати свій бюджет.

Вести облік фінансів не так вже й складно. Головне робити це регулярно і не лінуватися фіксувати кожний дохід та витрату. Дуже важливо, щоб користувач мобільного застосунку виробив в собі звичку стежити за всіма доходами та фіксувати кожну фінансову операцію. В сучасному світі інформація поширюється дуже швидко, це допомагає нам ефективно працювати та вчитися, разом з тим, існує можливість втратити фокус на головному.

Тому програмні продукти, що вимагають від користувача постійної активності повинні нагадувати про себе та заохочувати відкривати їх щодня, ось декілька таких механізмів.

Push-сповіщення – це сповіщення, які програми надсилають на пристрої користувачів, навіть якщо ці програми не відкриті. Вони розроблені, щоб привернути увагу на смартфонах, ноутбуках і навіть на розумних годинниках. Коли людина вперше перевіряє свій телефон вранці, вона, ймовірно, побачить декілька push-сповіщень, які надійшли за ніч – текстові повідомлення, пропущені дзвінки, електронні листи, сповіщення про погоду тощо. Згідно з останніми дослідженнями Google, час, необхідний для завантаження вебсайту мобільної цільової сторінки, становить 22 секунди. Однак дослідження також показує, що 53% людей залишать вебсайт мобільної сторінки, якщо вона завантажується довше 3 секунд. Apple App Store вимагає, щоб розроблений застосунок завантажувався протягом 15 секунд, але в ідеалі це має бути швидше. Хорошим рішенням буде створення фінансового асистента через push-сповіщення, який буде нагадувати протягом дня про облік доходів та витрат, а в кінці дня буде надавати коротку статистику про проведені фінансові операції, чим самим щоденно створюючи цінність для користувача при його мінімальних зусиллях.

Гейміфікація – це процес використання ігрової механіки, елементів та принципів і їх застосування до неігрового контексту для кращого залучення користувачів [7]. Для теми КвР доцільно використовувати наступні приклади ігрової механіки, яка використовується в гейміфікації. Цілі – виконайте завдання та отримайте нагороду, наприклад значок або бали. Навчання – підказки та тести надаються користувачеві протягом усього процесу. Спільнота – користувачі об’єднуються в пари або об’єднуються в групи для розв'язання проблем, виконання завдань або іншого досягнення мети. Статус – користувачі підвищують свій рівень або рейтинг, виконуючи дії. Нагороди за активне використання програмного продукту є популярною практикою. Як згадувалося вище, бали та значки є поширеними та корисними винагородами. Іншими винагородами можуть бути купони, знижки або подарункові картки. Це підживлює мотивацію користувача та підтримує високу зацікавленість. Успішна гейміфікація враховує внутрішню мотивацію користувача, як-от підвищення кваліфікації у своїй роботі, водночас пропонуючи зовнішню мотивацію, таку як винагороди, бали та значки.

Програмний продукт може мати декілька напрямків використання:

– створення записів про власні фінансові операції;

– допомога в керуванні бюджетом;

– спільне ведення бюджету;

– автоматичне формування фінансових звітів та візуалізація статистичних даних користувача: діаграми розподілу транзакцій за категоріями, за періодом часу;

– інструмент швидкого моніторингу за великою кількістю розподілених рахунків та їхніми транзакціями.

Продукт має великі перспективи для розвитку. Так, згідно з чинним законодавством України платники податків повинні зберігати свої записи про доходи та витрати протягом 3 років після подачі звітності, програмний модуль, що допоможе користувачам зберігати інвойси чудово доповнить ряди функцій ПП. Програма обліку фінансів може бути корисна не тільки для банківських рахунків та гаманців, але й для криптовалютних та ігрових облікових записів.

Загальний індекс фінансової грамотності України становить 55%. Це нижче, ніж середнє значення (58%) серед країн з аналогічним рівнем доходів, таких як Білорусь, Грузія, Угорщина, Польща, Туреччина. Понад 60% українців вважають рівень свого добробуту низьким та відчувають значний стрес через проблеми з грошима. Найкращий спосіб створити корисний для клієнта програмний продукт це розв’язувати його щоденні проблеми, наприклад: зменшити рівень переживань за гроші, надати йому декілька вільних хвилин в день шляхом автоматизації рутинних завдань по рахунках. Можна побачити, що українці справді мають підставити для стресу згідно з даними журналу «Forbes Ukraine» [8]:

– 75% – турбуються через оплату звичайних повсякденних витрат;

– 53% – не довіряють банкам;

– лише 30% громадян України мають власні кошти, щоб покрити витрати протягом місяця;

– тільки 12% українців заощаджують на рахунках в банку.

Щоб виявити кінцевих користувачів продукту був використаний принцип класифікації профілів користувачів за «Jobs to be Done» – це спосіб розуміти, які роботи виконують потенційні клієнти, і як ви можете допомогти їм досягти кращих результатів [9]. Таким чином були сформовані профілі користувачів та описані проблеми, які будуть розв'язані за допомогою майбутнього ПЗ:

– відстежування витрат в ритейл магазинах;

– встановлення лімітів на бюджет;

– інструмент для покращення фінансової грамотності;

– централізоване представлення балансу фінансових рахунків в єдиному програмному інтерфейсі;

– автоматична обробка статистики грошових операцій та створення фінансових звітів;

– об'єднання фінансових операцій із різних джерел в одну базу даних;

– створення та збереження інвойсів;

– забезпечення гнучкості програмному продукту для вирішення персональних потреб користувачів;

– можливість працювати з різними типами рахунків, їх групувати та розподіляти по робочих просторам.

Для опису вхідної інформації у вихідну було використано контексту діаграму потоків даних (DFD) в нотації Гейна-Сарсона (рисунок 1).

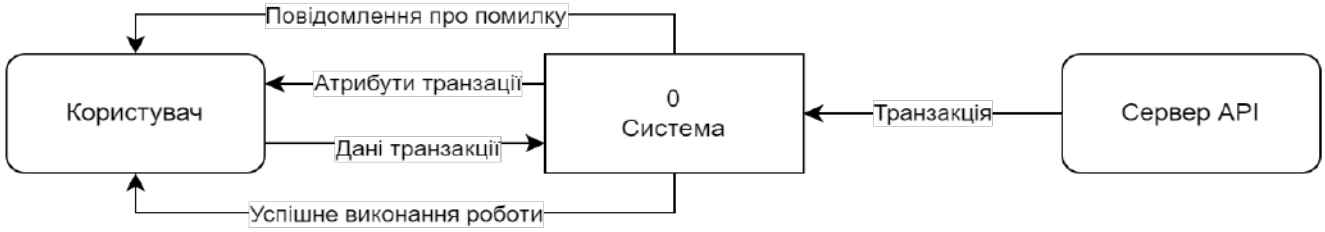


Рисунок 1 – Контекстна діаграма потоків даних мобільного застосунку

Проєктоване рішення містить дві зовнішні сутності, які взаємодіють із системою асинхронно. Дані від користувача можуть надходити за будь-яких умов,

тоді як дані від Сервера, що містить Application Programming Interface (API) надходять за умови доступу до мережі «Інтернет». Декомпозиція діаграми буде проводитися на наступних етапах детального проєктування.

Варто звернути увагу на психологічний аспект, тема кваліфікаційної роботи звучить як «витрати й доходи», мобільний застосунок повинен звучати як «доходи й витрати». Майбутнім користувачам буде приємніше працювати із доходами, ніж з витратами та боргами.

Безпека є найважливішим елементом будь-якої персональної фінансової програми. Користувачі мають бути впевнені, що їх спільна інформація захищена та конфіденційна за допомогою належних заходів безпеки в Інтернеті. Взаємодія користувача з інтерфейсом програми має бути швидка, оскільки програмний продукт має тенденцію до створення великої кількості записів, потрібний маршрут, що веде до результату, з мінімальною кількістю дій. Під час створення застосунку для особистих фінансів важливо, щоб все було просто. Як правило, 3 натискання має бути достатньо, щоб привести клієнта до результатів (наприклад, якщо клієнт хоче перевірити кредитні гроші). Безплатна доступність важлива при старті, надання безплатної програми привертає увагу користувачів і заохочує їх випробувати її. Функцію оплати та інструкцію можна додати пізніше. Програма для персональних фінансів може обробляти велику кількість даних, час відповіді має бути швидким, щоб користувач не втратив інтерес до використання програми. Користувачі завжди цінують доступність цілодобової підтримки клієнтів, яка може допомогти їм розв'язати будь-які проблеми, з якими вони можуть зіткнутися.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для дослідження предметної області було здійснено аналіз ринку і вивчення досвіду провідних фірм-розробників програмного забезпечення. Дані були взяті з відкритих сторінок продукту в магазині «Play Market», сервісу дослідження ринку «AppMagic» та шляхом UI тестування програмного забезпечення.

Аналіз мобільного застосунку «Saldo: Облік доходів та витрат». Основні інтерфейсні вікна подані на рисунку 2.

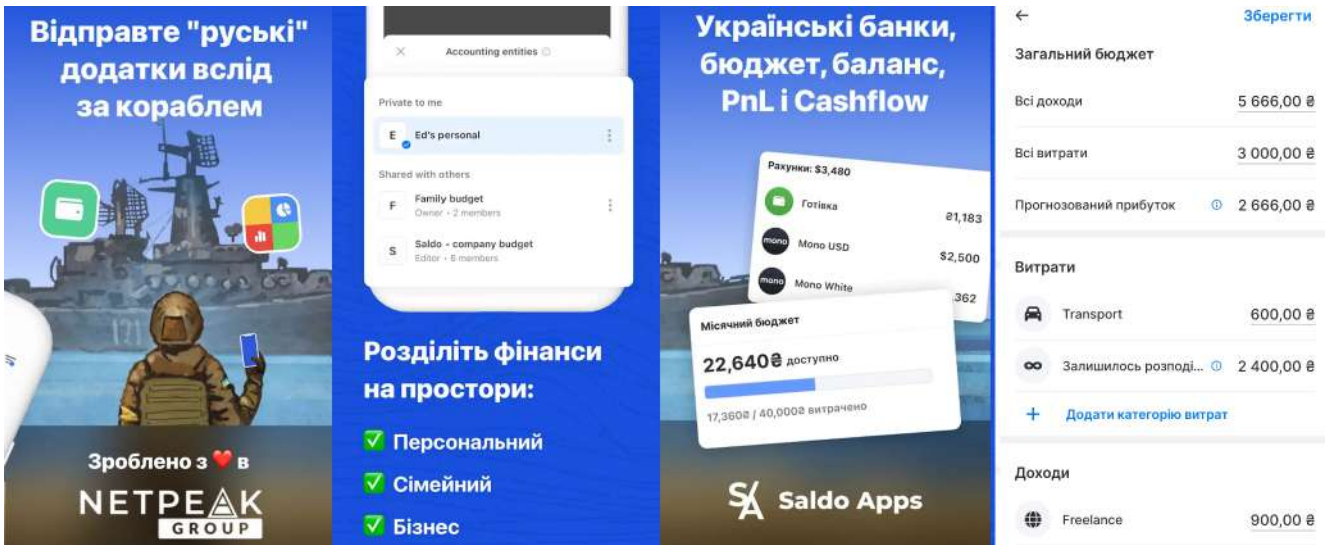


Рисунок 2 – Інтерфейс та обкладинки Saldo

Фірма-розробник «Saldo Apps», дата випуску – квітень 2022, країна реєстрації – США. Кількість завантажень – більша за 50 тис., рейтинг та кількість відгуків: 4.7/5 балів та 634 відповідно. Призначення: для фінансового планування та управління грошима, який робить особисті фінанси легкими та інтуїтивно зрозумілими. Переваги: хороші способи фільтрації даних, можливість створювати автоматичні правила, можна запрошувати людей, з якими ви хочете вести облік, через електронну пошту. Недоліки: інформаційна дошка переповнена даними, щоб додати нову транзакцію потрібно переходити на новий екран, мала кількість українських банків. Особливості: зроблений в Україні, має безпекові функції пін-коду та Touch/Face ID.

Аналіз мобільного застосунку «Wallet – Гроші, бюджет». Фірма-розробник «BudgetBakers.com», дата випуску – квітень 2011, країна реєстрації – Чехія. Кількість завантажень – більша за 10 млн, рейтинг та кількість відгуків: 4.4/5 балів та 310 тис. відповідно. Призначення: менеджер особистих фінансів, створений, щоб допомогти заощаджувати гроші, планувати майбутнє та переглядати всі свої фінансові операції в одному місці. Переваги: система сповіщень, зручний перегляд

бюджету, можливість встановлювати цілі, зручний інтерфейс користувача. Недоліки: мала кількість українських банків, повільно працює функція швидкого введення, перенавантажений елементами інтерфейс програми. Особливості: добре продумана система збирання коштів для досягнення цілей. Основні інтерфейсні вікна подані на рисунку 3.

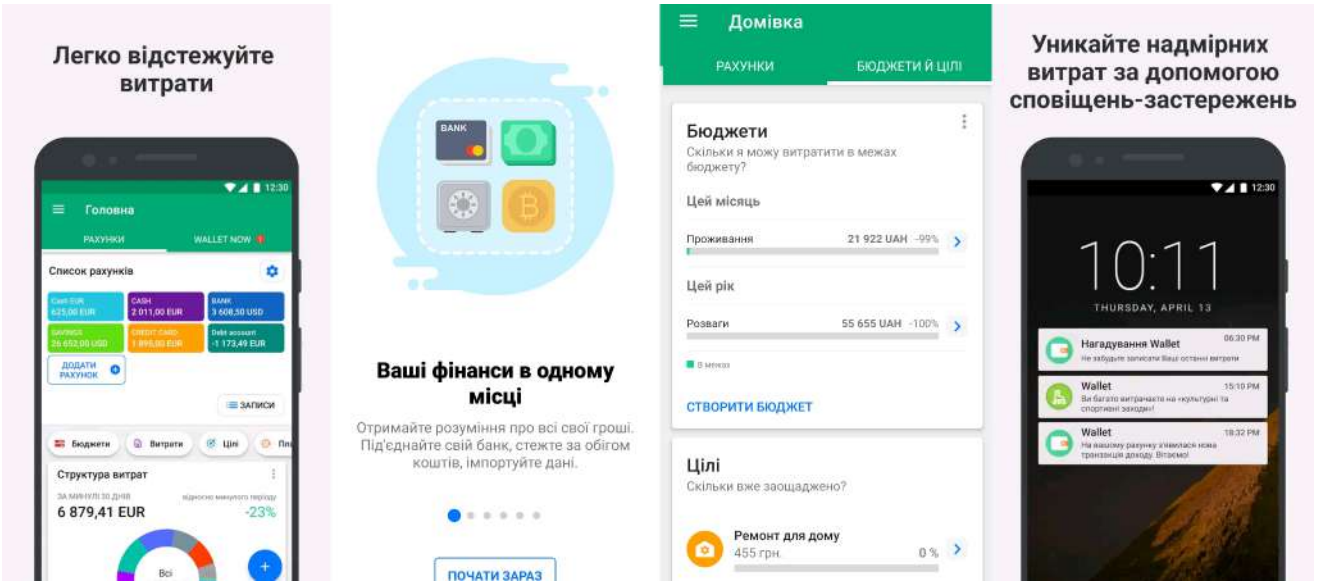


Рисунок 3 – Інтерфейс та обкладинки Wallet

Аналіз мобільного застосунку «Monefy – Розпорядник бюджету». Основні інтерфейсні вікна подані на рисунку 4.

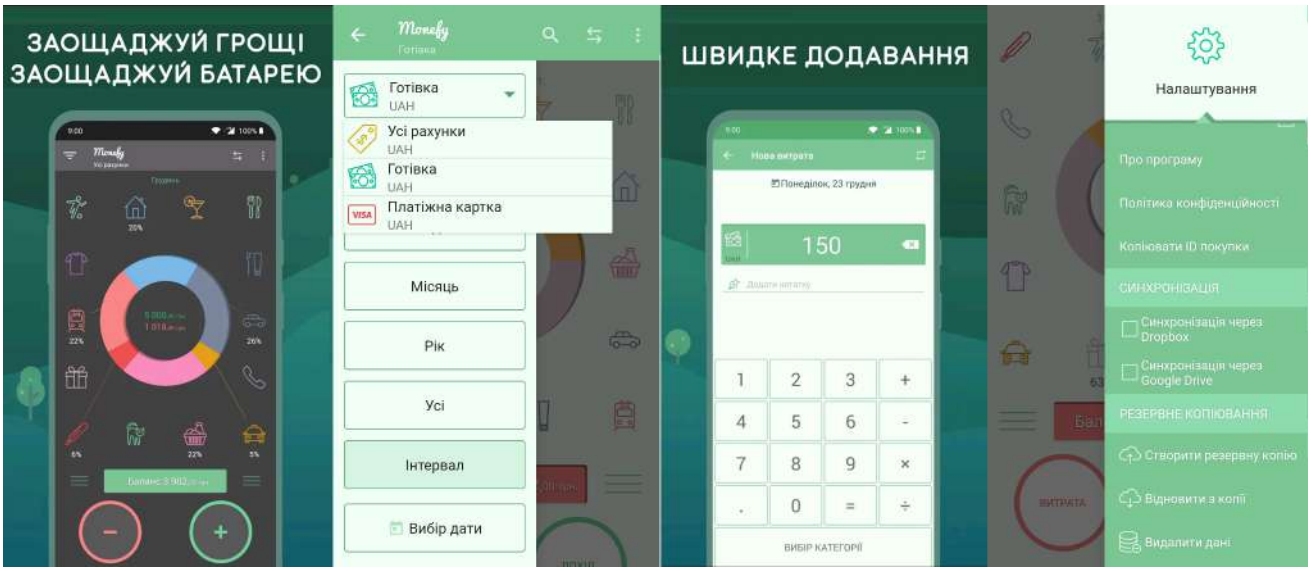


Рисунок 4 – Інтерфейс та обкладинки Monefy

Фірма-розробник «Reflectly ApS», дата випуску – лютий 2014, країна реєстрації – Данія. Кількість завантажень – понад 10 млн, рейтинг та кількість відгуків: 4.2/5 балів та 187 тис. відповідно. Призначення: фінансовий організатор для систематичного відстеження грошей, пропонує просте додавання нових записів щоразу, коли робиться покупка. Переваги: дуже швидка функція додавання, зручна система фільтрування даних, простий у використанні інтерфейс програми. Недоліки: немає функцій керування бюджетом та встановлення фінансових цілей. Особливості: надає функції синхронізації та резервного копіювання даних, після функції вибору категорії автоматично завершує і зберігає операцію, потім закриває робоче вікно програми.

Аналіз мобільного застосунку «Mony: Бюджет, Витрати, Фінанси». Основні інтерфейсні вікна подані на рисунку 5.

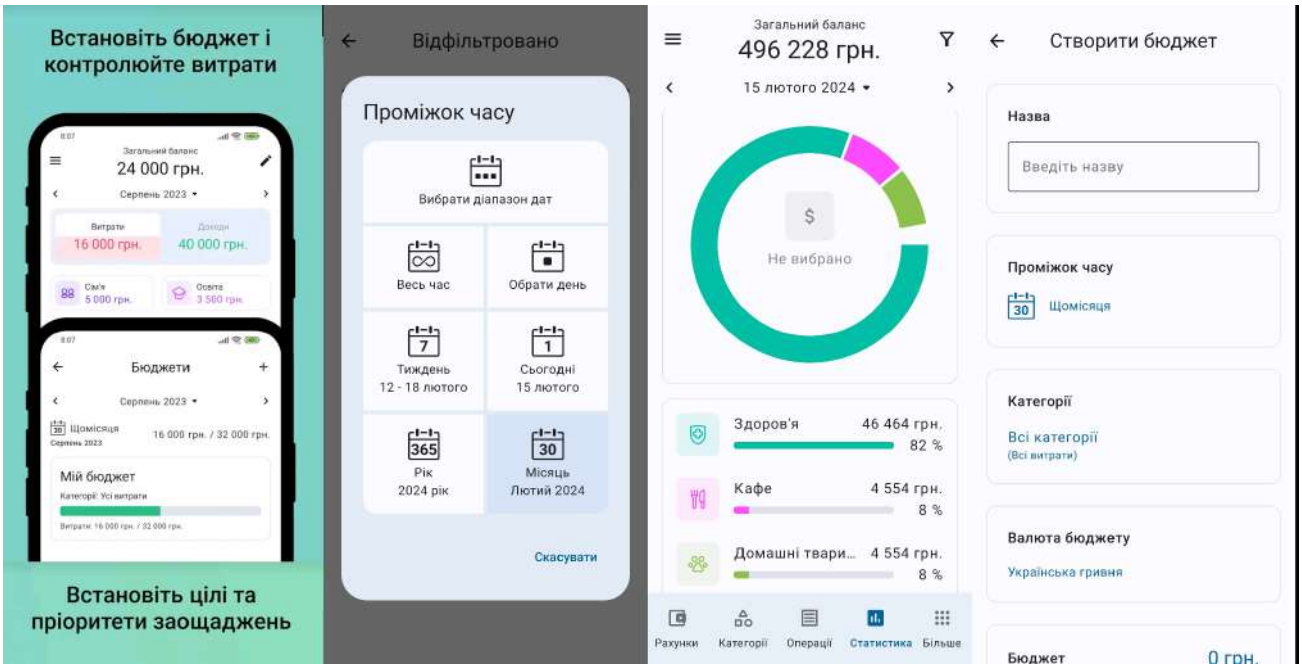


Рисунок 5 – Інтерфейс та обкладинки Mony

Фірма-розробник «jojDevx», дата випуску – листопад 2022, країна реєстрації – Україна. Кількість завантажень – більша за 100 тис., рейтинг та кількість відгуків: 4.4/5 балів та 1.24 тис. відповідно. Призначення: менеджер витрат веде облік ваших доходів і витрат, контролює щоденні витрати та допомагає відстежуйте, куди йдуть фінанси. Переваги: зручні вікна фільтрування даних, грамотно продуманий екран

статистики, великий функціонал для створення власних шаблонів. Недоліки: немає підключення до банківських рахунків, екрани перенавантажені елементами. Особливості: легкість використання екрана статистики, елементи, що рідко використовуються приховані в меню.

Аналіз мобільного застосунку «1Money: облік витрат, бюджет». Основні інтерфейсні вікна подані на рисунку 6.

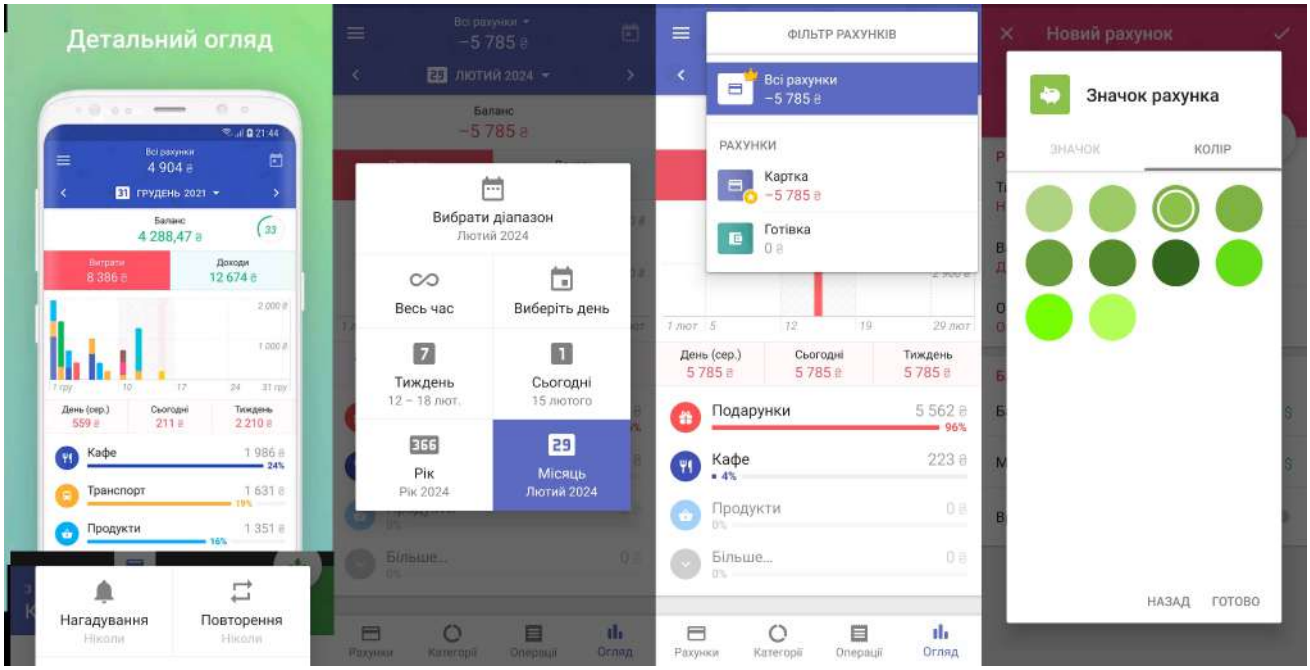


Рисунок 6 – Інтерфейс та обкладинки 1Money

Фірма-розробник «Svyatoslav Vlasov», дата випуску – вересень 2017, країна реєстрації – Російська федерація. Кількість завантажень – більша за 5 млн, рейтинг та кількість відгуків: 4.6/5 балів та 124 тис. відповідно. Призначення: застосунок покаже, куди йдуть гроші, програма, за допомогою якої буде легко відстежувати свої витрати в дорозі, додавання нових транзакцій одним дотиком, потрібно лише ввести суму. Переваги: зручна система створення значків, продумана система легкого вводу, можливість встановити режим повторення для операції. Недоліки: діаграми статистики переповнені елементами, велика кількість обмежень без преміумверсії. Особливості: в меню навігації можна додавати нові елементи.

Аналіз мобільного застосунку «Money Manager Expense & Budget». Фірма-розробник «Realbyte Inc», дата випуску – березень 2013, країна реєстрації –

Південна Корея. Кількість завантажень – більша за 20 млн, рейтинг та кількість відгуків: 4.6/5 балів та 383 тис. відповідно. Призначення: програма для фінансового планування, огляду, відстеження витрат і управління особистими активами для Android, робить управління особистими фінансами таким простим, як пиріг. Переваги: автоматичне фокусування на наступному елементі форми створення запису, можливість прикріплювати фото до витрат. Недоліки: діаграми завантажуються повільніше ніж в аналогічних продуктах. Особливості: наявний перегляд витрат у формі календаря, калькулятор відкривається окремим вікном для швидкого вводу. Основні інтерфейсні вікна подані на рисунку 7.

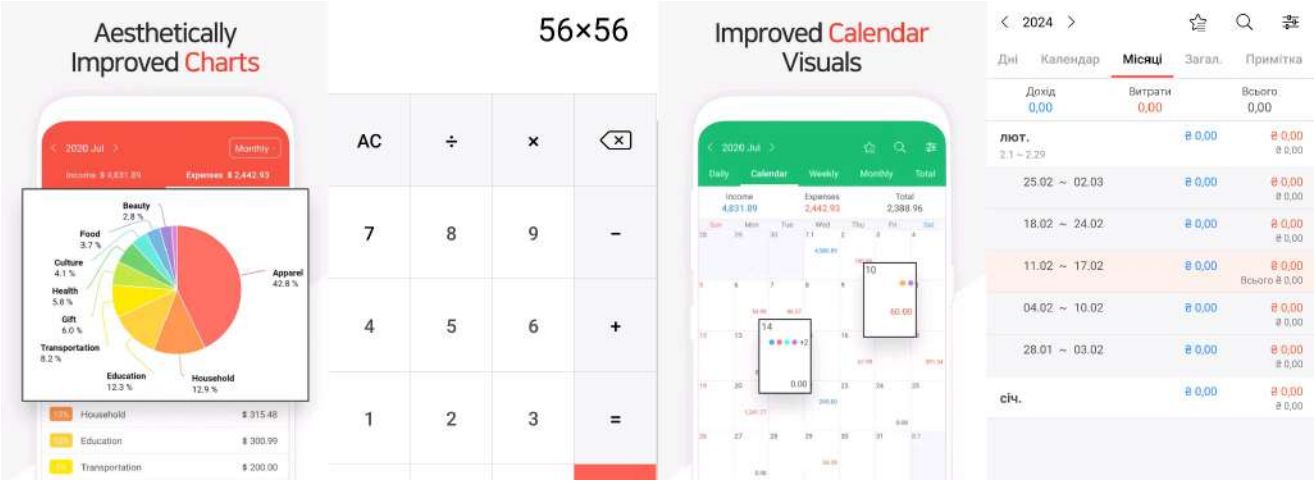


Рисунок 7 – Інтерфейс та обкладинки Money Manager Expense

Після отримання даних дослідження програмних продуктів можна провести порівняльний аналіз всіх розглянутих застосунків для обліку витрат та доходів за різними характеристиками. Фінансові програми часто працюють із великою кількістю даних, тому важливо, як швидко та зручно, їх можна додавати, чи є можливість автоматичної синхронізації операцій із банківських рахунків. Короткі маршрути навігації, можливості персоналізації, універсальний дизайн – все це сильно впливає на потенціал застосунку та оцінку користувачів. Фінансові дані – це чутливі дані, клієнтам програмного продукту важлива їх конфіденційність та наявність надійних механізмів захисту. Перелік недоліків та переваг аналогів програмних продуктів було зведено в порівняльній таблицю 4.

Таблиця 4 – Порівняльна таблиця аналогів програмного забезпечення

Функції / Назва ПП	Saldo: Облік доходів та витрат	Wallet – Гроші, бюджет	Monefy – Розпорядник бюджету	Mony: Бюджет, Витрати, Фінанси	1Money: облік витрат, бюджет	Money Manager Expense & Budget
Облік доходів та витрат	+	+	+	+	+	+
Працює з українськими банками	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Працює з закордонними банками	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Бізнес функції	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Розподіл на простори	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Спільний облік	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Планування бюджету	+	+	Відсутня	+	+	+
Шаблони	+	+	+	+	+	+
Фінансові цілі	Відсутня	+	Відсутня	+	+	Відсутня
Нагадування про платежі	Відсутня	+	Відсутня	Відсутня	Відсутня	Відсутня
Сповіщення	Відсутня	+	Відсутня	+	+	+
Синхронізація	+	+	+	+	+	+
Функції безпеки	+	+	Відсутня	+	+	+
Швидке введення	+	+	+	+	+	+
Інструкція	+	+	+	+	+	Відсутня

На основі даних порівняльної таблиці можна зробити висновки, що перші версії програмного продукту не обов’язково повинні включати функції по роботі з банківськими рахунками, спільного обліку, нагадувань про платежі, розподілу на простори чи бізнес інструменти.

Важливими є функції персоналізації та можливості створювати свої шаблони рахунків, категорій витрат, груп, валют. Всі програмні продукти мали функцію швидкого введення суми операції за допомогою штатних методів програми.

1.3 Визначення вимог до програмного забезпечення та розробка технічного завдання

Після аналізу аналогів мобільних застосунків предметної області було відібрано ряд функцій і обґрунтовано причини такого вибору. Кількість типів користувачів (акторів) становить одиниці. Перелік основної функціональності та їх атрибути подані в таблиці 5.

Таблиця 5 – Перелік основних функцій мобільного застосунку

Назва	Опис	Обґрунтування
Конструктор шаблонів	Містить об’єкти, які можна створювати, редагувати та видаляти, серед цього переліку є: валюти, групи рахунків, рахунки та категорії фінансових операцій.	Можливість персоналізації важлива для користувача, якщо цей процес буде досить легким, то це значно підвищень рівень конкурентоспроможності програмного продукту.
Робочі простори	Можна створювати нові, редагувати та видаляти старі простори, наприклад: особистий, сімейний, бізнес, для друзів.	За допомогою цього, користувачі зможуть розділити своє фінансове життя на окремі групи. Поточний модуль слугуватиме плацдармом для розробки спільного ведення обліку.
Створення транзакцій	Процес створення має два етапи: матриця вибору атрибутів транзакції та швидке введення суми.	Найчастіша дія, яка буде виконуватися при взаємодії із програмою, в основному після заходу в мобільний застосунок.
Перегляд статистики	Список стандартних та розроблених користувачем діаграм з аналітичною таблицею.	Аналітична частина програми, повнота та якість якої на пряму впливає на корисність мобільного застосунку.
Відстеження фінансів	Відображає рахунки із робочого простору та групує їх для зручного перегляду. Показ загальну інформацію, таку як: баланс, номер рахунку, дата закінчення валідності, опис.	Виконує основну роботу програмного продукту для обліку фінансів. Важливим є легка структура інтерфейсу, представлення складних фінансових даних у зручній формі для людей із синдромом дефіциту уваги та гіперактивності (ADHD).

Для моделювання варіантів використання програмного забезпечення було використано діаграму «Use Case» з універсальної мови моделювання (UML), система мобільного застосунку на рисунку 8.

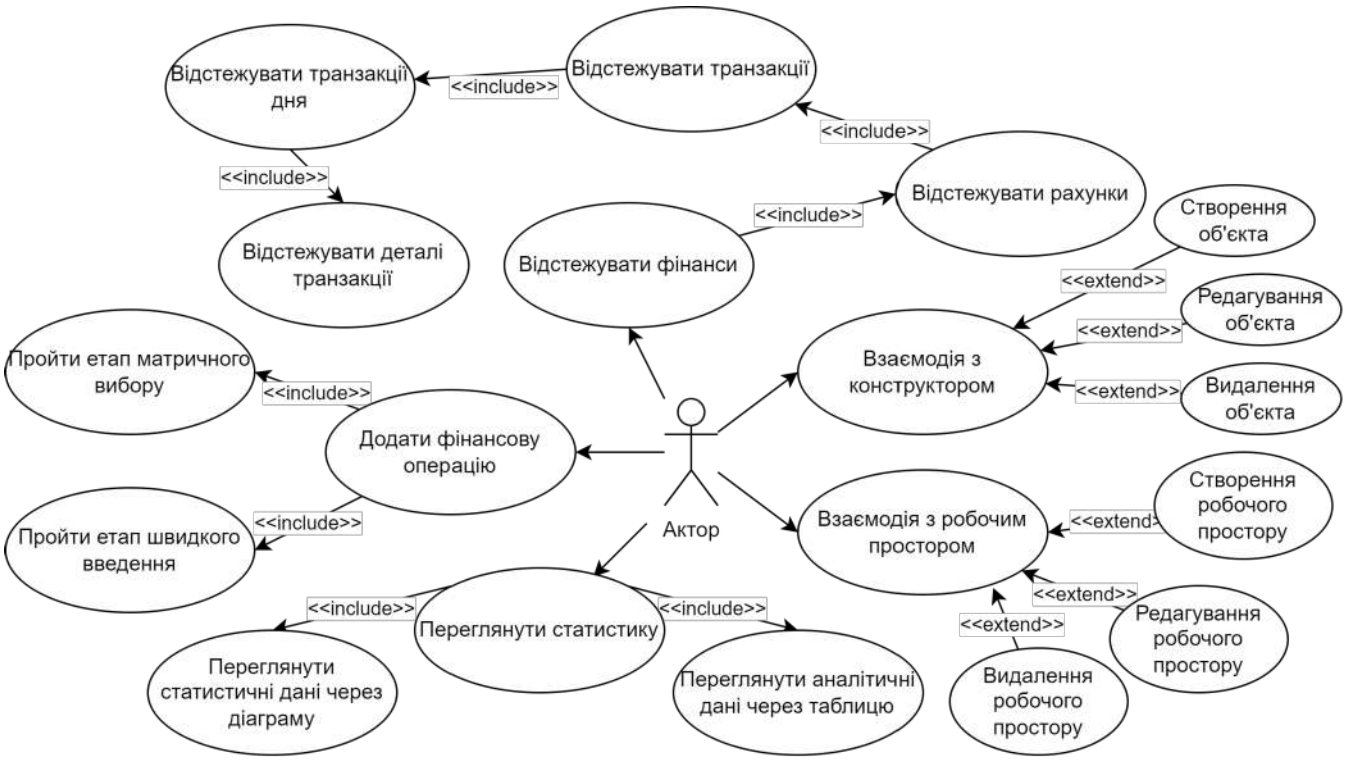


Рисунок 8 – Діаграма варіантів використання мобільного застосунку

Матричний вибір – це механізм, що надає користувачу всі можливі варіанти вибору у формі матриці елементів, які розміщені на одному екрані. Для завершення дії достатньо здійснити одне натискання в програмному інтерфейсі, це найшвидший спосіб ручного вводу інформації.

Швидке введення – це механізм, що дозволяє користувачу вводити суму транзакції за допомогою штатних методів програми, тому що вони швидші за стандартну віртуальну клавіатуру пристрою.

Наведені функціональні вимоги в таблиці 5 та варіанти використання на рисунку 8 потребують великого обсягу робіт, а отже і ресурсів – часових, людських. Тому було прийнято рішення про проєктування цілої програмної системи, а етап реалізації торкнеться тільки частини модулів, які матимуть цінність для користувача та розробника програмного забезпечення.

Розробка модулів відстеження фінансів, створення транзакцій та статистики неможливі без готовності модулів конструктора та робочих просторів, тому їх буде реалізовано в рамках виконання кваліфікаційної роботи. Інша частина функціональних модулів будуть реалізовані частково за наявності відповідних проєктних ресурсів. Перша версія мобільного застосунку повинна отримати наступні функціональні можливості:

– операції створення, перегляду, редагування та видалення для об’єктів конструктора та робочих просторів;

– валідація даних для форм створення та редагування елементів;

– адаптивне меню навігації для малих, середніх та великих за розміром екрана пристрою;

– інформаційні повідомлення для користувача;

– систему діагностики помилок;

– зміна мови інтерфейсу користувача.

Важливо дотримуватися однобічного дизайну інтерфейсу користувача серед всіх програмних модулів. Наприклад, взаємодія з об’єктами конструктора та робочими просторами повинна відбуватися в однаковій формі. Незрозумілість валідації даних у таких формах може стати перешкодою для персоналізації робочого простору, тому програмний продукт повинен забезпечити достатню кількість підказок та інформаційних повідомлень.

Висновки. В даному розділі було здійснено дослідження предметної області, виявлено галузеві проблеми та завдання, які можна розв’язати за допомогою інженерії програного забезпечення, сегментовано потреби користувачів, завершено аналіз ринку мобільних застосунків та його тенденції розвитку, висвітлено переваги та недоліки наявного програмного та інформаційного забезпечення предметної області. Такий збір інформації дозволив сформувати список функціональних вимог, які були відібрані та спроєктовані для розв’язання проблем в галузі фінансів. В результаті було визначено функціональні та нефункціональні вимоги до мобільного застосунку, що розробляється за темою кваліфікаційної роботи, повний список яких наведений в документі «Технічне завдання», що розміщений в додатку А.

2 **ПРОЄКТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ**

2.1 Проєктування структури бази даних

Мобільний застосунок для ведення особистих витрат та доходів згідно з контекстною діаграмою потоків даних (рисунок 1) оперує даними транзакцій та її атрибутами. Дані транзакцій при операції створення є новими й вносяться в базу даних, а при редагуванні читаються. Атрибути транзакцій завдяки можливостям персоналізації можуть створюватися і зберігатися в базі даних для повторного використання. Коли створюється нова транзакція, користувачу надаються можливості для додавання атрибутів до фінансової операції.

Для ефективного зберігання даних мобільного застосунку чудового підходять реляційні бази даних – це тип системи управління базами даних (СУБД), яка зберігає та надає доступ до даних, що взаємопов'язані та організовані у вигляді таблиць. Основні характеристики реляційних баз даних включають наступні.

Структура даних. Дані організовані у вигляді таблиць, де кожен рядок представляє об'єкт, а стовпці представляють атрибути цього об'єкта.

Незалежність даних. Схема даних (структура таблиць та відношень між ними) зберігається окремо від програм, які використовують ці дані.

Мова запитів. Реляційні бази даних використовують стандартну мову запитів SQL для виконання операцій з даними.

Цілісність даних. В реляційних базах даних можна встановити обмеження цілісності, щоб забезпечити точність та послідовність даних.

Транзакції. Реляційні бази даних підтримують транзакції, які дозволяють виконувати кілька операцій як одну логічну одиницю роботи.

Третя нормальна форма (3NF) є важливою частиною нормалізації даних в реляційних базах даних [10]. Вона вимагає, щоб кожен не ключовий атрибут був безпосередньо залежний від первинного ключа. Це допомагає уникнути дублювання даних та забезпечує цілісність даних, що може зменшити обсяг зберігання та покращити ефективність операцій з даними. Якщо база даних не

відповідає 3NF, це може призвести до аномалій вставки, видалення та оновлення, що спричинить стан непослідовності даних.

Спроєктована база даних відповідає третій нормальній формі. База даних мобільного застосунку містить додаткову інформацію в таблиці «DATABASEINFO», де інформацію із колонки «DATA» можна отримати за допомогою ключа «PROPERTY». Всі необхідні таблиці для роботи зображені на логічній ER-діаграмі (рисунок 9).

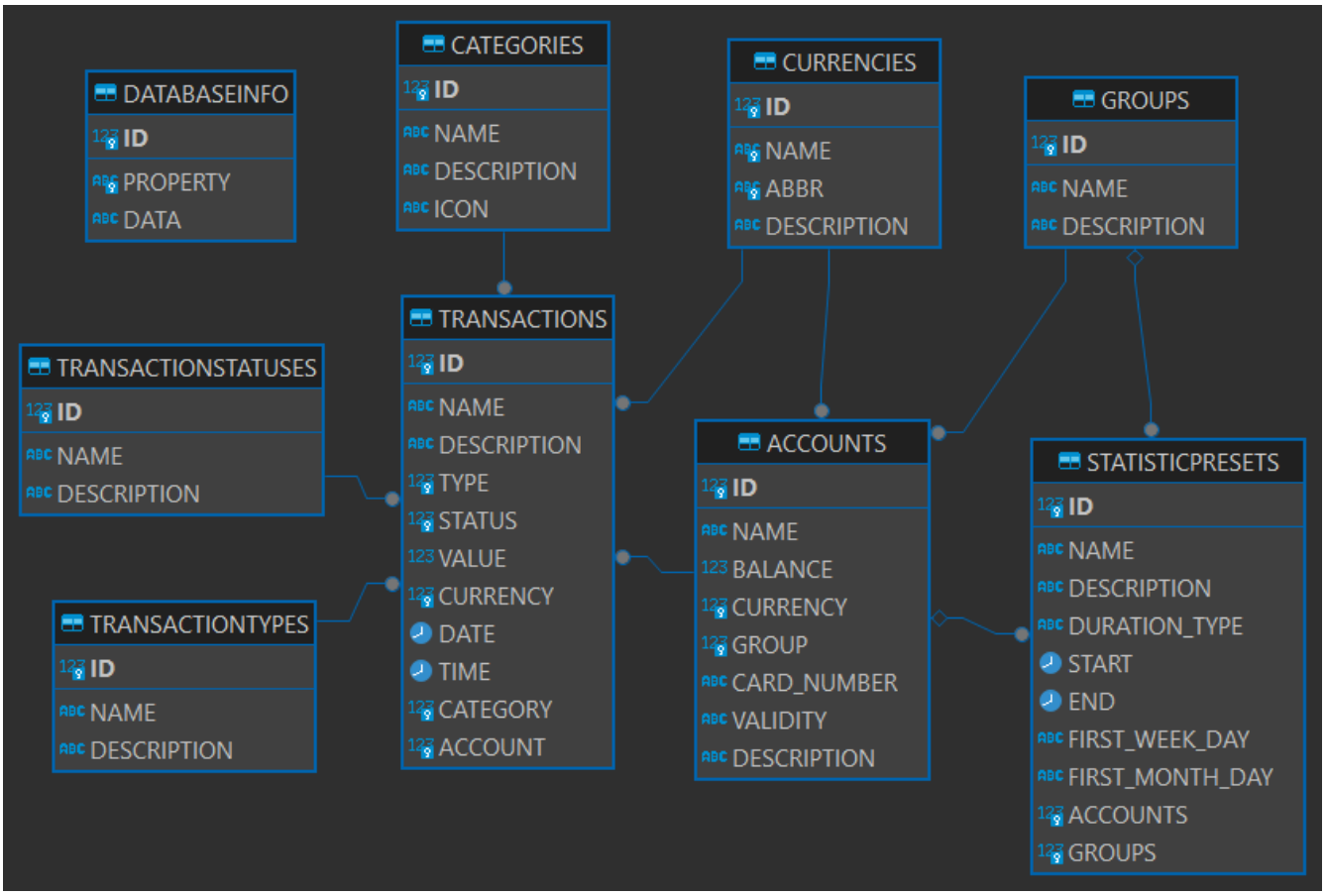


Рисунок 9 – Логічна ER-діаграма бази даних

Таблиця «CATEGORIES» містить колонки імені категорії, її опис та ідентифікатор іконки.

Таблиця «CURRENCIES» має колонки назви валюти, її абревіатуру та опис.

Таблиця «GROUPS» має властивості назви групи та її опис.

Таблиця «TRANSACTIONSTATUSES» містить колонки назви статусу та його опис для користувача.

Таблиця «TRANSACTIONTYPES» має властивості імені типу транзакції та його опис для користувача.

Таблиця «TRANSACTIONS» має колонки імені та опису транзакції, типу та статус, значення і валюту, дату і час проведення, категорію і рахунок, з якого було проведено фінансову операцію.

Таблиця «ACOUNTS» містить колонки імені рахунку та його опису, балансу, групу рахунку, номер карти й термін придатності.

Таблиця «STATISTICSPRESETS» містить колонки, що відповідають за збережені налаштування для діаграм. Серед них ім’я та опис, початок та кінець періоду відбору даних або тип тривалості (останній місяць, пів року тощо), перший день тижня та місяця, рахунки та групи.

Рахунки та групи будуть відображати безпосередньо в інтерфейсу, деталі транзакцій повинні поступово деталізуватися залежно від призначення екрана. Дані транзакцій відіграють ключову роль, оскільки є джерелом для створення аналітичної статистики та побудови фінансових діаграм.

2.2 Проєктування інтерфейсу користувача

Мобільний застосунок безпосередньо пов’язаний із фінансовими даними, інтерфейс програми відповідає за їх відображення. Багато чисел, фінансових звітів, статистика це дані, які є важкими для сприйняття, тому їх компактній і легкість читання є головними в процесі проєктування інтерфейсу програми. Під час процесу проєктування були використані наступні рекомендації.

Основний принцип – це універсальний дизайн, який враховує різноманіття людської природи. Мінімалістичний із явно вираженим одним яскравим кольором, задній фон темний. Дизайн, графи навігації, маршрути дій, структура представлення складних даних – повинні бути оптимізовані для людей із розладом гіперактивності з дефіцитом уваги (ГРДУ) / синдромом дефіциту уваги та гіперактивності (СДУГ, англ. ADHD).

Універсальний дизайн – це дизайн предметів, середовища, програм та послуг, що забезпечує їхню максимальну придатність для використання всіма особами без необхідної адаптації чи спеціального дизайну. Програмний інтерфейс реалізований із використанням такого дизайну можна схарактеризувати трьома правилами. Чітка структура навчального й ознайомчого процесу із застосунком та його використання допомагає людям з ГРДУ краще орієнтуватися [11]. Візуальні підказки та направні знаки можуть допомогти людям з ГРДУ зосередитися на завданні. Використання технологій, таких як нагадувань або організація завдань.

Розробка мобільних застосунків для людей із синдромом дефіциту уваги та гіперактивності передбачає розуміння їхніх унікальних потреб і поведінки.

Легкість навігації. Люди з СДУГ можуть мати проблеми з організацією та концентрацією уваги. Створення чітких, інтуїтивно зрозумілих навігаційних меню та мінімізація факторів, що здатні відвернути увагу з екрана сильно покращить роботу з інтерфейсом. Це може означати використання послідовного макета і колірної гами, зменшення кількості елементів, на які можна натиснути, або включення клавіатурних скорочень для часто використовуваних дій, можливість швидко повторювати останні виконані дії.

Допоміжні інструменти та функції. Надання інструментів та функцій, які відповідають їхнім унікальним потребам. Наприклад, програма для календаря може містити нагадування та сповіщення, щоб допомогти людям з СДУГ не відставати від графіка. Застосунок для нотаток може містити такі функції, як перетворення голосу в текст або розпізнавання рукописного тексту, щоб полегшити швидке фіксування ідей.

Інтерфейси, що захоплюють. Люди з СДУГ можуть бути більш схильні до взаємодії з візуально заохочувальним, інтерактивними або гейміфікованими інтерфейсами. Варто включити такі елементи, як анімація, мікровзаємодія або петлі зворотного зв'язку, щоб зробити досвід цікавішим і мотивувальним. Однак дотримуватися балансу між залученням і надмірною стимуляцією, оскільки надмірна кількість візуальних чи аудіальних даних може бути приголомшливою. Важливість та опис гейміфікації вже згадувалася в першому розділі.

Хороший UX-дизайн враховує потреби, цілі та поведінку користувачів, а також технічні обмеження платформи. Важливо також залучати людей з СДУГ до процесу розробки, щоб переконатися, що мобільний застосунок відповідає їхнім потребам і вподобанням.

В даній пояснювальній записці викладена частина вимог до дизайну та інтерфейсу користувача, повний обсяг зазначений в технічному завдані. Для реалізації програмного інтерфейсу варто використати інструменти та технології, що дозволяють змінювати основні складові дизайну мобільного застосунку, наприклад компонентний підхід, де при зміні головного елемента, змінюються всі дочірні. Проєктування інтерфейсу користувача варто почати зі створення схеми переходу між екранами (позначено закругленими прямокутниками) та діалоговими вікнами (еліпси) мобільного застосунку, рисунок 10.

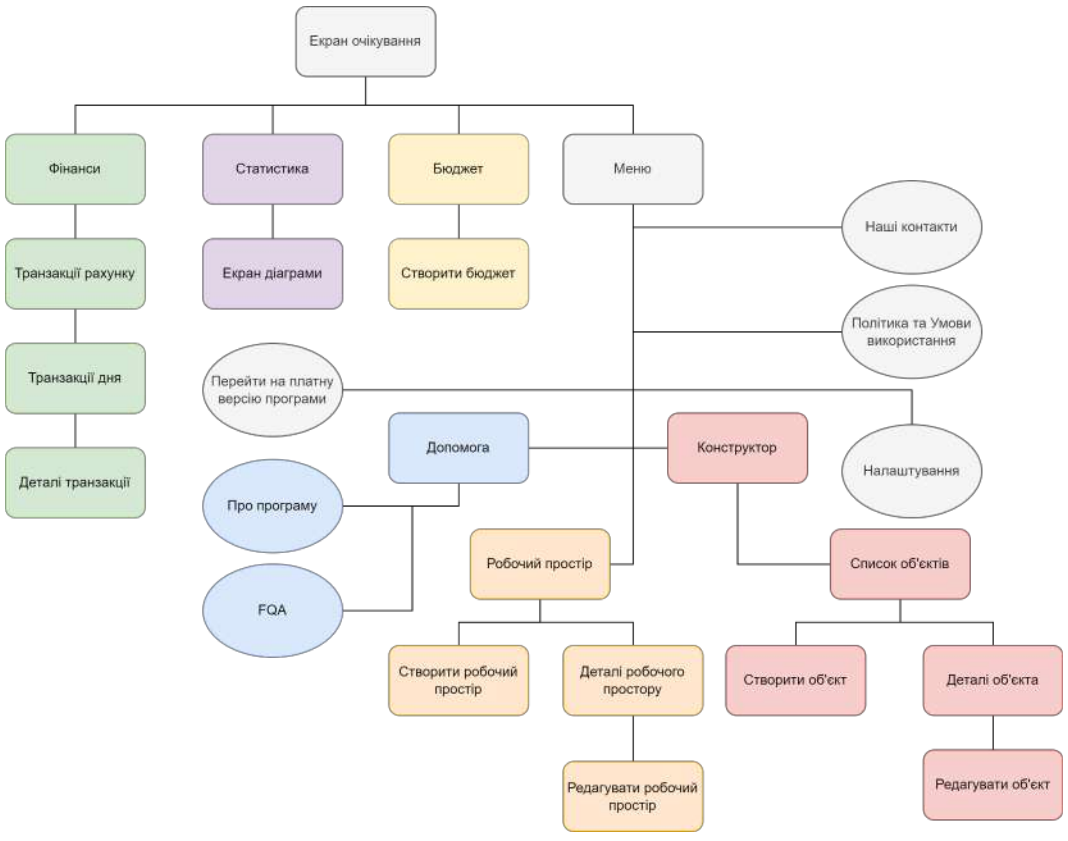


Рисунок 10 – Схема переходу між екранами мобільного застосунку

На даній схемі зображено 19 екранів та 6 вікон діалогів. Кожний екран містить кнопку, що повернутися на попередній, а кожне діалогове вікно містить

кнопку для закриття. Популярною серед користувачів також є навігація жестами, що надає можливість швидко пересуватися на попередній екран. Для послідовності та одноманітності дизайну було вибрано палітру кольорів та текстових шрифтів для різних типів заголовків, рисунок 11.

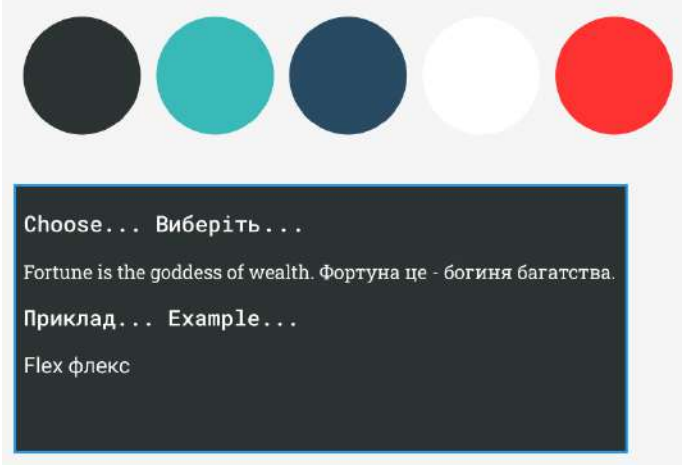


Рисунок 11 – Колірна гама та шрифти

Для створення прототипу дизайну було використано інструмент Figma – це інструмент для спільного проєктування інтерфейсів, який дозволяє командам проєктувати, узгоджувати та створювати продукти разом. Він надає платформу для проєктування, створення прототипів, розробки та тестування продуктів в одному просторі. Для побудови інтерфейсу користувача спочатку було створено компоненти, їх анімація при натисканні зображена на рисунках 12-14. На рисунку 12 показано верхній заголовок екрана та нижнє меню.

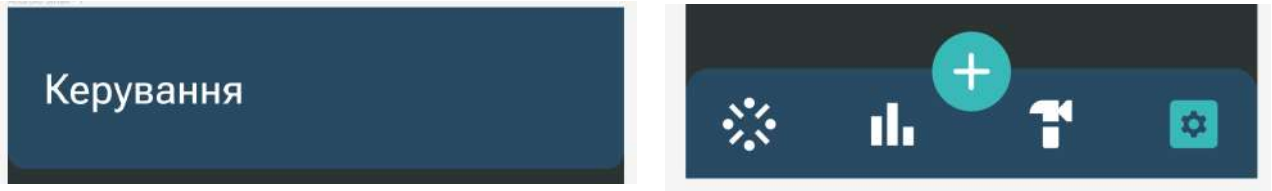


Рисунок 12 – Заголовок екрана та меню навігації

На рисунку 13 зображено кнопки, що керують переходами на екрани та діалоги, їх дизайн зберігається, а текстове наповнення та іконка буде змінюватися залежно від доступних операцій для користувача. Кнопка повернутися назад буде

доступна для всіх екранів, що починаюся з другого рівня вкладеності. Також варто передбачити спеціалізовані кнопки дій, які будуть привертати уваги, наприклад для покупки платної версії мобільного застосунку.

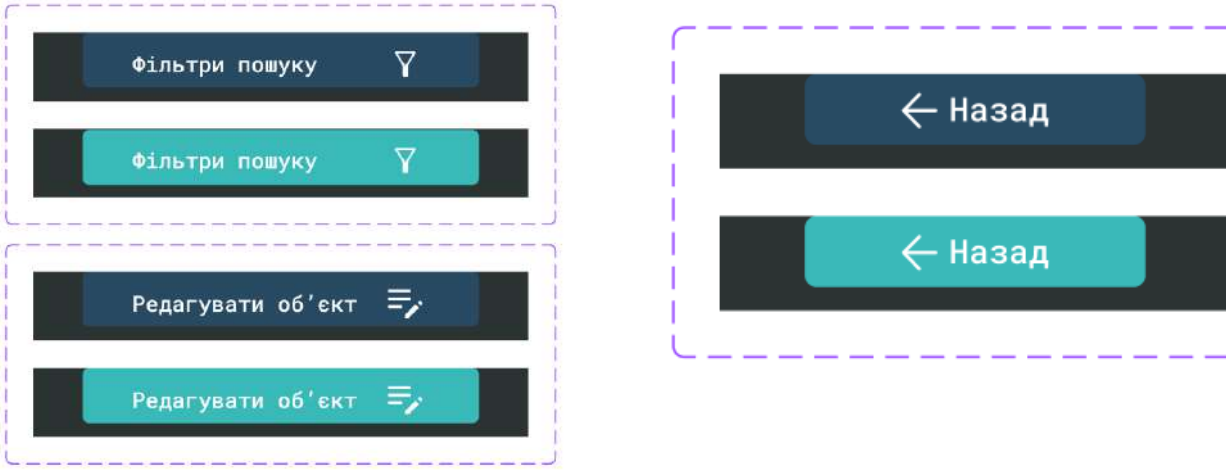


Рисунок 13 – Компоненти кнопок

На рисунку 14 зображено карточку рахунку (зліва), що містять інформацію про нього: ім’я рахунку, валюту, баланс, номер карти та термін придатності відображаються за наявності, оскільки мобільний застосунок дозволяє також створювати офлайн рахунки. Даний компонент буде присутній на екрані фінансів. Карточка (справа) є складовою можливостей персоналізації й буде використовуватися на екрані списку об’єктів. Користувачу пропонується відредагувати наявні об'єкти або створити нові. Серед доступних: валюти, групи, рахунки та категорії транзакцій.

Рисунок 14 – Динамічні компоненти списків

На рисунку 15 зображено зліва екран фінансів, посередині транзакцій рахунку, а справа екран транзакцій дня. Дані екрани містять відповідно групи та їх рахунки, транзакції згруповані по днях та категоріях, коротку інформацію про транзакції. Для відкриття екрана деталей транзакції достатньо натиснути на неї. Користувачу пропонують послідовну деталізацію даних, починаючи із загальних цифр балансу рахунку, чим глибше він занурюється в граф навігації, тим дрібнішу інформацію він отримує, аж до деталей транзакції.

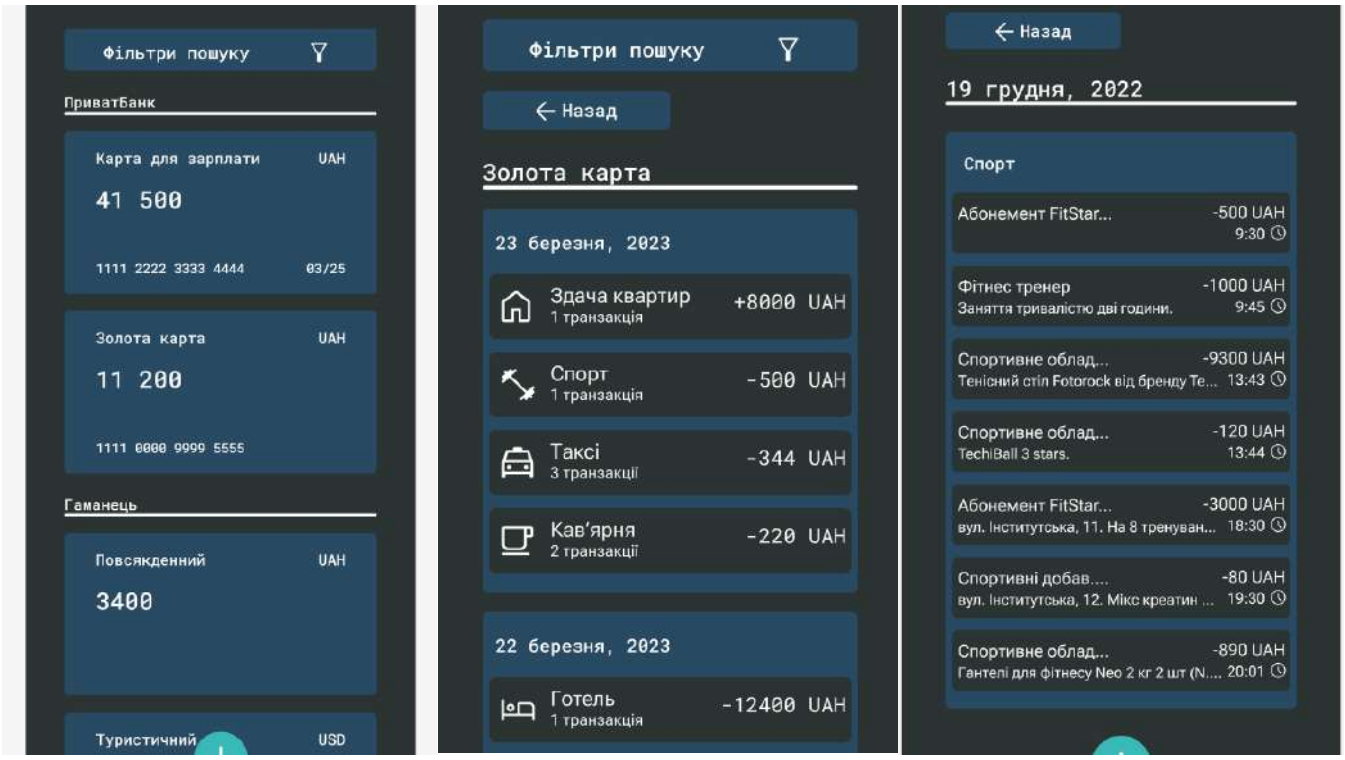


Рисунок 15 – Екрани фінансів

Важливо також надати користувачу підтримку при отриманні помилки чи неможливості завершити дію, наприклад, якщо введені дані у формі не пройшли валідацію, або видалення об’єкта пошкодить цілісність бази даних. Мобільний застосунок передбачає валідацію двох типів. Перший, під час введення, користувачу надаються підказки, а також розширені повідомлення при наявності помилок. Друга валідації відбувається вже під час виконання транзакції до бази даних, якщо вона повертає помилку, користувач обов’язково повинен бути повідомлений про неї. Варто також зазначити, що діалогові вікна використовуються для елементів з якими рідко взаємодіє користувач, вони

відкриваються поверх того екрану, до яких прикріплені та не можуть мати дворівневих вкладень. Це значно зменшує шлях маршруту навігації.

На рисунку 16 зображені екрани керування та форми створення об’єктів (валюти тощо). Екрани з формами заповнення даних передбачені для функціональності конструктора та робочого простора. В обох випадках можна подивитися список наявних елементів, відкрити їх деталі та відредагувати.

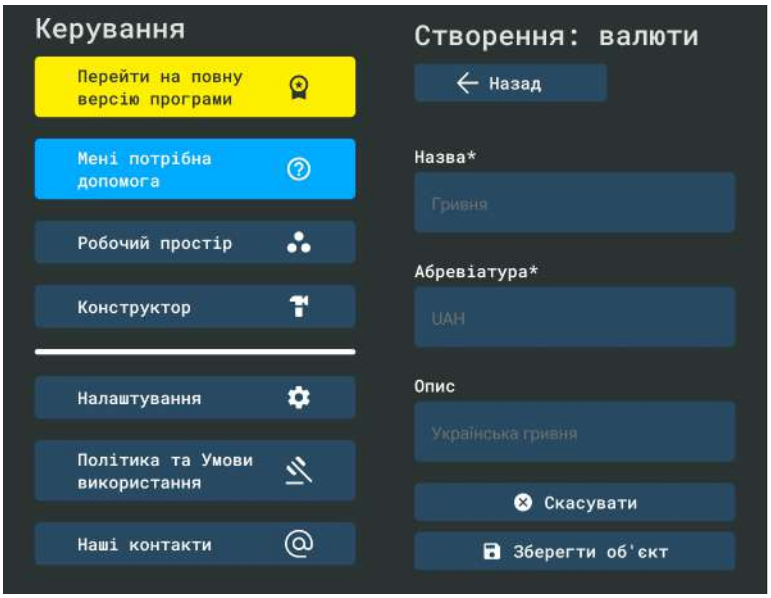


Рисунок 16 – Екран керування та форми конструктора

Завдяки інструменту Figma було створено інтерактивний прототип для перевірки гіпотез, чи зручно буде користуватися навігацією, чи достатніх розмірів інтерактивні елементи інтерфейсу. Створений інтерфейс користувача відповідає представленій діаграмі варіантів використання. Отримані результати будуть використані при реалізації компонентів інтерфейсу в наступному розділі.

2.3 Розробка алгоритму роботи мобільного застосунку

При запуску мобільного застосунку користувач бачить його логотип і назву. У фоновому режимі використовуючи багатопотоковість відбувається запуск підготовчих процесів, серед них:

– підключення модуля збереження інформації для налагодження (створення програмних звітів);

– створення екземпляра класу, що керує робочою текою програми;

– ініціалізації механізму баз даних DataStore (підтримується Android системою за умовчування, прийшов на зміну застарілому Preference) [12];

– створення класу контексту для отримання локалізованих повідомлень;

– завантаження мови інтерфейсу зі збережених налаштувань користувача;

– очищаємо застарілі файли збору інформації;

– збираємо інформацію про пристрій користувача;

– визначаємо місце збереження бази даних;

– налаштовуємо і підключаємо базу даних, здійснюється тестування;

– налаштовуємо повідомлення, що спливають.

Лише після успішного виконання всіх завдань, користувач має змогу повноцінно користуватися мобільним застосунком. Більшість завдань, що не мають важливості для інтерфейсу користувача і затримують його час до взаємодії можна оптимізувати, і виконувати вже після появи головного екрану. Поява інтерфейсу зрозумілою мовою для користувача показує прогрес навіть, якщо відсутня актуальна інформація із бази даних. Пошук необхідних даних може зайняти декілька секунд, тому важливо, щоб користувач мав розуміння про поточний стан готовності програми для використання. Основною та повторювальною дією, яка буде виконуватися в рамках програмного забезпечення є створення транзакції. Діаграма активностей подана на рисунку 17 із використанням нотацій від SysML.

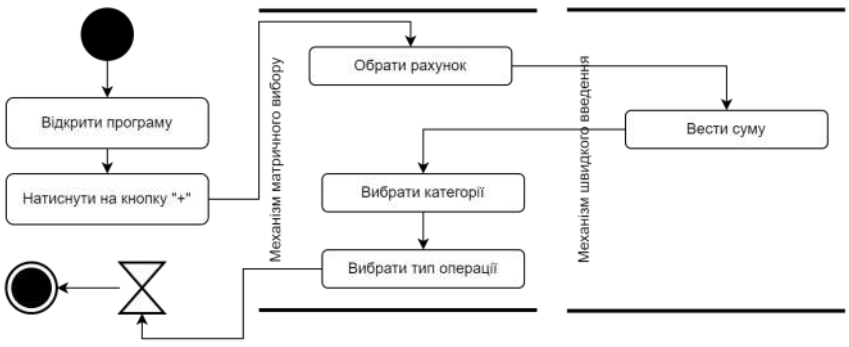


Рисунок 17 – Діаграма активностей створення транзакції

Оскільки ця дія може виконуватися десятки раз за день, то важливо, щоб це було максимально швидко. Тому в даному алгоритмі використовують механізми матричного вибору та швидкого введення, про які згадувалися в підрозділі 1.3. Основна робота алгоритму починається з місця, коли користувач натискає на кнопку плюса (створити транзакцію), тоді у вигляді матриці шириною 2 елементи (для пристроїв із великим екраном 3) завантажують доступні рахунку. За наявності більше ніж однієї групи рахунків, відбувається спочатку вибір поміж них, а далі надаються рахунки, що залишилися. Введення суми відбувається через цифрову клавіатуру, що складається із чисел від 0 до 9 та знака крапки, наявна кнопка переходу до наступного кроку. Вибір категорій відбувається аналогічним чином. Якщо деяка категорія містить підкатегорії, то відкривається додатковий екран із кнопкою завершення дії. Якщо користувач натискає продовжити, то переходить до вибору типу транзакції, який визначає чи введена сума буде прибутком, чи витратою, користувач має обрати серед цих двох або більше опцій. Оскільки це фінальна операція, процес створення транзакції завершується автоматично.

Доцільно здійснити навігацію користувача на екран деталей транзакції, де він зможе переглянути остаточний результат. Якщо потрібно внести зміни, то він зможе натиснути кнопку для редагування. Всі екрані діалоги мають кнопку зліва для повернення на попередній крок, по центру знаходиться кнопка для внесення детальної інформації, справа кнопка про перехід до наступного кроку. Рахунки та їх групи будуть відображатися в першу чергу, тому їх завантаження із бази даних в оперативну пам’ять повинно відбуватися найшвидше. Далі не варто очікувати послідовних дій від користувача, а заздалегідь відправити запис для отримання решти інформації із бази даних та зберегти її в приватні структури даних. Такий підхід дозволить зробити дію створення нової транзакції максимально швидкою. Варто зазначити, що більшість операцій здійснюється завдяки одному натисканню, в найкращому випадку тривалість операції дорівнюватиме шести натисканням та часу введення суми транзакції. Для середнього випадку кількість натискань дорівнює дев’яти.

Мобільний застосунок має великий функціонал персоналізації, серед них і можливість створення нових об'єктів, наприклад валюти. Розглянемо сценарій, де користувачу потрібно змінити абревіатуру для деякої наявної в робочому просторі валюти. Діаграма активностей зображена на рисунку 18.

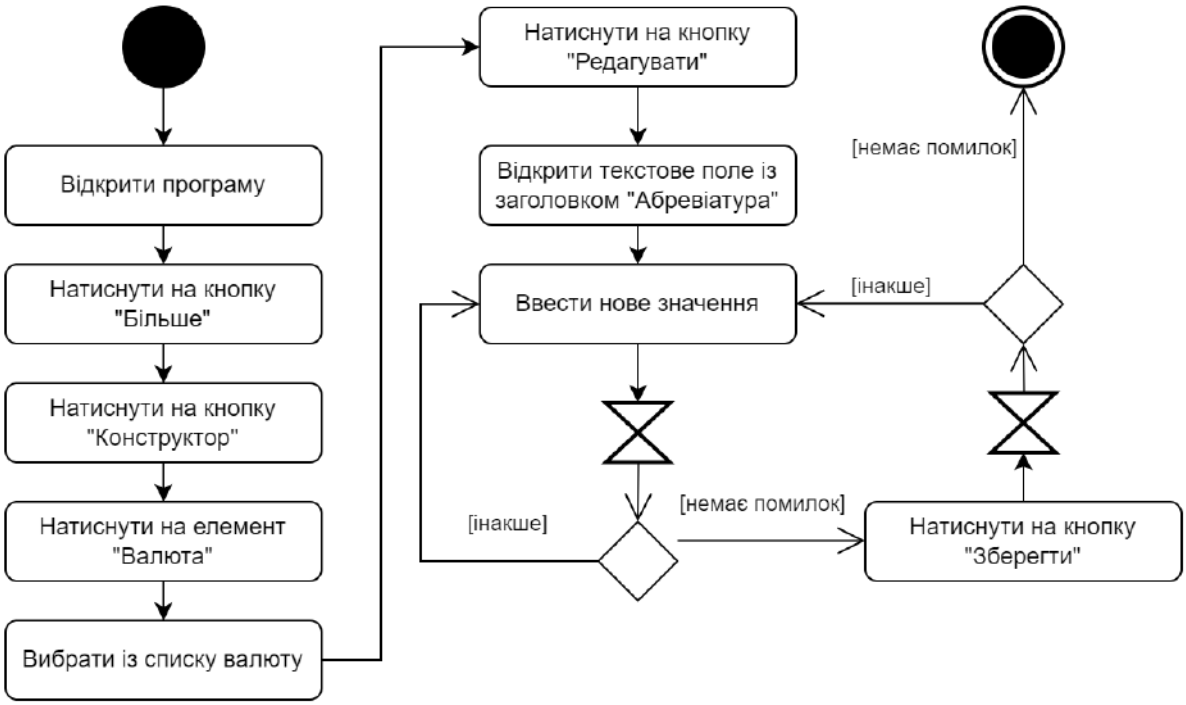


Рисунок 18 – Діаграма активності редагування об’єкта валюти

Починаючи від відкриття мобільного застосунку, користувач без зупинки йде до списку валют, де йому необхідно буде знайти бажаний елемент серед списку, що завантажується поступово. При великій кількості елементів списку, варто показати приховану кнопку фільтрів пошуку. Коли користувач знаходить бажаний елемент він натискає на нього, а також на кнопку редагування в новому вікні. Далі перед ним з’являється форма редагування, він бачить поле із заголовком абревіатур, а також старе значення. При зміні значень, нові дані проходять перевірку на валідацію, якщо значення поля не підходить під визначені вимоги, користувач буде повідомлений спеціальним сповіщенням. Весь процес редагування чи внесення даних супроводжується підказками для текстових полів. Якщо дані проходять перевірку на валідність, кнопка збереження змін буде активна, в іншому випадку, вона є недоступною для взаємодії. Наступний процес валідації

відбувається вже на програмній частині бази даних. Надіслані дані можуть знову не пройти валідацію, про що буде сповіщений користувач. При виникненні помилок користувач завжди отримує достатньо інформації. Після завершення сценарію, користувача перенаправляють на екран деталей обраної валюти. Він знову може переглянути всі атрибути, побачити зміни та за потреби знову почати процес редагування. Також на цьому етапі доступна кнопка видалення.

Іншою потужною функцією персоналізації є робочі простори. В межах одного такого робочого простору користувач має змогу вести облік доходів та витрат окремо від інших даних. Також кожний простір має власні атрибути транзакцій: валюти, рахунки, групи рахунків та категорії. Для зміни робочого простору базовий маршрут займає чотири натискання. Оскільки ця дія для великих фінансових рахунків може часто повторюватися, був розроблений шлях альтернативного способу із використанням кнопки верхнього меню, тоді процес займає лише три натискання. Щоб почати процес створення нового простору достатньо трьох натискань, а для видалення чотири. Діаграма активностей зміни робочого простору для короткого маршруту зображена на рисунку 19.

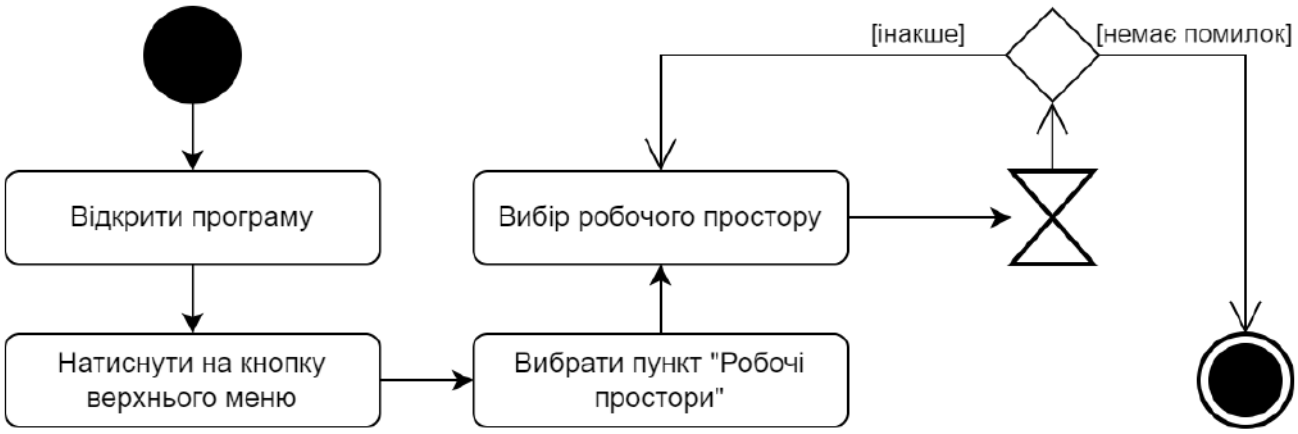


Рисунок 19 – Діаграма активності зміни робочого простору

Робочий простір – це насамперед інформація про місце знаходження бази даних з якою будуть проводити операції. Всі фінансові операції залежать від бази даних, не можливо зберегти щось чи відобрази без підключення до джерела інформації. Список наявних робочих просторів потрібно тримати завжди в

оперативній пам’яті, а також їх назви, оскільки почергове з’єднання до кожної із баз даних задля отримання властивостей вимагає багато часу. Даний динамічний список повинен формуватися при запуску мобільного застосунку. При подіях монтування чи від’єднання карти пам’яті або інших накопичувальних пристроїв, список повинен оновлятися актуальними робочими просторами. Варто зазначити, що мобільний застосунок надає можливість користувачу вибирати тип пам’яті (внутрішня пам’ять пристрою або зовнішня) на якій варто зберігати базу даних із всією інформацією. Дана опція при створенні робочого простору вибирається самостійно, де більше вільного місця для даних, та пам’ять буде використовуватися. Якщо відвідати екран деталей робочого простору, то там знаходитиметься кнопка для зміни місця збереження. Будь-які помилки, що можуть виникнути через нестачу пам’яті для запису даних обробляються програмою, а також користувач отримає докладні інструкції щодо їх усунення.

Мобільний застосунок не потребує особливих математичних алгоритмів для досягнення результатів роботи. Весь процес взаємодії із програмним продуктом зводиться до внесення фінансової інформації в базу даних, а потім послідовне її відображення через інтерфейс користувача. Тому зручність використання, швидкість виконання та продуктивність дій є головними метриками оцінки якості мобільного застосунку. Швидкодія неможлива без використання ефективних структур даних [13] та розуміння розробником нотації Ландау [14].

2.4 Аналіз та вибір технологій і методів реалізації застосунку

Після опрацювання результатів дослідження предметної області було прийнято рішення про створення нативного мобільного застосунку для ОС Android, оскільки це дозволить надати користувачу більш функціональний продукт та ефективніше використовувати програмні ресурси пристрою, на якому буде працювати програма. Для реалізації програмного забезпечення доступні наступні технології та методи реалізації.

Основні мови програмування, які використовуються для розробки нативних застосунків Android є Kotlin/Java. Kotlin є більш сучасною та безпечною мовою [15], але Java все ще широко використовується. Kotlin має більш виразний та зручний синтаксис, що робить код більш читабельним та зрозумілим. Kotlin надає безпечніший код і запобігає деяким помилкам, які можуть виникнути при використанні Java. Був розроблений як мова, сумісна з Java, але з розширеним набором функцій і синтаксисом. Kotlin був офіційно підтриманий Google для розробки мобільних застосунків під платформу Android. Разом з Kotlin рекомендовано використовувати сучасний інструментарій для створення нативного інтерфейсу для Android – Jetpack Compose. Він спрощує та прискорює розробку інтерфейсу, включаючи адаптацію до будь-якого формфактора – від смартфонів, складаних пристроїв та планшетів до телевізорів та пристроїв, що носяться.

Extensible Markup Language – розширювана мова розмітки вже давно є фундаментальною частиною розробки застосунків для Android [16]. Файли XML-макетів використовуються для визначення структури та зовнішнього вигляду інтерфейсу користувача. До переваг можна віднести наступне:

– середовище інтегрованої розробки Android Studio надає візуальний XML-редактор, який спрощує процес розробки макетів інтерфейсу користувача;

– XML дозволяє чітко розділити дизайн і логіку інтерфейсу користувача;

– можливість повторного використання XML-макетів у різних видах діяльності та фрагментах, що сприяє підвищенню ефективності та узгодженості коду програми.

До недоліків можна віднести наступне:

– схильність до багатослівності, XML-макети можуть стати складними, особливо для складних дизайнів інтерфейсу користувача;

– обмежена виразність, XML-макети мають обмеження у вираженні складної поведінки інтерфейсу або анімації без зв'язку з кодом Java/Kotlin.

Jetpack Compose використовує можливості декларативного програмування інтерфейсу користувача [17,18], його перевагами є:

– декларативний інтерфейс, Jetpack Compose дозволяє описувати інтерфейс як функцію поточного стану програми, що робить його більш декларативним;

– менше шаблонів, з Jetpack Compose розробник пише менше шаблонного коду, оскільки елементи інтерфейсу є частиною коду на Kotlin.

Недоліками Jetpack Compose є те, що він є відносно новим і може мати більш круту криву навчання для розробників, які звикли до XML-макетів.

Варто також зазначити, що Jetpack Compose є майбутнім напрямком розвитку користувацького інтерфейсу Android, рекомендованим Google. Однак XML-макети все ще залишаються життєздатним варіантом, особливо для проєктів з обмеженими термінами або для розробників, яким зручніше працювати з імперативною парадигмою програмування.

Розглянемо Kotlin/Jetpack Compose більш детально (таблиця 6) в порівняні із його найближчим конкурентом Flutter – це фреймворк для розробки мобільних застосунків, створений компанією Google. Він використовує мову програмування Dart і дозволяє розробникам створювати високопродуктивні, візуально привабливі додатки як для Android, так і для iOS [19,20]. Віджети Flutter враховують усі критичні відмінності платформ, такі як прокрутка, іконки та шрифти, щоб забезпечити повну нативну продуктивність як на iOS, так і на Android. Варто також зазначити, що існує технологія Kotlin Multiplatform [21], яка покликана спростити розробку кросплатформних проєктів. Вона скорочує час на написання та підтримку однакового коду для різних платформ, зберігаючи при цьому гнучкість та переваги нативного програмування. Тож обидві технології можна широко використовувати для різних платформ і при цьому отримувати доступ до нативної функціональності.

Таблиця 6 – Порівняльна таблиця Kotlin/Jetpack Compose і Dart/Flutter

Стек мови. Kotlin – це багатоплатформова, статично типізована мова програмування загального призначення, яка повністю сумісна з Java.	Flutter використовує Dart як основну мову програмування.
Мінімально підтримувані платформи всі. Kotlin сумісний з JDK 6, тому додатки з Kotlin безпечно працюють на старих версіях Android.	Flutter підтримує Android 4.1 (рівень API 16) та iOS 8.0, а також веб-, десктопні та вбудовані пристрої.

Продовження таблиці 6

Продуктивність Kotlin порівнянна з продуктивністю Java завдяки тому, що її байт-код компілюється у байт-код JVM, який потім виконується віртуальною машиною Java (JVM) [22]. Крім того, він використовує оптимізаційний компілятор, який ще більше покращує його продуктивність порівняно з інтерпретованими мовами, такими як Python або JavaScript.	Додатки Flutter відомі своєю високою продуктивністю та плавною анімацією завдяки використанню графічного движка Skia та реактивної моделі програмування [23].
Інтерфейс користувача. Пропонує широкий спектр бібліотек та фреймворків для створення користувацьких інтерфейсів, таких як Android XML макети та Android Jetpack compose для розробки користувацьких інтерфейсів.	Flutter пропонує унікальний і багатий набір віджетів, що налаштовуються, який дозволяє створювати виразні та гнучкі інтерфейси.
Можливості. Kotlin пропонує широкий спектр можливостей, включаючи безпеку нуля, функції розширення, класи даних, підпрограми та виведення типів.	Flutter пропонує широкий спектр функцій, включаючи гаряче перезавантаження, багатий набір віджетів, що налаштовуються, і швидкий цикл розробки, що дозволяє легко створювати високопродуктивні й візуально привабливі програми.
Спільнота. Kotlin має велику та активну спільноту, з багатьма ресурсами для навчання та усунення несправностей.	Хоча вона все ще відносно невелика порівняно з деякими більш відомими мовами фреймворків, спільнота навколо Flutter швидко зростає.
Переваги. Kotlin має хорошу продуктивність, нульову безпеку, функції розширення, класи даних, підпрограми та виведення типів, а також він повністю сумісний з Java.	Flutter пропонує високу продуктивність, швидкий цикл розробки та широкий спектр віджетів, що налаштовуються, що дозволяє легко створювати візуально привабливі та високопродуктивні додатки на різних платформах: розробляйте один раз і для Android, і для iOS. Вибір Flutter зменшує ці витрати й забезпечує майже однаковий інтерфейс застосунку на обох платформах.
Недоліки. Обмежені ресурси у порівнянні з іншими популярними мовами, такими як Java. Обмежена кількість бібліотек у порівнянні з іншими мовами.	Крива навчання для Flutter може бути крутою, і він може мати обмежену кількість сторонніх бібліотек у порівнянні з іншими фреймворками.

Після проведення порівняльного аналізу можна зробити наступні висновки. Kotlin завдяки своїй сумісності із Java має більше доступних бібліотек та велику спільноту. Kotlin разом із декларативним інтерфейсом від Jetpack Compose гарантує швидший процес розробки мобільного застосунку із великою кількістю екранів та компонентів. Це значно зменшить об'єм програмного коду та його читабельність. Оскільки Kotlin/Jetpack Compose має більшу підтримку від Google, цей інструмент розробки буде активно розвиватися, а також він з самого початку більше уваги приділяє розробці під операційну систему Android, що не може не відобразитися на якості реалізації програмних продуктів. Було прийняте рішення про використання Kotlin та Jetpack Compose, як основних інструментів розробки, із наступними технологіями це дозволить відкрити їх повний потенціал.

Android Studio – це офіційне інтегроване середовище розробки (IDE) для розробки мобільних застосунків Android та програм під всі інші пристрої [24,25]. Надає широкий набір інструментів для тестування застосунків. Пропонує шаблони проєктів та інтеграцію з GitHub.

H2 – це високошвидкісна база даних, написана на Java [26]. Вона має ряд переваг, що позитивно будуть впливати на якість реалізації мобільного застосунку:

– невелика вага, розмір двигуна H2 становить лише близько 2,5 МБ;

– продуктивність бази даних H2 є однією з найкращих серед популярних реляційних баз даних;

– база даних H2 створена на мові Java і спеціально для розробки під Java;

– підтримує стандартний SQL та JDBC API (Java Database Connectivity), він також може використовувати драйвер ODBC (Open Database Connectivity) від розробників бази даних PostgreSQL;

– має як вбудований, так і серверний режим;

– підтримує кластеризацію та багатоверсійну конкуренцію;

– має потужні функції безпеки;

– надзвичайно швидкий движок баз даних;

– не потребує додаткової конфігурації.

Ці переваги роблять H2 чудовим вибором для розробки та тестування застосунків. Dbeaver – це універсальний і багатоплатформний інструмент управління базами даних, який підтримує широкий спектр баз даних SQL і NoSQL [27]. Це цінний інструмент для розробників, адміністраторів баз даних та аналітиків даних. Основні його призначення та можливості:

– дозволяє створювати, керувати та маніпулювати базами даних у різних системах керування базами даних (СКБД);

– на відміну від інструментів командного рядка, DBeaver надає зручний інтерфейс, за допомогою якого можна переміщатися по базах даних;

– можна редагувати дані безпосередньо в таблицях, подібно до редагування електронних таблиць;

– дозволяє створювати візуальні діаграми об'єктів і схем бази даних та експортуйте їх за потреби;

– створення та виконання спеціальних SQL-запитів з такими функціями, як підсвічування синтаксису та автозаповнення;

– підтримує функції, характерні для різних СУБД (наприклад, представлення, збережені процедури, типи даних).

Exposed – це фреймворк для роботи з базами даних, розроблений JetBrains, який надає ідіоматичний Kotlin API для деяких реалізацій реляційних баз даних, включаючи H2. Кожна операція з базою даних в Exposed потребує активної транзакції. Exposed може використовуватися як високорівневий DSL [28] над SQL, так і як легкий ORM (Object-Relational Mapping) [29]. DSL в Exposed – це SQL-обгортка для різних типів даних, яка дозволяє використовувати SQL-запити в Kotlin з більшою безпекою та зручністю. Він є одним з двох способів взаємодії з базою даних, другий – це легкі об’єкти доступу до даних (DAO).

Іншим потужним інструментом, що використовується для автоматизації процесу збірки є Gradle [30]. Наявний в Android Studio для розробки Android застосунків. Автоматизує багато процесів збірки, що допомагає уникнути багатьох поширених помилок збірки. Дозволяє визначати гнучкі, настроювані конфігурації

збірки. Кожна конфігурація збірки може визначати свій власний набір коду та ресурсів, повторно використовуючи частини, що є загальними для всіх версій програмного забезпечення. Gradle підтримує мову програмування Kotlin, що дозволяє використовувати всі переваги цієї мови при написанні скриптів збірки. Android Studio має власний плагін для підтримки Gradle.

2.5 Архітектура та функціональна структура застосунку

Архітектура та функціональна структура застосунку відіграють вирішальну роль у забезпеченні високої продуктивності, масштабованості та підтримки якості програмного рішення [31]. В цьому розділі розглянемо ключові компоненти архітектури мобільного застосунку, включаючи вибір архітектурного патерну, який найкраще відповідає потребам проєкту, та розробку функціональної структури, яка забезпечить наочний та ефективний процес розробки мобільного застосунку.

MVP – це патерн проєктування архітектури програмного забезпечення, який відокремлює рівень презентації від бізнес-логіки [32]. Він є похідним від традиційного патерну MVC (Model-View-Controller) [33] і використовується для того, щоб зробити додатки більш тестованими та легшими в обслуговуванні. Складається із трьох складових MVP.

Model відповідає за обробку логіки домену (реальних бізнес-правил) та зв'язок з базою даних і мережевим рівнем. Він керує даними, а також бізнес-логікою мобільного застосунку.

Компонент представлення (View) інтерфейсу користувача (UI) забезпечує візуалізацію даних і відстежує дії користувача, щоб повідомити про них ведучого. Він відповідає за відображення інтерфейсу користувача на екрані та обробку всіх взаємодій користувача.

Компонент ведучого (Presenter) отримує дані з моделі та застосовує логіку інтерфейсу, щоб вирішити, що саме відображати. Він керує станом представлення і

виконує дії відповідно до повідомлень користувача про введення даних з представлення. Особливості використання:

– відсутність концептуального зв'язку в android компонентах;

– якщо розробник не дотримується принципу єдиної відповідальності коду, то рівень ведучого має тенденцію розширюватися до величезного класу, що все знає і це зашкодить якості коду;

– легке обслуговування та тестування коду, оскільки модель, представлення та рівень ведучого застосунку розділені.

MVI розшифровується як Model-View-Intent. Це архітектурний патерн, який був натхненний односпрямованим потоком даних і концепцією незмінності з функціонального програмування [31]. Має наступні компоненти. Model представляє стан програми, це незмінна структура даних, яка містить всю необхідну інформацію для відображення представлення. Оскільки вона незмінна, замість того, щоб змінювати наявну модель, створюється нова модель кожного разу, коли відбуваються зміни. Представленням (View) відповідає за рендеринг моделі та перенаправлення намірів користувача до наміру. В Android це може бути активність, фрагмент або користувацьке представлення. Намір (Intent) представляє намір або бажання виконати дію, як користувачем, так і самим застосунком. Це результат взаємодії користувача з представленням або внутрішньої події, наприклад, мережевого запиту. Користувач взаємодіє з представленням, генеруючи наміри. Ці наміри потім використовуються для створення нової моделі. Дані моделі потім показуються в представлені. Цикл повторюється з кожним новим наміром. Особливості MVI архітектури:

– підтримка стану більше не є проблемою з цією архітектурою, оскільки вона зосереджена в основному на станах;

– створює багато об'єктів для всіх станів, це робить його занадто дорогим для управління пам'яттю програми;

– оскільки потік даних є односпрямованим, його можна легко відстежувати та прогнозувати;

– легко налагоджувати програму, оскільки відомо стан об'єкта при якій виникла помилка;

– кожен компонент виконує свою власну відповідальність;

– тестувати застосунок буде простіше, оскільки можна відобразити бізнес-логіку для кожного стану;

– забезпечує безпеку потоку, оскільки об'єкти стану є незмінними;

– призводить до великої кількості шаблонного коду, оскільки потрібно підтримувати стан для кожної дії користувача.

MVVM – це архітектурний патерн, який полегшує розподіл завдань в Android застосунку [32,35]. Він розділяє програму на три основні компоненти. Model представляє джерела даних та бізнес-логіку. Вона абстрагує операції пошуку та зберігання даних. View обробляє компоненти інтерфейсу користувача (UI). Спостерігає за ViewModel та інформує її про дії користувача. ViewModel діє як міст між моделлю та представленням. Вона відкриває відповідні потоки даних для інтерфейсу користувача та інкапсулює бізнес-логіку. Особливості застосування:

– підвищення можливості повторного використання коду;

– файли проєкту легко підтримувати та вносити зміни.

– всі модулі є незалежними, що покращує можливість тестування кожного рівня окремо;

– виникають труднощі в налагоджені для програм зі складною логікою зв'язування даних.

Clean Architecture – це підхід до розробки програмного забезпечення при якому можна легко зрозуміти функціональність програми просто аналізуючи її вихідний код [31]. Чиста архітектура створює системи, які можна тестувати і які не залежать від інтерфейсу користувача, баз даних, фреймворків, зовнішніх агентств та бібліотек. Так само як і інші концепції проєктування програмного забезпечення, чиста архітектура спрямована на створення економічно ефективного процесу розробки високоякісного коду, який працює краще, легко модифікується і має менше залежностей. Головне правило, яке змушує чисту архітектуру працювати

говорить, що ніщо у внутрішньому колі не повинно залежати від чогось у зовнішньому колі. Зокрема, прикладні та бізнес-правила не повинні залежати від інтерфейсу користувача, бази даних. Особливості використання:

– код легше тестувати, ніж зі стандартним MVVM;

– зручна структура пакунків;

– ідеально контрольоване розділення (найголовніша перевага);

– важка адаптованість до вже готового проєкту;

– швидке впроваджувати нових функцій;

– легко підтримувати проєкт в робочому стані;

– містить багато додаткових класів, тому не підходить для мобільних застосунків з низьким рівнем складності.

VIPER – це абревіатура від View, Interactor, Presenter, Entity і Router (подання, взаємодіючий, ведучий, сутність і маршрутизатор). Це патерн проєктування, який реалізує принцип поділу проблем у дуже структурований спосіб, що полегшує ізоляцію залежностей та тестування компонентів програми [34]. Основною перевагою архітектури є її модульність. Кожен компонент має чітко визначені обов'язки, що знижує залежності та спрощує розуміння коду. Це робить VIPER ідеальним для великих та складних застосунків, де потрібна висока масштабованість та легкість обслуговування. Схема його компонентів зображена на рисунку 20.

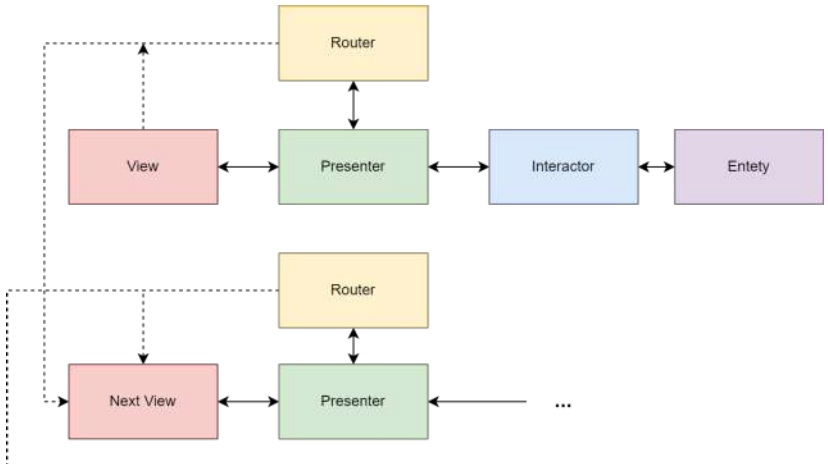


Рисунок 20 – Архітектура VIPER

Його можна описати наступним чином. Подання (View) відповідає активності або фрагменту в застосунку Android. Він відповідає за відображення користувацького інтерфейсу і збір даних, введених користувачем. Він надсилає дії користувача ведучому.

Взаємодіючий це місце, де відбувається вся бізнес-логіка. Взаємодіючий піклується про виконання будь-якої дії, коли ведучий дає йому вказівки. Він нічого не знає про інтерфейс користувача.

Ведучий діє як посередник між поданням та взаємодіючим. Він отримує дії користувача від подання, обробляє їх (за допомогою взаємодіючого, якщо потрібно) і відповідно оновлює подання.

Сутність (Entity) представляє дані програми. Зазвичай це рівень моделі в програмі, яка містить основні структури даних, якими маніпулює взаємодіючий.

Маршрутизатор (Router): Маршрутизатор відповідає за навігацію між екранами в мобільному застосунку. Зазвичай його реалізує ведучий, повідомляючи поданню, куди рухатися далі. Кожен блок в архітектурі VIPER відповідає об'єкту з певними завданнями, входами та виходами. Це робить кодову базу набагато більш організованою і легшою для навігації, а також покращує можливість тестування програмного коду.

Отже, після отримання артефактів проєктування баз даних та інтерфейсу, розробки алгоритму роботу мобільного застосунку, аналізу наявних засобів реалізації було прийняте рішення про використання гібридної VIPER архітектури, що використовує компонентний підхід, який реалізовується засобами Jetpack Compose та Kotlin. Наслідує принцип поділу проблем із VIPER архітектури, в дуже структурований спосіб, що полегшує ізоляцію залежностей та складових програми, тестування компонентів. Відповідно до VIPER її можна описати наступним чином.

Обов'язки View. Початкова активність (Activity) [36] в застосунку Android реалізовується вручну розробником. Вона відповідає за відображення користувацького інтерфейсу і збір даних, введених користувачем. Всі інші активності створюються під час компіляції декларативним способом. Створені активності надсилають дії користувача ведучому.

Обов'язки Interactor (взаємодіючого). Місцем, де відбувається вся бізнес-логіка, є транзакція, вони передаються, як функції аргументи, що описані явно в програмному коді, на кожну можливу подію користувача. Взаємодіючий піклується про виконання будь-якої дії, коли ведучий дає йому вказівки.

Обов'язки Presenter (ведучого). Ця роль покладена на модулі. Діє як посередник між View та Interactor. Він отримує дії користувача від подання, обробляє їх за допомогою взаємодіючого і відповідно оновлює подання.

Обов'язки Entity. Сутності представляють дані програми, вони явно прописані в пакеті моделей, що містять основні структури даних, якими маніпулює взаємодіючий при зверненні до бази даних.

Обов'язки Router. Реалізуються через обгортку клас, що використовує можливості сторонньої бібліотеки маршрутизації, відповідає за навігацію між екранами в мобільному застосунку. Архітектура ПЗ зображена на рисунку 21.

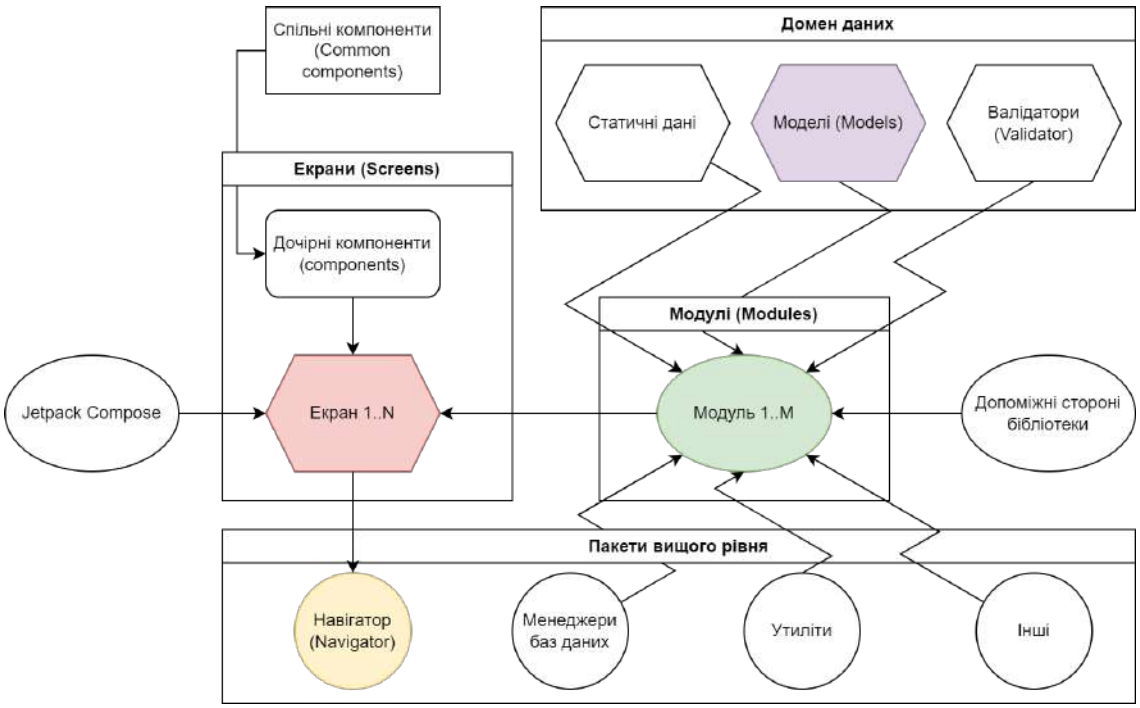


Рисунок 21 – Гібридна VIPER архітектура мобільного застосунку

Спільні особливості між VIPER архітектурою:

– наявність сутностей (Entity);

– наявність маршрутизатора (Router);

– однакова логіка роботи представлень (View);

– наявність Interactor та Presentor тільки в іншій формі.

Відмінності між VIPER архітектурою:

– View це збірка базових та дочірніх компонентів, які декларативно описують деякий програмний екран, який під час компіляції генерує Activity;

– Interactor це не один клас, а багато функцій, що містять бізнес-логіку;

– кожний View може мати декілька класів Presenter;

– кожний внутрішній пакет екранів мобільного застосунку є центральним компонентом до якого підключаються всі інші складові архітектури.

Щоб побачити використання компонентного підходу можна побудувавши таблицю батьківських та дочірніх компонентів, які використовуються для побудови всіх екранів мобільного застосунку.

Завдяки специфіці роботи Jetpack Compose, головним елементом є екран, який будується компонентами. Користувач надсилає події в Presenter (представлений модулями), що запускає роботу Interactor (представлений функціями бізнес-логіки).

Інші використані принципи:

– використання зовнішніх бібліотек через класи обгортки;

– патерн програмування Factory для створення компонентів [37].

Переваги гібридної архітектури:

– принцип розподілу відповідальності;

– однозначність розташування складових програми в архітектурній структурі, маючи функціональні особливості роботи, можна чітко визначити розташування програмних компонентів в програмній архітектурі;

– використання декларативного методу створення екранів застосунку;

– швидке створення компонентів за допомогою Jetpack Compose.

Слабкі місця гібридної архітектури:

– велика крива написання класів обгорток;

– головні класи мають велику міжмодульну залежність.

Щоб показати функціональність програмного забезпечення було спроєктовано діаграму пакетів, що містить варіанти використання. Даний артефакт добре відображає залежності між основними елементами великомасштабних систем. На схемі виділено відповідними кольорами шість типів пакетів за своїм призначенням. Діаграма зображена на рисунку в додатку Б.1.

Після завершення етапу проєктування стає очевидний обсяг необхідної роботи для реалізації. Було проведено аналіз ризиків та обмежень.

Тісна залежність між модулями може сильно повпливати на терміни виконання, незавершеність одного може блокувати прогрес інших. Обмежені ресурси (людські, фінансові, технічні) можуть повпливати на якість та повноту реалізації проєкту. Відсутність менеджера проєкту, некомпетентність розробника керувати проєктом та процесом розробки може затримати терміни повної реалізації висунутих вимог до програмного забезпечення.

Зміни, що вносяться під час розробки, можуть спричинити затримки та збільшення витрат. Рекомендуються наступні дії для керування ризиками:

– регулярно перевіряти стан проєкту для виявлення та розв'язання проблем на ранніх стадіях;

– проводити реалізацію пріоритетних програмних модулів в першу чергу;

– використовувати гнучку методологію розробки, наприклад Scrum.

Після завершення етапів дослідження предметної області та проєктування мобільного застосунку настає етап реалізації. Кожний програмний продукт має модель життєвого циклу програмного забезпечення [38], що є ключовою концепцією в програмній інженерії, оскільки вона визначає етапи створення та супроводу програмних систем. Ця модель допомагає командам розробників систематизувати процес від ідеї до випуску продукту та його подальшої підтримки. Недостатньо знати, що робити, а ще потрібно відповісти на питання «Як?». Тому потрібно обрати методологію розробки для досягнення кращого результату. Scrum – це гнучка методологія розробки, яка допомагає командам ефективно працювати над складними проєктами [39]. Вона базується на принципах Agile і зосереджена на співпраці, неперервному вдосконаленні та гнучкості у відповіді на зміни.

Основні складові Scrum. Продуктовий беклог (Product Backlog) – перелік усіх вимог до продукту, який постійно оновлюється. Спринт (Sprint) – основний цикл розробки, який триває від одного до чотирьох тижнів. Спринт беклог (Sprint Backlog) – перелік завдань, вибраних для реалізації в поточному спринті. Ревізія спринту (Sprint Review) – зустріч наприкінці спринту для демонстрації зробленого та отримання зворотного зв'язку. Ретроспектива спринту (Sprint Retrospective) – зустріч для аналізу минулого спринту та планування поліпшень.

Scrum підкреслює важливість зворотного зв'язку від користувачів та стейкхолдерів, а також необхідність швидкої адаптації до змін. Це дозволяє командам створювати високоякісні продукти, які відповідають потребам ринку та користувачів. Scrum не є жорсткою методологією, а радше набором рекомендацій, які можуть бути адаптовані під конкретні умови проєкту.

Висновки. В даному розділі було здійснено проєктування бази даних, створено концепцію дизайну та відібрано його ідейні принципи, описано логіку мобільного застосунку, був зроблений акцент на швидкості та зручності виконання фінансових операцій. Здійснено вибір технологій та методів реалізації, наголошено про важливість використання гнучкої методології розробки Scrum. Також було розглянуто п'ять типів архітектур мобільного застосунку, вибір здійснено в сторону гібридної VIPER архітектури з компонентним підходом, що реалізовується засобами Kotlin та Jetpack Compose.

В результаті проведення проєктування мобільного застосунку було отримано наступні артефакти:

– моделі «сутність-зв'язок» (ERD) та їх опис для створення бази даних;

– схему екранної навігації та прототип інтерфейсу користувача;

– діаграми активностей для реалізації логіки роботи програми;

– обґрунтований вибір технологій і методів реалізації мобільного застосунку;

– обґрунтований вибір архітектури програмного забезпечення та діаграму пакетів, що відображає функціональну структуру мобільної програми.

Артефакти проєктування будуть використані для якісної реалізації програмного продукту в наступному розділі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1 Реалізація логіки мобільного застосунку

Першим етапом реалізації є створення класу активності від «ComponentActivity». Фреймворк Jetpack Compose дозволяє декларативно будувати інтерфейс користувача. Один із його механізмів є створення теми – спосіб застосування схожих візуальних і поведінкових властивостей у різних застосунках, що робить їх унікальними та уніфікованими. Всі кольори, як для світлого, так і для темного режиму екрана, а також форми, шрифти об’єднуються в спеціально створену тему «MonetaTheme». Це дозволяє вкладеним елементам посилатися на головну тему, і використовувати збережені властивості. Всі налаштування прописані в стандартному пакеті «ui.theme». У наступному фрагменті коду, елемент тексту використовує колір, який добре видно на фоні, а також стиль від заголовка другого рівня, який містить розмір тексту та тип шрифту.

```
MonetaTheme {
     Text(
           LocalContext.current.resources.getString(R.string.app_name),
           style = MaterialTheme.typography.h2,
           color = MaterialTheme.colors.onBackground
     )
}
```

Запуск мобільного застосунку супроводжується початковим екраном, що відображає ім’я та логотип програми. На даному етапі відбуваються запуск всіх процесів підготовки, результатом яких є стан програми, що дозволяє повноцінно використовувати застосунок. Повний список процесів наведений в розділі 2.3. Велика кількість підготовчих дій затримує показ першого екрану, тому їх потрібно оптимізувати, для цього було створено клас «OpConductor». Важливо розуміти, що деякі модулі програми мають міжмодульні залежності, наприклад база даних не може завантажитися без отримання із DataStore шляху, де вона знаходиться.

Асинхронне або не блокувальне програмування є важливою частиною сучасного процесу розробки. При створенні мобільних застосунків важливо

забезпечити не лише зручність роботи з погляду користувача, але й можливість масштабування за потреби. Kotlin розв'язує цю проблему гнучко, надаючи підтримку підпрограм на рівні мови та делегуючи більшу частину функціональності бібліотекам. Крім того, що підпрограми відкривають двері до асинхронного програмування, вони також надають безліч інших можливостей, таких як паралелізм та модель акторів [40]. Клас «OpConductor» використовує можливості паралельних обчислень від бібліотеки «kotlinx.coroutines» для оптимізації виконання процесів. Для наочності було створено діаграму станів мобільного застосунку, де показано події від яких залежить початок інших процесів, рисунок 21.

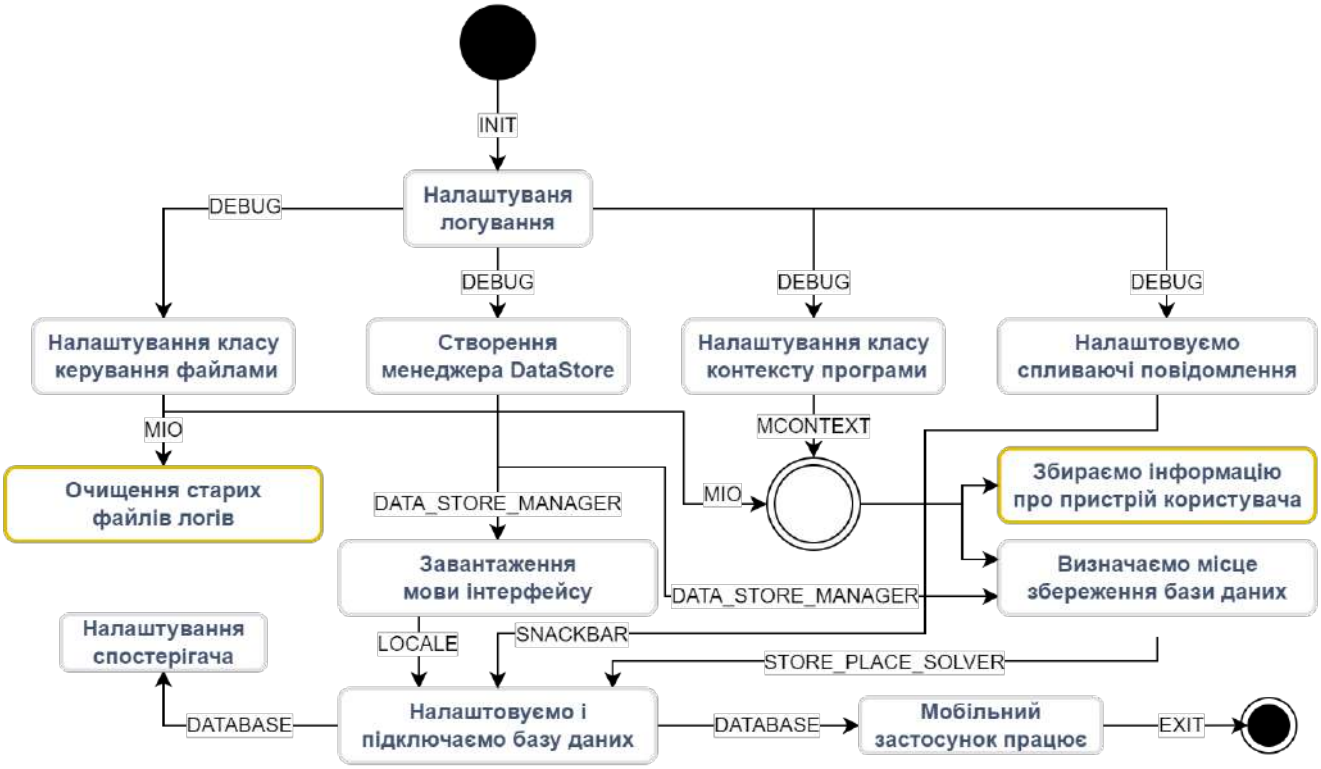


Рисунок 21 – Діаграма станів підготовчих процесів запуску ПЗ

Стани, що мають жовту рамку не є критично важливими для правильної роботи мобільного застосунку, їх виконання потрібно затримати до показу першого функціонального екрана, механізмом, що відповідає за це є таймер.

Під час запуску програми, головна активність створює першу подію «INIT», що впливає на перший програмний стан. Подія «DEBUG» є критично важливою,

оскільки сповіщає готовність програмного забезпечення коректно обробляти будь-які помилки, а також фіксувати їх у файлах звітах (англ. Logs). Подія «MIO» сповіщає про готовність класу, що надає методи по роботі із теками та файлами, всі процеси, які потребують цієї функціональності можуть почати використовувати його. Подія «MCONTEXT» має важливе значення для всіх модулів, адже вона сповіщає про доступність класу, який відповідає за внутрішньо програмні властивості мобільного застосунку. Подія «DATA_STORE_MANAGER» свідчить про готовність класу, що відповідає за отримання даних із механізму збереження інформації «DataStore». Подія «LOCALE» сповіщає про готовність локалізації текстових повідомлень відповідно до тієї мови, що обрав користувач або стандартної мови пристрою. Подія «SNACKBAR» стверджує про доступність використання компонента рухомих повідомлень. Подія «DATABASE» повідомляє інші процеси про можливість використання бази даних. Стан «Мобільний застосунок працює» триває доти, поки користувач не захоче вийти або не виникне критична помилка. Подія виходу із мобільної програми «EXIT» генерується операційною системою, де вона встановлена.

Одна із ключових подій «APP_OBSERVER» відповідає за діагностику та тестування програмних помилок, самообслуговування та виправлення помилок, надання рекомендацій щодо виправлення проблем. Також відіграє важливу роль в безпеці та цілісності даних користувача. Має ряд тестів, що виконуються для діагностики проблем:

– визначення доступності місця збереження бази даних;

– тестування місця збереження бази даних на можливість запису даних та читання інформації;

– проводиться верифікація карти пам’яті, якщо вона вибрана, як місце збереження робочого простору;

– перевірка наявності файлів бази даних;

– перевірка підключення до бази даних шляхом виконання запитів до неї.

Завдяки даним тестам можливе виявлення наступних перешкод для користування мобільним застосунком:

– відсутність або помилкове вимкнення карти пам’яті;

– пошкодження карти пам’яті або її блокування;

– випадкове використання іншої карти пам’яті або навмисна заміна;

– порушення цілісності файлової структури бази даних;

– проблеми з підключенням до бази даних, авторизацією користувача, недостатньою кількістю системних ресурсів для коректної взаємодії.

Після завершення обов’язкових процесів відбувається перехід на екран стартера, який здійснює фінальні кроки для показу першого функціонального екрана. Також він зберігає клас навігації для подальшого використання внутрішніми компонентами.

Розробка нової функціональності, модуля чи інформаційного екрана має певний архітектурний та програмний шаблон. Його структурна складність залежить від кількості покладеної на нього відповідальності, модульної зв’язаності, довжини списку використаних компонентів, об’єму бізнес-логіки. Загальну програмну будову та структурні елементи можна побачити на рисунку 22.

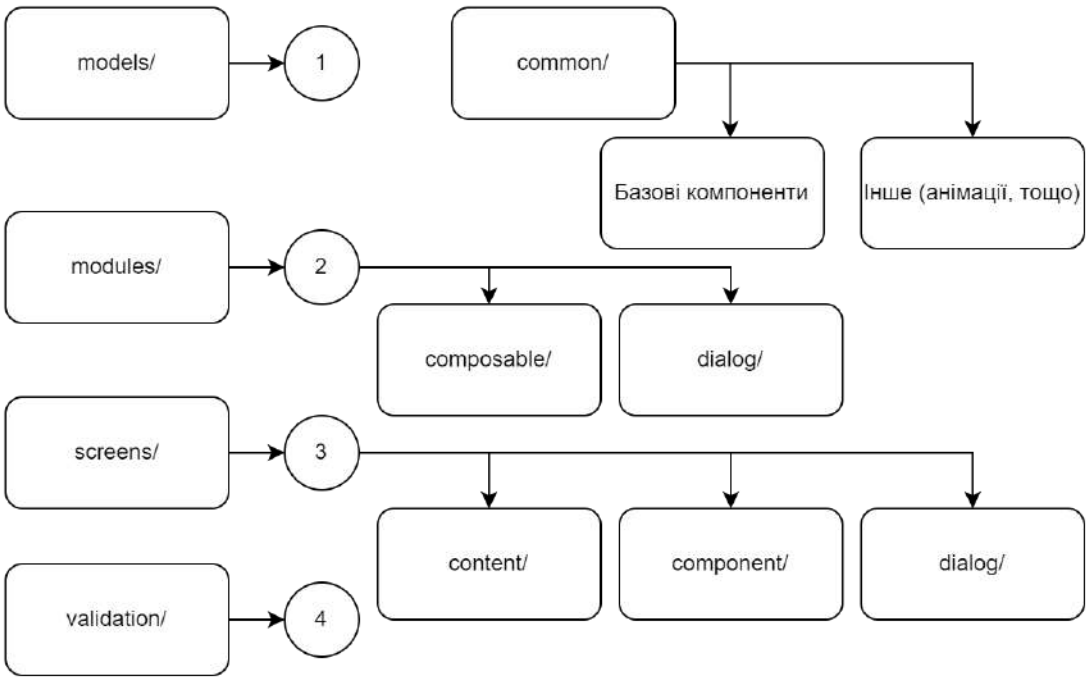


Рисунок 22 – Структурні та архітектурні елементи нової функціональності

Як показано на діаграмі для створення нової функціональності використовуються різні пакети. Елемент, що позначений цифрою 1 відповідає за модель даних, вона створюється тільки за необхідності збереження інформації в базі даних. Елемент під номером 4 відповідає за створення класу валідатора для моделі даних. Якщо дані не повинні мати жодної перевірки, його не потрібно створювати. Спільні складові знаходяться в пакеті «common/», так базові компоненти можуть використовуватися в будь-якому місці, де створюється інтерфейс користувача. Якщо нова функціональність екрана має складну бізнес-логіку, то вона буде знаходитися в пакеті «modules/» (елемент 2). Модулі можуть не мати спеціального екрана і використовуватися як UI компоненти, що є будівельними блоками для інших екранів і знаходяться у внутрішньому пакеті «composable/». Елемент, що позначений цифрою 3 створюється за умови, коли функціональність представлена новим екраном. Його внутрішніми складниками є:

– класи екранів, що використовують патерн програмування «Міст» (англ. Bridge) [37], вони є з'єднанням між активностями та його UI компонентами;

– пакет «content/» містить класи, що декларативно описують інтерфейс користувача для відповідного екрану;

– пакет «component/» наслідує базові батьківські компоненти й змінює їх логіку чи стилізацію за потреби;

– пакет «dialog/» містить діалоги в одному просторі.

Класи екранів були спеціально розділені від UI компонентів для гнучкості впровадження логіки високого рівня. Кожний із чотирьох елементів можуть мати внутрішні пакети для групування пов'язаних складових. Пакети діалогів наслідують батьківські елементи із базових компонентів. Таким чином, наявна структура використовує принципи широкого розділення відповідальності, а також повторного використання. Батьківські елементи описують тільки загальну логіку і можуть бути перевизначені у вкладених пакетах. Якщо розширення батьківського елемента не дає бажаного результату, завжди можна створити його дублікат із бажаними властивостями.

Мобільний застосунок має потужну систему персоналізації, серед її можливостей є створення нових атрибутів для транзакцій. Модуль, що відповідає за це має назву «Конструктор». Серед його екранів є форма створення, щоб показати деталі реалізації була створена діаграма послідовностей, рисунок 23.

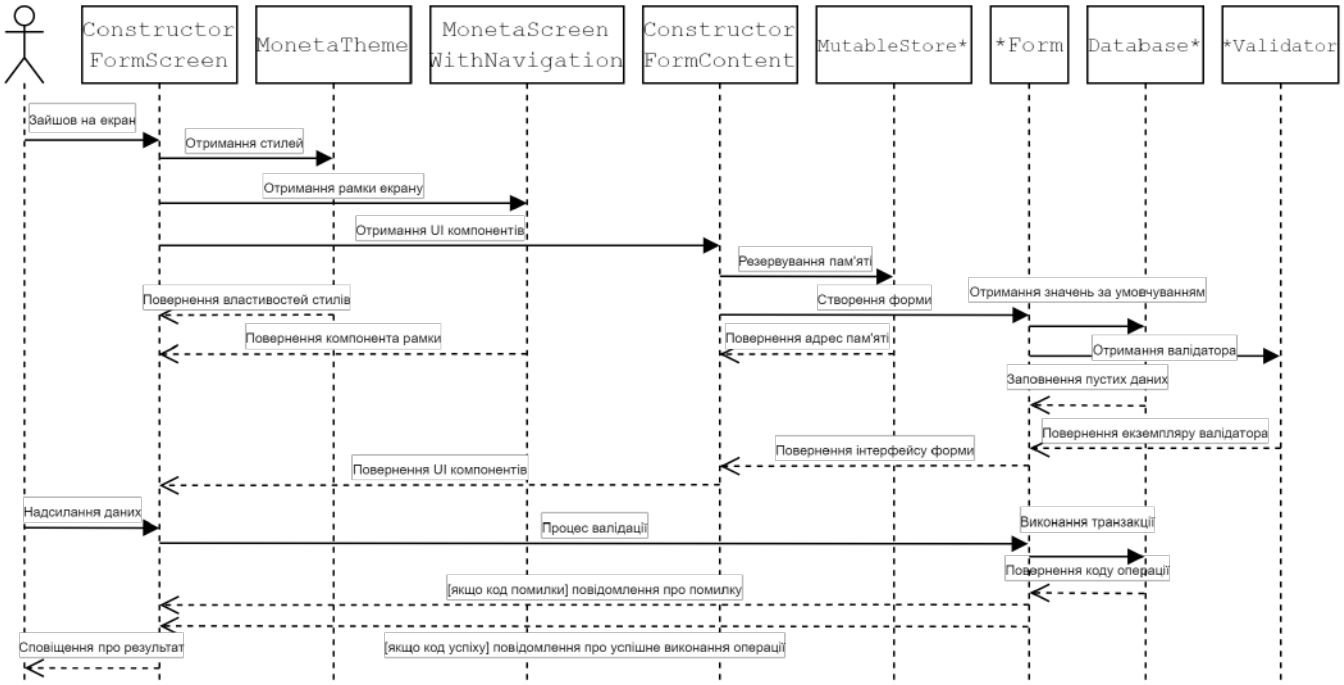


Рисунок 23 – Діаграма послідовностей екрана форми конструктора

Алгоритм роботи побудований на тому, щоб об’єднати необхідні компоненти, які здатні обробляти бізнес-логіку через інтерфейс користувача. Перші чотири елементи зліва на право відповідають тільки за створення загальний рис інтерфейсу. Наприклад «MonetaScreenWithNavigator» надає рамку для екранів, яка вже містить три види меню, які змінюються залежно від розмірності екранів. Сукупність класів, які дозволяють зберігати дані простих типів та об’єкти моделей даних була об’єднана в елемент «MutableStore*». Варіацій класів «*Form» та «*Validator» існує стільки ж, скільки об’єктів для створення в конструкторі. Елемент «Database*» на діаграмі об’єднаний в один клас для зручності, його реалізація складається із декількох окремих частин.

Розглянемо реалізовану структуру пакетів. Пакет «common» – містить базові компоненти, що наслідуються дочірніми.

Пакет «data» – статичні дані, наприклад назви заголовків екранів, клас даних елементів меню. Пакет «database» – класи, що надають можливість взаємодії з базою даних. Пакет «model» – містить сутності, що представляють дані програми.

Пакет «module» – містить модулі, кожний з яких має чітко визначену відповідальність, що можуть реалізовувати бізнес-логіку окремих функціональних екранів або надавати незалежні UI компоненти.

Пакет «navigator» – класи, що надають можливість пересуватися між функціональними екранами мобільного застосунку. Пакет «screen» – екрани описані декларативним методом.

Пакет «store» – реалізації декількох баз даних. Пакет «ui.theme» – стандартний пакет стилізації Jetpack Compose. Пакет «utils» – містить мікропрограми загально використання (робота із форматом часу і дати, хеш-функції тощо). Пакет «validation» – містить класи валідації для моделей даних.

Файл «MainActivity» – початкова активність для запуску програми.

Програмна реалізація має велику кількість модулів. Модуль «CrashHandler» надає функції обгортки аналогічним чином, як і «try-catch» блок. Якщо в ньому виникне помилка, він запускає автоматичну діагностику помилок модуля «AppObserver». Якщо результат діагностики буде незадовільний, то користувач може отримати обмеження щодо використання функціональних екранів, які представляють фінансові дані. В такому випадку користувачу будуть запропоновані дії для виправлення проблеми. Можливі наступні сценарії рекомендованих дій:

– потрібно під'єднати карту пам’яті, якщо вона не працює належним чином або замінена на іншу;

– змінити місце збереження для робочого простору;

– прохання звільнити більше системних ресурсів (оперативна пам’ять, пам’ять пристрою або накопичувача);

– прохання створити новий робочий простір;

– повторне виконання операції;

– провести перезапуск мобільного застосунку.

Бізнес-логіка мобільного застосунку описана в транзакціях до бази даних, що викликається для виконання при певних діях користувача. Було спроєктовано ряд функцій обгорток для транзакцій, що містять в собі механізми відловлювання помилок певного типу, наприклад системних, як в наступному коді.

```
private fun systemCatch(transaction: () -> Unit) {
    try {
        transaction()
    } catch (e: ExposedSQLException) {
        when (e.sqlState) {
            "53100" -> { // disk_full
                MLog.d("ExposedSQLException", "disk_full")
                SnackbarMaster.showError("Frame",
R.string.db_message_error_disk_full)
            }
            "53200" -> { // out_of_memory
                MLog.d("ExposedSQLException", "out_of_memory")
                SnackbarMaster.showError("Frame",
R.string.db_message_error_out_of_memory)
            }
            "90067" -> { // session closed
                MLog.d("ExposedSQLException", "session closed")
                AppObserver.setDatabaseErrorFlag()
            }
            else -> {
                MLog.d("systemCatch", "${e.sqlState} ${e.message}")
                throw e
            }
        }
    }
}
```

Дана функція зможе попередити користувача про відсутність місця для збереження введеної інформації, недостачу оперативної пам’яті для виконання роботи, помилку розриву з’єднання із базою даних. Також була створена функція «deleteCatch» для обробки операцій видалення, що повідомляє користувачу про неможливість видалення об’єкта, оскільки він використовується в інших місцях та має залежності. Операції створення та оновлення можуть мати найбільшу кількість помилок, це пов’язано із валідацією даних, а також великою залежністю інформації, серед них існують наступні:

– значення повинно бути унікальним;

– значення не може бути пустим;

– значення виходить за дозволений діапазон або його довжина більша за встановлені межі.

Функція «newAndUpdateCatch» розв'язує можливі проблеми. Таким чином, забезпечується цілісність даних та легкість використання операцій над базою даних. Весь процес взаємодії із формою даних супроводжується підказками, індикаторами помилок, а також сповіщеннями успіху.

3.2 Реалізація компонентів мобільного застосунку

Всі UI компоненти походять від бібліотеки «Compose Material 3» та попередніх версій. Основними будівельними блоками компонентів мобільного застосунку є текст, горизонтальні та вертикальні колонки, іконки. Дерево основних компонентів можна побачити на рисунку 24, елементи, що відмічені символом «*» мають велику кількість складників, які не доцільно показувати на діаграмі.

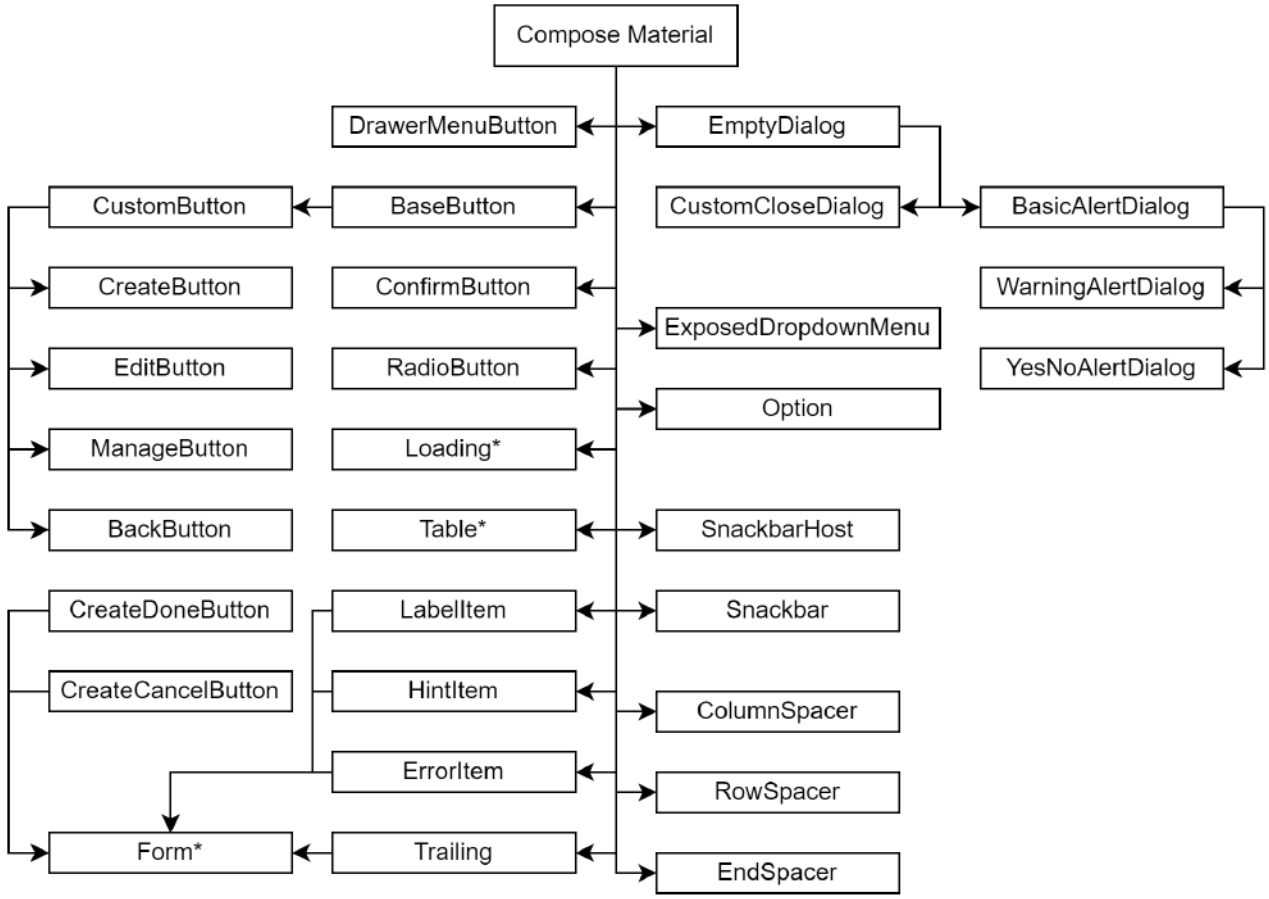


Рисунок 24 – Дерево основних компонентів

Важливу роль у створені функціонального екрана відіграє компонент «MonetaScreenWithNavigation», його частковий програмний код викладений далі.

```
@Composable
fun MonetaScreenWithNavigation(
    modifier: Modifier = Modifier,
    header: String,
    baseRoute: String,
    drawerIsOpen: Boolean = true,
    navigator: DestinationsNavigator,
    content: @Composable () -> Unit
) {
```

Функція, що будує даний компонент приймає наступні параметри відповідно. Додаткові стилі, заголовок екрана, базовий маршрут для виділення елементу в меню навігації. Наступний параметр відповідає за те, чи буде відкрите бічне меню. Також передаються стандартний клас навігації та внутрішні UI компоненти.

В мобільному застосунку реалізовано три типи меню під різні розмірності екранів, що надає зручну навігацію для кожного пристрою користувача. Для пристроїв із широкими екранами вигляд меню показано на рисунку 25. Також доступна кнопка гамбургер для згортання лівого меню.

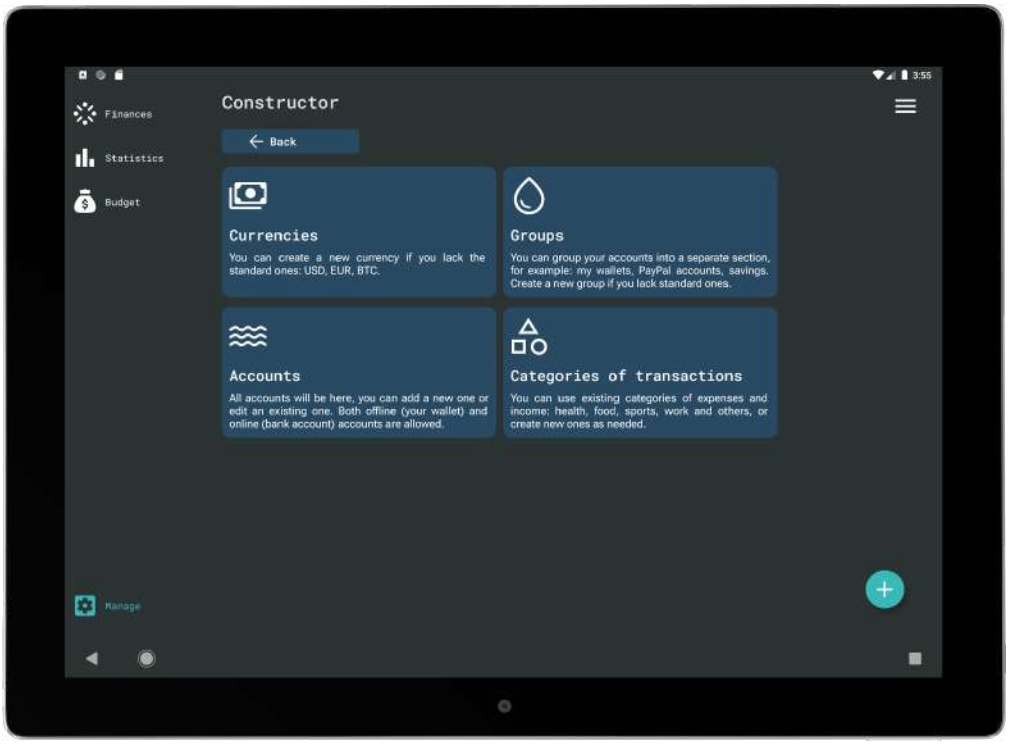


Рисунок 25 – Бокове меню навігації згортання

Для більшості смартфонів в альбомному режимі (зліва на рисунку 26) та портретному режимі (на рисунку справа).

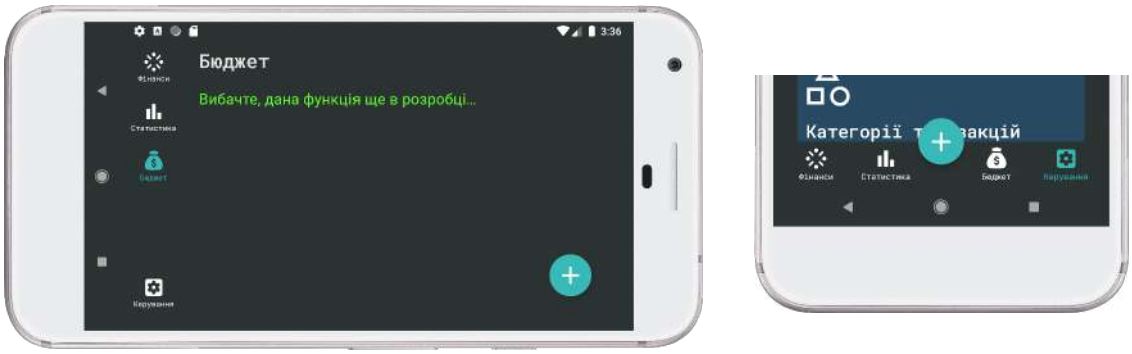


Рисунок 26 – Бокове та нижнє меню навігації

На екрані керування можна виконати велику кількість дії, наприклад отримати контакти розробників мобільного застосунку. Також можна викликати діалог покупки платної версії програми, перейти на екран допомоги, почитати політику використання, викликати діалог налаштувань, і скористатися можливостями персоналізації застосунку.

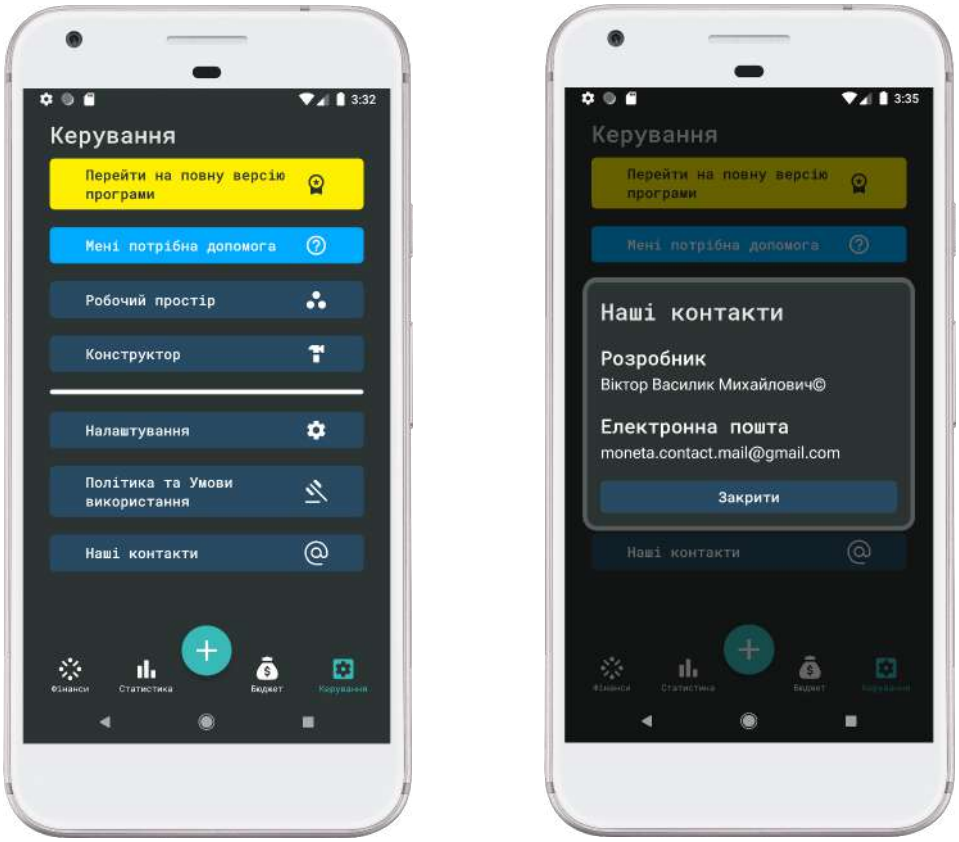


Рисунок 27 – Екран керування та діалог наших контактів

В мобільному застосунку реалізована можливість зміни мови інтерфейсу, якщо користувач не обрав ще жодної мови, то використовується системна. Згідно з технічним завданням є доступними українська та англійська мови, рисунок 28.

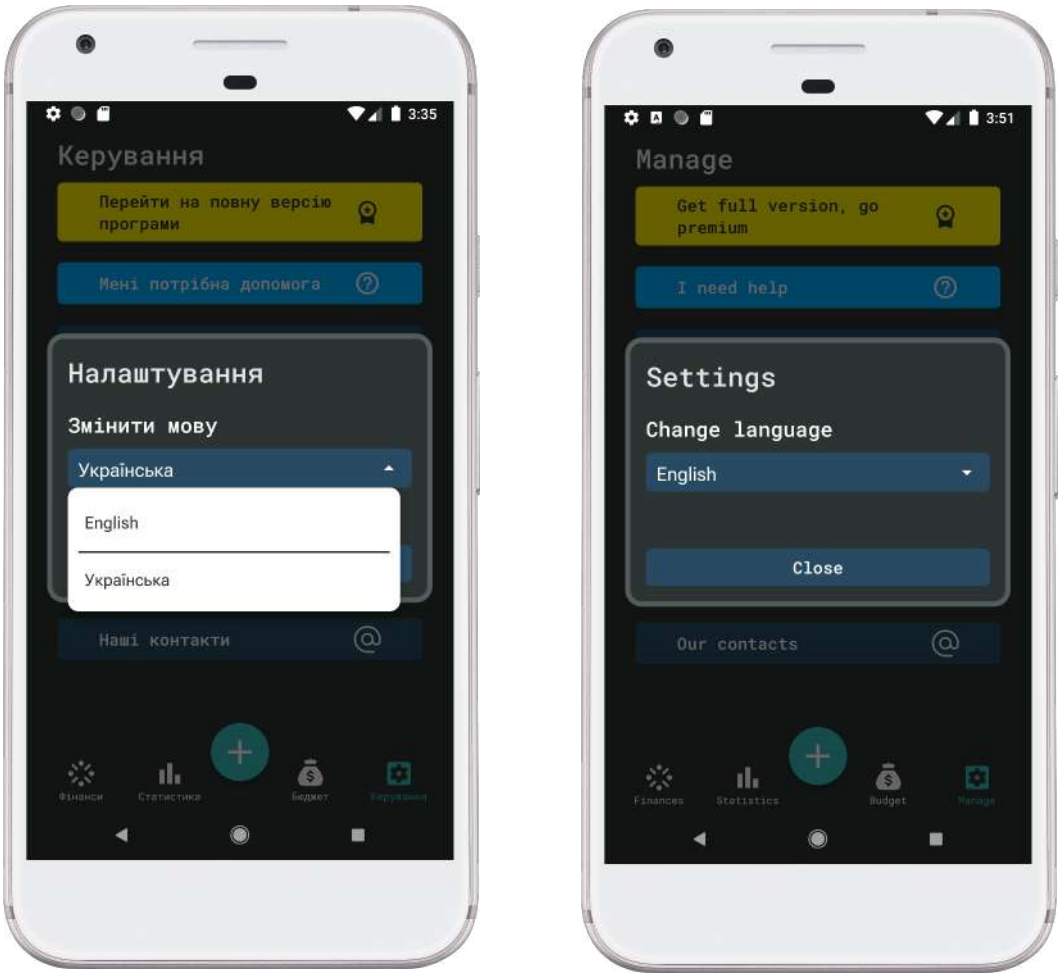


Рисунок 28 – Функціонал зміни мови інтерфейсу

На рисунку 29 показано вигляди інтерфейсу для пустого списку рахунків.

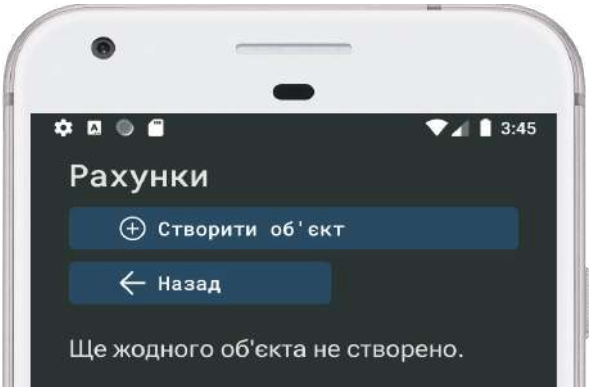


Рисунок 29 – Пустий список рахунків

На рисунку 30 показано вигляд інтерфейсу при наявності одного робочого простору на карті пам’яті.

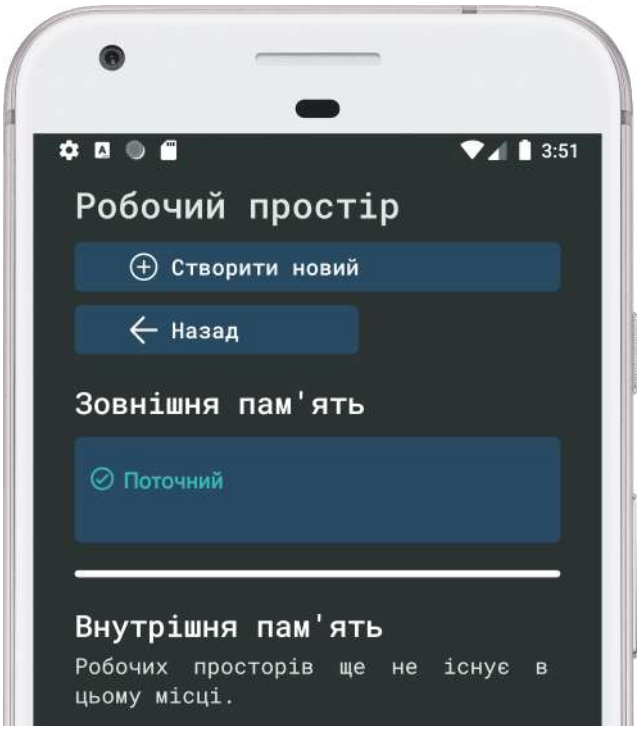


Рисунок 30 – Початковий список робочих просторів

Результати, що відображають функціональні можливості описані та зображені на рисунках в розділі тестування мобільного застосунку 3.4.

3.3 Розробка бази даних

База даних H2 для мобільного застосунку створюється автоматично засобами Exposed. Для всіх таблиць були створені відповідні моделі, наступний програмний код демонструє описові класи для об'єкта групи.

```
object GroupTable : IntIdTable(
    name = "Groups"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
}
class GroupEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<GroupEntity>(GroupTable)
    var name by GroupTable.name
    var description by GroupTable.description}
```

Завдяки «GroupTable» можна звертатися до бази даних, а через «GroupEntity» отримувати властивості деякого об’єкта. Для роботи з базою даних необхідно встановити наступні залежності.

```
// H2
implementation("com.h2database:h2:2.1.214")

// Exposed
def exposed_version = '0.44.0'
implementation "org.jetbrains.exposed:exposed-core:$exposed_version"
implementation "org.jetbrains.exposed:exposed-dao:$exposed_version"
implementation "org.jetbrains.exposed:exposed-jdbc:$exposed_version"
implementation "org.jetbrains.exposed:exposed-java-time:$exposed_version"
```

Мобільний застосунок при оновленнях має автоматичні механізми для зміни структури бази даних. Розглянемо наступний програмний код.

```
tryCreateTable(DatabaseInfoTable)

val currentDBVersion = try {
     val databaseInfo = DatabaseInfoEntity.find {
           DatabaseInfoTable.property eq "current_version"
     }.first()
     databaseInfo.data
} catch (e: Exception) {
     "not exist"
}

if (currentDBVersion != MONETA_DB_VERSION) {
     MLog.i(
           "CURRENT_DB_VERSION",
           "MONETA DB VERSION: %s | CURRENT DB VERSION: %s",
           MONETA_DB_VERSION,
           currentDBVersion
     )

     tryCreateTable(CurrencyTable)
     tryCreateTable(GroupTable)
     tryCreateTable(AccountTable)
     tryCreateTable(TransactionTypeTable)
     tryCreateTable(TransactionStatusTable)
     tryCreateTable(CategoryTable)
     tryCreateTable(TransactionTable)
     tryCreateTable(StatisticsPresetTable)

     DatabaseInfoEntity.createOrUpdate(
           "current_version",
           MONETA_DB_VERSION
     )
}
```

Спочатку створюються відсутні таблиці та колонки для моделі «DatabaseInfoTable», потім звідти читається версія бази даних і порівнюється із

поточною. Якщо було виявлено різницю, то застосовується оновлення для всіх інших моделей. Такий алгоритм споживає мало програмних ресурсів, що дозволяє перевіряти актуальність схеми бази даних. Базу даних H2 після створення можна проаналізувати через фізичну ER-діаграму, рисунок 31.

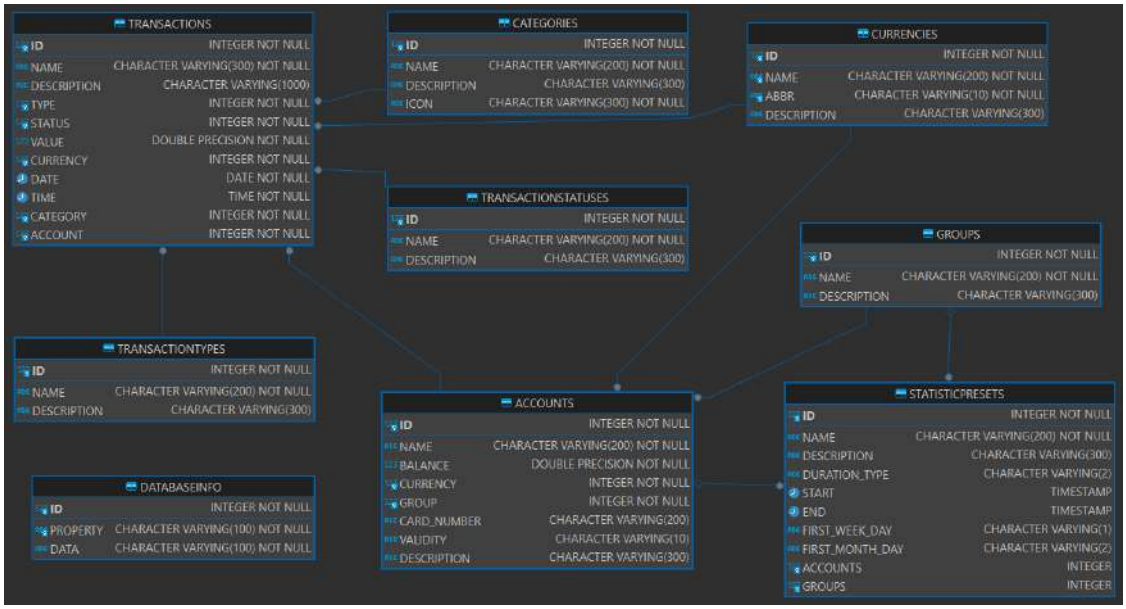


Рисунок 31 – Фізичка ER-діаграма бази даних H2

Файл бази даних без збереженої інформації займає 60 кілобайтів, це вважається хорошим показником для мобільних пристроїв. Фізичним екземпляром робочого простору є база даних із властивостями назви та опису. Новий користувач не зіштовхується із потребою вибору місця розташування, алгоритм побудований таким чином, що простір вибирається той, де більше вільного місця. Функції персоналізації передбачають можливість зміни місця збереження бази даних. Також в наступних версіях планується реалізація функцій експорту/імпорту даних, автозбереження на віддаленому сервері або в сервісі хмарі.

3.4 Тестування мобільного застосунку

Існує велика кількість методів тестування мобільних застосунків [41,42]. Для тестування було використано три основні методи та обґрунтовано їх вибір.

Функціональне тестування – це методика перевірки програмного забезпечення, яка зосереджена на визначенні відповідності програми її функціональним специфікаціям. Головна мета функціональних тестів полягає в перевірці кожної окремої функції програмного продукту шляхом подачі відповідних вхідних даних і аналізу вихідних даних на предмет відповідності очікуваним результатам. Функціональне тестування, яке в основному базується на методології чорного ящика, не зосереджується на внутрішній структурі програми. Цей тип тестування оцінює роботу інтерфейсу користувача, API, бази даних, системи безпеки, а також інших функцій програми. Функціональне тестування може проводитися як вручну, так і з використанням автоматизованих інструментів.

Першим методом тестування було вибрано функціональне тестування чорного ящика за допомогою сценаріїв. Це процес, в якому тестувальник не має знань про внутрішню структуру програмного забезпечення. Він працює тільки з вхідними й вихідними даними, не звертаючи уваги на внутрішній код програми. Сценарії для тестування – це послідовність кроків, які виконуються для перевірки певного аспекту або функції програми. Вони включають вхідні дані, очікувані результати та кроки, необхідні для досягнення цих результатів. Спочатку перевіряються вимоги до системи. Дефекти, якщо такі є, усуваються та перевіряються повторно. Список деяких сценаріїв наведений в таблиці 7.

Таблиця 7 – Сценарії тестування

Ідентифі-катор	Модуль	Мета	Вхідні значення	Очікуваний результат
1	2	3	4	5
К1	Конструктор	Створити валюту	Назва: Гривня Абревіатура: UAH Опис: Українська гривня.	Повідомлення про успіх
К2	Конструктор	Створити групу	Назва: Гаманці	Повідомлення про успіх
К3	Конструктор	Створити рахунок	Назва: Основний гаманець Валюта: Гривня Група: Гаманці	Повідомлення про успіх

Продовження таблиці 7

1	2	3	4	5
Р1	Робочий простір	Створити робочий простір	Назва: Новий	Повідомлення про успіх
Н1	Налаштування	Змінити мову інтерфейсу.	Опція: Англійська мова	Інтерфейс перекладений англійською мовою

Тестування мобільного застосунку відбувалося для кожного модуля в середовищі розробки Android Studio з використанням емулятора. Емулятори є простим способом тестувати програму на мобільному пристрої, не використовуючи його фізично [43]. Такий спосіб тестування зменшує економічні витрати бюджету. Отже, другим способом тестування є використання емуляторів, це дозволяє швидко виправляти недоліки інтерфейсу користувача та покращити зручність використання шляхом адаптації на нових пристроях. Дане тестування може проводитися на різних варіаціях розмірностей екранів та версій операційної системи Android.

Було обрано два шляхи використання цього методу. Перший, це загальні оцінки відповідності реалізації інтерфейсу до прототипу дизайну, завдяки цьому шляху було налагоджено коректну роботу всіх трьох типів меню навігації. Другий, це проведення раніше створених сценаріїв тестування, матриця результатів показана в таблиці 8.

Таблиця 8 – Результати тестових сценаріїв для різних пристроїв

Назва пристрою та його характеристики	Ідентифікатор тестового сценарію				
	К1	К2	К3	Р1	Н1
Pixel XS, API 27, 1440x2560, 412x732 dps	+	+	+	+	+
Pixel C, API 27, 2560x1800, 1280x900 dps	+	+	+	+	+
Pixel 6 Pro, API 34, 1440x3120, 412x892 dps	+	+	+	-	+
ZTE Blade V10, API 28, 1080x2280, 400x840 dps	+	+	+	+	-

Як показано в таблиці 8, два тестові сценарії не пройшли успішно, після детального розгляду проблеми вдалося здійснити успішне налагодження.

Третім методом було обрано тестування юзабіліті, це процес оцінки продукту, системи або сервісу з погляду кінцевого користувача. Це охоплює розуміння того, наскільки легко користувач може виконувати основні завдання, наскільки інтуїтивно зрозумілим є інтерфейс, і як продукт відповідає на очікування користувача. Тестування юзабіліті також вимірює задоволення користувачів від використання продукту. За допомогою тестування юзабіліті можна виявити та розв'язати проблеми ще до того, як вони стануть проблемами для кінцевих користувачів. Виявлені проблеми зведені в таблицю 9.

Таблиця 9 – Результати тестування юзабіліті

Кодове ім’я тестувальника	Стан продукту	Дефект
Олег	Інтерактивний прототип	Необхідність збільшити розмір кнопок навігації.
Юля	Інтерактивний прототип	Вибрана колірна гама має низьку контрастність.
Ігор	Інтерактивний прототип	Анімація переходу на новий екран проходить зашвидко.
Тарас	Реалізований продукт	Виявлено неприємний ефект блимання під час переходу на новий екран.
Сергій	Реалізований продукт	Під час зміни орієнтації екрана заповнені дані у формі введення пропадають.
Таня	Реалізований продукт	Надмірний опис дій, що відбудуться після натискання на кнопку.
Іван	Реалізований продукт	Елементи із меню навігації знаходяться надто низько, що не зручно для використання.

Всі виявлені дефекти були взяти до уваги, більшість виправлена. Дефекти отримані тестувальниками Олегом та Іваном будуть виправлені в наступних версіях програмного продукту.

Під час розробки модулів також використовувався метод білої скриньки, коли тестувальнику відомо деталі програмної реалізації. Таким чином було перевірено всі доступні варіанти поведінки програмного забезпечення.

На рисунку 32 відображені знімки екранів результатів після виконання тестових сценаріїв К1-К2. Успішно створені об’єкти валют та груп.

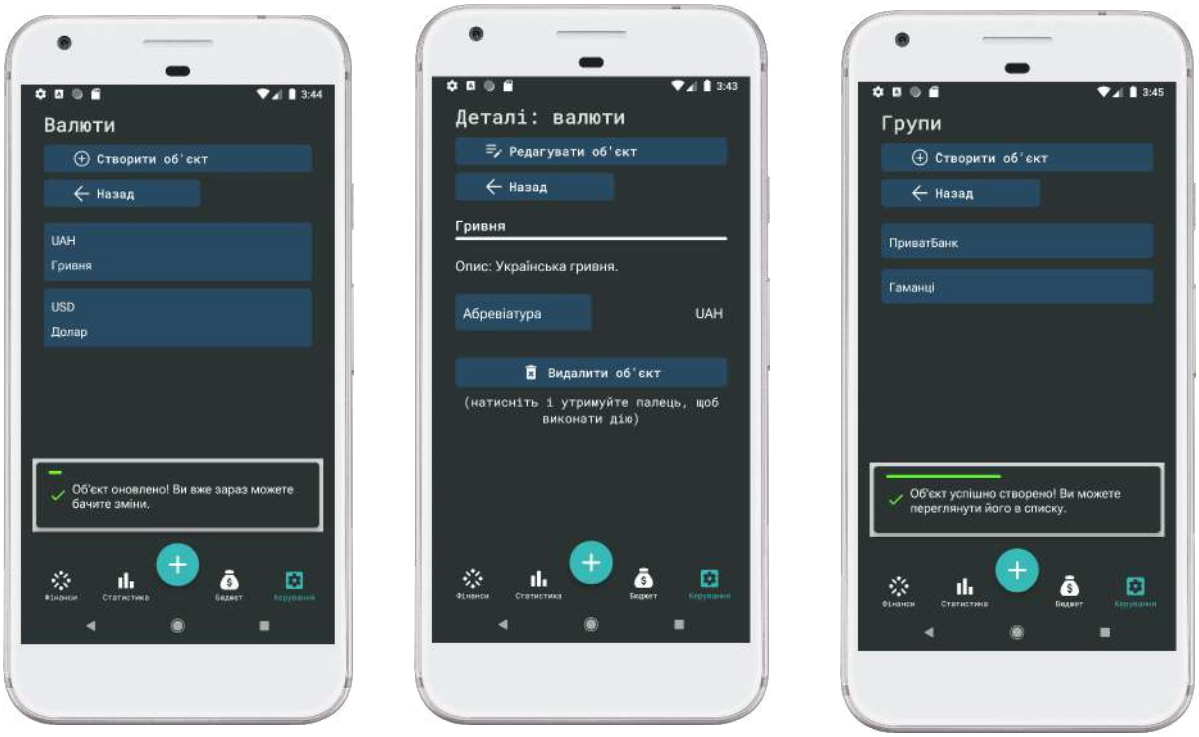


Рисунок 32 – Вікна інтерфейсу результатів тестових сценаріїв К1-К2

На рисунку 33 відображені знімки екранів результату після виконання тестового сценарію К3. Успішно створено рахунок, можна також переглянути його деталі. Всі форми створення об’єктів мають однакову стилізацію.

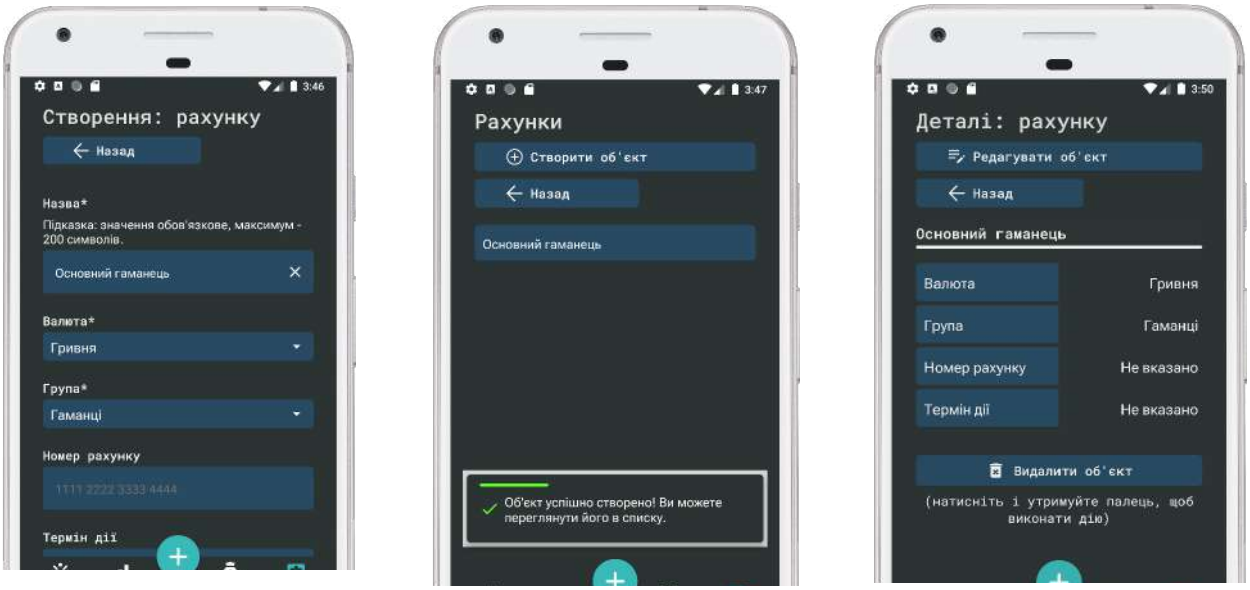


Рисунок 33 – Вікна інтерфейсу результату тестового сценарію К3

На рисунку 34 відображені знімки екранів результату після виконання тестового сценарію Р1. Робочий простір може зберігатися в пам’яті пристрою або на карті пам’яті. В наступних версіях буде впроваджена функціональність зміни, даний модуль частково реалізований в поточній версії.

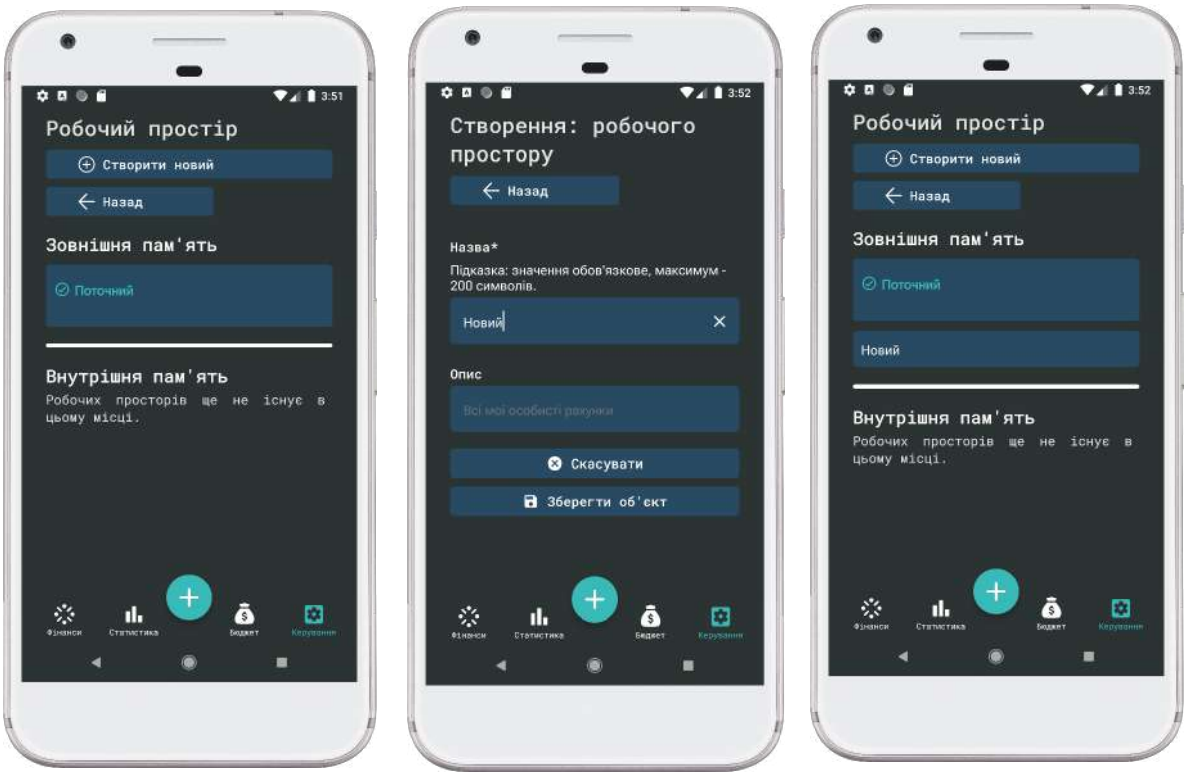


Рисунок 34 – Вікна інтерфейсу результату тестового сценарію Р1

Вибрані методи тестування дозволили завчасно виявити та виправити дефекти на ранніх стадіях розробки програмного забезпечення. Існує ще велика кількість методів тестування на які варто звернути увагу. Наприклад, продуктивності під час створення елементів інтерфейсу, що відображають наявні артефакти транзакцій. Модульне, інтеграційне, безпеки, регресійне, локалізації – це тести, що повинні проводитися з кожною версією програмного продукту.

3.5 Керівництво користувача

Мобільний застосунок варто завантажувати тільки з офіційних джерел розповсюдження. Власники програмного продукту розуміють чутливість

фінансових даних, і намагатимуться надати безпечний та надійний спосіб обробки даних. При відкритті програмного продукту користувачі погоджуються, що розробник не несе відповідальності за збитки, отримані в будь-якій формі через втрату частини або всіх даних, їх компрометування або пошкодження, часткове або повне оприлюднення, втрати доступу до інформації.

Мобільний застосунок інсталюється в пам’ять пристрою, дані мобільного застосунку можуть бути частково розподілені між зовнішньою та внутрішньою пам’яттю. Питання щодо переміщення та транспортування бази даних розглядаються в блоці «FQA» (часті запитання та відповіді), також там розглядаються можливості міграції даних з одного пристрою на інший. Оновлення мобільного застосунку не спричиняє втрати попередніх даних користувача. При виникненні додаткових питань або партнерства, звертайтеся на електронну адресу підтримки мобільного застосунку.

При виникненні труднощів у використані, користувач може перейти на екран допомоги за кнопкою, що знаходиться меню. Користувач має право отримати безплатну технічну допомогу, оцінювати та залишати відгуки в офіційних дистриб'юторів програмного забезпечення. Розробки мобільного застосунку залишає за собою право змінювати умови використання програмного продукту в будь-який час.

3.6 Технічні характеристики мобільного застосунку

Мобільний застосунок розроблявся під операційну систему Android 8.1 та вище. Інтегроване середовище розробки Android Studio має великі можливості для аналізу продуктивності мобільної програми. Серед його інструментів є профайлер, що дозволяє визначити навантаження на операційну систему та споживані ресурси під час використання мобільного застосунку. Результати на рисунку 35 свідчать про те, що програмне забезпечення помірно споживає ресурси процесора та заряд батареї, оперативна пам’ять використовується в межах 200-400 МБ.

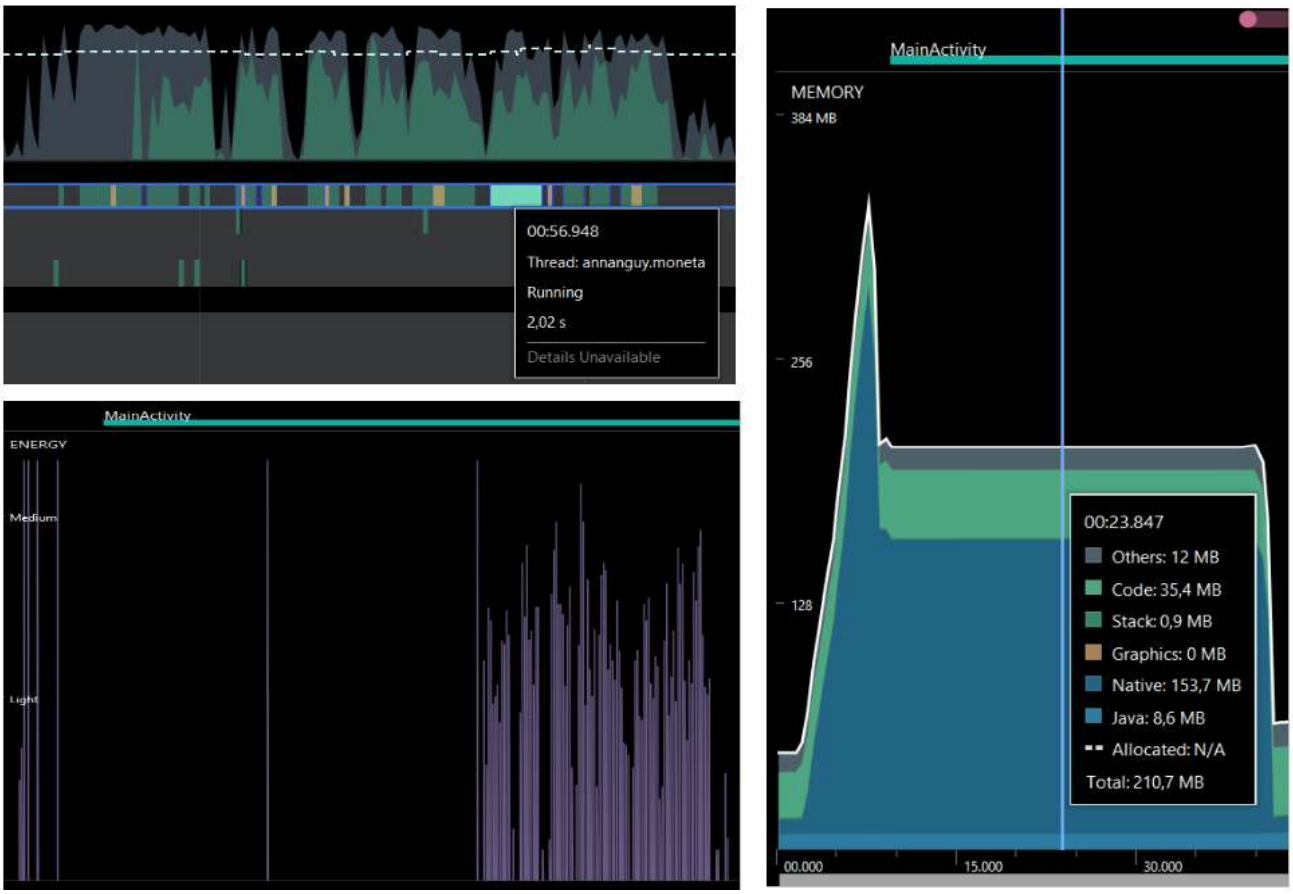


Рисунок 35 – Результати роботи профайлера

Отже, після проведення аналізу були отримані наступні програмні характеристики мобільного застосунку. Мінімальна конфігурація:

– Android версії 8.1;

– частота процесора 600 МГц;

– 400 МБ ОЗП;

– 20 МБ вільної пам'яті для встановлення застосунку;

– 1 МБ для мінімальної структури бази даних.

Конфігурація, що рекомендується:

– Android версії 8.1 та вище;

– частота процесора 800 МГц;

– 600 МБ ОЗП;

– 40 МБ вільної пам'яті для встановлення застосунку;

– 10 МБ для мінімальної структури бази даних.

В мобільному застосунку були реалізовано всі функціональні можливості, що визначені в першому розділі під час постановки задачі. Серед них було реалізовано два основні модулі конструктора та робочого простору. Також наявна велика кількість допоміжних функцій: різні типи навігаційних меню, можливість зміни мови інтерфейсу, модуль діагностики помилок. Програмний продукт має великий потенціал для розвитку і функціональних оновлень у своїх майбутніх версіях. Можна виділити наступні перспективи розвитку мобільного застосунку:

– розробка модуля створення транзакцій;

– розробка модуля відстеження фінансів;

– модуль статистики;

– безпекові функції, наприклад ПІН-код, біометрія;

– автозбереження даних на серверах;

– отримання можливості синхронізації фінансових операцій з інтернет-банкінгу для рахунків;

– можливість збереження інвойсів для податкової звітності.

Отже, можна зробити висновки. В даному розділі було проведено роботу по реалізації та тестуванню мобільного застосунку. Спочатку було описано принципи створення стилів та тем інтерфейсу користувача. Далі розглянуто діаграму станів підготовчих процесів мобільної програми, описано проєктування пакета, що відповідає за їх виконання. Зроблено акцент на функціональних можливостях модуля діагностики помилок. Також було описано структурні та архітектурні шаблони нової функціональності, розглянуто загальну структуру пакетів, показано діаграму послідовностей для одного з варіантів використання. Наступним кроком описано, як працюють функції взаємодії з базою даних. Складено діаграму реалізованих UI компонентів та показано результати реалізації інтерфейсу користувача. Створено базу даних та наведено її фізичну ER-діаграму. Потім було проведено належне тестування, що гарантує відповідність програмного продукту специфікаціям вимог технічного завдання. Було викладено та оцінено технічні характеристики програмного забезпечення, написано інструкцію користувача.

ВИСНОВКИ

При виконанні кваліфікаційної роботи було проведено аналіз предметної області та дослідження основних тенденцій її розвитку, визначено проблеми цільової аудиторії та слабкі сторони наявного програмного забезпечення, визначено особливості та специфіку процесу обліку доходів і витрат.

Після цього було визначено функціональні задачі мобільного застосунку, на основі яких було створено вимоги до програмного забезпечення. Для якісного виконання поставлених задач було розроблено технічне завдання.

Далі було спроєктовано базу даних, що відповідає інформаційним потокам системи, розроблено інтерфейс користувача, який дозволить розв'язувати функціональні задачі, спроєктовано логіку роботи мобільного застосунку, відібрано актуальні технології для реалізації програмного продукту. Опираючись на отримані артефакти проєктування було здійснено пошук архітектурних рішень та синтезовано гібридну VIPER архітектуру, чиї переваги покращують якість рішення, а недоліки не заважатимуть процесу розробки.

На основі проєктної документації, виконано програмну реалізацію двох відібраних модулів застосунку. Після розробки проведено функціональне тестування чорного ящика згідно з розробленими сценаріями та емуляторами, здійснено перевірку юзабіліті. Отримані результати дозволити вдосконалити ПП.

В результаті цього розроблений застосунок отримав значні функції персоналізації робочого простору, яких немає більшість аналогів даного виду програмного забезпечення.

При виконанні кваліфікаційної роботи було розроблено мобільний застосунок під операційну систему Android для ведення особистих витрат і доходів, що допомагає зберігати інформацію про проведені грошові операції, керувати власними фінансами, переглядати історію транзакцій в зручній формі. Виконано всі поставлені задачі проєктування, дотримано вимог якості, що були визначені в технічному завдані.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Raynor de Best. Number of cryptocurrencies worldwide from 2013 to January 2024. Statista. 2024. URL: https://www.statista.com/statistics/863917/number-crypto-coins-tokens/ (дата звернення: 08.02.2024).

2. Проєкт USAID «Трансформація фінансового сектору». Фінансова грамотність, фінансова інклюзія та фінансовий добробут в Україні у 2021. 2021. С. 83. URL: https://bank.gov.ua/admin_uploads/article/Research_Financial_Literacy_Inclusion_Welfare_2021.pdf?v=4 (дата звернення: 10.02.2024).

3. Національний банк України. Оприлюднено Рамку фінансових компетентностей дітей та молоді України. 2024. URL: https://bank.gov.ua/ua/news/all/oprilyudneno-ramku-finansovih-kompetentnostey-ditey-ta-molodi-ukrayini (дата звернення: 09.02.2024).

4. Наскрізна практична підготовка : програма та методичні вказівки щодо її організації та виконання студентами спеціальності 121 «Інженерія програмного забезпечення» / ред.: В. С. Яремчук, В. П. Карпанасюк ; уклад.: Г. І. Радельчук, Л. П. Бедратюк. Хмельницький : Хмельн. нац. ун-т, 2021. 40 с.

5. Sherif A. Global market share held by mobile operating systems from 2009 to 2023, by quarter. Statista. 2024. URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (date of access: 16.02.2024).

6. Statcounter GlobalStats. Statcounter GlobalStats. Mobile operating system market share worldwide. Statcounter GlobalStats. URL: https://gs.statcounter.com/os-market-share/mobile/worldwide (date of access: 17.02.2024).

7. Bitrián P., Buil I., Catalán S. Enhancing user engagement: the role of gamification in mobile apps. Journal of business research. 2021. Vol. 132. P. 170–185. URL: https://doi.org/10.1016/j.jbusres.2021.04.028 (date of access: 29.05.2024).

8. Ролік В. Прості інструменти, які допоможуть навчити дітей фінансовій грамотності. Forbes Ukraine. 2023. URL: https://forbes.ua/lifestyle/prosti-instrumenti-

yaki-dopomozhut-navchiti-ditey-finansoviy-gramotnosti-13042021-1357 (дата звернення: 16.02.2024).

9. Kalbach J. The jobs to be done playbook: align your markets, organization, and strategy around customer needs. Two Waves Books, 2020. 320 p.

10. Hernandez J M. Database design for mere mortals. 4th ed. Addison-Wesley Professional, 2020. 640 p.

11. Maslin S. Designing Mind-Friendly Environments. Jessica Kingsley Publishers, 2021. 272 p.

12. Saving data on android (second edition): learn jetpack datastore, room, firebase & sqlite with kotlin / Raywenderlich Tutorial Team et al. Razeware LLC, 2021. 344 p.

13. Liang Y. Introduction to Java programming and data structures, comprehensive version. 12th ed. Pearson, 2021. 1240 p.

14. Brown M. Big O notation simplified: mastering time complexity analysis. Independently published, 2023. 45 p.

15. Coders Hub. Kotlin java but better: lined Kotlin android programming. Independently published, 2024. 110 p.

16. Phang C. L. XML: a quick guide to all aspects. Independently published, 2022. 114 p.

17. Künneth T. Android UI development with Jetpack Compose: bring declarative and native UI to life quickly and easily on android using Jetpack Compose and Kotlin. 2nd ed. Birmingham, United Kingdom : Packt Publishing, 2023. 278 p.

18. Ghita C. Kickstart Modern Android Development with Jetpack and Kotlin: Enhance your applications by integrating Jetpack and applying modern app architectural concepts. Packt Publishing, 2022. 472 p.

19. Rose R. Flutter and dart cookbook: developing full-stack applications for the cloud. O'Reilly Media, 2023. 307 p.

20. Alessandria S. Flutter Cookbook - Second Edition: 100+ real-world recipes to build cross-platform applications with Flutter 3.x powered by Dart 3 (alpha). 2nd ed. Packt Publishing, 2023. 712 p.

21. Sedunov A. Kotlin in-depth: a guide to a multipurpose programming language for server-side, front-end, android, and multiplatform mobile. BPB Publications, 2022. 687 p.

22. Spilca L. Troubleshooting Java: Read, debug, and optimize JVM applications. Manning, 2023. 328 p.

23. Hajian M. Flutter engineering. Staten House, 2024. 545 p.

24. Smyth N. Android Studio Jellyfish essentials: developing android apps using Android Studio 2023.3.1 and Kotlin. Payload Media, Inc., 2024. 1331 p.

25. Smyth N. Android studio iguana essentials - Kotlin edition: developing android apps using Android Studio 2023.2.1 and Kotlin. Payload Media, 2024. 754 p.

26. Darmawikarta D. Learning SQL in h2 database. Independently published, 2022. 96 p.

27. Santana O. Java Persistence with NoSQL: Revolutionize your Java apps with NoSQL integration (English Edition). BPB Publications, 2024. 366 p.

28. Jeffery C. L., Ahmad I. Build your own Programming Language - Second Edition: A programmer's guide to designing compilers, interpreters, and DSLs for modern computing problems. 2nd ed. Packt Publishing, 2024. 556 p.

29. Kurur N. P. Mastering Java Persistence API (JPA): realize java's capabilities spanning RDBMS, ORM, JDBC, caching, locking, transaction management, and JPQL (english edition). BPB Publications, 2022. 258 p.

30. Gregory T. Gradle Build Bible: The ultimate guide to mastering Gradle projects. Kindle, 2023. 410 p.

31. Shahwaiz A. Effective android app architecture: best practices: be awesome. Independently published, 2023. 65 p.

32. Kumar P. Building android projects with Kotlin: use android SDK, Jetpack, material design, and Junit to build android and JVM apps that are secure and modular. BPB Publications, 2022. 589 p.

33. Jain D. Ultimate Laravel for modern web development: build robust and interactive enterprise-grade web apps using Laravel's MVC, authentication, APIs, and cloud deployment (english edition). Orange Education Pvt Ltd, 2024. 280 p.

34. García R. F. IOS architecture patterns: MVC, MVP, MVVM, VIPER, and VIP in Swift. Apress, 2023. 415 p.

35. Nijs P. The MVVM Pattern in .NET MAUI: the definitive guide to essential patterns, best practices, and techniques for cross-platform app development. Packt Publishing, 2023. 386 p.

36. Trivedi H. Android application development with Kotlin: build your first android app in no time (english edition). BPB Publications, 2020. 402 p.

37. Cardoso A. F. M. Implementing design patterns in C# and .NET 5: build scalable, fast, and reliable .NET applications using the most common design patterns (english edition). BPB Publications, 2021. 290 p.

38. Hochwender J. The complete guide to the software development lifecycle. Independently published, 2023. 182 p.

39. Сазерленд Дж. Scrum. Навчись робити вдвічі більше за менший час. Наш Формат, 2022. 280 с.

40. Soshin A. Kotlin Design Patterns and Best Practices - Third Edition: Elevate your Kotlin skills with classical and modern design patterns, coroutines, and microservices. 3rd ed. Packt Publishing, 2024. 474 p.

41. Knott D. Hands-On mobile app testing: a guide for mobile testers and anyone involved in the mobile app business. 2nd ed. Independently published, 2022. 267 p.

42. Heusser M. Software testing strategies: a testing guide for the 2020s. Packt Publishing, 2023. 378 p.

43. Roybal III G. T. Penetration Testing with Kali NetHunter: Hands-on Android and iOS penetration testing. BPB Publications, 2024. 350 p.

Додаток А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ НА РОЗРОБКУ МОБІЛЬНОГО ЗАСТОСУНКУ

Технічне завдання

1 Словник

Абревіатури

1. ПЗ – Програмне Забезпечення
2. ПП – Програмний Продукт
3. ID – Identification
4. СКБД – Система Управління Базами Даних
5. DP – Density-Independent Pixels
6. ADHD – Attention Deficit Hyperactivity Disorder

Терміни

1. Програмне забезпечення – це сукупність програм системи оброблення інформації та програмних документів, необхідних для експлуатації цих програм.
2. Програмний продукт – це програмне забезпечення, розроблене для розв'язання задачі масового попиту та призначене для постачання користувачам.
3. Система управління базами даних – це набір взаємопов'язаних даних і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.
4. ID – ім'я об'єкта програми, що дозволяє звернутись до об'єкта; ознака, яка цілком визначає сутність, в наперед визначеному просторі.
5. Логи (англ. logs) – спеціальні файли, в яких накопичується зібрана службова та статистична інформація про події в мобільному застосунку.
6. Density-Independent Pixels – гнучкі одиниці вимірювання, які масштабуються, щоб мати однакові розміри на будь-якому екрані.

7. СДУГ (англ. ADHD) – неврологічно-поведінковий синдром розладу розвитку, що починається в дитячому віці. Проявляється такими симптомами, як труднощі концентрації уваги, гіперактивність і погано керована імпульсивність.

2 Загальна інформація

Найменування розробки: «Mon Money: фінанси та спокій».

Замовник: Василик Віктор Михайлович, Хмельницький національний університет.

Розробник: Василик Віктор Михайлович.

Початок роботи – 21 січня 2024 року, завершення – 30 травня 2024.

Підставою для розробки є документ «Завдання на кваліфікаційну роботу» затверджене завідувачем кафедри інженерії програмного забезпечення.

Сфера застосування: фінанси, бізнес, особисте користування.

Цільова аудиторія: україномовні та англомовні користувачі, що здійснюють грошові витрати чи отримують прибуток, мають фінансову грамотність.

Мета: розробити мобільний застосунок з дотримання всіх вимог технічного завдання при цьому не порушити графік роботи, використовувати гнучку методологію розробки та по фазну перевірку цілісності початкових вимог програмного забезпечення.

Якщо проєкт вийде за часові рамки через свій великий обсяг функціональних вимог та ризики керування, акцент варто зробити на системі персоналізації.

Задачі, що вирішує застосунок: відстеження грошового потоку, аналіз та розрахунок статистики для грошових рахунків, наочне представлення даних у формі діаграм, ручне введення даних про операції.

Призначення та опис розробки. Український застосунок для ведення щоденних витрат, доходів, планування бюджету, як особистого, так і колективного.

Наділений унікальними функціями та ефективними інструментами, щоб зробити ваше життя легшим, а думки спокійними. Вам більше не потрібні десятки банківських програм для відстеження балансу, всі ваші рахунки будуть в одній програмі. Простий інтерфейс дозволить відстежувати всі операції, в цьому вам допоможе наша аналітика та наочні діаграми. Скоро на платформі Play Market!

3 Технології та сценарії

Платформи: Android.

Пристрої, що підтримуються: Android версії 8.1 та вище.

Мова та інструменти реалізації: Kotlin, Jetpack Compose.

Локалізація: українська мова, англійська мова.

4 Вимоги до програмного продукту

4.1 Функціональні вимоги

А Меню

1. Мобільний застосунок має три типи меню для різних розмірів екранів, згідно зі специфікацією Jetpack Compose їх назви: Bottom Navigation (малі екрани, менш як 600 dp), Drawer (середні екрани, менш як 840 dp), Rail (великі екрани, більше або рівно 840 dp).
2. Наявність по середині меню завислої кнопки дії для малих екранів, на середніх та великих вона знаходиться в лівому нижньому куті.

В Екран «Фінанси», «Finances»

1. Виводить список рахунків.
2. Відображає групи рахунків.
3. Відображає дані про рахунки.

C Екран «Рахунок», «Account»

1. Показує дні коли відбувалися транзакції.
2. Відображає категорії транзакції.
3. Відображає загальну інформацію про ці транзакції.

D Екран «Транзакції дня», «Day Transactions»

1. Відображає транзакції певної категорії за вибраний день.
2. Відображає інформацію про ці транзакції.

E Екран «Деталі транзакції», «Transaction Details»

1. Відображає таблицю з деталями вибраної транзакції.
2. Містить кнопку для редагування об’єкта.
3. Містить кнопку для видалення об’єкта.

F Екран «Створення транзакції», «New transaction»

1. Можливість введення даних про нову транзакцію.
2. Збереження запису транзакції в базу даних.

G Екран «Статистика», «Statistics»

Показує можливі діаграми до перегляду та їх опис.

H Екран «Розподіл за категоріями», «Distribution by categories»

1. Наявні дві опції для уточнення вибірки даних для яких рахунків та груп, за який часовий період.

2. Формує статистику розподілу за категоріями для двох розділів: витрат й доходів.
3. Кожний розділ містить кругову діаграму для візуалізації даних.
4. Кожний розділ містить таблицю всіх даних статистики.

I Екран «Бюджет», «Budget»

Пустий екран, з повідомленням про розробку функції.

J Екран «Керування», «Manage»

1. Містить кнопки посилання на інші екрани та їх опис.
2. Можливість відкрити потрібні діалоги та інші екрани.

K Екран «Перейти на повну версію програми», «Get full version, go premium»

1. Містить переваги платної версії.
2. Наявна кнопка для покупки платної версії програми.

L Діалог «Про програму», «About program»

Ознайомлює користувача з основними можливостями застосунку.

M Діалог «Налаштування», «Settings»

1. Має можливість змінити локалізацію інтерфейсу.
2. Має можливість для зміни місця зберігання даних (телефон, SD-карта).

N Діалог «Політика та Умови використання», «Politics and Term of use»

Містить кнопки для від відкриття всіх підрозділів політики окремо.

O Діалог «Наші контакти», «Out contacts»

1. Містить ім’я розробника і робочу електронну пошту.
2. Має можливість скопіювати пошту.

P Екран «Конструктор», «Constructor»

Містить категорії об’єктів, які можна створювати й редагувати.

Q Екран списку об’єктів

Відображає список уже створених об’єктів.

R Екран «Деталі об’єкта», «Object Details»

1. Відображає таблицю з деталями об’єкта.
2. Містить кнопку для редагування об’єкта.
3. Містить кнопку для видалення об’єкта.

S Екран «Створення об’єкта», «Creating an object» / «Редагування об’єкта», «Editing an object»

1. Містить поля для введення даних про об’єкт.
2. Має можливість редагувати попередньо завантажену інформацію.
3. Має кнопку для збереження та скасування змін.

T Значення за умовчуванням

При першому запуску програми наявні базові об’єкти для оперування.

U Екран «Робочий простір», «Workspace»

1. Містить список доступних робочих просторів.
2. Містить функціонал створення та перемикання робочих просторів.

V Екран «Деталі робочого простору», «Workspace details»

1. Відображає таблицю з деталями робочого простору.
2. Містить кнопку для редагування робочого простору.
3. Містить кнопку для видалення робочого простору.

W Екран «Створення робочого простору», «Creating a workspace» / «Редагування робочого простору», «Editing a workspace»

1. Містить поля для введення даних про робочий простір.
2. Має можливість редагувати попередньо завантажену інформацію.
3. Має кнопку для збереження та скасування змін.

4.2 Нефункціональні вимоги та опис екранів застосунку

0 Загальні

1. Мінімалістичний, універсальний дизайн із явно вираженим одним яскравим кольором, задній фон темний.
2. Дизайн, графи навігації, маршрути дій, структура представлення складних даних – повинні бути оптимізовані під людей із синдромом дефіциту уваги та гіперактивності (СДУГ, англ. ADHD).
3. Ввід тексту відбувається системною клавіатурою клієнта.
4. Правила застосунку мають версію у формі дати, при зміні дати правил користувачі повинні заново прийняти їх.
5. Кожний екран та діалог має заголовок.
6. Кожний екран, що не має навігації через меню, має кнопку «Назад», «Back» для повернення до попереднього екрана.
7. Кожний діалог має кнопку «Закрити», «Close».

8. Важкі та довготривалі дії супроводжують анімацією очікування.

A Меню

1. Містить 4 кнопки із відповідними іконками для переходу між інтерфейсами програми: «Фінанси», «Finances»; «Статистика», «Statistics»; «Бюджет», «Budget»; «Керування», «Manage».
2. Містить активну кнопку,що розміщується залежно від розміру екранів пристрою, що відкриває екран «Створення транзакції», «New transaction».

B Екран «Фінанси», «Finances»

1. Вивід списку рахунків із розподілом на групи.
2. Кожний рахунок має власну назву та групу.
3. При натисканні на карту, відбувається перехід в екран «Рахунок», «Account».
4. Екран показує наступну інформацію:
   - Рахунки у формі карточок, із назвою, загальною сумою та валютою.
   - Номер рахунку і термін придатності карти, якщо встановлений.
   - Групує рахунки в блоки із заголовками.

C Екран «Рахунок», «Account»

1. Містить детальну інформацію про транзакції, що відбувалися, у формі списку для гортання, де розміщені карточки транзакцій.
2. Групує всі транзакції по днях у великі блоки карточки.
3. Групує всі транзакції по категоріях в маленькі блоки карточки.
4. Екран показує наступну інформацію:
   - Дату великої групи.
   - Кількість транзакцій певної категорії в малій групі.
   - Загальна сума транзакцій в певній категорії.
   - Загальну суму транзакцій за весь день.

D Екран «Транзакції дня», «Day Transactions»

1. Відображає транзакції певної категорії за вибраний день у формі списку карточок.
2. Блок карточки складається з:
    - Назви та опису транзакції.
    - Суми та валюти.
    - Дати й часу проведення.

E Екран «Деталі транзакції», «Transaction Details»

1. Відображає деталі вибраної транзакції у формі списку.
2. Атрибути транзакції для показу:
    - Назву та опис.
    - Дата і час проведення.
    - Суму і валюту.
    - Тип транзакції.
    - Категорія.
    - Прив’язка до рахунку.
    - Група рахунку.

F Екран «Створення транзакції», «New transaction»

1. Матриця вибору в якій вводиться інформація про нову транзакцію: назва, сума, дата і час, вибір категорії транзакції, вибір рахунку, тип транзакції, опис.
2. Для збереження транзакції користувач має натиснути на клавішу «Зберегти».
3. Наявна валідація вводу користувача.

G Екран «Статистика», «Statistics»

1. Виводить список доступних діаграм у вигляді карточок для аналізу грошового потоку.

2. Кожний елемент списку має назву, іконку та опис.

H Екран «Розподіл за категоріями», «Distribution by categories»

1. Перша кнопка уточнення вибірки даних дозволяє вибрати як окремі рахунки, так і декілька, або ж цілу групу, декілька груп.
2. Друга кнопка уточнення вибірки даних дозволяє вибрати стандартні періоди часу або вказати свій. За умовчуванням: цей місяць. Можливі варіанти:
   - Цей тиждень.
   - Цей місяць.
   - Цей квартал.
   - Цей рік.
   - Останні 30 днів.
   - Останні 365 днів.
   - Власний діапазон.

I Екран «Бюджет», «Budget»

1. На екран має базові елементи, заголовок і елемент переходу із меню.
2. Контент містить тільки повідомлення з вибаченням, що дана функція не доступна, оскільки ведеться її розробка.

J Екран «Керування», «Manage»

1. Кнопка перейти на повну версію програми відкриває екран «Перейти на повну версію програми», «Get full version, go premium»
2. Кнопка про програму відкриває діалог «Про програму», «About program».
3. Кнопка налаштування відкриває діалог «Налаштування», «Settings.
4. Кнопка конструктор відкриває екран «Конструктор», «Constructor».
5. Кнопка політика та умови використання відкриває діалог «Політика та Умови використання», «Politics and Term of use».
6. Кнопка наші контакти відкриває діалог «Наші контакти», «Out contacts».

K Екран «Перейти на повну версію програми», «Get full version, go premium»

Один екраном показуються переваги платної версії, кожний пункт містить яскраву іконку, в низу списку кнопка для оформлення платної підписки.

L Діалог «Про програму», «About program»

У формі слайдів користувачу розказується про наявні функції програми.

M Діалог «Налаштування», «Settings»

1. Користувач може змінити мову інтерфейсу, список доступних мов у формі випадного списку.
2. Користувач може змінити місце для збереження даних, опції представлені у вигляді радіо кнопок, одночасно може бути вибрана тільки одна опція.

N Діалог «Політика та Умови використання», «Politics and Term of use»

Кожний документ для читання представлений у формі кнопки, що відкриває діалог другого рівня, де можна прочитати його.

O Діалог «Наші контакти», «Out contacts»

1. Містить два пункти у формі заголовків та тексту.
2. Першим пунктом йде інформація про розробника, другим пунктом йде електронна пошта, яку можна скопіювати.

P Екран «Конструктор», «Constructor»

У формі списку карточок показує можливі об’єкти для взаємодії, при натисканні на елемент, відкривається екран списку об’єктів.

Q Екран списку об’єктів

У формі списку карточок показує імена та опис об’єктів, при натисканні на них відбувається відкриття екрана «Деталі об’єкта», «Object Details».

R Екран «Деталі об’єкта», «Object Details»

1. Відображає всі атрибути об’єкта у формі таблиці.
2. Над таблицею знаходиться кнопка для редагування.
3. Під таблицею знаходиться кнопка для видалення.

S Екран «Створення об’єкта», «Creating an object» / «Редагування об’єкта», «Editing an object»

1. Містить форму, що будується зі стилізованих під дизайн програми компонентів. Якщо об’єкт був раніше створений, то його дані завантажуються в наявні поля для редагування. При створенні нового об’єкту ці поля пусті.
2. Під полями знаходяться кнопка скасувати, що закриває екран та кнопка зберегти, яка створює або оновлює об’єкт.
3. У формі наявна валідація вводу користувача.
4. Сповіщення про помилки відображаються під назвою поля або за допомогою сповіщень, що спливають. Наявний сценарій обробки дублікатів, тоді користувача повідомляють про помилку і просять змінити необхідні поля.

T Значення за умовчуванням

1. Популярні валюти: гривня, американський долар, євро, злотий, біткоїн, британський фунт.
2. Групи рахунків: гаманець, заощадження, банк.
3. Рахунки: мій гаманець, моя банківська карта, сейф.

4. Категорії: платежі, машина, одяг, комунікація, їжа, розваги, подарунки, здоров’я, дім, спорт, кафе, тварини, книги, навчання, робота, транспорт, таксі.
5. Типи: витрати, доходи.

U Екран «Робочий простір», «Workspace»

У формі списку карточок показує наявні робочі простори для взаємодії, при натисканні на елемент, відкривається екран деталей робочого простору.

V Екран «Деталі робочого простору», «Workspace details»

1. Відображає всі атрибути об’єкта у формі таблиці.
2. Над таблицею знаходиться кнопка для редагування.
3. Під таблицею знаходиться кнопка для видалення.

W Екран «Створення робочого простору», «Creating a workspace» / «Редагування робочого простору», «Editing a workspace»

1. Містить форму, що будується зі стилізованих під дизайн програми компонентів. Якщо об’єкт був раніше створений, то його дані завантажуються в наявні поля для редагування. При створенні нового робочого простору ці поля пусті.
2. Під полями знаходяться кнопка скасувати, що закриває екран та кнопка зберегти, яка створює або оновлює об’єкт.
3. У формі наявна валідація вводу користувача.
4. Сповіщення про помилки відображаються під назвою поля або за допомогою сповіщень, що спливають. Наявний сценарій обробки дублікатів, тоді користувача повідомляють про помилку і просять змінити необхідні поля.

4.3 Вимоги до надійності

1. Програма повинен повідомляти про виникнення альтернативних сценаріїв через повідомлення, що спливають.
2. При виникненні фатальних помилок, користувач буде повідомлений, а його екран зміниться на перший екран.

4.4 Вимоги до безпеки

1. Інформація записується як є, без шифрування та іншої обробки.
2. Інформація не передається через інтернет, обробляється та зберігається на пристрої користувача.

4.5 Технічні вимоги

1. База даних повинна бути побудована завдяки СКБД H2.
2. Мова програмування Kotlin.
3. Інструмент розробки Jetpack Compose.

4.6 Платіжні вимоги

Покупка платної версії програми здійснюється за допомогою сервісів Play Market.

4.7 Вимоги до складу та параметрів технічних засобів

Операційна система: Android; версія операційної системи 8.1 Oreo; оперативна пам’ять: 2 ГБ; вільна пам’ять пристрою: 100 МБ.

4.8 Вимоги до програмної документації

1. Документація генерується за допомогою інструменту Dokka із коментарів KDoc у формат локального сайту.
2. Інструкція користувача міститься в діалозі «Про програму», «About program».

5 Стадії та етапи проєкту

Стадії розробки	Етапи робіт	Зміст роботи
Ескізний проєкт початок: 1 січня 2024 кінець: 31 січня 2024	Пошук ідей, формування цілі проєкту, обґрунтування економічної доцільності.	Розробка ідеї; Формування гіпотез; Перевірка гіпотез; Знаходження майбутньої аудиторії користувачів. Вивчення попиту програмного забезпечення на ринку.
	Аналіз та вивчення предметної області	Пошук наявних рішень; Аналіз конкурентів.
	Розробка технічного завдання	Формування вимог до програмного продукту; Створення календарного плану роботи.
	Затвердження проєкту	Виключення суперечних вимог; Перевірка реальності термінів виконання; Підтвердження економічної доцільності.
Прототип початок: 1 лютого 2024 кінець: 29 лютого 2024	Створення прототипу	Створення продукту з деякою функціональністю.
	Тестування прототипу	Залучення користувачів для тестування; Перевірка системи на стресостійкість.
	Отримання відгуку від користувачів	Збір побажань та думок від користувачів.
Дослідження початок: 1 березня 2024	Аналіз тенденцій на ринку	Повторне вивчення конкурентів;

кінець: 31 березня 2024	Впровадження отриманого досвіду від прототипування	Корекція проєкту згідно з отриманим досвідом.
	Зміна внутрішніх документів.	Покращення фінансового плану; Адаптація технічного завдання для MVP;
	Затвердження проєкту	Повторна перевірка реальності термінів виконання; Повторне обґрунтування економічної доцільності. Розробка маркетингової стратегії.
Розробка MVP початок: 1 квітня 2024 кінець: 15 травня 2024	Виправлення недоліків прототипу	Вилучення слабких місць; Масштабування компонентів системи.
	Паралельне тестування	Після кожної ітерації розробки відбувається тестування.
	Розробка робочої версії	Впровадження покладених функцій технічного завдання.
Впровадження початок: 16 травня 2024 кінець: 31 травня 2024	Оформлення юридичних документів	Оформлення ліцензій використаних програмних компонентів; Перевірка авторського права використаних матеріалів; Розробка стратегій захисту програмного забезпечення.

6 Порядок контролю та приймання

6.1 Атрибути якості

1. Однорідний дизайн застосунку.
2. Стабільність програми до фатальних помилок.
3. Коректність роботи основних функцій програми.

6.2 Перевірка якості програмного забезпечення

Кожний етап розробки супроводжується тестуванням користувачів. Перевіряється відповідність вимогам.

6.3 Приймання роботи

Відповідальна особа: Василик Віктор Михайлович.

Особи що приймають: Василик Віктор Михайлович, Хмельницький національний університет.

Дата здачі: 1 червня 2024 року.

Додаток Б
(додатковий)

ДІАГРАМА ПАКЕТІВ

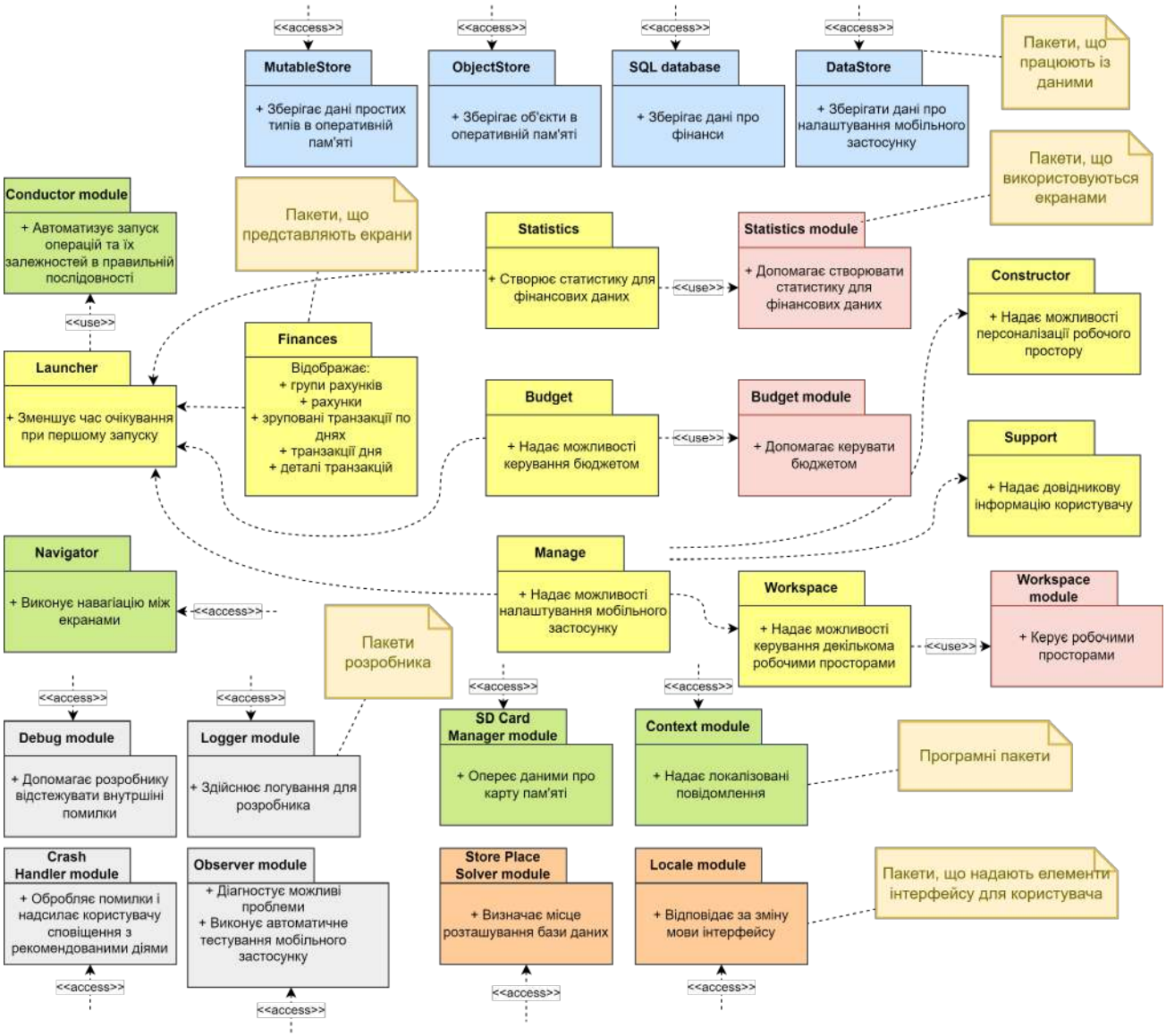


Рисунок Б.1 – Діаграма пакетів

Додаток В
(додатковий)

ЛІСТИНГ КОДУ

```
package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable
import org.jetbrains.exposed.sql.javatime.date

object AccountTable : IntIdTable(
    name = "Accounts"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
    var balance = float("balance")
    var validity = date("validity").nullable()
    var card_number = varchar("card_number", 200).nullable()
    var currency = reference("currency", CurrencyTable)
    var group = reference("group", GroupTable)
}

class AccountEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<AccountEntity>(AccountTable)
    var name by AccountTable.name
    var description by AccountTable.description
    var balance by AccountTable.balance
    var validity by AccountTable.validity
    var card_number by AccountTable.card_number
    var currency by CurrencyEntity referencedOn AccountTable.currency
    var group by GroupEntity referencedOn AccountTable.group
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable

object CategoryTable : IntIdTable(
    name = "Categories"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
    val icon = varchar("icon", 300)
}

class CategoryEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<CategoryEntity>(CategoryTable)
```

```
    var name by CategoryTable.name
    var description by CategoryTable.description
    var icon by CategoryTable.icon
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable

object CurrencyTable : IntIdTable(
    name = "Currencies"
) {
    val name = varchar("name", 200).uniqueIndex()
    val abbr = varchar("abbr", 10).uniqueIndex()
    val description = varchar("description", 300).nullable()
}

class CurrencyEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<CurrencyEntity>(CurrencyTable)
    var name by CurrencyTable.name
    var abbr by CurrencyTable.abbr
    var description by CurrencyTable.description
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable

/**
 * Store information about the database version etc.
 */
object DatabaseInfoTable : IntIdTable(
    name = "DatabaseInfo"
) {
    val property = varchar("property", 100).uniqueIndex()
    val data = varchar("data", 100)
}

class DatabaseInfoEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<DatabaseInfoEntity>(DatabaseInfoTable) {
        fun createOrUpdate(property: String, data: String) {
            val existingSettings = DatabaseInfoEntity.find
{ DatabaseInfoTable.property eq property }.singleOrNull()
            if (existingSettings != null) {
                existingSettings.data = data
            } else {
                DatabaseInfoEntity.new {
                    this.property = property
                    this.data = data
                }
            }
        }
```

```
    }

    var property by DatabaseInfoTable.property
    var data by DatabaseInfoTable.data
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable

object GroupTable : IntIdTable(
    name = "Groups"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
}

class GroupEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<GroupEntity>(GroupTable)
    var name by GroupTable.name
    var description by GroupTable.description
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable
import org.jetbrains.exposed.sql.Table
import org.jetbrains.exposed.sql.javatime.datetime

object StatisticsPresetTable : IntIdTable(
    name = "StatisticPresets"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
    val duration_type = varchar("duration_type", 2).nullable()
    val start = datetime("start").nullable()
    val end = datetime("end").nullable()
    val first_week_day = varchar("first_week_day", 1).nullable()
    val first_month_day = varchar("first_month_day", 2).nullable()
    val accounts = reference("accounts", AccountTable).nullable()
    val groups = reference("groups", GroupTable).nullable()
}

class StatisticsPresetEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<StatisticsPresetEntity>(StatisticsPresetTable)
    var name by StatisticsPresetTable.name
    var description by StatisticsPresetTable.description
    var duration_type by StatisticsPresetTable.duration_type
    var start by StatisticsPresetTable.start
    var end by StatisticsPresetTable.end
    var first_week_day by StatisticsPresetTable.first_week_day
    var first_month_day by StatisticsPresetTable.first_month_day
```

```
    var accounts by AccountEntity via StatisticsPresetAccounts
    var groups by GroupEntity via StatisticsPresetGroups
}

object StatisticsPresetAccounts : Table() {
    val statistics_preset = reference("statistics_preset", StatisticsPresetTable)
    val account = reference("account", AccountTable)
    override val primaryKey = PrimaryKey(statistics_preset, account)
}

object StatisticsPresetGroups : Table() {
    val statistics_preset = reference("statistics_preset", StatisticsPresetTable)
    val group = reference("group", GroupTable)
    override val primaryKey = PrimaryKey(statistics_preset, group)
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable
import org.jetbrains.exposed.sql.javatime.date
import org.jetbrains.exposed.sql.javatime.time

object TransactionTable : IntIdTable(
    name = "Transactions"
) {
    val name = varchar("name", 300)
    val description = varchar("description", 1000).nullable()
    var type = reference("type", TransactionTypeTable)
    var status = reference("status", TransactionStatusTable)
    var value = float("value")
    var currency = reference("currency", CurrencyTable)
    var date = date("date")
    var time = time("time")
    var category = reference("category", CategoryTable)
    var account = reference("account", AccountTable)
}

class TransactionEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<TransactionEntity>(TransactionTable)
    var name by TransactionTable.name
    var description by TransactionTable.description
    var type by TransactionTypeEntity referencedOn TransactionTable.type
    var status by TransactionStatusEntity referencedOn TransactionTable.status
    var value by TransactionTable.value
    var currency by CurrencyEntity referencedOn TransactionTable.currency
    var date by TransactionTable.date
    var time by TransactionTable.time
    var category by CategoryEntity referencedOn TransactionTable.category
    var account by AccountEntity referencedOn TransactionTable.account
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
```

```
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable

object TransactionStatusTable : IntIdTable(
    name = "TransactionStatuses"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
}

class TransactionStatusEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object :
IntEntityClass<TransactionStatusEntity>(TransactionStatusTable)
    var name by TransactionStatusTable.name
    var description by TransactionStatusTable.description
}

package com.bannanguy.moneta.model

import org.jetbrains.exposed.dao.IntEntity
import org.jetbrains.exposed.dao.IntEntityClass
import org.jetbrains.exposed.dao.id.EntityID
import org.jetbrains.exposed.dao.id.IntIdTable

object TransactionTypeTable : IntIdTable(
    name = "TransactionTypes"
) {
    val name = varchar("name", 200)
    val description = varchar("description", 300).nullable()
}

class TransactionTypeEntity(id: EntityID<Int>) : IntEntity(id) {
    companion object : IntEntityClass<TransactionTypeEntity>(TransactionTypeTable)
    var name by TransactionTypeTable.name
    var description by TransactionTypeTable.description
}

package com.bannanguy.moneta.common

import android.annotation.SuppressLint
import androidx.compose.animation.animateContentSize
import androidx.compose.animation.core.Ease
import androidx.compose.animation.core.tween
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.*
import androidx.compose.material.rememberScaffoldState
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import com.bannanguy.moneta.common.component.ColumnSpacer
import com.bannanguy.moneta.common.component.button.DrawerMenuButton
import com.bannanguy.moneta.common.component.loading.Loading
import
com.bannanguy.moneta.common.component.snackbar.composable.GetMonetaSnackbarHost
import com.bannanguy.moneta.module.crashHandler.unknownCrashHandler
```

```
import com.bannanguy.moneta.module.observer.AppObserver
import com.bannanguy.moneta.navigation.MNavigator
import com.bannanguy.moneta.navigation.NavigationCallbackManager
import com.bannanguy.moneta.navigation.composable.*
import com.bannanguy.moneta.util.WindowInfo
import com.bannanguy.moneta.util.getWindowInfo
import com.ramcosta.composedestinations.navigation.DestinationsNavigator

/**
 * The default screen frame that contain content and have 3 types of menus.
 * Also have title, snackbar place, loading animation element.
 */
@SuppressLint("UnusedMaterialScaffoldPaddingParameter")
@Composable
fun MonetaScreenWithNavigation(
    modifier: Modifier = Modifier,
    header: String,
    baseRoute: String,
    drawerIsOpen: Boolean = true,
    navigator: DestinationsNavigator,
    content: @Composable () -> Unit
) {
    unknownCrashHandler {
        MNavigator.update(navigator)
    }

    var isRecomposition by rememberSaveable { mutableStateOf(false) }
    if (!isRecomposition) {
        // Call method only on change screen,
        // not change a device orientation
        NavigationCallbackManager.processQueueInFrame()
    }

    if (!MNavigator.isMainScreen()) {
        AppObserver.resetErrorState()

        if (!AppObserver.canDbInteract) {
            AppObserver.runChainTest()
        }
    }

    val windowInfo = getWindowInfo()
    var rememberDrawerIsOpen by rememberSaveable {
        mutableStateOf(drawerIsOpen)
    }

    val scaffoldState = rememberScaffoldState()

    val isCompact = windowInfo.screenWidthInfo is WindowInfo.WindowType.Compact
    val isMedium = windowInfo.screenWidthInfo is WindowInfo.WindowType.Medium

    val minDrawerWidth = 180.dp
    val defaultMaxDrawerWidth = 280.dp
    val currentMaxDrawerWidth = if (rememberDrawerIsOpen) defaultMaxDrawerWidth else
0.dp
    val (maxDrawerWidth, setMaxDrawerWidth) = remember
{ mutableStateOf(currentMaxDrawerWidth) }
```

```
    @Composable
    fun TitleRow(
        content: @Composable () -> Unit
    ) {
        Row(
            modifier = Modifier
                .fillMaxWidth()
                .padding(top = 10.dp, bottom = 10.dp),
            horizontalArrangement = Arrangement.SpaceBetween,
            verticalAlignment = Alignment.Top
        ) {
            content()
        }
    }

    @Composable
    fun Title() {
        Text(
            text = header,
            style = MaterialTheme.typography.h5
        )
    }

    Scaffold(
        scaffoldState = scaffoldState,
        snackbarHost = {
            GetMonetaSnackbarHost(
                "Frame",
                scaffoldState.snackbarHostState
            )
        },
        bottomBar = {
           if (isCompact) {
               MonetaBottomNavigation(baseRoute)
           }
        },
        floatingActionButton = {
            MonetaFloatingActionButton()
       },
        floatingActionButtonPosition = if (isCompact) FabPosition.Center else
FabPosition.End,
        isFloatingActionButtonDocked = if (isCompact) true else false,
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colors.background)
            .then(modifier)
    ) { paddingValues ->

        if (isCompact) {
            Column(
                modifier = Modifier
                    .padding(
                        bottom = paddingValues.calculateBottomPadding(),
                        start = 30.dp,
                        end = 30.dp
                    )
            ) {
                TitleRow {
```

```
            Title()
        }
        content()
        ColumnSpacer(50.dp)
    }
} else if (isMedium) {
    Row(
        modifier = Modifier
            .background(MaterialTheme.colors.background)
            .then(modifier)
    ) {
        MonetaNavigationRail(baseRoute)
        Column(
            modifier = Modifier
                .padding(
                    start = 15.dp,
                    end = 30.dp
                )
        ) {
            TitleRow {
                Title()
            }
            content()
            ColumnSpacer(50.dp)
        }
    }
} else {
    Row {
        Column(
            modifier = Modifier
                .animateContentSize(
                    animationSpec = tween(600, 0, Ease)
                )
        ) {
            Box(
                modifier = Modifier
                    .widthIn(min = minDrawerWidth, max = maxDrawerWidth)
            ) {
                MonetaNavigationDrawerContent(
                    baseRoute,
                    minDrawerWidth,
                    defaultMaxDrawerWidth
                )
            }
        }
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(start = 30.dp, end = 30.dp)
        ) {
            TitleRow {
                Title()
                DrawerMenuButton(
                    clickProcess = {
                        rememberDrawerIsOpen = if (rememberDrawerIsOpen) {
                            setMaxDrawerWidth(0.dp)
                            false
                        } else {
```

```
                                    setMaxDrawerWidth(defaultMaxDrawerWidth)
                                    true
                                }
                            }
                        )
                    }
                    content()
                    ColumnSpacer(50.dp)
                }
            }
        }

        Loading.Get()
    }

    if (!isRecomposition) {
        isRecomposition = true
    }
}

package com.bannanguy.moneta.common.component.button

import androidx.compose.animation.animateColorAsState
import androidx.compose.animation.core.*
import androidx.compose.foundation.background
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.*
import androidx.compose.material.*
import androidx.compose.runtime.*
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.style.TextAlign
import androidx.compose.ui.unit.Dp
import androidx.compose.ui.unit.dp
import com.bannanguy.moneta.R
import com.bannanguy.moneta.module.crashHandler.unknownCrashHandler
import com.bannanguy.moneta.navigation.MNavigator
import com.bannanguy.moneta.ui.theme.DisabledColor
import com.bannanguy.moneta.module.context.MContext
import com.bannanguy.moneta.util.WindowInfo
import com.bannanguy.moneta.util.getWindowInfo

@Composable
private fun BaseButton(
    nameId: Int,
    iconId: Int? = null,
    clickProcess: () -> Unit,
    onceClick: Boolean? = false,
    surfaceModifier: Modifier = Modifier,
    formErrorState: State<Boolean>? = null,
    iconSize: Dp = 20.dp,
    centerAlignment: Boolean? = false
) {
    var isEnable = rememberUpdatedState(newValue = if (formErrorState == null) true
else !formErrorState.value )

    var isButtonClicked = remember { mutableStateOf(false) }
```

```
    val animatedDisOrNoColor by animateColorAsState(
        if (isEnable.value) MaterialTheme.colors.onPrimary
        else DisabledColor,
        animationSpec = tween(
            durationMillis = 150,
            easing = EaseIn
        ),
    )

    val animatedBgColor by animateColorAsState(
        if (isButtonClicked.value && isEnable.value) MaterialTheme.colors.secondary
        else MaterialTheme.colors.primary,
        animationSpec = tween(
            durationMillis = 250,
            easing = EaseOut
        ),
        finishedListener = {
            if (isEnable.value) isButtonClicked.value = false
        }
    )

    Box(
        modifier = Modifier
            .clickable(
                onClick = {
                    unknownCrashHandler {
                        if (onceClick == true && isButtonClicked.value) {
                            return@unknownCrashHandler
                        }
                        isButtonClicked.value = true
                        clickProcess()
                    }
                },
                enabled = isEnable.value
            )
            .background(
                animatedBgColor,
                MaterialTheme.shapes.small
            )
            .then(surfaceModifier)
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.Center,
            modifier = Modifier
                .padding(horizontal = 40.dp, vertical = 7.dp)
                .then(if (centerAlignment == true) { Modifier.fillMaxWidth() } else
Modifier)
        ) {
            if (iconId != null) {
                Icon(
                    painterResource(iconId),
                    null,
                    Modifier.size(iconSize),
                    animatedDisOrNoColor,
                )
                Spacer(modifier = Modifier.width(10.dp))
```

```
            }
            Text(
                MContext.getString(nameId),
                style = MaterialTheme.typography.button,
                color = animatedDisOrNoColor,
                textAlign = if (centerAlignment == true) TextAlign.Center else
TextAlign.Start
            )
        }
    }
}

@Composable
fun NavigationBackButton(
    clickProcess: () -> Unit = {
        MNavigator.back()
    }
) {
    val windowInfo = getWindowInfo()

    BaseButton(
        nameId = R.string.button_back,
        iconId = R.drawable.ic_button_back_arrow,
        clickProcess = {
            unknownCrashHandler {
                clickProcess()
            }
        },
        onceClick = true,
        surfaceModifier = when (windowInfo.screenWidthInfo) {
            is WindowInfo.WindowType.Compact -> Modifier
                .widthIn(min=windowInfo.screenWidth/2)
            else -> Modifier.requiredWidthIn(min = 200.dp)
        }
    )
}

@Composable
fun DrawerMenuButton(
    clickProcess: () -> Unit
) {
    IconButton(
        onClick = {
            unknownCrashHandler {
                clickProcess()
            }
        }
    ) {
        Icon(
            painterResource(R.drawable.ic_drawer_menu_button),
            MContext.getString(R.string.menu_drawer_contentDescription_button),
            Modifier.size(40.dp),
            MaterialTheme.colors.onBackground
        )
    }
}

@Composable
```

```
fun CustomButton(
    nameId: Int,
    iconId: Int? = null,
    clickProcess: () -> Unit,
    onceClick: Boolean? = true,
    surfaceModifier: Modifier? = null,
    formErrorState: State<Boolean>? = null,
    centerAlignment: Boolean? = false
) {
    val windowInfo = getWindowInfo()

    BaseButton(
        nameId = nameId,
        iconId = iconId,
        clickProcess = clickProcess,
        onceClick = onceClick,
        surfaceModifier = surfaceModifier?: when (windowInfo.screenWidthInfo) {
            is WindowInfo.WindowType.Compact -> Modifier
                .fillMaxWidth()
            else -> Modifier.requiredWidthIn(min = 200.dp)
        },
        formErrorState = formErrorState,
        centerAlignment = centerAlignment
    )
}
```

```
package com.bannanguy.moneta.screen.budget

import androidx.compose.runtime.Composable
import com.bannanguy.moneta.R
import com.bannanguy.moneta.common.MonetaScreenWithNavigation
import com.bannanguy.moneta.screen.budget.content.BudgetContent
import com.bannanguy.moneta.screen.destinations.BudgetScreenDestination
import com.bannanguy.moneta.ui.theme.MonetaTheme
import com.bannanguy.moneta.module.context.MContext
import com.ramcosta.composedestinations.annotation.Destination
import com.ramcosta.composedestinations.navigation.DestinationsNavigator

@Destination
@Composable
fun BudgetScreen(
    navigator: DestinationsNavigator
) {
    MonetaTheme {
        MonetaScreenWithNavigation(
            header = MContext.getString(R.string.screen_budget_header),
            baseRoute = BudgetScreenDestination.route,
            navigator = navigator
        ) {
            BudgetContent()
        }
    }
}
```

```
package com.bannanguy.moneta.screen.budget.content

import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.material.MaterialTheme
```

```
import androidx.compose.material.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.unit.dp
import com.bannanguy.moneta.R
import com.bannanguy.moneta.common.component.EndSpacer
import com.bannanguy.moneta.common.component.SpaceBeforeContent
import com.bannanguy.moneta.module.crashHandler.unknownCrashHandler
import com.bannanguy.moneta.ui.theme.SuccessColor
import com.bannanguy.moneta.module.context.MContext

@Composable
fun BudgetContent () {
    LazyColumn {
        unknownCrashHandler {
            item {
                SpaceBeforeContent()
                Text(
                    text =
MContext.getString(R.string.message_sorry_feature_under_development_message),
                    style = MaterialTheme.typography.h6,
                    color = SuccessColor
                )

                EndSpacer(50.dp)
            }
        }
    }
}


package com.bannanguy.moneta.module.logger

import com.bannanguy.moneta.BuildConfig
import com.bannanguy.moneta.module.debug.printMyStack
import timber.log.Timber
import java.time.LocalDateTime
import java.time.format.DateTimeFormatter

/*
    v A verbose message. It is typically used for debugging and provides the lowest
level of log output.

    d A debug message. It is commonly used to provide debug information during
development and troubleshooting.

    i An informational message. It is generally used to provide important information
about the application's state or progress.

    w A warning message. It indicates a potential issue or condition that might cause
problems in the future but is not an error.

    e An error message with an associated throwable. It is used to report errors or
exceptions that occur during the execution of the application.

    wtf A catastrophic failure message. It stands for "What a Terrible Failure" and
is typically used to log critical errors that lead to application crashes or severe
issues.
* */
```

```
/*
* In DEBUG MODE: Logcat output
* In RELEASE MODE: v & d -> not sending; i & w & e & wtf -> save at log files
* */

/**
 * Custom wrapper for the Timber.
 */
class MLog {
    companion object {
        private fun getSuffix(): String {
            val currentDateTime = LocalDateTime.now()
            val formatter = DateTimeFormatter.ofPattern("HH:mm:ss dd-MM-yyyy")
            return " [" + currentDateTime.format(formatter) + "]"
        }

        fun d(tag: String, msg: String?, vararg args: Any?) {
            if (msg == null || msg.trim() == "") return

            Timber.tag(tag).d(msg + getSuffix(), *args)
        }

        fun v(tag: String, msg: String?, vararg args: Any?) {
            Timber.tag(tag).v(msg, *args)
        }

        fun i(tag: String, msg: String?, vararg args: Any?) {
            Timber.tag(tag).i(msg, *args)
        }

        fun w(tag: String, t: Throwable?, msg: String?, vararg args: Any?) {
            if (BuildConfig.DEBUG) {
                printMyStack()
                Timber.tag(tag).w(t, msg, *args)
            }
            else Timber.tag(tag).w(t, msg, *args)
        }

        fun e(tag: String, t: Throwable?, msg: String?, vararg args: Any?) {
            if (BuildConfig.DEBUG) {
                printMyStack()
                Timber.tag(tag).e(t, msg, *args)
            }
            else Timber.tag(tag).e(t, msg, *args)
        }

        fun wtf(tag: String, t: Throwable?, msg: String?, vararg args: Any?) {
            if (BuildConfig.DEBUG) {
                printMyStack()
                Timber.tag(tag).wtf(t, msg, *args)
            }
            else Timber.tag(tag).wtf(t, msg, *args)
        }
    }
}
```

Додаток Г
(додатковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

1

Мобільний застосунок для ведення особистих витрат і доходів

Шифр КвРІПЗ.200158.01.03.ПЗ

- Автор: Василик Віктор Михайлович, студент групи ІПЗ-20-1
- Керівник: Бедратюк Ганна Іванівна, старший викладач

Хмельницький національний університет, 2024

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

2

Мета роботи

Розробити мобільного застосунок під Android ОС для ведення особистих витрат та доходів, яке допомагає користувачам економити свій час і бути спокійним за свої гроші, за допомогою представлення даних з різних джерел в єдиному програмному інтерфейсі, а також системи персоналізації робочого простору під індивідуальні потреби кожного клієнта.

3

Актуальність теми

Цільова аудиторія програмного продукту має декілька фінансових потоків із різних джерел, дані яких є децентралізованими, український ринок програмного забезпечення не може покрити запити клієнтів на розв'язання проблем предметної області, тому було прийнято рішення про необхідність розробки нового програмного забезпечення.

4

Задачі проєктування

- провести дослідження предметної області, проблем користувачів, ринку та основні тенденції їх розвитку;
- визначити особливості та специфіку процесу обліку доходів та витрат;
- розробити архітектуру та функціональну структуру, алгоритм роботи та програмну логіку мобільного застосунку;
- здійснити реалізацію та тестування мобільного застосунку.

Порівняльна таблиця аналогів програмного забезпечення

5

Функції / Назва ПП	Saldo: Облік доходів та витрат	Wallet – Гроші, бюджет	Monefy – Розпорядник бюджету	Money: Бюджет, Витрати, Фінанси	1Money: облік витрат, бюджет	Money Manager Expense & Budget
Облік доходів та витрат	+	+	+	+	+	+
Працює з українськими банками	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Працює з закордонними банками	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Бізнес функції	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Розподіл на простори	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Спільний облік	+	+	Відсутня	Відсутня	Відсутня	Відсутня
Планування бюджету	+	+	Відсутня	+	+	+
Шаблони	+	+	+	+	+	+
Фінансові цілі	Відсутня	+	Відсутня	+	+	Відсутня
Нагадування про платежі	Відсутня	+	Відсутня	Відсутня	Відсутня	Відсутня
Сповіщення	Відсутня	+	Відсутня	+	+	+
Синхронізація	+	+	+	+	+	+
Функції безпеки	+	+	Відсутня	+	+	+
Швидке введення	+	+	+	+	+	+
Інструкція	+	+	+	+	+	Відсутня

6

- На основі даних порівняльної таблиці можна зробити висновки, що перші версії програмного продукту не обов'язково повинні включати функції по роботі з банківськими рахунками, спільного обліку, нагадувань про платежі чи бізнес інструменти.

- Важливими є функції персоналізації та можливості створювати свої шаблони рахунків, категорій витрат, груп, валют. Всі програмні продукти мали функцію швидкого введення суми операції за допомогою штатних методів програми.

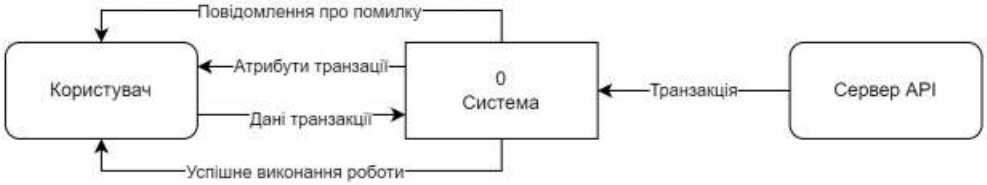
7

Функціональні можливості програмного продукту

- Форма створення транзакцій
- Аналітична таблиця
- Діаграми статистики
- Покрокова деталізація даних
- Конструктор об'єктів
- Робочі простори
- Матричний вибір
- Швидке введення

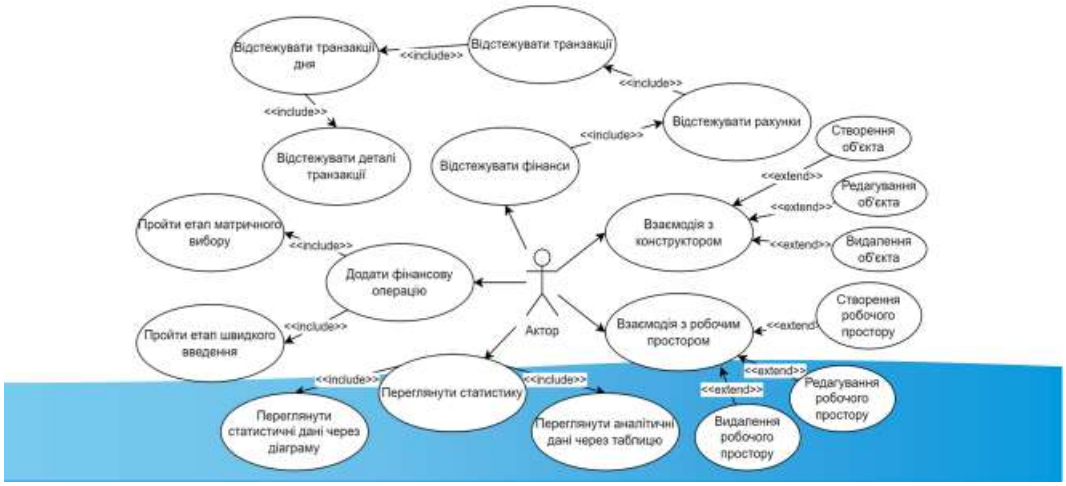
8

Контекстна діаграма потоків даних



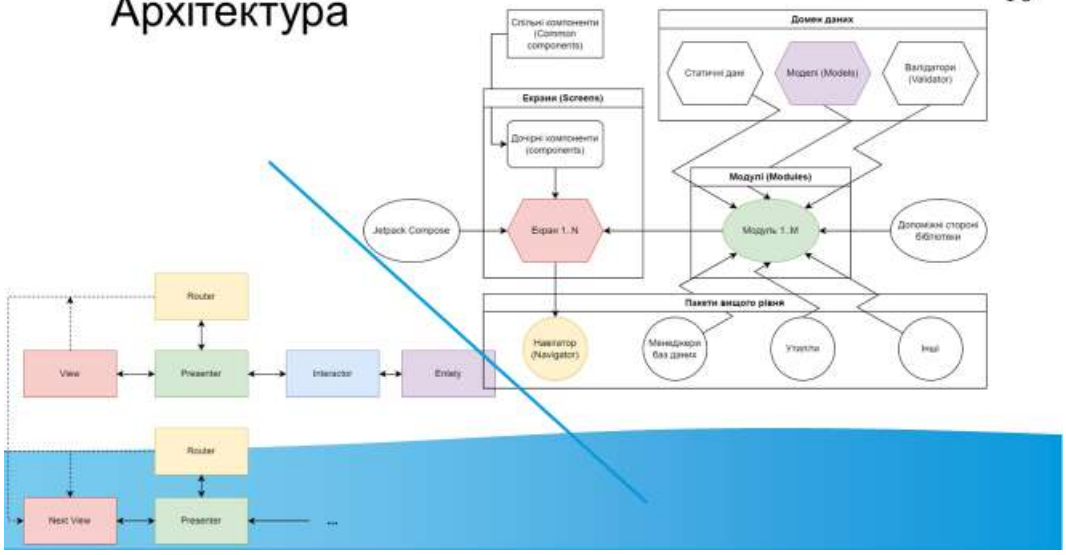
9

Діаграма варіантів використання



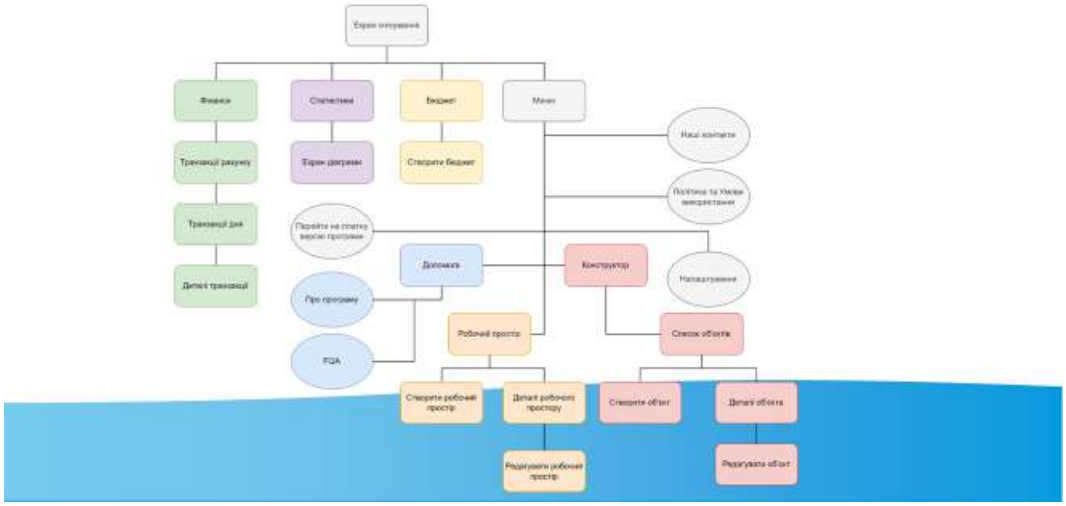
10

Архітектура



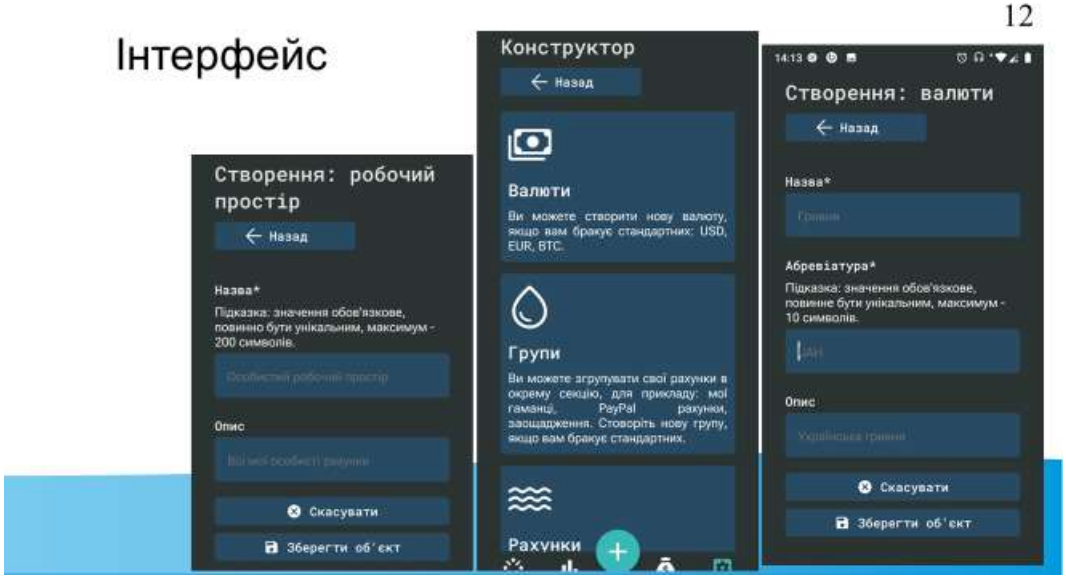
11

Блок-схема навігації



12

Інтерфейс



13

14

Результати тестування

- було проведене функціональне тестування за допомогою тестових сценаріїв — які були виконані успішно;
- проводилося тестування із використання Android емуляторів для різних версій операційних систем та розмірностей екранів — була виконана робота по адаптації інтерфейсу;
- проводилося тестування зручності (англ. Usability Testing) із залучення семи користувачів — їхні побажання були взяті до уваги.

15

Практична цінність результатів КвР

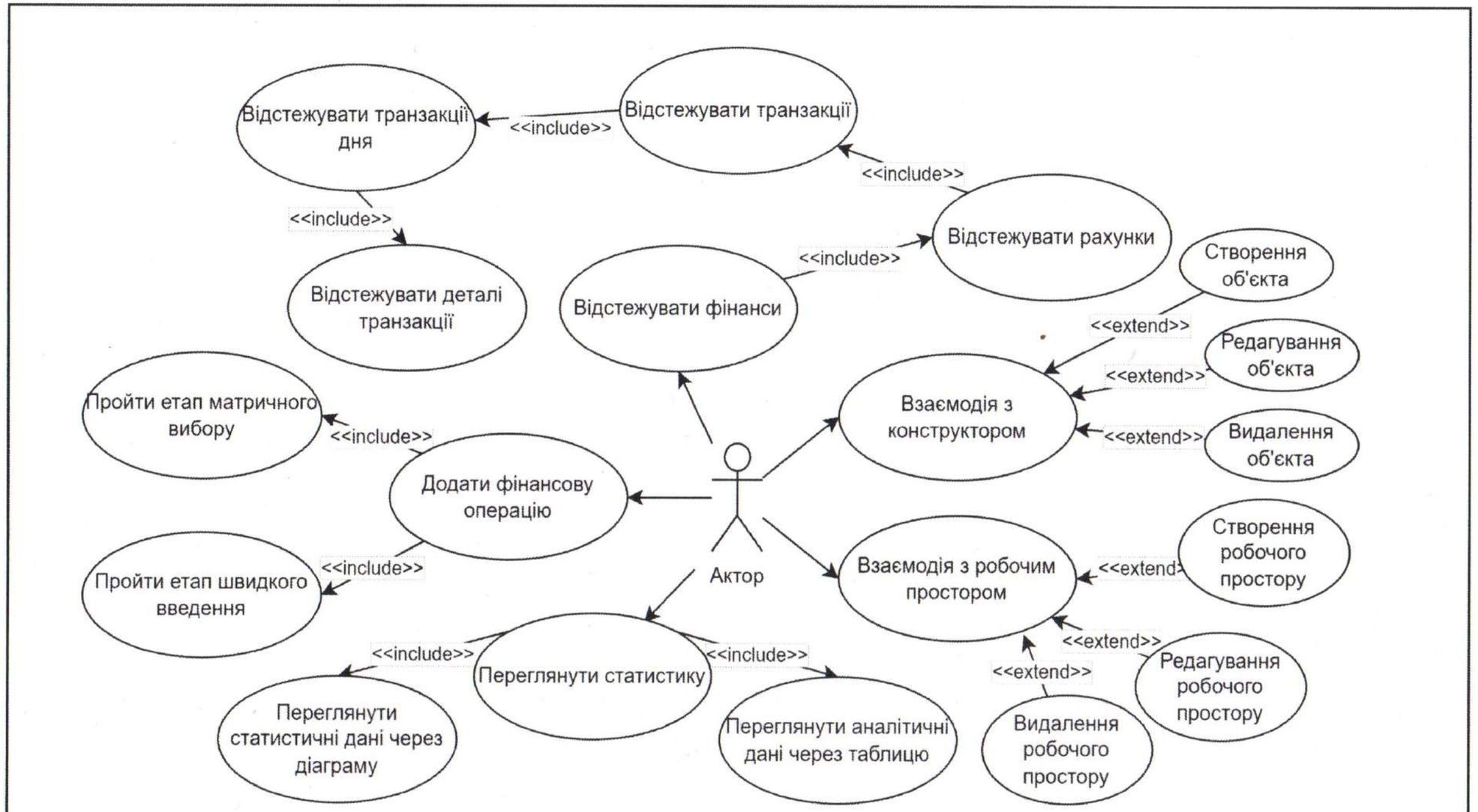
- створено програмне забезпечення для українського ринку мобільних застосунків;
- програмний продукт отримав значні можливості персоналізації, яких немає у більшості аналогів;
- на основі поточної версії мобільного застосунку буде розроблена наступна версія з можливістю обліку банківських онлайн рахунків;
- користувачі, які ведуть облік доходів та витрат мають менше фінансового стресу в житті.

16

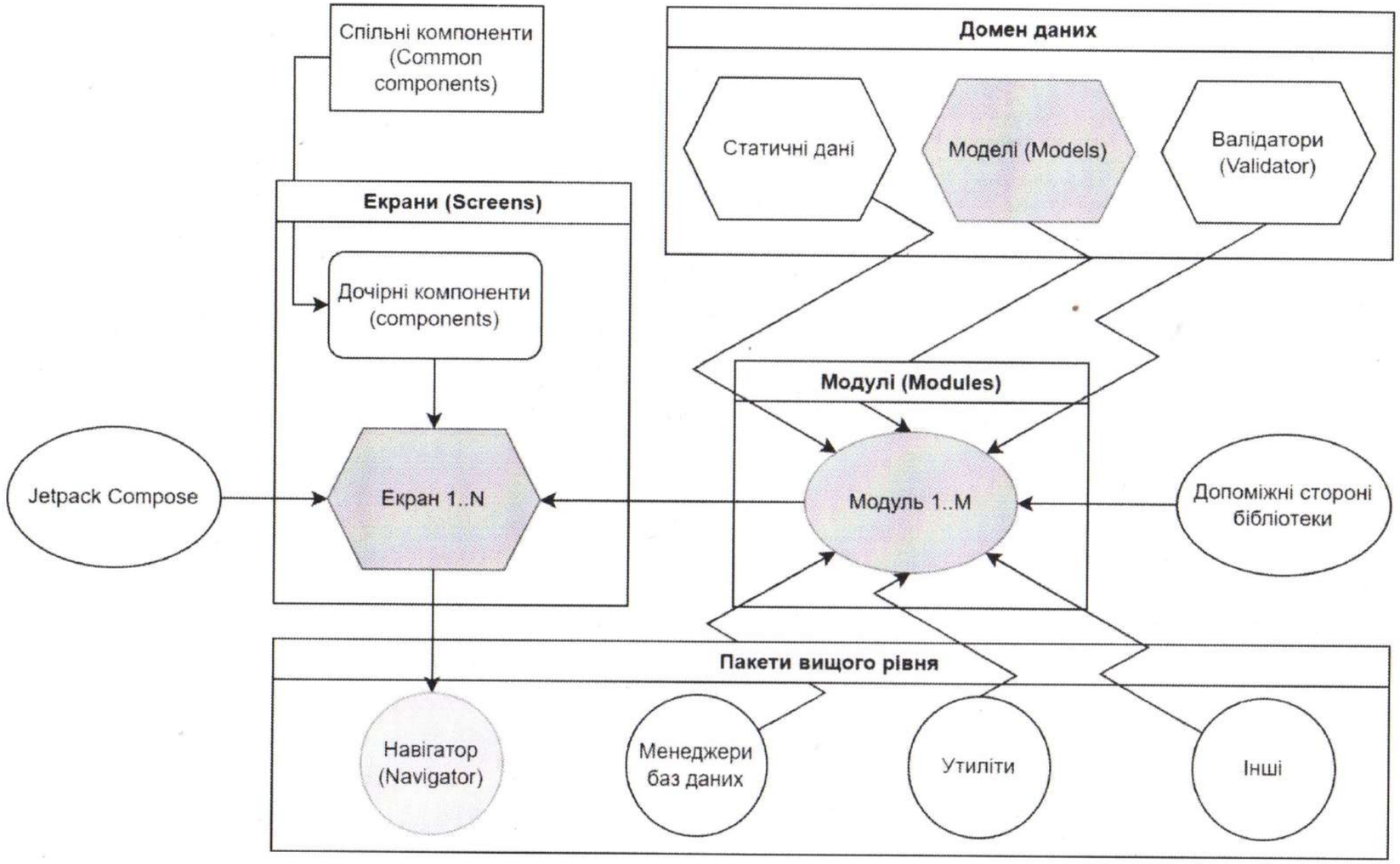
Висновки

- В результаті виконання кваліфікаційної роботи було розроблено мобільний застосунок під операційну систему Android для ведення особистих витрат і доходів, який допомагає зберігати інформацію про проведені грошові операції, керувати власними фінансами, переглядати історію транзакцій в зручній формі.
- Виконано всі поставлені задачі проєктування, дотримано вимог якості, що були визначені в технічному завдані.

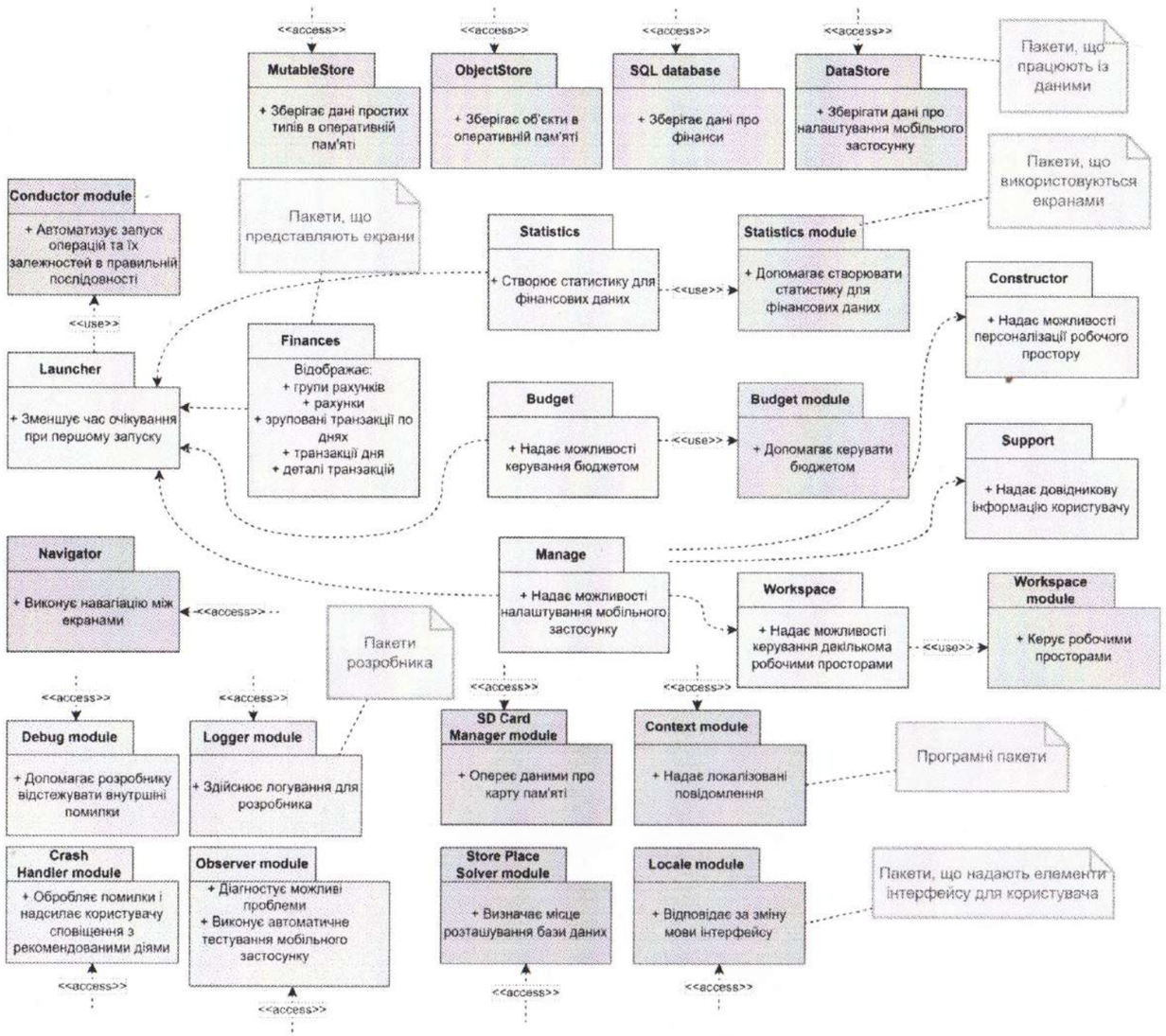
ГРАФІЧНА ЧАСТИНА



					КвРІПЗ.200158.01.03.Е1			
						Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Василик В. М.		30.05.24				
Керівник		Бедратюк Г. І.		30.05.24		Аркуш 1	Аркушів 1	
Н. Контр.		Праворська Н. І.		30.05.24	UML-діаграма варіантів використання	ХНУ. ІПЗ-20-1		
Зав. каф.		Бедратюк Л. П.		30.05.24				



					КвРІПЗ.200158.01.03.Е2	
Зм.	Арк.	№ докум.	Підпис	Дата		Літера / Маса / Масштаб
Розробив		Василик В. М.		30.05.24		
Керівник		Бедратюк Г. І.		30.05.24		Аркуш 1 / Аркушів 1
Н. Контр.		Праворська Н. І.		30.05.24	Гібридна VIPER архітектура мобільного застосунку	ХНУ. ІПЗ-20-1
Зав. каф.		Бедратюк Л. П.		30.05.24		



					КвРІПЗ.200158.01.03.Е3		
Зм.	Арк.	№ докум.	Підпис	Дата		Літера	Маса / Масштаб
Розробив		Василик В. М.					
Керівник		Бедратюк Г. І.				Аркуш 1	Аркушів 1
Н. Контр.		Праворська Н.			UML-діаграма пакетів, що містить варіанти використання	ХНУ: ІПЗ-20-1	
Зав. каф.		Бедратюк Л. П.					

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Василика В. М.

Прізвище, ініціали

факультет ІТ, 4 курс, група ІПЗ-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті», згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

02.01.2024

дата

підпис

UNICHECK
by Turnitin

Ім'я користувача:
ІПЗ

ID перевірки:
1016286729

Дата перевірки:
27.05.2024 12:18:50 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
30.05.2024 19:13:23 EEST

ID користувача:
100012953

Назва документа: **Дипломне проєктування ХНУ ІПЗ-20-1 Василик. В. М**

Кількість сторінок: **82** Кількість слів: **15123** Кількість символів: **122453** Розмір файлу: **12.10 MB** ID файлу: **1016080658**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

7.56%
Схожість

Найбільша схожість: **1.42%** з джерелом з Бібліотеки (ID файлу: **1015017362**)

6.97% Джерела з Інтернету	687	Сторінка 84
2.63% Джерела з Бібліотеки	213	Сторінка 89

0.07% Цитат

Цитати	2	Сторінка 90

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1
Підозріле форматування	20 сторінок

Mon May 27 09:40:05 EEST 2024, Форкун Юрій Вікторович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

ID: 127274 Назва: Мобільний застосунок для ведення особистих витрат і доходів Додано в БД: 2024-05-27 Автора: Віктор ВАСИЛЮК Керівники: старший викладач Ганна БЕДРАТЮК Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	98777	1400	2662 (3%)	36 (3%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Василик Віктор Михайлович

Тема Мобільний застосунок для ведення особистих витрат і доходів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 83

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі здійснено дослідження предметної області, завдяки чому виявлено галузеві проблеми, проаналізовано ринок наявного програмного забезпечення, визначено функціональні та нефункціональні вимоги до нового програмного продукту із розширеною системою персоналізації. Створено концепцію дизайну, описано логіку мобільного застосунку, здійснено вибір технологій та методів реалізації, наголошено про важливість використання гнучкої методології розробки. Використано гібридну архітектуру, реалізовано базу даних та описано процес розробки. В результаті чого створено мобільний застосунок. Проведено три типи тестування, які дозволили виявити дефекти та виправити їх на різних стадіях розробки, завдяки чому в розробленому програмному забезпеченні коректно працюють спроєктовані модулі.

2. Висновок про відповідність роботи поставленому завданню В роботі виконані всі поставлені задачі, вимоги технічного завдання не порушені.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки й техніки та передових методів роботи У вступі висвітлено стан предметної області та її проблеми, таким чином визначено актуальність теми та завдання кваліфікаційної роботи. В першому розділі проведено аналіз предметної області та наявних технічних рішень. В другому розділі проаналізовані сучасні архітектури, розглянуто їх особливості та визначено, що мобільний застосунок буде побудований на гібридній VIPER архітектурі, такий вибір обґрунтовано. В третьому розділі виконано практичну розробку бази даних, програмних модулів, екранів та компонентів інтерфейсу користувача. Також в цьому розділі виконано функціональне тестування чорного ящика згідно з розробленими сценаріями та використанням емуляторів, здійснено перевірку зручності, в результаті було здійснено налагодження та підтверджено коректну роботу мобільного застосунку.

4. Позитивні сторони роботи Згідно з останніми дослідженнями, що також наведені в роботі, індекс фінансової грамотності українців нижчий за середній, розроблене програмне забезпечення має позитивний вплив на розвиток фінансових компетентностей, тому тема кваліфікаційної роботи є актуальною. Важливо, що в процесі розробки інтерфейсу користувача було взято до уваги потреби людей з синдромом дефіциту уваги та гіперактивності. Також було засновано новітні архітектурні рішення.

5. Негативні сторони роботи Проєктування та реалізація мобільного застосунку має великий обсяг виконання, було б краще конкретизувати тему кваліфікаційної роботи на системі персоналізації, оскільки на ній зроблений акцент уваги. Попри наведені переваги та недоліки проаналізованих архітектурних рішень, мало сказано про причини відмови розглянутих. Також можна було знайти кращі відповідники в українській мові для перекладу англомовних складових архітектурних рішень.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм. Пояснювальна записка оформлена згідно з вимогами чинних стандартів та методичних рекомендацій Хмельницького національного університету. Бібліографічні посилання відповідають стандарту ДСТУ 8302:2015.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки послідовно та чітко описує виклад дій в кожному зі структурних розділів. Візуальні матеріали ефективно доповнюють текст, надаючи можливість детально ознайомитись з особливостями проєктованої системи. Розроблене програмне забезпечення слугує меті, покращення фінансової грамотності. Важливо похвалити те, що застосунок прагне доступності для людей з обмеженнями.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «відмінно».

РЕЦЕНЗЕНТ (прізвище, ім'я, по батькові, посада, місце роботи) Говорущенко Тетяна Олександрівна, доктор технічних наук, професор, зав. кафедри комп'ютерної інженерії та інформаційних систем (КІІС) ХНУ

"6" 06 2024 р.

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів:

Назва кваліфікаційної роботи: «Мобільний застосунок для ведення особистих витрат і доходів»
Автор: Василик Віктор Михайлович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Бедратюк Ганна Іванівна, старший викладач
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є академічним плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені у роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат Unicheck виявлено схожість з деякими документами у частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів, у рамках основних написів, у назвах публікацій переліку джерел посилання;

2) в якості запозичень системою Unicheck було зафіксовано деякі послідовності вихідного коду і посилання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) запозичення, виявлені в тексті роботи, є фрагментарними.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 1.0%. Обсяг запозичень, визначений системою Unicheck виявлення збігів ідентичності/схожості, складає 7.56% і адресується до 687 джерел з Інтернету і 213 джерела з бібліотеки, що, з урахуванням наведених обґрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 06.06.2024 р.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Ганна БЕДРАТЮК