Міністерство освіти і науки України

Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ANDROID-ДОДАТКУ «MONEY FLOW» З ВИКОРИСТАННЯМ CAPACITOR.JS, REACT.JS ТА TYPESCRIPT

DEVELOPMENT OF ANDROID APPLICATION «MONEY FLOW» USING CAPACITOR.JS, REACT.JS AND TYPESCRIPT

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗс-21
Антонюк Андрій Олексійович
_____
(підпис)

Керівник:
к.т.н., доцент
Гулієва Наталія Михайлівна
_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«8» червня 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна
_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРИСТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Н. Лимина
«2» 02 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Антонюку Андрію Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: розробка Android-додатку «Money Flow» з використанням *Capacitor.js, React.js та TypeScript*
Керівник роботи: *к.т.н., доцент, Гулієва Наталія Михайлівна*
затверджені наказом закладу вищої освіти від «*30» грудня 2023 р.* № *477/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*

3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення, опис функціонального наповнення об'єкта проектування, практична реалізація об'єкта проектування.*

5. Перелік графічного матеріалу: *14 рисунків, 2 таблиці.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Гулієва Н. М.*		
Специфікації вимог до розробленої системи	*Гулієва Н. М.*		
Розробка об'єкта проєктування	*Гулієва Н. М.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту		58%	
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«2» лютого 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	Вик.
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	Вик.
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	Вик.
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	Вик.
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	Вик.
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	Вик.
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	Вик.

Здобувач вищої освіти _____ *Антонюк А. О.*
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ *Гулієва Н. М.*
(підпис) (прізвище, ініціали)

Міністерство освіти і науки України
Луцький національний технічний університет

Рецензія
на кваліфікаційну роботу

Здобувач вищої освіти: Антонюк Андрій Олексійович

Тема: «Розробка Android-додатку "Money Flow" з використанням Capacitor.js, React.js та TypeScript»

Коротка характеристика кваліфікаційної роботи: Кваліфікаційна робота бакалавра присвячена розробці мобільного додатку для обліку витрат та доходів. Додаток дозволяє користувачам вести облік фінансових операцій, включаючи витрати, доходи та перекази між рахунками, підтримуючи необмежену кількість валют та рахунків. У роботі використано сучасні технології, такі як React.js, Capacitor.js та TypeScript.

Самостійні розробки і пропозиції автора: У роботі детально описаний процес розробки мобільного додатку з обліку фінансів. Розроблено інтерфейс користувача, який забезпечує інтуїтивно зрозуміле керування фінансовими операціями, а також реалізовано функціонал для детального аналізу та статистики по транзакціям.

Практичне значення роботи: Мобільний додаток був успішно реалізований і може бути використаний широкою аудиторією для обліку особистих фінансів. Додаток протестовано в умовах, наближених до реальних, і він показав високу стабільність та зручність у користуванні. Кваліфікаційна робота має значне практичне значення.

Недоліки: В роботі варто було б провести більш глибокий аналіз конкурентних рішень на ринку.

Загальний висновок: Робота виконана на високому рівні, має логічну та послідовну структуру, відповідає встановленим вимогам, може бути представлена до захисту на державній екзаменаційній комісії та заслуговує відмінної оцінки.

Рецензент: Багнюк Наталія Володимирівна

Рецензент кваліфікаційної роботи _____
(підпис)

«07» червня 2024 р.

Міністерство освіти і науки України
Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

ВІДГУК
КЕРІВНИКА НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
Здобувач вищої освіти *Антонюк Андрій Олексійович*
група *ІПЗс-21*

Тема кваліфікаційної роботи: *Розробка Android-додатку «Money Flow» з використанням Capacitor.js, React.js та TypeScript*
Керівник *Гулієва Наталія Михайлівна*

Актуальність теми: тема дослідження та розробка мобільного додатку для обліку витрат та доходів є актуальним завданням у сучасних умовах, коли фінансова грамотність та управління особистими фінансами набувають все більшого значення. Незважаючи на існування численних фінансових додатків, ще залишається потреба у вирішенні питань зручності користування, підтримки багатьох валют та рахунків, а також можливості детального аналізу фінансових операцій..

Об'єкт дослідження – це мобільний додаток для обліку витрат та доходів «Money Flow», розроблений з використанням технологій Capacitor.js, React.js та TypeScript.

Характеристика теоретичного рівня, наявності самостійних розробок і практичної значимості роботи: студентом була створена ефективна система для обліку фінансових операцій, яка дозволяє користувачам зручно і швидко вести облік витрат та доходів. В додатку реалізовано можливість створення та ведення необмеженої кількості валют і рахунків, що робить його універсальним для різних користувачів. Було розроблено функції для детального аналізу транзакцій, включаючи пошук, фільтрацію та статистику. Додаток має сучасний і інтуїтивно зрозумілий інтерфейс, створений з використанням React.js та Capacitor.js, що забезпечує високу продуктивність. Усі розробки були протестовані в умовах, наближених до реальних, і показали високий рівень стабільності та зручності використання.

Зауваження та недоліки: істотних недоліків не виявлено.

Загальний висновок робота виконана на високому рівні та заслуговує оцінки «відмінно» за умови успішного захисту.

Керівник _____ (Гулієва Н.М.)
(підпис) (прізвище, ім'я, по батькові)

«07» червня 2024 р.

Міністерство освіти і науки України

Луцький національний технічний університет

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ANDROID-ДОДАТКУ «MONEY FLOW» З ВИКОРИСТАННЯМ CAPACITOR.JS, REACT.JS ТА TYPESCRIPT

DEVELOPMENT OF ANDROID APPLICATION «MONEY FLOW» USING CAPACITOR.JS, REACT.JS AND TYPESCRIPT

спеціальність 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

освітня програма «Інженерія програмного забезпечення»
(назва освітньої програми)

Виконав: здобувач вищої освіти
Групи ІПЗс-21
Антонюк Андрій Олексійович

_____
(підпис)

Керівник:
к.т.н., доцент
Гулієва Наталія Михайлівна

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2024 р.
к.т.н., доцент
Гарант освітньої програми:
Ліщина Наталія Миколаївна

_____
(підпис)

Луцьк – 2024 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРИСТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *121 Інженерія програмного забезпечення*
Освітня програма: *«Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Антонюку Андрію Олексійовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи: розробка Android-додатку «Money Flow» з використанням *Capacitor.js, React.js та TypeScript*
Керівник роботи: *к.т.н., доцент, Гулієва Наталія Михайлівна*
затверджені наказом закладу вищої освіти від «*30*» *грудня 2023 р.* № *477/01-02*

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«5» червня 2024 р.*

3. Вихідні дані до роботи: *технічне та програмне забезпечення ЕОМ, вимоги до розробки програмного забезпечення, ергономічні вимоги до функціонування програмного засобу.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): *аналіз сучасного стану проблеми, існуючих методів і засобів її розв'язання, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення, опис функціонального наповнення об'єкта проектування, практична реалізація об'єкта проектування.*

5. Перелік графічного матеріалу: *14 рисунків, 2 таблиці.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Гулієва Н. М.*		
Специфікації вимог до розробленої системи	*Гулієва Н. М.*		
Розробка об'єкта проєктування	*Гулієва Н. М.*		
Нормоконтроль	*Повстяна Ю. С.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	___%		
Академічна доброчесність	*Ящук А. А.*		

7. Дата видачі завдання *«2» лютого 2024 р.*

КАЛЕНДАРНИЙ ПЛАН

№ З/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Огляд літературних джерел по темі кваліфікаційної роботи бакалавра*	*07.02.2024 р.*	
2	*Аналіз проблеми розробки та впровадження об'єкту проектування*	*27.02.2024 р.*	
3	*Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання*	*12.03.2024 р.*	
4	*Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних*	*17.04.2024 р.*	
5	*Практична реалізація об'єкта проектування та розробка бази даних*	*18.05.2024 р.*	
6	*Тестування та налагодження об'єкта проектування*	*01.06.2024 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*05.06.2024 р.*	

Здобувач вищої освіти _____ *Антонюк А. О.*
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ *Гулієва Н. М.*
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Антонюк А. О. Розробка Android-додатку «Money Flow» з використанням Capacitor.js, React.js та TypeScript. Рукопис.

Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2024.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел.

У першому розділі здійснено аналіз сучасного стану проблеми розробки мобільних додатків і сформульовані завдання. В другому розділі визначено вимоги до розроблюваної системи, а також засоби, методи і алгоритми розробки. У третьому розділі розроблено Android-додаток «Money Flow» з використанням Capacitor.js, React.js та TypeScript та здійснено тестування додатку. У висновках узагальнено інформацію, відображену у попередніх частинах. Результати розробки демонструють можливості розробки мобільних додатків за допомогою веб технологій.

Ключові слова: Android, Capacitor.js, React.js, TypeScript.

ABSTRACT

Antoniuk A. O. Development of Android Application "Money Flow" Using capacitor.js, react.js and Typescript. Manuscript.

Bachelor's qualifying thesis of the educational program "Software Engineering". Lutsk National Technical University. Lutsk, 2024.

The bachelor's qualifying thesis consists of an introduction, three sections, conclusions, a list of used sources.

In the first chapter, an analysis of the current state of the problem of developing mobile applications was carried out and the task was formulated. The second section defines the requirements for the developed system, as well as means, methods and development algorithms. In the third chapter, "Money Flow" Android-application was developed using Capacitor.js, React.js and TypeScript. The conclusions summarize the information presented in the previous parts. The development results demonstrate the possibilities of developing mobile applications utilizing web technologies.

Keywords: Android, Capacitor.js, React.js, TypeScript.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі контроль над фінансовими потоками є одним із ключових аспектів стабільного та успішного життя. Люди прагнуть не лише заробляти гроші, але й ефективно їх розподіляти та зберігати. Однак, у швидкому ритмі життя відстеження витрат та доходів часто стає складним завданням. Саме тут мобільні додатки можуть стати у пригоді, надаючи зручний і ефективний спосіб управління фінансовими ресурсами.

Розробка мобільного додатка для обліку витрат та доходів має велику актуальність у наш час. Додаток дозволяє користувачам вести облік своїх фінансів, контролювати бюджет і приймати обґрунтовані рішення щодо витрат і заощаджень. Завдяки цьому, люди можуть покращити свою фінансову грамотність та досягти своїх фінансових цілей.

Метою кваліфікаційної роботи є розробка мобільного додатка для обліку витрат та доходів, який надає користувачам простий та зручний інструмент для управління своїми фінансовими справами.

Об’єктом дослідження є мобільний додаток для обліку витрат та доходів, розроблений з використанням технологій Capacitor.js, React.js та TypeScript.

Предметом дослідження є процес розробки, тестування та оцінки ефективності цього додатка.

Завданнями дослідження є:

– проведення аналізу сучасного стану проблеми;

– проведення аналізу аналогічних рішень проблеми дослідження;

– визначити вимоги до розроблюваного мобільного додатку;

– проведення аналізу сучасних підходів та методів у сфері розробки мобільних додатків;

– здійснити вибір та обґрунтування технологій, які будуть використані при розробці додатка;

– практична реалізація об’єкта проектування із використанням Capacitor.js, React.js та TypeScript;

– проведення тестування додатка для перевірки його функціональності та надійності.

Таким чином, дана робота спрямована на вирішення проблеми ефективного управління фінансовими ресурсами за допомогою сучасних технологій мобільної розробки.

РОЗДІЛ 1
АНАЛІЗ РОЗРОБКИ МОБІЛЬНИХ ДОДАТКІВ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

У сучасному цифровому світі існує безліч мобільних додатків для обліку фінансів, які пропонують різноманітні функціональні можливості для користувачів. Більшість з них дозволяють вести облік доходів і витрат, створювати бюджети та переглядати статистику витрат. Деякі з додатків також пропонують можливість синхронізації з банківськими рахунками, що значно полегшує контроль за фінансами.

Одним із ключових аспектів, на який звертають увагу користувачі, є простота використання додатка та інтуїтивний інтерфейс. Мало сучасних додатків забезпечують саме такий досвід, пропонуючи прості та зручні механізми введення даних та доступ до звітності.

Також важливою характеристикою є можливість налаштування додатка під індивідуальні потреби користувачів, зокрема можливість створення «кастомізованих» категорій витрат та доходів, встановлення лімітів та цілей заощаджень.

Однак, незважаючи на широкий спектр функціональності, існуючі додатки можуть мати свої недоліки, такі як незручний та незрозумілий інтерфейс, закритий вихідний код, неможливість використовувати весь функціонал безкоштовно або відсутність певних функцій, важливих для користувачів.

Дослідження та аналіз існуючих мобільних додатків дозволяє виявити основні тренди та потреби користувачів у сфері обліку витрат та доходів. Це дає змогу сформулювати вимоги до розробки, що допоможе створити більш досконалі та ефективні рішення для контролю за фінансами.

Одним з прикладів існуючого рішення є мобільний додаток Ivy Wallet (рис. 1.1).

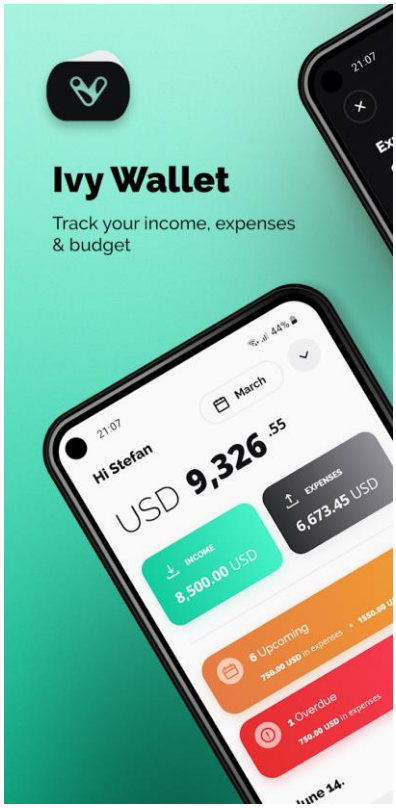

Рисунок 1.1 – Мобільний додаток Ivy Wallet

Ivy Wallet – це додаток для управління фінансами, який пропонує різноманітні переваги та функції для користувачів [8].

Додаток надає можливість запису витрат, доходів, переказів між рахунками, перегляду статистики місячних витрат, а також допомагає створювати бюджети відповідно до фінансових цілей користувача. Він підтримує багато валют, включаючи міжнародні і провідні криптовалюти, що дозволяє ефективно керувати різними активами.

Функціональність Ivy Wallet також включає планування майбутніх витрат (рис. 1.2).

Крім того, є функція резервного копіювання та експорту транзакцій у форматі CSV, що дає можливість відновити свої дані в будь-який момент. Візуалізація витрат за категоріями представлена у вигляді графіків та діаграм.

Додаток забезпечує конфіденційність, у ньому дані зберігаються тільки на пристрої користувача. Крім того, він має відкритий вихідний код, що дає

можливість проаналізувати його та виявити чи він не містить шкідливих для користувача функцій.

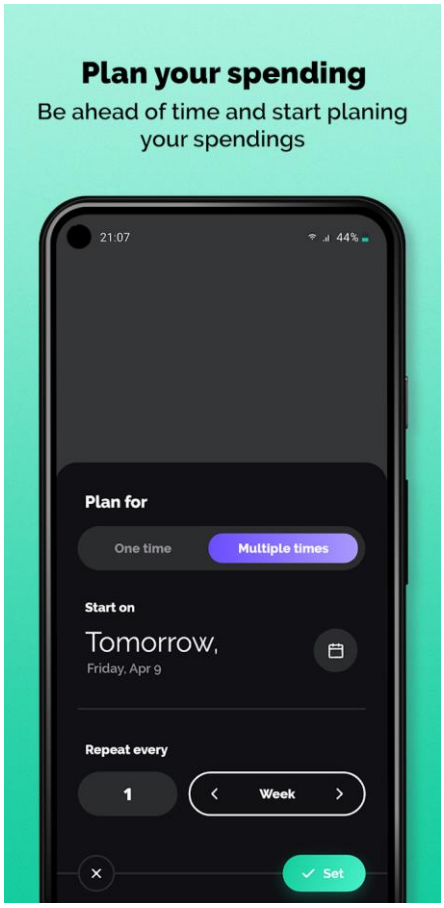

Рисунок 1.2 – Планування майбутніх витрат в Ivy Wallet

Також додаток підримує віджети на робочому столі та має вбудований калькулятор для розрахунку витрат.

Серед недоліків можна відзначити незручний і заплутаний інтерфейс користувача, відсутність можливості створювати підкатегорії, а також проблеми з продуктивністю та стабільністю.

Ще одним прикладом існуючого рішення є мобільний додаток Money Manager Expense & Budget (рис. 1.3) [10].

Перш за все, його функціонал дозволяє користувачам вводити та зберігати фінансові транзакції. Сортування витрат за різними параметрами,

такими як дата, категорія та сума, дозволяє швидко знаходити необхідну інформацію.

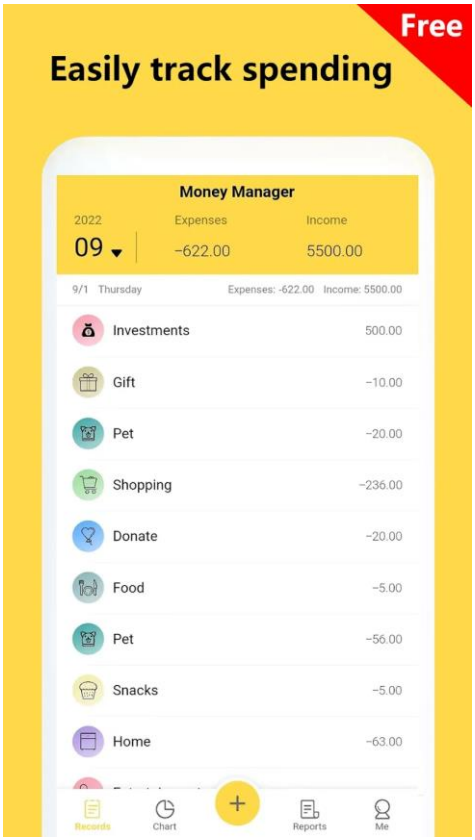

Рисунок 1.3 – Мобільний додаток Money Manager Budget & Expense

Користувачі можуть встановлювати місячні бюджети для різних категорій витрат. Аналіз витрат порівняно з встановленими бюджетами допомагає користувачам здійснювати свідомі фінансові рішення.

Є можливість створювати власні категорії.

Додаток дозволяє додавати різні валютні рахунки та виконувати транзакції в різних валютах.

Звіти та аналітика надають користувачам можливість отримувати інформацію про витрати та доходи за різними періодами (рис. 1.4).

Наявна синхронізація між пристроями за допомогою облікового запису користувача.

Рисунок 1.4 – Звіт витрат в Money Manager

Серед недоліків варто зазначити відсутність можливості створення підкатегорії, наявність реклами та платної версії для її відключення. Також додаток має застарілий користувацький інтерфейс.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Проаналізувавши існуючі мобільні додатки було прийнято рішення розробити додаток, який не буде мати розглянутих недоліків. Було поставлено наступні завдання, які повинні бути виконанні в результаті кваліфікаційної роботи:

– визначити актуальність поставленої проблеми;
– дослідити існуючі рішення;
– сформувати вимоги до розроблюваного мобільного додатку;

– провести аналіз сучасних підходів та методів у сфері розробки мобільних додатків;

– вибрати та обґрунтувати технології, які будуть використані при розробці додатка;

– розробити дизайн та безпосередньо сам мобільний додаток;

– провести тестування додатка для перевірки його функціональності та надійності.

Висновки до розділу 1

Було проведено детальний аналіз сучасного стану мобільних додатків для обліку фінансів, а саме додатки Ivy Wallet та Money Manager. Це дозволило визначити основні тенденції, переваги та недоліки існуючих рішень.

Були визначені ключові завдання кваліфікаційної роботи, серед яких визначення вимог, вибір технологій, розробка дизайну, реалізація та тестування додатка.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проектування програмного забезпечення

Аналіз вимог до програмного забезпечення є ключовим етапом у розробці мобільного додатку для обліку витрат та доходів. Основною метою цього етапу є збір, аналіз та документування вимог, які визначають, що саме має робити програмне забезпечення і як воно має це робити. Для досягнення цієї мети використовуються різні методи та інструменти.

У процесі розробки мобільного додатку для обліку витрат та доходів важливо вибрати певний тип моделі для виконання завдання з моделювання програмної системи та її елементів, яка забезпечить найбільш ефективний та надійний підхід до вирішення поставлених завдань. Існує декілька підходів до моделювання програмного забезпечення, серед яких:

– об’єктно-орієнтована модель (ООП);
– функціонально-орієнтована модель;
– модель на основі компонентів;
– модель, орієнтована на дані.

Розглянемо кожну з них детальніше.

Об’єктно-орієнтована модель заснована на концепції об’єктів, які являють собою інстанції класів. Цей підхід дозволяє описати систему через класи, що містять атрибути (дані) та методи (функції), які оперують цими даними [1]. Основні переваги ООП включають:

– інкапсуляція – захист даних об’єктів від зовнішнього втручання;
– поліморфізм – здатність різних класів використовувати однакові інтерфейси;
– наслідування – можливість створення нових класів на основі існуючих, що сприяє повторному використанню коду;
– модульність – полегшення підтримки та масштабування додатку.

Функціонально-орієнтована модель фокусується на функціях або процедурах, що виконуються системою. Вона використовує діаграми потоку даних для моделювання функцій та їх взаємодій. Основні переваги цього підходу:

– чітке визначення функцій та їх взаємодії;

– фокус на функціональних аспектах системи.

Проте, цей підхід має певні недоліки, такі як складність у підтримці та розширенні системи через відсутність інкапсуляції та модульності.

Модель на основі компонентів заснована на використанні компонентів як основних будівельних блоків системи. Компоненти представляють собою незалежні модулі, які можуть бути повторно використані в різних системах. Основні переваги:

– можливість повторного використання компонентів у різних проєктах;

– полегшення підтримки та масштабування завдяки модульності;

– недоліки включають можливі проблеми з інтеграцією компонентів та складністю управління залежностями між ними.

Модель, орієнтована на дані фокусується на структурах даних і їх взаємодії. Він використовується для систем, де основною є робота з великими обсягами даних. Переваги включають оптимізацію роботи з даними та ефективне зберігання даних. Однак, цей підхід може бути менш ефективним для систем, де важливіші функціональні аспекти.

Для мобільного додатку з обліку витрат та доходів найкраще підходить об’єктно-орієнтована модель. Основні причини цього вибору включають наступні переваги моделі:

– об’єктно-орієнтована модель дозволяє чітко структурувати додаток на окремі модулі (класи), що спрощує розробку, тестування та підтримку;

– наслідування та поліморфізм дозволяють створювати гнучкі та багаторазово використовувані компоненти, що зменшує час розробки та підвищує якість коду;

– багато аспектів фінансового обліку легко моделюються у вигляді об’єктів (наприклад, транзакції, категорії витрат) з реального світу, що робить систему інтуїтивно зрозумілою для розробників і користувачів;

– захист даних та функцій всередині об’єктів допомагає підтримувати безпеку та цілісність даних, що особливо важливо для фінансових додатків;

– можливість легко додавати нові функції та компоненти без значних змін у вже існуючій системі.

Для успішної розробки мобільного додатку для обліку витрат та доходів необхідно чітко визначити вимоги до програмного забезпечення. Це включає як функціональні, так і нефункціональні аспекти, що забезпечують відповідність додатку потребам користувачів та вимогам бізнесу. Вимоги до програмного забезпечення можна розділити на дві основні категорії: функціональні та нефункціональні вимоги.

Функціональні вимоги описують, що система повинна робити. Вони визначають специфічні функції, задачі або дії, які має виконувати додаток, щоб відповідати потребам користувачів [2].

Нефункціональні вимоги визначають, як система повинна виконувати ці функції. Вони включають характеристики якості, такі як продуктивність, безпека, надійність, зручність використання та масштабованість [2].

У таблиці 2.1 наведено список функціональних вимог до системи.

Таблиця 2.1 – Функціональні вимоги

Вимога	Опис	Деталі
Створення транзакцій	Користувач має можливість створювати нові транзакції, що включають витрати, доходи та перекази між рахунками.	Транзакції витрат включають назву, категорію витрати, суму, рахунок, дату. Транзакції доходів включають назву, категорію доходу, суму, рахунок, дату. Перекази між рахунками включають назву, суму, рахунок відправника, рахунок отримувача, дату.

Продовження таблиці 2.1

Вимога	Опис	Деталі
Створення необмеженої кількості валют	Користувач має можливість додавати нові валюти для використання в додатку.	Кожна валюта повинна мати назву (наприклад, UAH, USD, EUR), колір для візуальної ідентифікації та можливість налаштовувати відображення чисел для цієї валюти (кількість знаків після коми, розташування назви валюти відносно числа).
Створення необмеженої кількості рахунків	Користувач може створювати необмежену кількість рахунків.	Рахунки можуть бути різних типів, таких як готівкові, банківські, кредитні картки тощо. Кожен рахунок має мати назву, іконку, колір для візуальної ідентифікації, початковий баланс на момент створення та валюту до якої рахунок прив’язаний.
Пошук транзакцій	Користувач може швидко шукати транзакції за назвою.	Пошук проводиться за назвою транзакції та повинен бути розташований біля списку транзакцій. Пошук повинен враховувати помилки при наборі.
Створення необмеженої кількості валют	Користувач має можливість додавати нові валюти для використання в додатку.	Кожна валюта повинна мати назву (наприклад, UAH, USD, EUR), колір для візуальної ідентифікації та можливість налаштовувати відображення чисел для цієї валюти (кількість знаків після коми, розташування назви валюти відносно числа).
Створення необмеженої кількості рахунків	Користувач може створювати необмежену кількість рахунків.	Рахунки можуть бути різних типів, таких як готівкові, банківські, кредитні картки тощо. Кожен рахунок має мати назву, іконку, колір для візуальної ідентифікації, початковий баланс на момент створення та валюту до якої рахунок прив’язаний.
Пошук транзакцій	Користувач може швидко шукати транзакції за назвою.	Пошук проводиться за назвою транзакції та повинен бути розташований біля списку транзакцій. Пошук повинен враховувати помилки при наборі.
Фільтрація транзакцій	Користувач може фільтрувати список транзакцій за різними параметрами.	Параметри фільтрації включають дату, категорії витрат, категорії доходів, рахунки та валюти.

Продовження таблиці 2.1

Вимога	Опис	Деталі
Статистика по транзакціям	Додаток надає користувачу статистику по його транзакціям.	Статистика включає загальні витрати та доходи за певний період, розподіл витрат по категоріях, розподіл доходів по категоріях, графіки та діаграми. Статистика повинна рахуватись для кожної валюти.
Створення категорій з підтримкою необмеженого рівня вкладеності	Користувач може створювати категорії для транзакцій з необмеженою кількістю рівнів вкладеності.	Кожна категорія може містити підкатегорії, що дозволяє створювати деревоподібну структуру категорій. Категорії поділяються на категорії витрат та категорії доходів.
Можливість створення резервних копій	Користувач має можливість створювати резервні копії даних додатку для запобігання втрати даних.	Резервні копії повинні експортуватися у вигляді файлу.

У таблиці 2.2 наведено список нефункціональних вимог.

Таблиця 2.2 – Нефункціональні вимоги

Вимога	Опис
Продуктивність	Додаток повинен забезпечувати швидку обробку транзакцій та відображення результатів фільтрації та пошуку.
Масштабованість	Додаток повинен підтримувати можливість додавання нових функцій та розширення існуючих без значних змін в архітектурі.
Юзабіліті	Інтерфейс додатку повинен бути інтуїтивно зрозумілим та зручним для користувачів різного рівня підготовки.
Надійність	Додаток повинен забезпечувати безперебійну роботу та мінімізувати можливість втрати даних при збоях.

Таким чином, зазначені вимоги забезпечують функціональність та надійність мобільного додатку для обліку витрат та доходів, що відповідає потребам користувачів у ефективному управлінні своїми фінансами.

На основі цієї специфікації вимог було створено UML діаграму класів (рис. 2.1), яка відображає структуру та взаємозв’язки основних класів системи.

Діаграма допомагає візуалізувати архітектуру додатку та взаємодію між його компонентами.

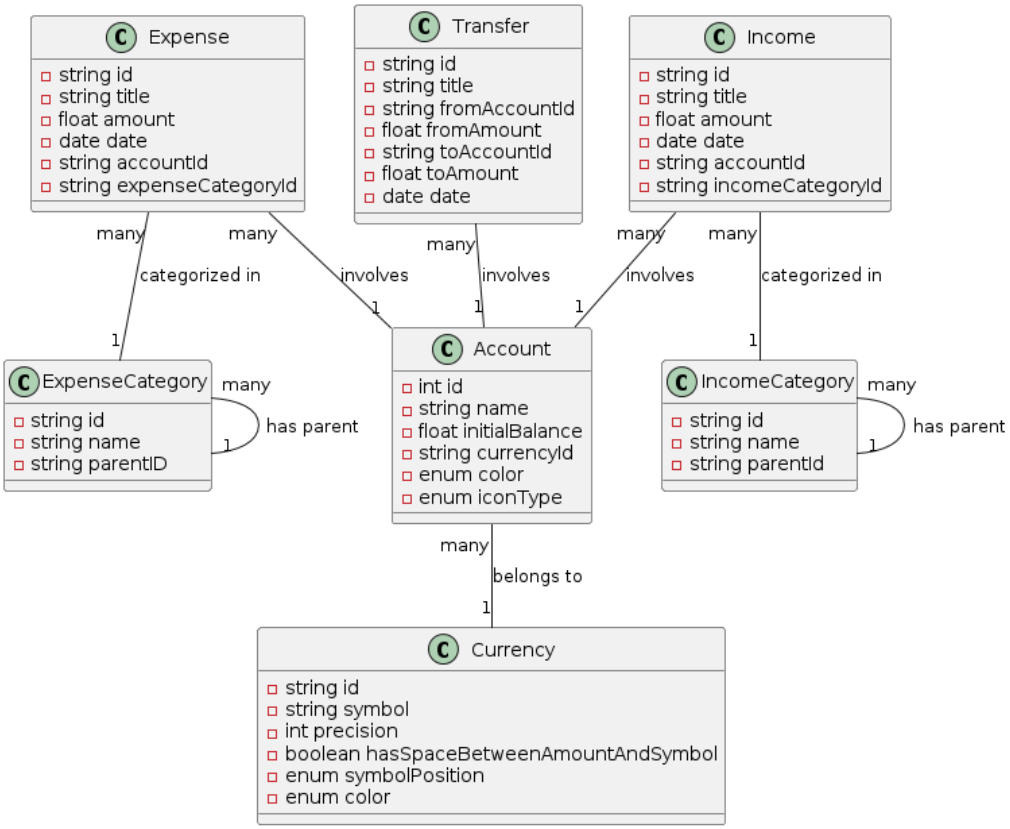

Рисунок 2.1 – UML діаграма класів

На основі вимог до програмного забезпечення був розроблений UI/UX дизайн у Figma. Дизайн забезпечує інтуїтивно зрозумілий інтерфейс та зручну навігацію для користувачів мобільного додатку з обліку витрат та доходів. Нижче наведено пояснення кожного скріншота відповідно до специфікації вимог.

Головний екран. На цьому екрані (рис. 2.2) повинен відображатися список усіх транзакцій. Кожен тип транзакції візуально повинен відрізнятися кольором для того, щоб користувач міг зразу зрозуміти яка це транзакція. Червоний колір для витрат, зелений для доходів та блакитний для переказів між рахунками.

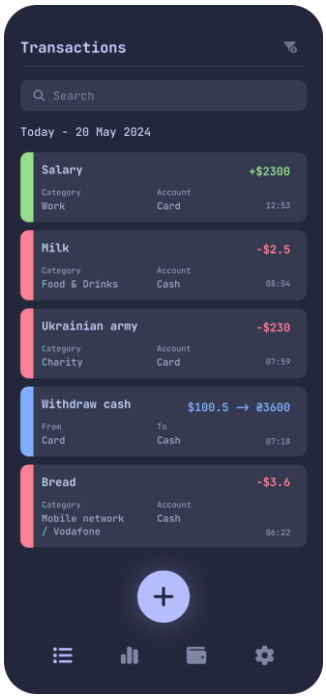

Рисунок 2.2 – Головний екран

Зверху розташоване поле для пошуку транзакцій за назвою та кнопка для відкриття параметрів фільтрування. Знизу по центру розташована велика кнопка, яка відповідає за створення транзакцій, при її натисканні повинен з’являтися вибір типу транзакції (рис. 2.3).

Рисунок 2.3 – Вибір типу транзакції

Після вибору типу транзакції, користувач повинен переходити до сторінки створення транзакції цього типу, наприклад сторінки створення витрати (рис. 2.4).

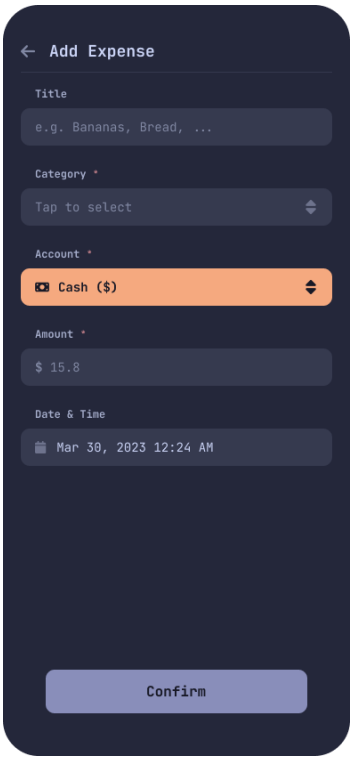

Рисунок 2.4 – Створення витрати

На цій сторінці користувач повинен ввести назву, вибрати категорію, рахунок, ввести суму, та вибрати дату проведення транзакції. Після натиснення кнопки підтвердження користувач повертається на головний екран, де бачить щойно створену транзакцію.

З головного екрану користувач може перейти на екран зі списком рахунків (рис. 2.5), натиснувши третю кнопку в меню.

На цьому екрані відображається список усіх рахунків користувача та кнопка для створення нових рахунків. Картка кожного рахунку показує інформацію про його баланс та назву.

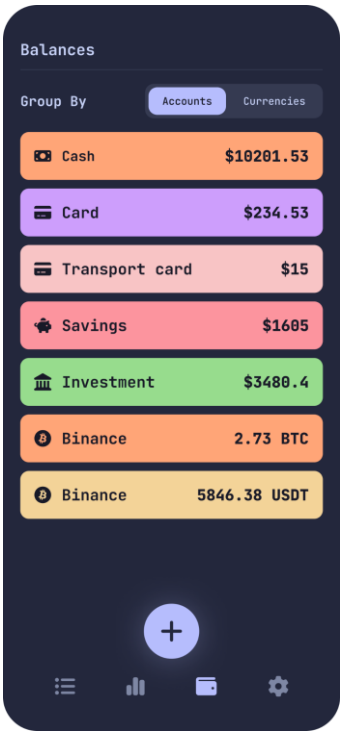

Рисунок 2.5 – Список рахунків

Також є можливість перейти на екран зі списком валют (рис. 2.6).

Рисунок 2.6 – Список валют

При натисканні на картку рахунку відкривається екран детального огляду рахунку (рис. 2.7).

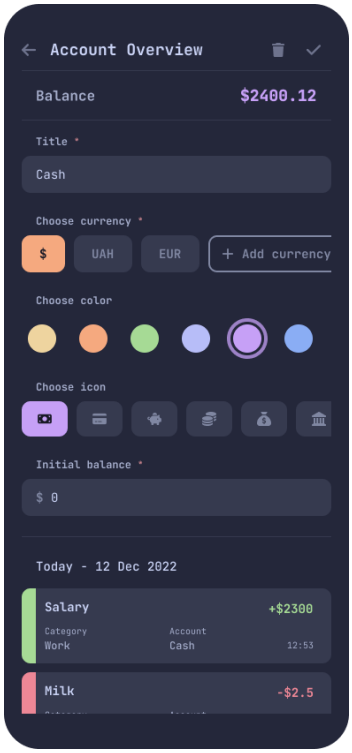

Рисунок 2.7 – Екран рахунку

На цьому екрані користувач може редагувати рахунок, змінюючи назву, колір, іконку, початковий баланс. Також користувачу відображається список транзакцій пов’язаних з цим рахунком.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Розробка мобільних додатків зазвичай здійснюється за допомогою нативних, кросплатформних чи гібридних технологій.

Нативна розробка мобільних додатків передбачає використання мов програмування та інструментів, специфічних для конкретної платформи. Це дозволяє максимально використовувати можливості операційної системи та апаратного забезпечення, забезпечуючи високу продуктивність та інтеграцію з

нативними API. Для Android і iOS основними нативними мовами програмування є Kotlin та Swift відповідно.

Kotlin є сучасною мовою програмування, офіційно підтримуваною Google для розробки додатків під Android. Вона була створена компанією JetBrains і стала популярною завдяки своїй зручності та ефективності [9].

Переваги Kotlin:

– повністю сумісний з Java, що дозволяє використовувати існуючий код на Java та поступово переходити на Kotlin. Це забезпечує зворотну сумісність і спрощує процес міграції;

– пропонує більш лаконічний та виразний синтаксис порівняно з Java, що дозволяє писати менше коду та робить його більш читабельним;

– має вбудовану систему, що допомагає уникнути помилок, пов’язаних з null-посиланнями (NullPointerException), що є однією з найбільш поширених проблем у Java;

– підтримує корутини, що дозволяє спрощувати асинхронне програмування та покращує продуктивність додатків.

Недоліки Kotlin:

– хоча Kotlin набирає популярності, він все ще відносно новий, і деякі розробники можуть стикатися з обмеженою кількістю ресурсів та документації.

– розробникам, які звикли до Java, може знадобитися час на освоєння нових концепцій та синтаксису Kotlin.

Swift – це мова програмування, розроблена компанією Apple для створення додатків під iOS, macOS, watchOS та tvOS. Swift була представлена в 2014 році як більш сучасна та безпечна альтернатива Objective-C.

Переваги Swift:

– створений для швидкої та ефективної роботи, забезпечуючи високу продуктивність додатків;

– багатьом типам помилок на стадії компіляції завдяки строгій типізації та управлінню пам’яттю;

– має лаконічний і зрозумілий синтаксис, що полегшує процес навчання та підвищує продуктивність розробників;

– Apple активно розвиває Swift, регулярно випускаючи оновлення та нові функції.

Недоліки Swift:

– продовжує розвиватися, і з кожним новим випуском можуть з’являтися зміни, що вимагають адаптації існуючого коду;

– на відміну від Kotlin, Swift не є сумісним з попередніми мовами розробки (Objective-C), що може ускладнити використання існуючих бібліотек.

Нативна розробка забезпечує найвищу продуктивність та стабільність додатків, оскільки вони безпосередньо використовують всі можливості операційної системи та апаратного забезпечення. Це особливо важливо для додатків, які вимагають високої продуктивності, складної графіки або тісної інтеграції з нативними функціями пристроїв (наприклад, камери, датчики, GPS).

Нативні додатки мають швидкий час відгуку і кращу оптимізацію ресурсів, що покращує користувацький досвід. Крім того, нативна розробка забезпечує кращу підтримку нових функцій операційної системи та апаратного забезпечення, оскільки оновлення і нові API зазвичай стають доступними для нативних інструментів раніше, ніж для кросплатформних рішень.

Основним недоліком нативної розробки є необхідність створення та підтримки окремих версій додатку для кожної платформи (Android та iOS), що збільшує витрати на розробку та підтримку. Це потребує більше часу і ресурсів, а також команди розробників, які володіють навичками розробки для обох платформ.

Кросплатформені рішення дозволяють розробникам створювати мобільні додатки, які можуть працювати на різних операційних системах, таких як iOS та Android, використовуючи єдину базу коду. Це значно скорочує час та витрати на розробку, оскільки не потрібно створювати окремі версії додатку

для кожної платформи. Серед найпопулярніших кросплатформних фреймворків можна виділити React Native та Flutter.

React Native – це популярний фреймворк, розроблений компанією Facebook, який дозволяє створювати мобільні додатки з використанням JavaScript та React [11].

Переваги React Native:

– дозволяє використовувати один код для обох платформ (iOS та Android), що значно знижує витрати на розробку та підтримку додатків;

– забезпечує близьку до нативної продуктивність завдяки використанню нативних компонентів;

– можливість гарячого перезавантаження (hot reload) дозволяє розробникам бачити зміни в коді в режимі реального часу, що прискорює процес розробки;

– має велику спільноту розробників та багато готових бібліотек і компонентів, що спрощує розробку та впровадження нових функцій.

Недоліки React Native:

– для складних інтерфейсів та анімацій React Native може поступатися нативним рішенням у продуктивності;

– деякі нативні функції можуть бути недоступними або потребувати написання додаткових нативних модулів.

Flutter – це фреймворк від Google, який дозволяє створювати мобільні додатки з використанням мови програмування Dart [7].

Переваги Flutter:

– використовує власний механізм рендерингу, що забезпечує високу продуктивність та швидкість роботи додатків;

– надає потужні засоби для створення складних анімацій та графіки, що дозволяє створювати привабливі користувацькі інтерфейси;

– як і React Native, Flutter дозволяє використовувати один код для обох платформ, знижуючи витрати на розробку та підтримку;

– Google активно розвиває Flutter, що забезпечує постійні оновлення та нові можливості.

Недоліки Flutter:

– хоча Flutter швидко набирає популярність, його спільнота все ще менша порівняно з React Native;

– є відносно новим фреймворком, тому можуть виникати проблеми з інтеграцією з деякими нативними API та бібліотеками.

Гібридні рішення для розробки мобільних додатків поєднують в собі елементи веб та нативної розробки. Вони дозволяють створювати додатки за допомогою веб-технологій, таких як HTML, CSS і JavaScript, які потім вбудовуються в нативний додаток за допомогою WebView.

Ionic – це популярний фреймворк для гібридної розробки, який використовує веб-технології для створення кросплатформених додатків.

Переваги Ionic:

– дозволяє використовувати один код для обох платформ (iOS та Android), що значно знижує витрати на розробку та підтримку додатків;

– використання знайомих веб-технологій (HTML, CSS, JavaScript) дозволяє швидко створювати та змінювати інтерфейс користувача;

– має велику спільноту розробників та багато готових компонентів, що спрощує розробку та впровадження нових функцій;

– легко інтегрується з популярними фреймворками, такими як Angular, React та Vue.js.

Недоліки Ionic:

– додатки Ionic працюють через WebView, що може впливати на продуктивність, особливо для складних та графічно насичених додатків;

– для додатків, які потребують високої продуктивності та швидкого відгуку, Ionic може поступатися нативним рішенням.

Apache Cordova (раніше PhoneGap) – це один з перших фреймворків для гібридної розробки мобільних додатків [6]. Він надає інструменти для упаковки веб-додатків у нативні.

Переваги Cordova:

– дозволяє використовувати один код для обох платформ, що знижує витрати на розробку та підтримку додатків;

– забезпечує доступ до нативних функцій пристроїв через плагіни, що дозволяє використовувати камери, GPS, файлову систему тощо;

– має довгу історію та велику спільноту, що забезпечує доступ до багатьох ресурсів та плагінів.

Недоліки Cordova:

– як і у випадку з Ionic, додатки Cordova працюють через WebView, що може впливати на продуктивність та відгук додатку;

– інтеграція та налаштування деяких плагінів можуть бути складними та потребувати додаткових зусиль.

Capacitor – це сучасний фреймворк для гібридної розробки, створений командою Ionic. Він надає інструменти для створення веб-додатків, які можуть працювати як нативні додатки на різних платформах [4].

Переваги Capacitor:

– забезпечує більш сучасний підхід до гібридної розробки, ніж Cordova, з кращою інтеграцією з нативними функціями та API;

– дозволяє використовувати один код для обох платформ, що знижує витрати на розробку та підтримку додатків;

– легко інтегрується з популярними веб-фреймворками, такими як React, Angular та Vue.js;

– забезпечує доступ до нативних функцій через плагіни та дозволяє створювати власні плагіни для специфічних потреб.

Недоліки Capacitor:

– як і у випадку з іншими гібридними рішеннями, додатки Capacitor працюють через WebView, що може впливати на продуктивність;

– як і з усіма кросплатфомними рішеннями можуть бути складнощі з взаємодією з нативним API та недоступність усіх функцій на відміну від нативних додатків.

Гібридні технології зазвичай використовуються у поєднанні з фронтенд фреймворками, такими як React.js.

React.js – це популярна бібліотека JavaScript, розроблена компанією Facebook, яка використовується для створення користувацьких інтерфейсів, зокрема односторінкових додатків (SPA). Вона забезпечує швидку та ефективну розробку інтерактивних UI завдяки низці унікальних властивостей та підходів до управління станом і рендерингом компонентів.

Однією з основних концепцій React.js є компонентний підхід, який передбачає розбиття інтерфейсу на невеликі, незалежні та повторно використовувані компоненти. Кожен компонент є самостійним модулем з власною логікою та станом, що полегшує розробку, тестування та підтримку додатку. Це дозволяє створювати складні додатки, що складаються з великої кількості взаємодіючих частин.

React.js використовує віртуальний DOM – ефективну модель об’єктів документу, яка дозволяє мінімізувати кількість операцій з реальним DOM. Віртуальний DOM створює легку копію реального DOM і використовує алгоритм узгодження (reconciliation), щоб визначити найменшу кількість змін, необхідних для синхронізації віртуального DOM з реальним. Це значно підвищує продуктивність додатків, особливо при роботі з великими даними або складними інтерфейсами.

TypeScript – це надбудова над JavaScript, яка додає статичну типізацію і розширені можливості для розробки додатків. Вона була розроблена компанією Microsoft і призначена для того, щоб полегшити написання, розуміння і підтримку коду великих і складних проєктів. TypeScript додає до JavaScript статичну типізацію, що означає, що типи даних визначаються під час написання коду і перевіряються на етапі компіляції [12, с. 8]. Це дозволяє виявляти багато помилок на ранніх стадіях розробки, що значно покращує якість коду і знижує витрати на його підтримку. Наприклад, якщо функція очікує параметр типу string, TypeScript видасть помилку, якщо передати їй значення типу number. Однією з великих переваг TypeScript є його відмінна

інтеграція з сучасними середовищами розробки (IDE). Завдяки статичній типізації та метаданим про типи, IDE можуть надавати розширені можливості автозавершення коду, рефакторингу, навігації по коду та виявлення помилок. Це значно підвищує продуктивність розробників і полегшує роботу з великими кодовими базами.

Після аналізу доступних технологій для розробки мобільного додатку з обліку витрат та доходів було прийнято рішення використовувати Capacitor.js разом з React.js та TypeScript. Цей вибір обґрунтований рядом факторів.

По-перше, Capacitor.js дозволяє створювати додатки, що працюють як на iOS, так і на Android, з використанням єдиного коду. Це значно знижує витрати на розробку та підтримку порівняно з нативною розробкою для кожної платформи окремо.

По-друге, використання React.js для розробки інтерфейсу користувача забезпечує швидкість розробки та легкість у підтримці. React.js має велику спільноту розробників та багато готових компонентів, що сприяє швидкому впровадженню нових функцій. Capacitor.js дозволяє легко інтегрувати веб-технології з нативними функціями пристроїв, забезпечуючи доступ до файлової системи, сповіщень та інших необхідних нативних функцій.

По-третє, поєднання Capacitor.js та React.js забезпечує достатню продуктивність для додатку з обліку витрат та доходів, порівняно з нативними рішеннями такими як Kotlin чи Swift, оскільки додаток не містить високонавантажених компонентів.

По-четверте, використання єдиного стека технологій для веб та мобільної розробки знижує поріг входження для розробника, полегшує підтримку та розширення додатку. Не менш важливим фактором є наявність досвіду з JavaScript та React.js, що дозволяє уникнути витрат часу на вивчення Kotlin та Swift.

Як React.js, так і Capacitor.js мають активну спільноту та хорошу документацію, що забезпечує доступ до великої кількості ресурсів, прикладів та готових рішень для вирішення різноманітних завдань. Крім того, використання

всієї веб екосистеми, яка містить багато готових веб компонентів, напрацьованих роками в індустрії веб розробки, є значною перевагою.

Висновки до розділу 2

Було проведено детальний аналіз та визначення вимог до програмного забезпечення для мобільного додатку з обліку витрат та доходів. Розглянуто функціональні та нефункціональні вимоги до системи, а також створено специфікацію вимог, яка включає основні функціональні можливості додатку. На основі специфікації було розроблено UML діаграму класів, яка відображає структуру та взаємозв’язки між компонентами системи.

Крім того, було розроблено UI/UX дизайн додатку за допомогою інструменту Figma, який відповідає визначеним вимогам та забезпечує інтуїтивно зрозумілий інтерфейс для користувачів. В результаті, обрані інструменти та підходи дозволять створити ефективний, зручний та функціональний мобільний додаток, що задовольнить потреби користувачів в обліку витрат та доходів.

Також було проведено аналіз різних технологій для розробки мобільних додатків, включаючи нативні рішення (Kotlin для Android та Swift для iOS), кросплатформні рішення (React Native та Flutter) та гібридні рішення (Ionic, Apache Cordova та Capacitor.js). Було зроблено вибір на користь використання Capacitor.js разом з React.js та TypeScript, що обумовлено їхніми перевагами у швидкості розробки, зручності підтримки та інтеграції з нативними функціями пристроїв.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ОБЛІКУ ВИТРАТ ТА ДОХОДІВ

3.1 Практична реалізація об’єкта проєктування

Додаток розробляється за допомогою мови програмування TypeScript, Capacitor.js та React.js. Для розробки на TypeScript для початку потрібно встановити Node.js з офіційного сайту nodejs.org. Node.js – це середовище для запуску JavaScript поза браузером. Node.js постачається з менеджером залежностей npm, який дозволяє встановлювати бібліотеки та керувати їх версіями в проєкті. Для створення Capacitor.js проєкта потрібно виконати у командному рядку команду «npm init @capacitor/app». Після виконання команди розгортач проєктів Capacitor запитає назву проєкту і створить папку з ним.

Перед тим як безпосередньо приступити до розробки мобільного додатку за допомогою Capacitor, потрібно також встановити середовище розробки Android Studio, яке має в собі інструменти для запуску та збірки мобільних додатків на Android. Це можна зробити на офіційному сайті Android Studio.

Після встановлення Android Studio можна приступати до розробки мобільного додатку. Для початку потрібно визначитися із архітектурою коду в проєкті, щоб зручно та зрозуміло писати код. Код буде поділений на шари, кожен з яких буде мати свою зону відповідальності. Для розробки була вибрана архітектура, яка складається з наступних шарів:

– shared – тут розміщується загальний код, який може бути перевикористаний в інших частинах коду. Сюди відносяться різні допоміжні функції, ui-kit та інший код, який може бути корисний навіть для розробки інших мобільних додатків;

– entities – тут розміщується код, який стосується основних сутностей розроблюваної системи, таких як транзакції, рахунки, валюти, налаштування;

– features – тут розміщується код, який стосується дій, які користувач може робити в системі, наприклад створення / редагування / видалення транзакцій, створення / редагування / видалення рахунків, створення / редагування / видалення валют;

– widgets – тут знаходяться компоненти, які поєднують компоненти з шарів entities та features;

– pages – тут знаходяться сторінки мобільного додатку, наприклад сторінка зі списком транзакцій, сторінка для створення рахунку чи сторінка налаштувань;

– app – цей шар компонує усі вищезгадані шари і містить загальні налаштування, такі як конфігурації роутингу для переходу між сторінками та інші.

Ця структура дозволить легко орієнтуватись в коді та добавляти новий функціонал при потребі.

Розглянемо розробку головної сторінки зі списком транзакцій. На основі UI/UX дизайну, який був описаний в розділі 2, головна сторінка містить такі елементи: заголовок сторінки, поле для пошуку транзакцій за назвою та список транзакцій. Розпочнемо з компонента транзакції у списку (ліст. 3.1).

Лістинг 3.1 – Компонент транзакції TransactionCard

```
import { twMerge } from "tailwind-merge";

export const TransactionCard = ({
  title,
  formattedAmount,
  time,
  firstProperty,
  secondProperty,
  className,
  amountClassName,
}: TransactionCardProps) => {
  return (
    <div
      className={twMerge(
        "flex flex-col gap-2 rounded border-l-[1rem]",
```

```
        "py-3 pl-2.5 pr-5 transition-colors bg-surface0 active:bg-
surface1",
        className,
      )}
    >
      <div className="flex gap-4 justify-between overflow-x-auto">
        <span className="text-sm font-bold text-text
truncate">{title}</span>
        <span
          className={twMerge(
            "text-sm font-bold min-w-max text-right overflow-x-auto
whitespace-nowrap",
            amountClassName,
          )}
        >
          {formattedAmount}
        </span>
      </div>
      <div className="flex justify-between items-end gap-4">
        <div className="flex flex-1 justify-between gap-4">
          <TransactionCardProperty className="flex-1" {...firstProperty}
/>
          <TransactionCardProperty className="flex-1" {...secondProperty}
/>
        </div>
        <span className="text-xxs text-subtext0">{time}</span>
      </div>
    </div>
  );
};
```

Кінець лістингу 3.1

У лістингу 3.1 зображений код компонента «TransactionCard», цей компонент є абстрактним і містить спільну верстку для усіх видів транзакцій: витрат, доходів та переказів між рахунками. Стилізація HTML відбувається за допомогою бібліотеки Tailwind CSS, яка дозволяє легко і швидко задавати CSS властивості для тегів, використовуючи короткі імена HTML класів. У лістингу 3.2 зображений код компонента «ExpenseCard», який побудований на базі компонента «TransactionCard» і є компонентом витрати.

Лістинг 3.2 – Компонент витрати ExpenseCard

```
export const ExpenseCard = ({ expense }: ExpenseCardProps) => {
```

```
  const formattedAmount =
createExpenseAmountString(expense.formattedAmount);
  return (
    <Link to={`/expenses/${expense.id}`}>
      <TransactionCard
        title={expense.title}
        formattedAmount={formattedAmount}
        time={expense.time}
        firstProperty={{ title: "Category", value: expense.categoryTitle
}}
        secondProperty={{ title: "Account", value: expense.accountTitle
}}
        className="border-red"
        amountClassName="text-red"
      />
    </Link>
  );
};
```

Кінець лістингу 3.2

Дивлячись на код лістингу 3.2 видно, що витрата має рамку зліва червоного кольору, це задається за допомогою класа «border-red». Компонент «ExpenseCard» приймає інформацію про витрату через props компонента, а саме такі параметри як назву витрати (title), суму (formattedAmount), назву категорії (categoryTitle), назву рахунка з якого була зроблена витрата (accountTitle) та час коли витрата була створена (time). Також приймається параметр id, який потрібен для того, щоб сформувати посилання, яке веде на сторінку для детального огляду витрати. По аналогії було створено компоненти для доходів (IncomeCard) та переказів між рахунками (TransferCard). Компонент IncomeCard має ідентичний код до компонента ExpenseCard окрім кольору рамки, у випадку IncomeCard вона зеленого кольору. Стосовно TransferCard, то цей компонент приймає два рахунки в параметрах, це рахунок з якого роблять переказ та рахунок на який роблять переказ.

Після створення компонента транзакції можна розпочати розробку компонента списка транзакцій. Оскільки на сторінці повинні відображатися усі транзакції користувача, то при великій кількості транзакцій у користувача будуть проблеми з швидкодією додатка, оскільки потрібно відрендерити велику

кількість компонентів на екрані. Щоб це оптимізувати доцільно використати такий підхід як віртуалізація списків. Цей підхід дозволяє рендерити на екрані тільки ті компоненти які знаходяться у зоні видимості користувача [3]. Розробка рішення з нуля, яке буде реалізовувати віртуалізацію списків є нелегкою задачею і вимагає багато часу на розробку та тестування. Відповідно було прийнято рішення використовувати готову бібліотеку, яка реалізує цей функціонал для React. Ця бібліотека називається «react-virtuoso». Для встановлення бібліотеки потрібно виконати у командному рядку команду «npm install react-virtuoso».

Як тільки бібліотека була встановлена, можна приступити до реалізації списка. У лістингу 3.3 зображений код списку транзакцій.

Лістинг 3.3 – Компонент списку транзакцій

```
export const GroupedTransactionList = ({
  searchTerm, filters = {}, showEmptyState = false,
}: GroupedTransactionListProps) => {
  const { accounts } = useAccountsStore();
  const { expenseCategories } = useExpenseCategoriesStore();
  const { incomeCategories } = useIncomeCategoriesStore();
  const transactions = useTransactions();
  const filteredTransactions = searchTransactionsByTitle(
    filterTransactions(
      transactions, filters, accounts,
      expenseCategories, incomeCategories,
    ),
    searchTerm,
  );
  const transactionGroups = groupTransactionsByDay(filteredTransactions);
  return (
    <div>
      {transactionGroups.length ? (
        <Virtuoso
          useWindowScroll
          data={transactionGroups}
          itemContent={(_, transactionGroup) => (
            <TransactionListGroup
              className="pb-6 group-last:pb-0"
              transactionGroup={transactionGroup}
            />
          )}
        />
```

```
      ) : showEmptyState ? (
        <p
          className={twMerge(
            "absolute left-1/2 top-1/2 -translate-x-1/2 -translate-y-1/2
w-[75%]",
            "text-center text-base/[1.75] font-medium text-subtext0
whitespace-pre-line",
          )}
        >
          You don’t have any transactions yet. To add first tap add
button
        </p>
      ) : null}
    </div>
  );
};
```

Кінець лістингу 3.3

Реалізувавши список транзакцій можна приступити до реалізації пошуку транзакцій за назвою. Відповідно до вимог, пошук повинен враховувати помилки при формуванні пошукового запиту, для цього можна скористатись таким алгоритмом як Bitap. Цей алгоритм дозволяє реалізувати нечіткий пошук, він шукає підрядок в рядку і видає відсоток схожості. Доцільно використати готову бібліотеку, яка реалізує цей алгоритм. Була вибрана бібліотека «fuse.js». Для того, щоб встановити її, потрібно виконати у командному рядку команду «npm install fuse.js». У лістингу 3.4 зображений код функції, яка шукає транзакції за назвою.

Лістинг 3.4 – Функція пошуку транзакцій за назвою

```
import Fuse from "fuse.js";

interface SearchTransaction {
  title: string;
}

export const searchTransactionsByTitle = <T extends SearchTransaction>(
  transactions: T[],
  searchTerm?: string,
): T[] => {
  searchTerm = searchTerm?.trim();
  if (!searchTerm) {
```

```
    return transactions;
  }
  const fuse = new Fuse(transactions, {
    keys: ["title"] as (keyof SearchTransaction)[],
    ignoreLocation: true,
    threshold: 0.3,
  });
  const result = fuse.search(searchTerm);
  return result.map((i) => i.item);
};
```

Кінець лістингу 3.4

Дана функція приймає два параметри. Перший – це список транзакцій серед яких шукати (transactions), а другий – це пошуковий запит (searchTerm). Функція створює об’єкт класу «Fuse» і задає параметри для нього такі як поле за яким шукати транзакції (keys), відсоток схожості (threshold) та параметр ignoreLocation, який вказує, що співпадіння повинно «шукатися» не тільки з початку рядка. Threshold був встановлений експериментальним шляхом у значення 0.3, що означає відсоток схожості 70%. Далі за допомогою об’єкта класу «Fuse» функція шукає транзакції і повертає результат. У результаті функція «fuse.search» повертає список транзакцій, які відсортовані за відсотком схожості у спадаючому порядку.

Після створення функції, яка шукає транзакції можна розпочати створення компонента поля для пошуку. Код компонента поля для пошуку транзакцій зображений у лістингу 3.5.

Лістинг 3.5 – Компонент поля для пошуку транзакцій

```
import { SearchIcon } from "@shared/ui/icons";
import { Input } from "@shared/ui/inputs";

interface TransactionsSearchBarProps {
  value: string;
  onChange: (value: string) => void;
}

export const TransactionsSearchBar = ({
  value,
  onChange,
}: TransactionsSearchBarProps) => {
```

```
  return (
    <Input
      inputBoxClassName="gap-2.5 py-2.5"
      value={value}
      onChange={(event) => onChange(event.target.value)}
      placeholder="Search"
      leftAddon={<SearchIcon size="xxs" className="" />}
    ></Input>
  );
};
```

Кінець лістингу 3.5

Отже, головний екран було розроблено успішно. Окрім цього було також розроблено обчислення статистики для транзакцій, керування рахунками, валютами та категоріями, створення резервних копій.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Етап тестування дуже важливий у розробці будь-якого програмного забезпечення, оскільки воно дозволяє переконатися, що програмне забезпечення відповідає поставленим вимогам до нього. Для тестування мобільного додатку для обліку витрат та доходів було застосовано автоматизоване тестування, а саме написання unit тестів. Unit тести дозволяють тестувати окремі частини коду ізольовано один від одного, їх легко запускати та вони швидко виконуються [5, с. 14]. Для написання unit тестів була обрана бібліотека для написання тестів під назвою «vitest». Для того, щоб її встановити потрібно виконати у командному рядку команду «npm install -D vitest». Додаткових конфігурацій бібліотека не потребує. Тестом вважається файл, назва якого закінчується на «.test.ts». Розглянемо написання тесту для функції, яка виконує пошук транзакцій. Створимо файл за шляхом «src/features/search-transactions/search.test.ts», код цього файлу зображений в лістингу 3.6.

Лістинг 3.6 – Unit тест для функції для пошуку транзакцій

```
describe("search transactions", () => {
```

```
  const now = DateTime.local();

  it("empty list", () => {
    const transactions: Transaction[] = [];
    const actual = searchTransactionsByTitle(transactions, "term");
    assert.deepEqual(actual, transactions);
  });

  it("without matches", () => {
    const transactions: Transaction[] = [
      {
        id: "1", type: TransactionType.expense,
        title: "Some title", categoryId: "10",
        accountId: "100", amount: "1",
        datetime: now, createdAt: now,
      },
      {
        id: "2", type: TransactionType.expense,
        title: "Some title 2", categoryId: "10",
        accountId: "100", amount: "1",
        datetime: now, createdAt: now,
      },
    ];
    const actual = searchTransactionsByTitle(transactions, "term");
    assert.deepEqual(actual, []);
  });

  it("with 1 match", () => {
    const transactions: Transaction[] = [
      {
        id: "1", type: TransactionType.expense,
        title: "Erase your disk", categoryId: "10",
        accountId: "100", amount: "1",
        datetime: now, createdAt: now,
      },
      {
        id: "2", type: TransactionType.expense,
        title: "Some title 2", categoryId: "10",
        accountId: "100", amount: "1",
        datetime: now, createdAt: now,
      },
    ];
    const actual = searchTransactionsByTitle(transactions, "title");
    assert.deepEqual(actual, [transactions[1]]);
  });
});
```

Кінець лістингу 3.6

У лістингу 3.6 є три тести. Перший тест тестує функцію з пустим масивом транзакцій і перевіряє, що вона має повертати пустий масив. Другий тест тестує те, що функції має повертати пустий масив, якщо пошукових збігів не було знайдено. Третій тест тестує те, що функція має повернути лише одну транзакцію з двох, оскільки тільки одна транзакція задовольняє пошуковий запит.

3.3 Розробка бази даних

Оскільки транзакції, рахунки та валюти повинні зберігатися в мобільному додатку, то потрібна база даних. Для взаємодії з базою даних був обраний плагін для Capacitor.js під назвою Preferences. Цей плагін – це абстракція над SharedPreferences базою даних в Android та UserDefaults в iOS. Як і SharedPreferences так і UserDefaults є базами даних типу ключ-значення. Це не реляційні бази даних, у них дані зберігаються у вигляді пари ключ-значення. Було вирішено зберігати дані у вигляді JSON документів, де ключ – це назва колекції (наприклад expenses) документів, а значення – це сама колекція, яка є масивом з документами у форматі JSON.

Була спроектована схема бази даних, яка зображена на рисунку 3.1 у вигляді UML діаграми класів.

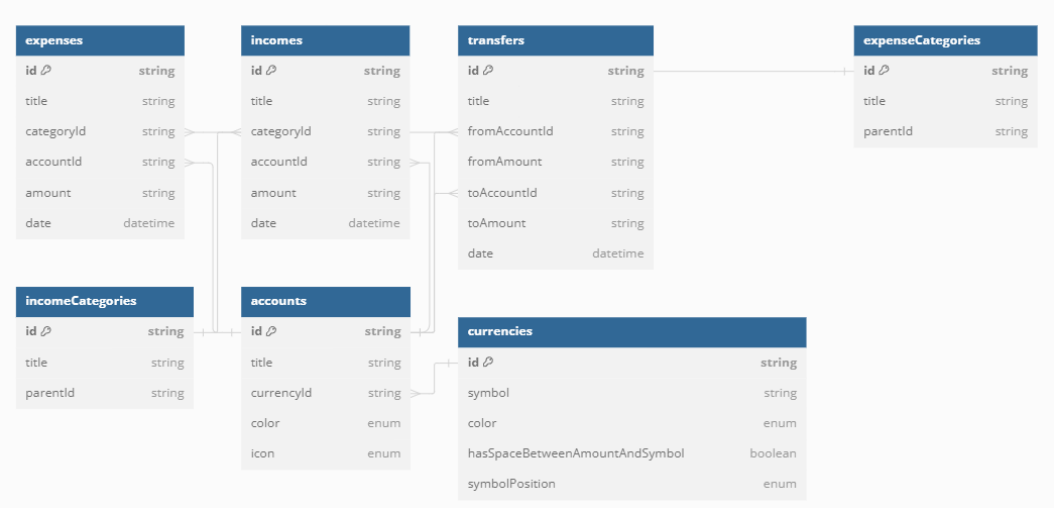

Рисунок 3.1 – Схема бази даних

Перед тим як почати взаємодіяти з базою даних потрібно встановити Capacitor Preferences плагін, для цього потрібно виконати у командному рядку команду «npm install @capacitor/preferences».

Взаємодіяти в коді напряму з базою даних не дуже зручно і призводить до дублікації коду, тому було створено абстракцію над базою даних, яка містить функції для зручної взаємодії з нею. Для кожної колекції був створений окремий клас, який має методи для отримання, створення, редагування чи видалення документів. Одним з таких класів є «PreferencesExpensesAPI», код якого зображений у лістингу 3.7. Він відповідає за керування витратами в базі даних.

Лістинг 3.7 – Клас PreferencesExpensesAPI

```
export class PreferencesExpensesAPI implements ExpensesAPI {
  private async getState(): Promise<ExpensesDTO> {
    const { value } = await Preferences.get({ key: "expenses" });
    if (value === null) {
      const state: ExpensesDTO = {};
      await this.setState(state);
      return state;
    }
    return JSON.parse(value);
  }

  private async setState(state: ExpensesDTO) {
    await Preferences.set({
      key: "expenses",
      value: JSON.stringify(state),
    });
  }

  async getExpenses() {
    return await this.getState();
  }

  async setExpenses(expenses: ExpensesDTO) {
    await this.setState(expenses);
  }

  async createExpense(expense: CreateExpenseDTO) {
    const expenses = await this.getState();
    const id = uuidv4();
    const createdExpense: ExpenseDTO = {
```

```
      ...expense,
      id,
      createdAt: Date.now(),
    };
    expenses[id] = createdExpense;
    await this.setState(expenses);
    return createdExpense;
  }

  async updateExpense(id: string, expense: CreateExpenseDTO) {
    const expenses = await this.getState();
    expenses[id] = { ...expenses[id], ...expense, id };
    await this.setState(expenses);
    return expenses[id];
  }

  async deleteExpense(id: string): Promise<void> {
    const expenses = await this.getState();
    delete expenses[id];
    await this.setState(expenses);
  }
}
```

Кінець лістингу 3.7

Взаємодія з базою даних безпосередньо знаходиться у приватних методах «getState» та «setState», які читають значення з ключа «expenses» чи записують в нього. При читанні з бази даних відбувається десеріалізація JSON рядка, а при записуванні, дані наоборот серіалізуються і записуються у базу даних у вигляді рядка. По аналогії до класу «PreferencesExpensesAPI» були створені класи для всіх інших колекцій: доходів, переказів між рахунками, рахунків та валют.

3.4 Розробка інсталяційного пакета

Є два види файлів через які поширюються Android додатки, це APK та AAB. У випадку APK файлу достатньо запустити його на пристрої і розпочнеться встановлення додатка. AAB файл використовується для публікації додатка у Play Store. Оскільки додаток повинен бути викладений у Play Store, розглянемо збірку AAB файлу.

Для початку потрібно зібрати React bundle, це можна зробити за допомогою команди «npx vite build». Після виконання команди, зібраний React додаток буде у папці «build». Після цього виконати команду «npx cap sync», що скопіює папку «build» у Android проєкт. Наступним етапом буде збірка самого AAB файла в Android Studio.

Для збірки додатка потрібно відкрити проєкт у Android Studio. Після відкриття натиснути Build > Generate Signed Bundle / APK. У результаті з’явиться вікно з вибором типу збірки (рис. 3.2), на якому потрібно вибрати «Android App Bundle».

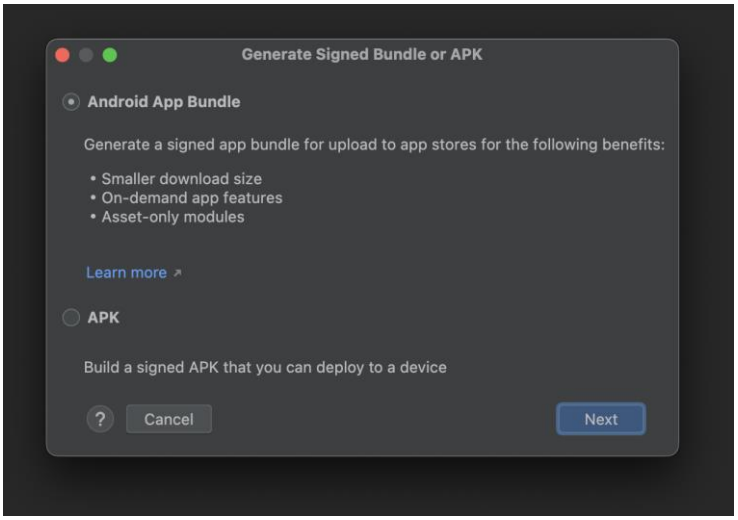

Рисунок 3.2 – Вибір AAB збірки

Після вибору AAB збірки, потрібно створити key store та задати пароль для нього. Key store використовується для підпису готової збірки. Далі з’явиться вікно в якому інструмент запропонує вибрати варіант збірки, «debug» чи «release». Оскільки додаток буде опублікований у Play Store, то був обраний варіант «release». Після цього потрібно натиснути «Create», після чого Android Studio розпочне збірку та відкриє папку із зібраним AAB файлом.

Наступним етапом буде реєстрація аккаунта розробника в Google Play. Реєстрація потребує одноразової оплати в 25 доларів США на момент написання бакалаврської роботи. Після реєстрації, потрібно натиснути «Create

app», після чого вписати назву мобільного додатку, мову, категорію. Після створення нового додатку, потрібно перейти у розділ Release та натиснути «Create new release». Відкриється сторінка, на якій потрібно завантажити зібраний AAB файл, вписати назву релізу, опис на різних мовах та завантажити скріншоти додатку. Після того як реліз був створений потрібно зачекати верифікацію релізу, це може займати до 1 тижня часу. Після того як реліз буде успішно верифікований, на його сторінці з'явиться надпис «Available on Google Play» (рис. 3.3).

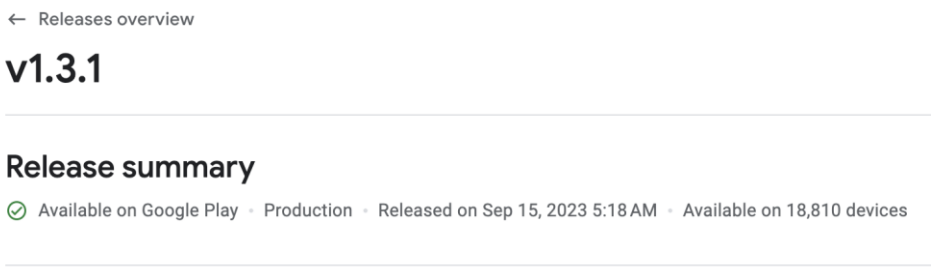

Рисунок 3.3 – Верифікований реліз

3.5 Розробка документації для програмного продукту

Документація програмного продукту є важливою складовою успішної розробки та підтримки мобільного додатку. Вона забезпечує детальну інформацію для користувачів, розробників та інших зацікавлених сторін про функціональність додатку, його налаштування та використання. Була створена користувацька документація для мобільного додатку.

Вступ. Вітаємо у мобільному додатку для обліку витрат та доходів! Цей додаток допоможе вам відстежувати ваші фінанси, управляти доходами та витратами та аналізувати фінансові звіти. Додаток підтримує необмежену кількість валют, категорій та рахунків, що робить його ідеальним інструментом для обліку фінансів.

Встановлення додатку:

– відкрийте Google Play Store на вашому мобільному пристрої;

– у рядку пошуку введіть назву додатку «Money Flow»;

– натисніть на іконку додатку в результатах пошуку;

– натисніть кнопку «Встановити»;

– після завершення встановлення натисніть «Відкрити» для запуску додатку.

Після першого запуску додатку ви попадете на головний екран з списком транзакцій.

Початкові налаштування:

– створіть вашу першу валюту (наприклад, «USD», «UAH»);

– створіть ваш перший рахунок (наприклад, «Гаманець», «Банк» тощо);

– створіть базові категорії витрат та доходів (наприклад, «Їжа», «Зарплата»).

Використання основних функцій:

1) створення транзакцій:

– на головному екрані натисніть кнопку «+» для створення транзакції;

– виберіть тип транзакції: витрата, дохід або переказ між рахунками;

– введіть назву, суму, виберіть категорію, рахунок та дату;

– натисніть «Save»;

2) управління рахунками:

– перейдіть до розділу «Balances» у меню;

– виберіть підрозділ «Accounts»;

– натисніть «+» для добавлення рахунку, введіть назву рахунку, виберіть валюту, початковий баланс, іконку та колір;

– для редагування або видалення рахунку натисніть на відповідний рахунок і виберіть потрібну дію;

3) управління валютами:

– перейдіть до розділу «Balances» у меню;

– виберіть підрозділ «Currencies»;

– натисніть «+» для добавлення валюти, введіть назву валюти, кількість знаків після коми та колір;

– для редагування або видалення валюти натисніть на відповідну валюту і виберіть потрібну дію;

4) управління категоріями витрат та доходів:

– перейдіть до розділу «Settings» у меню;

– натисніть «Categories», щоб попасти на сторінку з категоріями, виберіть тип категорії: витрат чи доходів;

– натисніть «+» для добавлення категорії, введіть назву та натисніть «Save»;

– для створення підкатегорії виберіть основну категорію, натисніть «+» і введіть її назву;

– для редагування або видалення категорій натисніть на відповідну категорію і виберіть потрібну дію;

5) пошук транзакцій:

– перейдіть до розділу «Transactions» у меню;

– використовуйте поле пошуку для введення пошукового запиту, транзакції будуть шукатися за назвою;

6) фільтрація транзакцій:

– перейдіть до розділу «Transactions» у меню;

– натисніть кнопку з іконкою фільтра у правому верхньому куті екрану;

– виберіть критерії фільтрації: дата, категорії, рахунки, валюти;

– натисніть «Apply» для відображення відфільтрованих транзакцій;

7) перегляд статистики:

– перейдіть до розділу «Statistics» у меню;

– виберіть потрібний період (день, тиждень, місяць, рік), валюти, рахунки, категорії;

– перегляньте графіки та діаграми, що відображають статистику витрат та доходів за вказаний період;

8) створення резервної копії:

– перейдіть до розділу «Settings» у меню;

– натисніть «Export backup» та виберіть місце для збереження.

Ця користувацька документація покликана допомогти вам максимально ефективно використовувати всі можливості мобільного додатку для обліку витрат та доходів. Сподіваємося, що додаток стане корисним інструментом у вашому фінансовому житті.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи був розроблений мобільний додаток для обліку витрат та доходів.

Була розроблена база даних, використовуючи плагін Capacitor Preferences, що дозволяє зберігати транзакції, рахунки, категорії та валюти.

Додаток був протестований та його функціонал відповідає поставленим вимогам, були написані unit тести, які дозволяють проводити автоматизоване тестування.

Описаний процес створення інсталяційного пакету для додатку та процедура його публікації у Play Store, що є важливим кроком для доступності додатку широкій аудиторії користувачів. Була розроблена користувацька документація, яка пояснює користувачеві як користуватися додатком.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи бакалавра було розроблено Android-додаток «Money Flow» з використанням Capacitor.js, React.js та TypeScript для обліку витрат та доходів користувачів. Було виконано наступні поставлені завдання:

– здійснено детальний аналіз актуальних проблем в управлінні фінансовими ресурсами, зокрема труднощів, з якими стикаються користувачі під час відстеження витрат та доходів;

– розглянуто та проаналізовано існуючі мобільні додатки для обліку фінансів Ivy Wallet та Money Manager, визначено їх сильні та слабкі сторони, що дозволило виявити можливості для вдосконалення;

– були сформульовані функціональні та нефункціональні вимоги до розроблюваного додатку;

– досліджено новітні підходи та методи у розробці мобільних додатків, що дало змогу вибрати найефективніші технології для реалізації проєкту, Були проаналізовані такі технології для розробки мобільних додатків як: Kotlin, Swift, React Native, Flutter, Ionic, Apache Cordova, Capacitor.js, React.js та TypeScript;

– обрано та обґрунтовано використання технологій Capacitor.js, React.js та TypeScript, які забезпечують високу продуктивність та зручність у розробці мобільних додатків;

– розроблено дизайн та на основі нього реалізовано мобільний додаток для обліку витрат та доходів, що відповідає поставленим функціональним та нефункціональним вимогам;

– проведено unit тестування розробленого додатка, яке підтвердило його функціональність, надійність та відповідність заявленим вимогам.

Таким чином, дана робота спрямована на вирішення проблеми ефективного управління фінансовими ресурсами за допомогою сучасних технологій мобільної розробки. Створений додаток надає користувачам

простий та ефективний інструмент для контролю витрат та доходів, що сприяє покращення їхньої фінансової грамотності та допомагає у досягненні фінансових цілей. Результати роботи можуть бути використані як основа для подальшого розвитку та вдосконалення мобільних рішень у сфері фінансового менеджменту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Учасники проєктів Вікімедіа. Об’єктно-орієнтоване моделювання. Вікіпедія. URL: https://uk.wikipedia.org/wiki/Об’єктно-орієнтоване _моделювання (дата звернення: 12.05.2024).

2. Що таке специфікація вимог: визначення, найкращі інструменти та методи. Visure Solutions. URL: https://visuresolutions.com/uk/blog/requirements-specification/ (дата звернення: 12.05.2024).

3. Banwar A. List virtualization in React. Medium. URL: https://medium.com/@atulbanwar/list-virtualization-in-react-3db491346af4 (дата звернення: 23.05.2024).

4. Capacitor – cross-platform native runtime for web apps | capacitor documentation. Capacitor by Ionic – Cross-platform apps with web technology. URL: https://capacitorjs.com/docs (дата звернення: 17.05.2024).

5. Costa L. D. Testing JavaScript Applications. Manning Publications Co. LLC, 2021. 511 с.

6. Documentation – apache cordova. Apache Cordova. URL: https://cordova.apache.org/docs/en/latest/ (дата звернення: 17.05.2024).

7. Flutter documentation. Docs | Flutter. URL: https://docs.flutter.dev/ (дата звернення: 17.05.2024).

8. Ivy wallet – apps on google play. Android Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.ivy.wallet (дата звернення: 27.04.2024).

9. Kotlin docs. Kotlin Help. URL: https://kotlinlang.org/docs/home.html (дата звернення: 17.05.2024).

10. Money manager: budget & expense – apps on google play. Android Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.freeman.moneymanager&pli=1 (дата звернення: 27.04.2024).

11. React Native – official documentation. React Native. URL: https://reactnative.dev/ (дата звернення: 17.05.2024).

12. Vanderkam D. Effective typescript: 62 specific ways to improve your typescript. O'Reilly Media, Incorporated, 2019. 250 с.