«ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ»
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп'ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____Олена ОЛЬХОВСЬКА
(підпис)

«_____»___202_ р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОБІЛЬНИХ ПЛАТФОРМ»

зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавра

Виконавець роботи Карабаш Вадим Євгенович
_____«____» _____202_ р.
(підпис)

Науковий керівник к. ф.-м. н., доцент, Черненко Оксана Олексіївна
_____«____»_____202_ р.
(підпис)

Рецензент

ПОЛТАВА 2024

РЕФЕРАТ

Записка: 47 с., 20 рис., 0 таблиць, 1 додаток, 14 джерел.

МОБІЛЬНИЙ ДОДАТОК, УПРАВЛІННЯ ФІНАНСАМИ, REACT NATIVE

Об’єкт розробки – процес управління особистими фінансами через мобільний додаток.

Мета роботи – розробка мобільного додатку для відстеження фінансів, який дозволяє користувачам керувати своїми витратами та доходами через інтуїтивно зрозумілий інтерфейс на основі кроссплатформної технології React Native.

Методи дослідження – методи програмування на мові JavaScript з використанням бібліотеки React Native, веб-сервіси Firebase для аутентифікації та збереження даних, а також використання Redux для управління станами додатку.

Розроблено мобільний додаток, який дозволяє користувачам реєструватися, авторизуватися та вести облік своїх фінансових операцій, з можливістю перегляду історії витрат за останні 7 днів. Крім того, додаток дозволяє здійснювати сортування витрат за датою та категорією, а також забезпечує валідацію введених даних і вивід користувацьких помилок.

Було проведено огляд існуючих рішень на ринку мобільних фінансових додатків, аналіз їх функціоналу та інтерфейсів. На основі аналізу визначено ключові вимоги до функціональності та дизайну власного додатку. Відбулося проектування архітектури додатку, його реалізація, тестування та оптимізація.

Результати роботи включають розроблену програму з документацією, що містить інструкції для користувачів та рекомендації щодо подальшого розвитку додатку. Висвітлено позитивні та негативні сторони використання кроссплатформних технологій у контексті розробки мобільних додатків.

Зміст

ВСТУП

З розвитком цифрових технологій відкрилися нові можливості для управління особистими фінансами. Сучасні мобільні додатки надають користувачам інструменти для моніторингу та аналізу їхніх витрат і доходів, сприяючи кращому фінансовому плануванню. Однак, не всі додатки забезпечують зручний, інтуїтивно зрозумілий інтерфейс та достатній рівень безпеки і персоналізації, які є критично важливими для користувачів.

У цьому контексті актуальним є створення нових, більш ефективних і користувацьки-орієнтованих мобільних рішень для управління фінансами. Це спонукає до розробки додатків, які б не лише відстежували транзакції, а й надавали корисні аналітичні інструменти та можливості кастомізації згідно з особистими потребами користувача.

Об’єктом розробки є процес управління особистими фінансами за допомогою мобільного додатку.

Предметом розробки є мобільний додаток для відстеження фінансів, створений на кроссплатформній основі React Native, що включає інтеграцію з Firebase для аутентифікації користувачів та зберігання даних.

Метою роботи є розробка мобільного додатку, який забезпечує зручний інтерфейс для відстеження фінансових транзакцій, їх категоризацію, аналіз витрат, а також високий рівень персоналізації та безпеки користувацьких даних.

Методи дослідження включають застосування технологій кроссплатформної розробки, роботу з базами даних і системами аутентифікації, а також впровадження функціональності за допомогою сучасних інструментів розробки програмного

забезпечення, таких як Redux для управління станом додатку, TailwindCSS для стилізації компонентів і TypeScript для підвищення надійності коду.

Структура роботи:

1. Вступ, де визначено актуальність, мету, об’єкт та предмет дослідження.
2. Опис теоретичних основ роботи, з акцентом на сучасні методи та технології розробки мобільних додатків.
3. Практична частина, що включає аналіз вимог, проектування та реалізацію додатку.
4. Висновки, де підсумовано результати роботи та оцінено досягнення поставлених цілей.
5. Додатки з кодом, ілюстраціями інтерфейсів та інші важливі матеріали проекту.

У роботі було використано кроссплатформну технологію React Native, що дозволило створити гнучке і ефективне рішення, яке може бути використане на різних типах мобільних пристроїв без потреби у розробці окремих версій для кожної платформи. Firebase використовувався для реалізації функцій аутентифікації та зберігання даних, що забезпечує швидкість та безпеку обробки персональної інформації користувачів.

1. ПОСТАНОВКА ЗАДАЧІ

Для розробки кваліфікаційної роботи на тему "Розробка програмного забезпечення для мобільних платформ" задача полягає у створенні мобільного додатку на базі React Native, призначеного для ефективного відстеження та управління особистими фінансами користувачів. План роботи такий:

- **Дослідити існуючі рішення** для управління фінансами, щоб визначити ключові функції та найкращі практики, які можна адаптувати або вдосконалити у власному додатку.
- **Аналізувати потреби користувачів** для забезпечення зручності та ефективності мобільного додатку, що дозволить користувачам легко вводити, переглядати та аналізувати свої фінансові дані.
- **Розробити детальний алгоритм функціонування додатку**, включаючи реєстрацію та аутентифікацію користувачів через Firebase, введення та зберігання даних про фінансові транзакції, їх категоризацію, а також функції фільтрації та сортування.

Додаток повинен надавати користувачам наступні можливості:

- **Стартова сторінка** з логіном та реєстрацією.
- **Головний інтерфейс** з оглядом недавніх та запланованих транзакцій, витрат за останні 7 днів, а також доступом до повної історії фінансових операцій.
- **Форми для вводу нових транзакцій** з автоматичною валідацією введених даних та можливістю вибору категорій витрат.
- **Функції сортування та фільтрації** даних за датами та категоріями, що дозволять користувачам ефективно управляти своїми фінансовими звітами.
- **Заходи безпеки** для захисту персональних і фінансових даних користувачів, включаючи шифрування даних та безпечну аутентифікацію.
- **Інструкції для користувачів** щодо використання додатку, включно з помилками, які можуть виникнути, та методами їх усунення.
- **Можливість завершення сесії роботи** в додатку у будь-який момент.

2. ІНФОРМАЦІЙНИЙ ОГЛЯД

2.1 Платформи для розробки мобільних додатків

У процесі розробки мобільного додатку для відстеження фінансів важливо розглянути різні платформи, які дозволяють створювати ефективні та функціональні додатки. Ключовим аспектом є вибір між нативною та кроссплатформеною розробкою. Кожен з цих підходів має свої переваги та недоліки, які слід враховувати залежно від цілей проекту.

Нативна розробка передбачає створення додатку спеціально для кожної операційної системи, такої як iOS або Android. Цей підхід забезпечує високу продуктивність, оптимальну сумісність з обладнанням та можливість доступу до всіх нативних API та функцій системи. Нативні додатки для iOS зазвичай розробляються в Xcode за допомогою Swift або Objective-C, тоді як для Android використовується Android Studio з Java або Kotlin [2], [3].

З іншого боку, **кроссплатформна розробка** дозволяє використовувати одну кодову базу для створення додатків, які працюють на декількох операційних системах. Це знижує час та витрати на розробку, але може мати певні компроміси в плані продуктивності та доступу до нативних функцій.

React Native є однією з найпопулярніших кроссплатформних технологій. Розроблена Facebook, ця технологія дозволяє розробникам створювати мобільні додатки за допомогою JavaScript та React, які взаємодіють із нативними компонентами операційної системи через спеціальний міст (див рис. 2.1). Це забезпечує гарну продуктивність та майже нативний користувацький досвід [1].

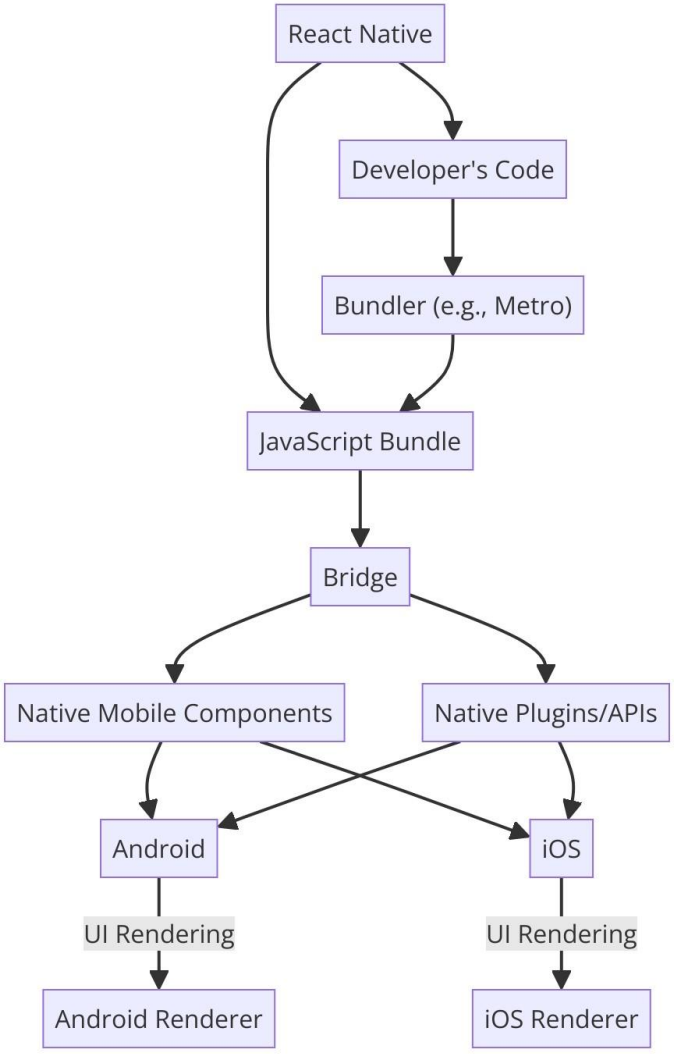

Рисунок 2.1 – Як працює React Native

Flutter, розроблений Google, є ще однією популярною кроссплатформною технологією. Він використовує мову Dart і компілює код в нативний машинний код, що забезпечує високу продуктивність додатків (див рис. 2.2). Flutter пропонує широкий спектр вбудованих віджетів та високий рівень налаштування інтерфейсу [4].

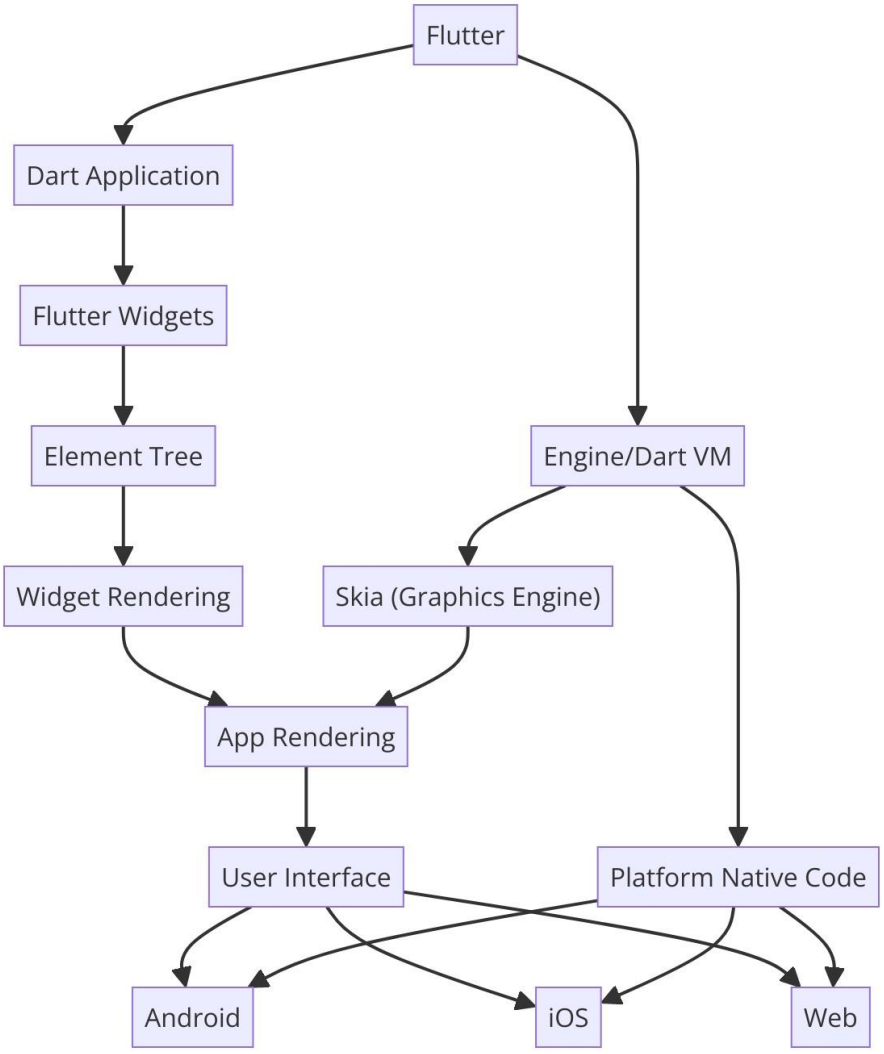

Рисунок 2.2 – Як працює Flutter

Вибір між цими підходами залежить від багатьох факторів, включаючи специфічні вимоги проекту, ресурси команди, час на ринок та очікувані характеристики додатку. В контексті розробки мобільного додатку для відстеження фінансів, React Native було обрано за його здатність швидко створювати стабільні, ефективні та доступні за вартістю додатки, які забезпечують високий рівень користувацького досвіду на обох платформах – iOS та Android [1].

2.2 Огляд інструментів і технологій для розробки на React Native

React Native є потужною кроссплатформною технологією, яка дозволяє розробникам створювати мобільні додатки, використовуючи JavaScript та реактивну

парадигму з фреймворку React. Ця технологія сприяє швидкій розробці додатків, які ефективно працюють як на Android, так і на iOS. Огляд основних інструментів і технологій, які використовуються в розробці на React Native, включає наступне:

1. **Expo CLI**: Expo є набором інструментів та сервісів, що побудовані навколо React Native, який дозволяє розробникам швидко розпочати роботу без необхідності взаємодії з нативним кодом. Expo надає безшовне середовище розробки з доступом до обширної бібліотеки API, яка включає камеру, геолокацію, нотифікації та багато іншого.
    - **Приклад**: Використання Expo дозволяє швидко створити додаток з інтегрованою підтримкою push-сповіщень та доступом до камери, що робить його ідеальним для розробки прототипів та додатків з базовою функціональністю. Однак для додатків, які вимагають глибокої інтеграції з нативними можливостями апаратного забезпечення, може знадобитися вийти за межі можливостей Expo та використати нативний код [4].
2. **React Navigation**: Для навігації в додатках React Native часто використовується React Navigation, який надає гнучкі рішення для руху між різними екранами. Він підтримує стекову, вкладену, табуляційну та висувну навігацію, що дозволяє створювати комплексні користувацькі інтерфейси, які легко масштабуються та адаптуються до різних потреб користувачів.
    - **Приклад**: Використання React Navigation у додатку для електронної комерції дозволяє легко переходити між головною сторінкою, сторінкою товарів, кошиком та сторінкою оплати, забезпечуючи при цьому плавний користувацький досвід.
3. **Redux та Redux Toolkit**: Redux є популярною бібліотекою управління станом для JavaScript-додатків, що дозволяє управляти станом аплікації в передбачуваний спосіб. Redux Toolkit подальше спрощує конфігурацію та використання Redux, забезпечуючи набори інструментів, які допомагають зменшити кількість коду, який потрібно написати та підтримувати [1].
    - **Приклад**: У додатку для відстеження фінансів використання Redux

забезпечує централізоване управління станом користувацьких даних, таких як транзакції, баланси рахунків та бюджети, що спрощує розробку та підтримку додатку.

4. **Styled Components та TailwindCSS**: Для стилізації компонентів в React Native, Styled Components дозволяють використовувати звичні CSS-подібні стилі у JavaScript, що значно спрощує процес стилізації. TailwindCSS, з іншого боку, може бути інтегрований через бібліотеки третіх сторін для використання утилітарних класів стилізації безпосередньо в компонентах React Native [5].
    - **Приклад**: Використання Styled Components у додатку для новин дозволяє розробникам швидко створювати стильні та консистентні інтерфейси, тоді як TailwindCSS може бути використаний для швидкого прототипування та створення адаптивних макетів.
5. **Firebase**: Firebase від Google надає широкий спектр бекенд-сервісів, таких як аутентифікація, бази даних, аналітика, файлоховище, які легко інтегруються з React Native. Це дозволяє розробникам швидко створювати масштабовані додатки без необхідності власної розробки складних серверних рішень [6].
    - **Приклад**: У додатку для соціальних мереж Firebase використовується для аутентифікації користувачів, зберігання їх фотографій та постів, а також для відстеження активності користувачів у реальному часі.

Ці інструменти та технології забезпечують міцну основу для розробки мобільних додатків на React Native, дозволяючи розробникам максимізувати продуктивність і забезпечити відмінний користувацький досвід. Використання таких інструментів допоможе оптимізувати розробку додатку для відстеження фінансів, забезпечуючи його ефективність, надійність та доступність для користувачів на різних платформах.

2.3 Приклади аналогічних фінансових додатків

Для розробки ефективного мобільного додатку для відстеження фінансів

важливо аналізувати існуючі рішення на ринку. Це допомагає зрозуміти які функції є популярними серед користувачів, які проблеми можуть виникати при їх використанні, та які інноваційні підходи можна застосувати у власному продукті. Нижче представлено кілька популярних фінансових додатків, які використовуються у всьому світі.

Mint: Mint є одним з найвідоміших додатків для управління особистими фінансами. Він дозволяє користувачам відстежувати свої витрати, створювати бюджети та моніторити стан своїх рахунків з одного інтерфейсу. Mint автоматично категоризує транзакції та надає детальні звіти, що допомагає користувачам краще розуміти свої фінансові звички.

YNAB (You Need A Budget): YNAB фокусується на методології бюджетування, яка допомагає користувачам досягати своїх фінансових цілей. Додаток пропонує детальні інструкції та підтримку для планування фінансів, спрямовану на зменшення боргів і збільшення заощаджень. YNAB також інтегрує навчальні матеріали для вдосконалення фінансової грамотності користувачів.

Personal Capital: Personal Capital комбінує елементи управління фінансами з інвестиційним консультуванням. Він надає користувачам інструменти для моніторингу активів та пасивів, аналізу інвестиційної продуктивності та планування на пенсію. Цей додаток ідеально підходить для осіб, які хочуть більш глибокого інтегрованого підходу до управління своїми фінансами.

PocketGuard: PocketGuard допомагає користувачам контролювати свої витрати і уникати перевитрат. Додаток автоматично виявляє повторювані платежі та пропонує стратегії для оптимізації витрат. Його інтерфейс простий та інтуїтивно зрозумілий, що робить його доступним навіть для початківців у фінансовому плануванні.

Аналіз цих додатків дозволяє ідентифікувати кілька ключових функціональних можливостей та інновацій, які можуть бути важливими для розробки власного мобільного додатку:

- **Автоматична категоризація транзакцій** з метою спрощення бюджетування та моніторингу витрат.

- **Генерація персоналізованих звітів** для кращого розуміння фінансового стану.
- **Прогнозування фінансових трендів** на основі історичних даних, що допомагає користувачам планувати майбутні витрати.
- **Інтеграція з банківськими рахунками** для надання актуальної інформації про стан коштів.

Ці особливості можуть стати основою для розробки додатка, який не тільки задовольнить потреби користувачів в управлінні їх фінансами, але й надасть додаткову цінність через зручність і інтуїтивність використання.

3. ТЕОРЕТИЧНА ЧАСТИНА

3.1 Принципи розробки мобільних додатків

Розробка мобільних додатків вимагає дотримання певних принципів, які забезпечують створення якісних, функціональних та зручних у використанні продуктів. Нижче наведено основні принципи розробки мобільних додатків разом із прикладами їх реалізації.

1. **Простота та зручність користування**: Один із ключових принципів — забезпечення інтуїтивно зрозумілого інтерфейсу користувача. Додаток має бути простим у використанні, з чіткою та логічною структурою.
    - **Приклад**: Додаток Google Keep має мінімалістичний дизайн, що дозволяє користувачам швидко створювати та організовувати нотатки без зайвих кроків або складних налаштувань. Використання простих іконок та кольорових тегів допомагає легко орієнтуватися в додатку.
2. **Висока продуктивність та швидкість роботи**: Мобільний додаток повинен працювати швидко та безперебійно, навіть на пристроях із обмеженими ресурсами.
    - **Приклад**: Додаток Instagram оптимізовано для швидкого завантаження зображень та відео. Це досягається за рахунок використання ефективних алгоритмів стиснення та асинхронного завантаження контенту, що забезпечує плавний досвід користування.
3. **Безпека даних**: Забезпечення захисту персональних даних користувачів є критично важливим аспектом розробки мобільних додатків. Це включає використання шифрування для передачі даних, захищене зберігання облікових даних та реалізацію надійних механізмів аутентифікації.
    - **Приклад**: Банківський додаток Monzo використовує двофакторну аутентифікацію та шифрування для захисту фінансової інформації користувачів. Крім того, додаток регулярно оновлюється для забезпечення захисту від нових загроз.
4. **Кроссплатформність**: Багато мобільних додатків розробляються для роботи

на різних платформах, таких як iOS та Android. Використання кроссплатформних фреймворків дозволяє зменшити витрати на розробку та забезпечити однаковий користувацький досвід на різних пристроях.

- **Приклад**: Додаток Airbnb був створений за допомогою React Native, що дозволило забезпечити однаковий інтерфейс та функціональність як на iOS, так і на Android, скоротивши час розробки та знизивши витрати.

5. **Модульність та масштабованість**: Розробка додатків повинна враховувати можливість подальшого розширення функціональності та підтримки коду.
    - **Приклад**: У додатку Spotify кожен функціональний блок, такий як управління плейлистами або пошук музики, реалізований як окремий модуль. Це полегшує його підтримку та розширення, дозволяючи додавати нові функції без значних змін у кодовій базі.
6. **Адаптивний дизайн**: Додаток повинен коректно відображатися на різних типах пристроїв з різними розмірами екранів. Адаптивний дизайн забезпечує зручність користування незалежно від пристрою, на якому запущено додаток.
    - **Приклад**: Додаток YouTube має адаптивний інтерфейс, який однаково зручно використовувати як на смартфонах, так і на планшетах та інших пристроях. Інтерфейс автоматично підлаштовується під розмір екрану, забезпечуючи оптимальний досвід користувача.
7. **Тестування та відладка**: Регулярне тестування та відладка є невід'ємною частиною процесу розробки мобільних додатків. Важливо забезпечити високу якість продукту шляхом проведення різних видів тестування, включаючи функціональне, інтеграційне та користувацьке тестування.
    - **Приклад**: Команда розробників WhatsApp регулярно проводить A/B тестування нових функцій, щоб зібрати відгуки користувачів та покращити досвід використання додатку. Крім того, автоматизоване тестування допомагає швидко виявляти та виправляти баги.
8. **Забезпечення сумісності з різними версіями операційних систем**: Додаток повинен коректно працювати на різних версіях операційних систем, щоб забезпечити максимальне покриття користувачів.

- **Приклад**: Додаток Facebook регулярно оновлюється для забезпечення сумісності з останніми версіями iOS та Android, а також для підтримки старіших версій, щоб уникнути проблем сумісності.

Дотримання цих принципів допомагає створювати мобільні додатки, які відповідають високим стандартам якості, задовольняють потреби користувачів та забезпечують надійну роботу в різних умовах. Реалізація зазначених принципів гарантує, що додаток буде зручним, швидким та безпечним, що, у свою чергу, підвищує його конкурентоспроможність на ринку.

3.2 Особливості роботи з React Native

React Native, розроблений Facebook, став одним з найпопулярніших інструментів для створення кроссплатформних мобільних додатків. Використання React Native дозволяє розробникам застосовувати той самий JavaScript-код для створення додатків як для iOS, так і для Android, що забезпечує значну економію часу та ресурсів. Розглянемо детальніше ключові особливості роботи з цією технологією:

1. **Компонентно-орієнтований підхід**: React Native використовує компонентно-орієнтований підхід, що дозволяє розробникам будувати додатки з використанням взаємозамінних та повторно використовуваних компонентів. Компоненти в React Native є аналогічними до HTML-елементів у веб-розробці, що робить їх зрозумілими та легкими у використанні. Наприклад, можна створити компонент кнопки, який буде використовуватись у різних частинах додатку без необхідності дублювання коду. Це сприяє модульності та легкості підтримки коду, оскільки зміни в одному компоненті не впливають на інші частини програми [1].
2. **Гаряче перезавантаження (Hot Reloading)**: Однією з найважливіших функцій React Native є гаряче перезавантаження, яке дозволяє розробникам миттєво бачити результати змін у коді в реальному часі. Це значно пришвидшує процес розробки, оскільки не потрібно перезапускати додаток для того, щоб побачити кожну невелику зміну. Наприклад, при зміні стилю

або поведінки компонента розробник може одразу бачити результат, що зменшує час на налагодження та тестування [2].

3. **Доступ до нативних API**: React Native дозволяє розробникам використовувати нативні API обох платформ. Це означає, що додатки можуть використовувати функціональні можливості камери, GPS, акселерометра та інших компонентів смартфону, що розширює можливості мобільних додатків. Наприклад, додаток для фітнесу може використовувати дані з GPS та акселерометра для відстеження активності користувача [3].
4. **Підтримка спільноти та бібліотек**: React Native має велику та активну спільноту розробників, яка постійно створює та підтримує велику кількість бібліотек та інструментів. Це дозволяє розробникам знаходити готові рішення для багатьох загальних завдань, забезпечуючи швидке впровадження нових функцій та підвищення надійності додатків. Наприклад, бібліотека React Navigation забезпечує просту та гнучку навігацію в додатку, а Redux дозволяє ефективно керувати станом додатку [4].
5. **Виклики з продуктивністю**: Хоча React Native забезпечує кроссплатформну сумісність та швидку розробку, він може стикатися з проблемами продуктивності на складних інтерфейсах або при інтенсивному використанні даних. Важливо використовувати профілювання продуктивності та оптимізацію, щоб забезпечити плавну роботу додатків на всіх пристроях. Наприклад, можна використовувати бібліотеку Reanimated для оптимізації анімацій або FlatList для ефективного рендерингу великих списків [5].
6. **Будівництво мостів до нативного коду**: Для деяких специфічних вимог може знадобитися писати нативний код для Android або iOS. React Native дозволяє "будувати мости" до нативного коду, що дозволяє інтегрувати високоспеціалізовану функціональність, недоступну через стандартні API. Наприклад, якщо потрібно реалізувати складний алгоритм обробки зображень, можна написати нативний модуль на Swift або Kotlin та інтегрувати його в React Native додаток [6].
7. **Платформо-залежні компоненти**: React Native дозволяє створювати

компоненти, специфічні для кожної платформи. Це означає, що розробники можуть створювати окремі реалізації для iOS та Android, якщо потрібно врахувати специфіку кожної платформи. Наприклад, компоненти можуть мати різний стиль або функціональність залежно від платформи, що дозволяє створювати більш нативний вигляд та відчуття додатків [7].

8. **Структура проекту та організація коду**: React Native пропонує гнучкість у структурі проекту та організації коду. Розробники можуть обирати між різними підходами до управління станом, маршрутизації та організації файлів. Використання таких інструментів, як Expo, може спростити початковий етап розробки, надаючи готові шаблони та інтеграції [8].

Розробка додатків з використанням React Native вимагає від розробників розуміння цих особливостей та готовності використовувати доступні інструменти та ресурси для оптимізації додатків під різні платформи і вимоги користувачів. Використання ресурсів спільноти та готових бібліотек може значно спростити процес розробки та забезпечити створення якісного продукту, орієнтованого на кінцевого користувача. Успішне використання React Native залежить від здатності розробників ефективно поєднувати нативні можливості платформ з перевагами кроссплатформного підходу, що дозволяє створювати потужні та інноваційні мобільні додатки.

3.3 Безпека даних у мобільних додаткахыф

Забезпечення безпеки даних є ключовим аспектом при розробці мобільних додатків, особливо коли йдеться про фінансові аплікації, що обробляють чутливу інформацію користувачів. Розуміння та застосування принципів безпеки даних може значно знизити ризики витоку даних, несанкціонованого доступу та інших потенційних загроз. Нижче представлені основні принципи та методи забезпечення безпеки даних у мобільних додатках:

1. **Шифрування даних**: Шифрування є основою безпеки даних. Всі чутливі дані, зберігані на пристрої або передавані через мережу, повинні бути

зашифровані. Це стосується не тільки фінансової інформації, але й особистих даних, паролів та іншої інформації, яка може бути використана для ідентифікації користувача. Використання сучасних алгоритмів шифрування, таких як AES (Advanced Encryption Standard), забезпечує високий рівень безпеки. Наприклад, при передачі даних між клієнтським додатком та сервером використовується TLS (Transport Layer Security), який забезпечує захищений канал зв'язку [1][3].

2. **Безпечне зберігання токенів та креденшіалів**: Токени аутентифікації, секретні ключі та інші креденшіали необхідно зберігати у безпечному сховищі. Для цього можна використовувати спеціалізовані бібліотеки, такі як Keychain для iOS та Keystore для Android, які забезпечують захищене зберігання чутливої інформації на пристрої. Наприклад, Keychain дозволяє зберігати паролі, сертифікати та інші облікові дані в захищеному вигляді, що значно знижує ризик їх компрометації [2].
3. **Захист API та серверної частини**: Забезпечення безпеки серверної частини та API є критично важливим для захисту даних, які передаються між мобільним додатком та сервером. Використання HTTPS для шифрування даних, переданих по мережі, та ретельне управління доступом до API допомагає запобігти несанкціонованому доступу та міжсайтовому скриптингу (XSS), міжсайтовій підробці запитів (CSRF) та іншим загрозам. Наприклад, обмеження доступу до API через використання OAuth 2.0 та JWT (JSON Web Tokens) допомагає забезпечити аутентифікацію та авторизацію користувачів [4].
4. **Регулярні оновлення та патчі**: Забезпечення актуальності програмного забезпечення є важливим для захисту від відомих вразливостей. Розробники повинні регулярно оновлювати мобільні додатки та їх компоненти, а також випускати патчі для виправлення будь-яких виявлених вразливостей. Наприклад, використання автоматичних оновлень та сповіщень про нові версії бібліотек допомагає забезпечити своєчасну реакцію на нові загрози [5].
5. **Реалізація принципу найменшого привілею**: Кожен компонент додатка

повинен мати доступ лише до тих ресурсів, які необхідні для його роботи. Цей принцип допомагає зменшити потенційні шкоди від можливих атак, обмежуючи можливості зловмисників при отриманні доступу до системи. Наприклад, якщо додаток використовує базу даних, то кожен запит повинен мати доступ лише до тих даних, які необхідні для виконання конкретної операції, і не більше [6].

6. **Аутентифікація та авторизація**: Використання надійних методів аутентифікації та авторизації є важливим аспектом безпеки даних. Наприклад, двофакторна аутентифікація (2FA) додає додатковий рівень захисту, вимагаючи від користувачів введення додаткового коду, відправленого на їхній мобільний телефон або електронну пошту. Також важливо використовувати надійні паролі та регулярно оновлювати їх, щоб запобігти несанкціонованому доступу [7].
7. **Моніторинг та аудит**: Постійний моніторинг та аудит безпеки додатків дозволяє виявляти та реагувати на потенційні загрози у реальному часі. Використання інструментів для моніторингу трафіку, а також журналювання дій користувачів допомагає відстежувати підозрілі активності та оперативно реагувати на них. Наприклад, інструменти для виявлення аномалій можуть виявляти несанкціоновані спроби доступу до системи та сповіщати адміністраторів про потенційні загрози [8].

Застосування цих методів забезпечує комплексний підхід до захисту даних у мобільних додатках, мінімізує ризики та підвищує довіру користувачів до аплікації. Важливо враховувати ці аспекти на всіх етапах розробки та експлуатації мобільного додатку, щоб забезпечити максимальний захист даних.

4. ПРАКТИЧНА ЧАСТИНА

4.1 Опис розробки програмного забезпечення

Аналіз вимог

На початковому етапі розробки додатку для відстеження витрат ключовим завданням було визначення та аналіз вимог (див рис. 4.1). Цей процес включав збір інформації про потреби користувачів та визначення функціональних вимог до додатку. Основні вимоги до додатку включали:

- **Відстеження витрат**: можливість користувачів вводити, переглядати та категоризувати свої витрати.
- **Управління бюджетом**: інструменти для планування бюджету на основі попередніх витрат.
- **Повідомлення про мінімальний бюджет**: система сповіщень, яка попереджає користувачів, коли вони наближаються до ліміту їхнього бюджету.

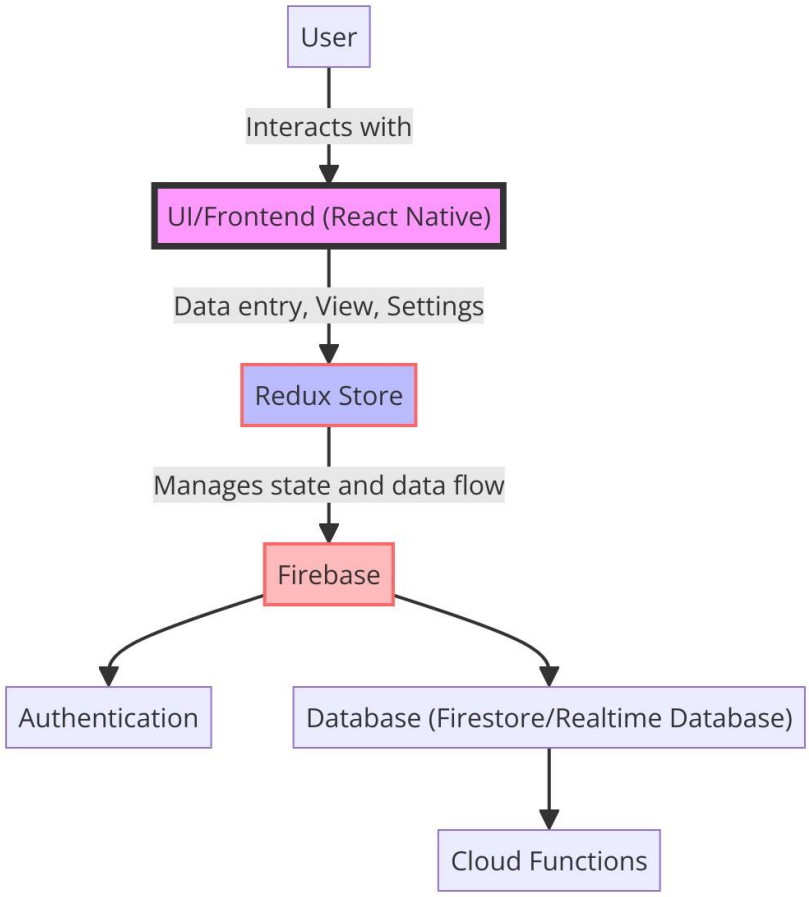

Рисунок 4.1 – Основні функції та потоки даних

Проектування

Процес проектування додатку охоплював розробку архітектури додатку, включаючи фронтенд та бекенд, а також вибір технологій (див рис. 4.2). Основні аспекти проектування включали:

- **Фронтенд (React Native)**: Використання React Native для створення гнучкого та реактивного інтерфейсу, який забезпечує однаково високу якість взаємодії на iOS та Android.
- **Бекенд (Firebase)**: Інтеграція з Firebase для аутентифікації користувачів, зберігання даних і взаємодії в реальному часі.
- **Управління станом (Redux)**: Використання Redux для управління станом додатку, забезпечуючи консистентність даних і легке управління станом через різні компоненти.

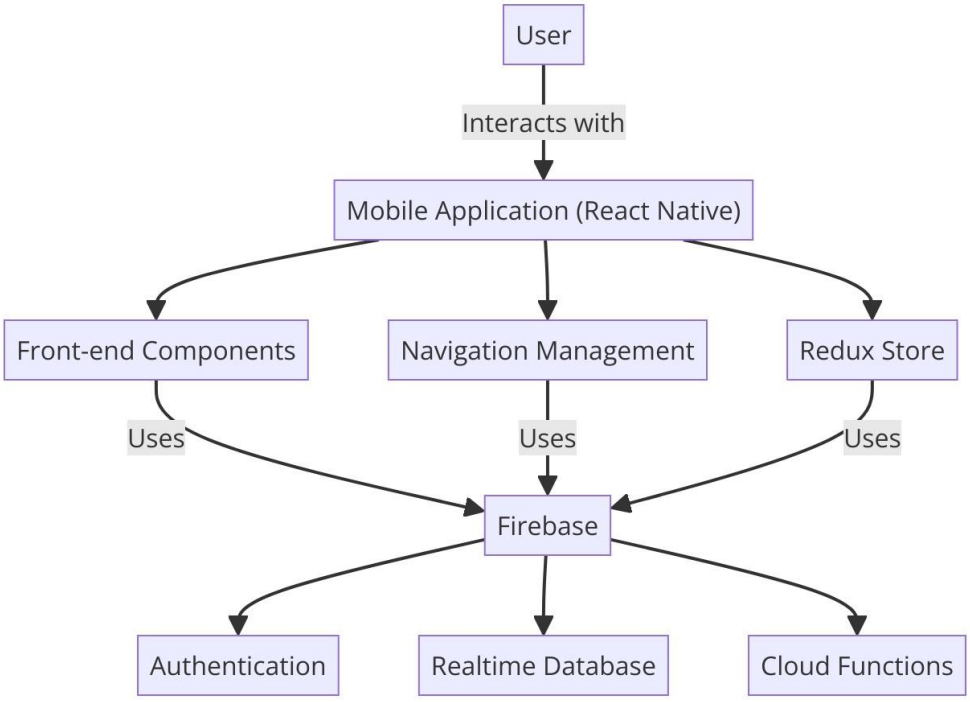

Рисунок 4.2 – Діаграма архітектури

Реалізація

Реалізація програмного забезпечення була розділена на дві основні частини: розробка фронтенду та бекенду:

- **Фронтенд**: Розробка користувальницького інтерфейсу виконувалася за допомогою компонентно-орієнтованого підходу React Native. Реалізація включала створення екранів для введення витрат, перегляду витрат за категоріями, налаштувань користувача та звітів.

```
const ExpenseForm = ({ onCancel, onConfirm, expense }: Props) => {
  const defaultVal = expense
    ? { ...expense, amount: `${expense.amount}` }
    : {
        amount: "",
        date: new Date().toISOString().slice(0, 10),
        desc: "",
      };
  const {
    control,
    handleSubmit,
    formState: { errors },
  } = useForm<ExpenseSchema>({
    resolver: zodResolver(expenseSchema),
    defaultValues: defaultVal,
  });

  const onSubmit: SubmitHandler<ExpenseSchema> = data => {
    const finalExpense = {
      ...data,
      amount: +data.amount,
    };
    onConfirm(finalExpense);
```

```
};
const isEditing = !!expense;

return (
  <View style={tw`mt-0`}>
    {/* <Text style={tw`text-2xl font-bold text-center text-white my-6`}>
      Ваша витрата
    </Text> */}
    <View style={tw`flex-row justify-between`}>
      <Controller
        control={control}
        name="amount"
        render={({ field: { onChange, onBlur, value } }) => (
          <Input
            label="Кількість"
            containerStyle="flex-1"
            isValid={!errors.amount}
            textInputConfig={{
              keyboardType: "decimal-pad",
              placeholder: "",
              autoFocus: true,
              onChangeText: onChange,
              onBlur,
              value: value.toString(),
            }}
          />
        )}
      />

      <Controller
```

```
        control={control}
        name="date"
        render={({ field: { onChange, onBlur, value } }) => (
          <Input
            label="Дата"
            containerStyle="flex-1"
            isValid={!errors.date}
            textInputConfig={{
              placeholder: "YYYY-MM-DD",
              keyboardType: "number-pad",
              maxLength: 10,
              onChangeText: onChange,
              onBlur,
              value,
            }}
          />
        )}
      />
    </View>

    <Controller
      control={control}
      name="desc"
      render={({ field: { onChange, onBlur, value } }) => (
        <Input
          label="Опис"
          isValid={!errors.desc}
          textInputConfig={{
            multiline: true,
            autoCorrect: false,
```

```
            autoCapitalize: "sentences",
            onChangeText: onChange,
            onBlur,
            value,
          }}
        />
      )}
    />
    <View style={tw`gap-1 items-center mb-2`}>
      {Object.entries(errors).map(([key, val]) => (
        <Text
          key={Math.random()}
          style={tw`text-error-500 font-semibold`}
        >
          {val.message}
        </Text>
      ))}
    </View>

    <View style={tw`flex-row justify-center items-center`}>
      <Button
        variant="flat"
        outerViewStyle={tw`min-w-[120px] mx-2`}
        onPress={onCancel}
      >
        Скасувати
      </Button>
      <Button
        variant="default"
        outerViewStyle={tw`min-w-[120px] mx-2`}
```

```
            onPress={handleSubmit(onSubmit)}
          >
            {isEditing ? "Оновити" : "Додати"}
          </Button>
        </View>
      </View>
  );
};
```

- **Бекенд**: Інтеграція з Firebase обслуговувала авторизацію, зберігання даних витрат та синхронізацію між різними пристроями. Розробка серверних функцій на Firebase Functions дозволила обробляти взаємодію з даними ефективно і безпечно.

```
const RecentExpensesScreen = () => {
  const user = useAppSelector(selectUser);
  if (!(user.id || user.token)) return <LoadingSpinner />;

  const { data, isLoading } = useGetExpensesQuery({
    token: user.token,
    userId: user.id,
  });

  const [recentExpenses, setRecentExpenses] = useState<Expense[]>([]);

  useEffect(() => {
    if (!data) return;

    setRecentExpenses(
      firebaseExpensesToExpenses(data).filter(expense => {
        const today = new Date();
        const date7DaysAgo = getDateMinusDays(today, 7);
```

```
            return new Date(expense.date) > date7DaysAgo;
        }),
      );
    }, [data]);

    if (isLoading) return <LoadingSpinner text="Завантаження ваших витрат" />;

    return (
      <ExpensesOutput
        expenses={recentExpenses}
        expensesPeriod="Останні 7 днів"
      />
    );
  };
```

Ці етапи забезпечили основу для наступних кроків тестування та розгортання додатку, що дозволило забезпечити його функціональність та надійність.

4.2 Опис програми

Огляд компонентів системи

Цей розділ розширює опис ключових компонентів системи, деталізуючи їх відповідальності та взаємодію з користувачем і данними.

1. Авторизація та реєстрація користувачів:

Ці компоненти відіграють критичну роль у забезпеченні безпеки і персоналізації досвіду користувачів. Авторизація забезпечує входження у систему з використанням Firebase, яка підтримує аутентифікацію через електронну пошту та пароль, а також може інтегрувати інші методи, такі як вход через соціальні мережі. Реєстрація дозволяє новим користувачам створювати обліковий запис, введення даних про який синхронізується та зберігається в базі даних Firebase для забезпечення цілісності сесій та доступу.

2. Обробка та зберігання витрат:

Ключові компоненти системи для роботи з витратами дозволяють користувачам вносити деталі про свої фінансові операції, які потім категоризуються та аналізуються. React Native сприяє створенню інтуїтивно зрозумілих форм введення, що забезпечує ефективний збір інформації, в той час як Firebase займається її асинхронним зберіганням та обробкою. Це включає реалізацію логіки для перегляду історії витрат, сортування за категоріями та агрегування даних.

Інтеграція та архітектурні вибори:

Система побудована на модульному принципі, де кожен компонент ізольовано відповідає за певний аспект функціональності:

- **Front-end**: Розроблено на базі React Native, що гарантує гладке та швидке реагування користувальницького інтерфейсу на взаємодії користувачів. Завдяки гнучкому управлінню станами, інтерфейс легко адаптується під різні завдання та операції користувачів.
- **Back-end**: Використання Firebase як хмарного рішення для всіх серверних операцій, таких як аутентифікація, зберігання даних та обробка аналітики, звільняє команду розробників від необхідності обслуговування власного серверного обладнання. Це сприяє швидкому масштабуванню і легкому оновленню системи.
- **Управління станом**: Використання Redux як ключового інструменту для централізованого управління станом додатку. Це не тільки поліпшує організацію і взаємодію між компонентами, але і значно підвищує продуктивність додатку за рахунок оптимального розподілу ресурсів і асинхронної взаємодії з сервером.

Ця багаторівнева архітектура забезпечує не тільки високу продуктивність і масштабованість, але й легкість у впровадженні нових функцій, що є важливим аспектом для адаптації додатку до змінних потреб користувачів та ринкових умов.

4.3 Необхідна користувачу інструкція програми

Інструкція користувача допоможе вам ефективно використовувати функціональність додатку для відстеження витрат. Нижче наведено кроки, які вам слід виконати для успішного користування додатком.

Крок 1: Реєстрація

1. Відкрийте додаток на своєму пристрої.
2. На стартовому екрані виберіть "Зареєструватися".
3. Введіть вашу електронну пошту, пароль та підтвердіть пароль у відповідні поля (див рис. 4.3).
4. Натисніть кнопку "Зареєструватися" для створення вашого нового облікового запису.
5. Після реєстрації ви автоматично перейдете до входу в додаток або можете повернутися на сторінку входу для введення облікових даних.

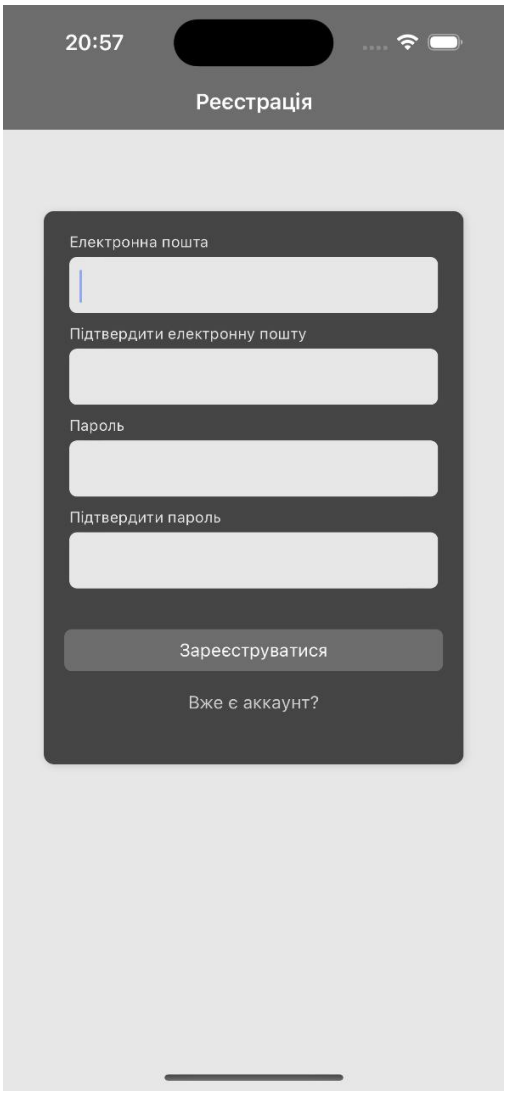

Рисунок 4.3 – Реєстрація

Крок 2: Вхід в додаток

1. На стартовому екрані виберіть "Увійти".
2. Введіть вашу електронну пошту та пароль у відповідні поля (див рис. 4.4).
3. Натисніть кнопку "Увійти" для доступу до основних функцій додатку.

Рисунок 4.4 – Вхід в додаток

Крок 3: Перегляд останніх витрат

1. На екрані "Останні витрати" ви знайдете перелік ваших недавніх витрат. Для кожної витрати вказані дата, категорія та сума (див рис. 4.5).
2. Натисніть на вкладку "Всі витрати" для доступу до повного переліку ваших витрат за весь час, що допоможе аналізувати ваші фінансові звички (див рис.

4.6).

3. Торкніться запису, щоб переглянути повний опис витрати і опції для її редагування або видалення, що забезпечує легке управління вашими фінансами (див рис. 4.7).

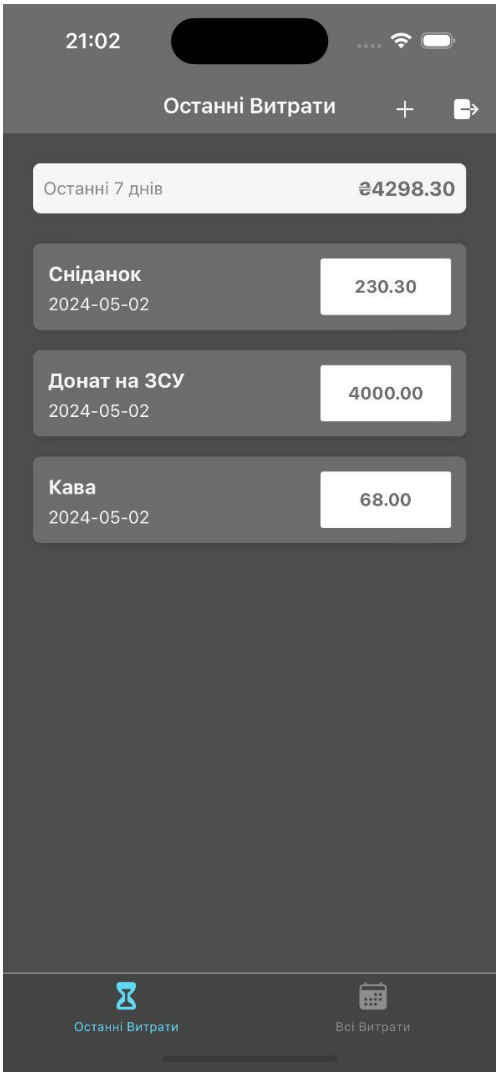

Рисунок 4.5 – Перегляд останніх витрат

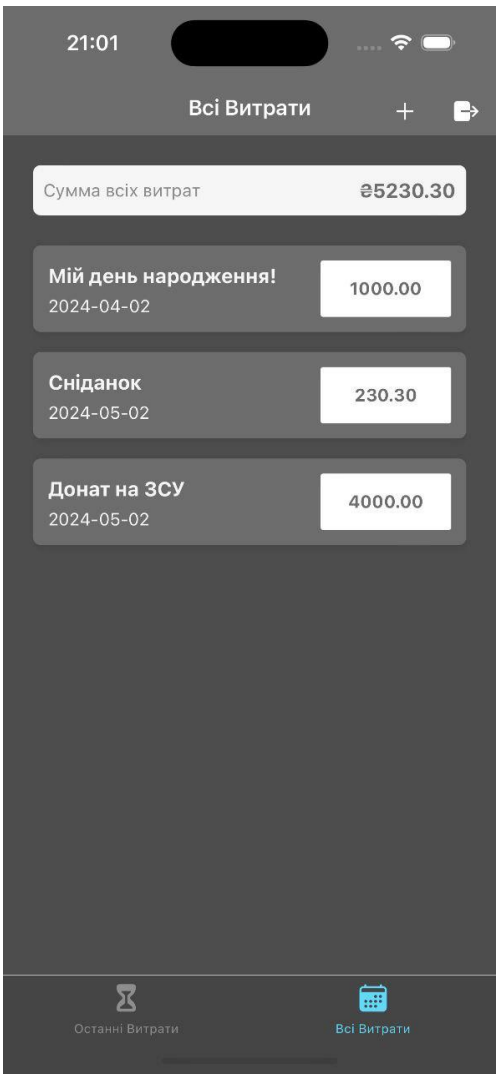

Рисунок 4.6 – Перегляд всіх витрат

Крок 4: Додавання витрат

1. На головному екрані натисніть кнопку "Додати витрату" для відкриття форми введення витрат (див рис. 4.5).
2. Введіть суму витрат, дату та опис (див рис. 4.7).
3. Натисніть кнопку "Додати" для збереження інформації, або "Скасувати", якщо потрібно внести зміни.

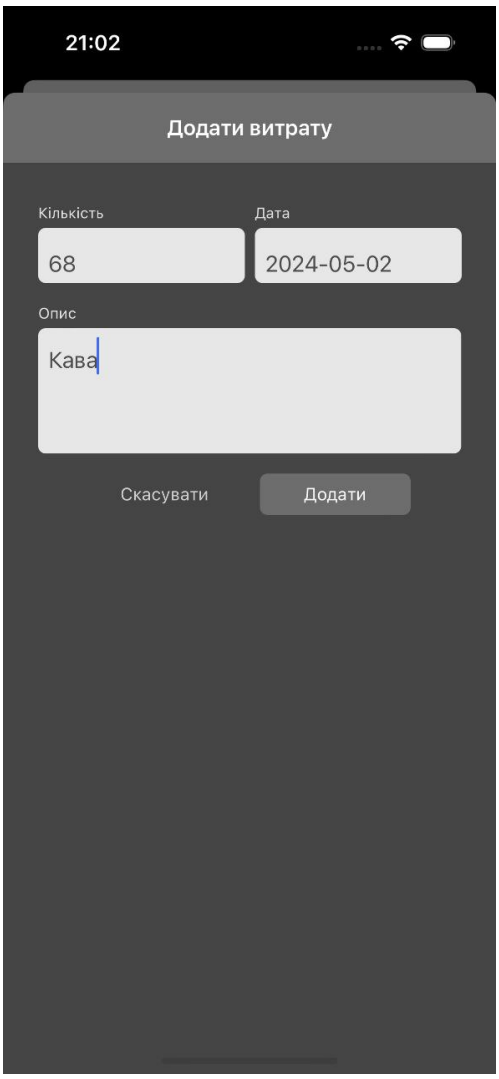

Рисунок 4.7 – Додавання витрат

Крок 5: Редагування та видалення витрат

1. Для редагування або видалення витрати торкніться відповідної витрати для відкриття екрана редагування.
2. Тут ви можете змінити деталі витрати або видалити її, використовуючи кнопку "Оновити" для збереження змін або "Скасувати" для скасування. Іконка сміттєвого відра використовується для видалення (див рис. 4.8).

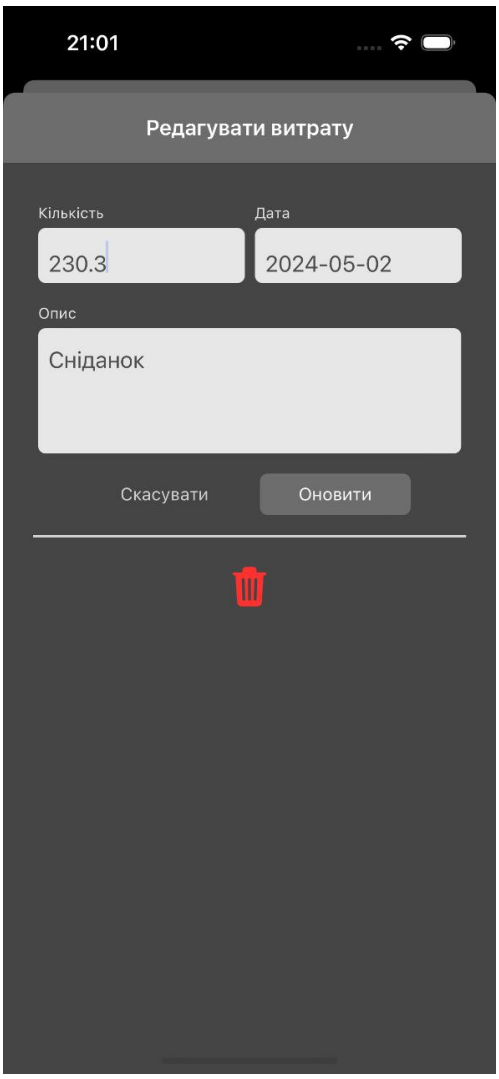

Рисунок 4.8 – Редагування та видалення витрат

Крок 6: Вихід із додатку

1. Для виходу з додатку або закриття сесії виберіть відповідний пункт у налаштуваннях або просто закрийте додаток, як це зазвичай робиться на вашому пристрої.

ВИСНОВКИ

У процесі виконання цього проекту з розробки мобільного додатку для відстеження витрат було проведено ряд важливих етапів, від теоретичного аналізу та проектування до практичної реалізації та тестування.

Результати роботи включають:

1. **Теоретичний аналіз**: Повторний огляд існуючих рішень та методів відстеження витрат, їхні переваги та недоліки.
2. **Вибір та обґрунтування технологічного стеку**: Вибрано React Native для фронтенду та Firebase для бекенду з метою оптимізації роботи та спрощення розробки.
3. **Проектування архітектури додатку**: Створено чітку та масштабовану архітектуру, що підтримує легке додавання нових функцій.
4. **Реалізація додатку**: Розроблені основні компоненти системи, що забезпечують введення, перегляд та управління витратами.
5. **Тестування та валідація**: Виконано ряд тестів для забезпечення надійності та ефективності роботи додатку.
6. **Документація та впровадження**: Підготовлено користувацьку документацію та інструкції, які спрощують засвоєння та використання додатку кінцевими користувачами.

Робота над проектом дозволила не тільки вирішити практичне завдання відстеження витрат в режимі реального часу, але й надала важливий досвід у галузі мобільної розробки та взаємодії з хмарними сервісами. Проект показав, що сучасні технології можуть ефективно вирішувати типові завдання управління персональними фінансами, роблячи цей процес максимально простим та зручним для користувачів.

СПИСОК ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. **Nader Dabit.** *React Native in Action*. Manning Publications, 2020. – 320 с.
2. **Erica Sadun, Richard E. Ward.** *The Core iOS Developer's Cookbook*. Addison-Wesley Professional, 2020. – 400 с.
3. **Jayant Varma.** *React Native Cookbook: Bringing the Web to Native Platforms*. Packt Publishing, 2018. – 350 с.
4. **Dan Ward.** *Firebase Essentials - Android Edition*. Neil Smyth, 2020. – 270 с.
5. **Fuller D.** *React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications*. Apress, 2019. – 250 с.
6. **Володимир Захаров.** *Кроссплатформенная разработка мобильных приложений на React Native*. ДМК Пресс, 2021. – 210 с.
7. **Marc L. Smith.** *Professional Redux*. Apress, 2021. – 330 с.
8. **Leanne Fitzpatrick.** *Hands-On Design Patterns with React Native*. Packt Publishing, 2019. – 290 с.
9. **Дмитро Дуденко.** *Основи TypeScript для професіоналів*. Видавництво "Академія", 2021. – 185 с.
10. **Michael Wanyoike.** *Build Mobile Apps with React Native and Firebase*. Independently Published, 2020. – 320 с.
11. **Axel Rauschmayer.** *Exploring JavaScript: From Intermediate to Advanced*. Dr. Axel Rauschmayer, 2019. – 416 с.
12. **Brad Green, Shyam Seshadri.** *AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps*. O'Reilly Media, 2014. – 302 с.
13. **Любов Галкіна.** *Введення в мобільну розробку з використанням PhoneGap та Ionic*. Фабула, 2020. – 230 с.
14. **Andrew J. Young, James J. Nuttall.** *Web Standards: Mastering HTML5, CSS3, and XML*. Apress, 2014. – 524 с.

ДОДАТОК А.

AllExpenses.tsx

```
import { useEffect, useState } from "react";

import ExpensesOutput from "../components/ExpensesOutput/ExpensesOutput";
import LoadingSpinner from "../components/UI/LoadingSpinner";
import { useAppSelector } from "../hooks/store";
import { useGetExpensesQuery } from "../redux/expensesAPI";
import { selectUser } from "../redux/userSlice";
import { Expense } from "../types/types";
import { firebaseExpensesToExpenses } from "../utils";

const AllExpensesScreen = () => {
  const user = useAppSelector(selectUser);
  if (!(user.id || user.token)) return <LoadingSpinner />;

  const { data, isLoading } = useGetExpensesQuery({
    userId: user.id,
    token: user.token,
  });
  const [expenses, setExpenses] = useState<Expense[]>([]);

  useEffect(() => {
    if (!data) return;
    setExpenses(firebaseExpensesToExpenses(data));
  }, [data]);

  if (isLoading) return <LoadingSpinner text="Завантаження ваших витрат" />;

  return (
```

```
      <ExpensesOutput
        expenses={expenses}
        expensesPeriod="Сумма всіх витрат"
      />
    );
};

export default AllExpensesScreen;
```

LoginScreen.tsx

```
import AuthContent from "../components/Auth/AuthContent";
import type { Props as LoginFormProps } from "../components/Auth/LoginForm";
import { useAppDispatch } from "../hooks/store";
import { loginUser } from "../redux/userSlice";

const LoginScreen = () => {
  const dispatch = useAppDispatch();

  const handleLoginUser: LoginFormProps["onSuccessfulLogin"] = data =>
    dispatch(loginUser(data));

  return <AuthContent mode="login" onAuth={handleLoginUser} />;
};

export default LoginScreen;
```

ManageExpenses.tsx

```
import React, { useEffect } from "react";

import { View } from "react-native";
import ExpenseDeleteButton from "../components/MangeExpense/ExpenseDeleteButton";
```

```
import ExpenseEditForm from "../components/MangeExpense/ExpenseForm/ExpenseEditForm";
import ExpenseNewForm from "../components/MangeExpense/ExpenseForm/ExpenseNewForm";
import tw from "../lib/tailwind";
import type { RootStackScreenProps } from "../types/navigation";

const ManageExpensesScreen = ({
  navigation,
  route: {
    params: { expenseId },
  },
}: RootStackScreenProps<"MangeExpense">) => {
  const isEditing = !!expenseId;

  useEffect(() => {
    // set title
    navigation.setOptions({
      title: isEditing ? "Редагувати витрату" : "Додати витрату",
    });
  }, [navigation, isEditing]);

  const handleFormCancel = () => {
    navigation.goBack();
  };

  const handleFormConfirm = () => {
    navigation.goBack();
  };
```

```
  return (
    <View style={tw`flex-1 p-6 bg-primary-800`}>
      {isEditing ? (
        <>
          <ExpenseEditForm
            expense={{ id: expenseId }}
            onCancel={handleFormCancel}
            onConfirm={handleFormConfirm}
          />
          <ExpenseDeleteButton
            expense={{ id: expenseId }}
            callback={handleFormCancel}
          />
        </>
      ) : (
        <ExpenseNewForm
          onConfirm={handleFormConfirm}
          onCancel={handleFormCancel}
        />
      )}
    </View>
  );
};

export default ManageExpensesScreen;
```

RecentExpenses.tsx

```
import { useEffect, useState } from "react";
import { firebaseExpensesToExpenses, getDateMinusDays } from "../utils";

import ExpensesOutput from "../components/ExpensesOutput/ExpensesOutput";
```

```
import LoadingSpinner from "../components/UI/LoadingSpinner";
import { useAppSelector } from "../hooks/store";
import { useGetExpensesQuery } from "../redux/expensesAPI";
import { selectUser } from "../redux/userSlice";
import { Expense } from "../types/types";

const RecentExpensesScreen = () => {
  const user = useAppSelector(selectUser);
  if (!(user.id || user.token)) return <LoadingSpinner />;

  const { data, isLoading } = useGetExpensesQuery({
    token: user.token,
    userId: user.id,
  });

  const [recentExpenses, setRecentExpenses] = useState<Expense[]>([]);

  useEffect(() => {
    if (!data) return;

    setRecentExpenses(
      firebaseExpensesToExpenses(data).filter(expense => {
        const today = new Date();
        const date7DaysAgo = getDateMinusDays(today, 7);
        return new Date(expense.date) > date7DaysAgo;
      }),
    );
  }, [data]);

  if (isLoading) return <LoadingSpinner text="Завантаження ваших витрат" />;
```

```
  return (
    <ExpensesOutput
      expenses={recentExpenses}
      expensesPeriod="Останні 7 днів"
    />
  );
};

export default RecentExpensesScreen;
```

SingUpScreen.tsx

```
import React, { useEffect } from "react";
import { useAppDispatch, useAppSelector } from "../hooks/store";
import { selectUser, signUpUser } from "../redux/userSlice";

import AuthContent from "../components/Auth/AuthContent";
import { Props as SignUpFormProps } from "../components/Auth/SignUpForm";

const SignUpScreen = () => {
  const dispatch = useAppDispatch();
  const user = useAppSelector(selectUser);

  const handleSignUpUser: SignUpFormProps["onSuccessfulSignUp"] = data =>
    dispatch(signUpUser(data));

  useEffect(() => {
    console.log("user", JSON.stringify(user, null, 2));
  }, [user]);

  return <AuthContent mode="signUp" onAuth={handleSignUpUser} />;
```

```
};

export default SignUpScreen;
```

App.tsx

```
import { StatusBar } from "expo-status-bar";
import { View } from "react-native";
import { Provider } from "react-redux";
import tw from "./src/lib/tailwind";
import AppNavigation from "./src/navigators";
import { store } from "./src/redux/store";

export default function App() {
  return (
    <View style={tw`flex-1`}>
      <Provider store={store}>
        <AppNavigation />
      </Provider>
      <StatusBar style="light" />
    </View>
  );
}
```

App.json

```
{
  "expo": {
    "name": "expense-tracker",
    "slug": "expense-tracker",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "userInterfaceStyle": "auto",
```

```
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#ffffff"
    },
    "assetBundlePatterns": [
      "**/*"
    ],
    "ios": {
      "supportsTablet": true
    },
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./assets/adaptive-icon.png",
        "backgroundColor": "#ffffff"
      }
    },
    "web": {
      "favicon": "./assets/favicon.png"
    }
  }
}
```

Tailwind.config.js

```
/** @type {import('tailwindcss').Config} */
module.exports = {
  content: [],
  theme: {
    screens: {
      sm: "380px",
      md: "420px",
```

```
      lg: "680px",
      tablet: "1024px",
    },
    extend: {
      colors: {
        primary: {
          50: "#f6f6f6",
          100: "#e7e7e7",
          200: "#d1d1d1",
          400: "#888888",
          500: "#6d6d6d",
          700: "#4d4d4d",
          800: "#454545",
        },
        accent: { 500: "#61d9fb" },
        error: { 50: "#fff1f1", 500: "#ff3232" },
        gray: { 500: "#39324a", 700: "#221c30" },
      },
    },
  },
  plugins: [],
};
```

.env.local

```
EXPO_PUBLIC_FIREBASE_API_KEY="AIzaSyAX9OhcS9ww7octRBBpNZ0pHCk9uVGsrLM"
EXPO_PUBLIC_AUTH_API_URL="https://identitytoolkit.googleapis.com/v1/"
EXPO_PUBLIC_DB_API_URL="https://expense-tracker-9b8d7-default-rtdb.firebaseio.com/"
```