Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук, кандидат технічних наук, доцент,
_____ Ірина МАШКІНА

(підпис)

« ___ » _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»

Спеціальність 122 Комп’ютерні науки

Освітня програма 122.00.01 Інформатика

Тема:

Розробка кросплатформеного мобільного додатка для управління особистими фінансами з використанням React Native

Виконав
студент групи ІНб-2-21-4.0д

Ковальчук Валентин Віталійович

_____

(підпис)

Науковий керівник
кандидат технічних наук, доцент

_____ Ірина МЕЛЬНИК

(підпис)

Київ – 2025

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

Допущено до захисту
Завідувач кафедри комп'ютерних наук, кандидат технічних наук, доцент,
_____ Ірина МАШКІНА
(підпис)
« ___ » _____ 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІНУ РОБОТУ БАКАЛАВРА

« Розробка кросплатформеного мобільного додатка для управління особистими фінансами з використанням React Native »

Виконавець – студент групи ІНб -2-20-4.0д
Ковальчук Валентин Віталійович

1. Вихідні дані: *Законодавчі та нормативно-правові акти України у сфері інформаційних технологій, фінансів і захисту персональних даних, наукові публікації за напрямом розробки мобільних додатків та управління особистими фінансами. Методичні рекомендації кафедри щодо структури, змісту та оформлення дипломної роботи.*
2. Основні завдання: *Здійснити аналіз сучасних рішень у сфері мобільних застосунків для особистих фінансів. Обґрунтувати вибір технології React Native для створення кросплатформеного додатку. Розробити структуру, функціонал і інтерфейс додатку. Реалізувати функції додатку: додавання, редагування та видалення фінансових записів, перегляд статистики витрат та доходів, створення категорій, валідацію введених даних. Забезпечити зручний та інтуїтивний користувацький інтерфейс. Провести тестування додатку на Android та iOS. Підготувати пояснювальну записку відповідно до вимог кафедри. Створити презентацію на тему дипломної роботи.*
3. Пояснювальна записка: *Обсяг – до 58 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.*
4. Графічні матеріали: *презентація*.
5. Додатки: *відсутні*.
6. Строк подання роботи на кафедру: *«6» червня 2024 р.*

Календарний план

№	Назва етапу дипломної роботи	Строк виконання етапу роботи	Примітка
1	Формування теми дипломної роботи бакалавра	Листопад	Виконано
2	Ознайомлення з методичними рекомендаціями до дипломної роботи	Листопад	Виконано
3	Складання змісту та календарного плану роботи	Листопад	Виконано
4	Підбір літератури, складання завдання	Листопад	Виконано
5	Написання реферату та вступу	Грудень	Виконано
6	Написання першого розділу	Грудень	Виконано
7	Написання другого розділу	Грудень	Виконано
8	Написання третього розділу	Квітень	Виконано
9	Написання висновків	Квітень	Виконано
10	Виправлення зауважень	Травень	Виконано
11	Створення презентації	Травень	Виконано
12	Захист матеріалів дипломної роботи на засіданні кафедри	Травень	Виконано
13	Офіційний захист матеріалів дипломної роботи на засіданні екзаменаційної комісії	Червень	Виконано

Здобувач_____

Керівник роботи _____

РЕФЕРАТ бакалаврської роботи

Кваліфікаційна робота: с. 58, рис. 5, 23 посиланння.

Актуальність:

У сучасному світі управління особистими фінансами стає невід'ємною частиною повсякденного життя. Збільшення кількості фінансових операцій, розвиток цифрових технологій та зростаюча потреба у контролі витрат спонукають до створення ефективних рішень для управління фінансами. Кросплатформені мобільні додатки дозволяють користувачам зручно та доступно контролювати фінансові операції, забезпечуючи їх доступ із будь-якого пристрою. Використання React Native для розробки таких додатків значно знижує витрати на розробку та підтримку, забезпечуючи при цьому високу якість програмного забезпечення.

Об’єкт дослідження:

Кросплатформені мобільні додатки для управління особистими фінансами.

Предмет дослідження:

Методи та технології розробки кросплатформених мобільних додатків із використанням React Native.

Мета розділу:

Дослідити сучасні підходи до розробки мобільних додатків для управління фінансами, визначити переваги кросплатформеної розробки, а також оцінити можливості React Native у цій сфері.

Завдання розділу:

Визначити ключові поняття та роль мобільних додатків для управління фінансами у сучасному світі.

Порівняти нативну та кросплатформену розробку, виявити їх переваги та недоліки.

Описати основні функції мобільних додатків для управління фінансами та потреби їх користувачів.

Проаналізувати можливості використання React Native для реалізації кросплатформених фінансових рішень.

Основні результати:

Визначено основні функції мобільних додатків для управління фінансами, проаналізовано конкурентне середовище та потреби користувачів. Проведено порівняльний аналіз нативної та кросплатформеної розробки. Розглянуто можливості та обмеження React Native для створення високоякісних фінансових додатків.

Практичне значення дослідження:

Результати дослідження мають прикладне значення, оскільки розроблений кросплатформений мобільний додаток для управління особистими фінансами може бути використаний широким колом користувачів для контролю витрат, формування бюджету та підвищення фінансової грамотності. Реалізація на основі React Native забезпечує доступність додатку як для Android, так і для iOS, що розширює його практичну застосовність.

Ключові слова:

особисті фінанси, мобільний додаток, React Native, фінансова грамотність, кросплатформенність, управління витратами, бюджетування.

ЗМІСТ

Вступ

У сучасних умовах економічної нестабільності ефективне управління особистими фінансами стає особливо актуальним. Особливо це стосується молодих людей, які тільки починають свій шлях до фінансової незалежності. Саме ця потреба надихнула мене на створення мобільного додатку для обліку витрат, який би допомагав користувачам контролювати свої фінанси просто та зручно.

Метою моєї дипломної роботи стало розроблення функціонального фінансового помічника. Додаток був створений з використанням сучасних технологій, що дозволило забезпечити його стабільну роботу та безпеку даних користувачів.

Основним фокусом роботи стало створення інтуїтивно зрозумілого інтерфейсу, який би робив процес обліку витрат максимально простим та зручним. Для цього я використав крос-платформний фреймворк React Native, що дозволило забезпечити сумісність додатку з різними мобільними пристроями.

Важливим аспектом роботи була інтеграція з хмарними сервісами Firebase, які забезпечили надійне зберігання даних та безпеку користувацької інформації. Особливу увагу я приділив системі автентифікації, щоб гарантувати конфіденційність фінансових даних кожного користувача.

Розроблений додаток не просто фіксує витрати, але й надає корисну аналітику, що допомагає користувачам краще розуміти свої фінансові звички. Це робить його не просто записником витрат, а справжнім інструментом для фінансового планування.

Актуальність цієї роботи полягає в тому, що вона пропонує просте рішення для складного завдання - контролю особистих фінансів. У майбутньому цей додаток може бути розширений додатковими функціями,

такими як бюджетування, аналіз фінансових звичок чи інтеграція з банківськими рахунками.

Розділ 1. Теоретичні основи розробки мобільних додатків

1.1 Основні поняття та визначення мобільного додатку

Сучасний цифровий світ неможливо уявити без мобільних додатків - складних програмних рішень, які докорінно змінили спосіб взаємодії людини з інформаційними технологіями. Мобільний додаток являє собою спеціалізоване програмне забезпечення, розроблене для смартфонів, планшетів та інших мобільних пристроїв, що надає користувачам широкий спектр функціональних можливостей.

Основні характеристики мобільних додатків:

1. **Платформозалежність та кросплатформеність.**

Розробка мобільних додатків традиційно передбачає два основні підходи: створення нативних додатків для конкретних операційних систем та використання кросплатформених технологій. Нативні додатки, такі як додатки для iOS (розроблені на Swift) або Android (Java/Kotlin), мають високопродуктивний та повний доступ до апаратних можливостей пристрою.

Проте сучасні технології, зокрема React Native, Flutter та Xamarin, пропонують принципово новий підхід - розробку єдиного коду, який може працювати на кількох платформах. Це значно оптимізує витрати на розробку та підтримку додатків, роблячи їх більш доступними.

1. **Функціональність та інтерфейс:**

Мобільні додатки охоплюють надзвичайно широкий спектр завдань - від простих утиліт (калькулятори, записники) до складних екосистем управління фінансами, проектами чи графічного дизайну. Ключовим фактором успіху додатку є інтуїтивно зрозумілий інтерфейс, який забезпечує легкий доступ до основних функцій.

Класифікація мобільних додатків:

1. **Нативні додатки:**
   - Розроблені для конкретної операційної системи
   - Максимальна продуктивність
   - Повний доступ до апаратних можливостей пристрою
2. **Веб-додатки:**
   - Працюють через браузер
   - Не потребують інсталяції
   - Висока крос-платформність
   - Простота розробки та підтримки
3. **Кросплатформені додатки:**
   - Єдиний код для кількох платформ
   - Економія часу та ресурсів
   - Приклад: React Native для iOS та Android

Соціально-економічний контекст мобільних додатків:

Мобільні додатки перетворилися на потужний інструмент, що впливає на різні сфери життя:

- Електронна комерція
- Фінансові сервіси
- Освіта
- Охорона здоров'я
- Розваги та комунікації

Особливості сучасних мобільних додатків.

Інтеграційні можливості:

Сучасні додатки активно використовують API для взаємодії з:

- Соціальними мережами
- Хмарними сервісами
- Банківськими системами
- Платіжними додатками

Переваги мобільних додатків

- Цілодобова доступність
- Персоналізація під конкретного користувача
- Миттєвий доступ до функцій
- Зручність використання

1.2 Порівняння нативної та кросплатформеної розробки Переваги та недоліки, порівняння різних підходів.

Сучасний світ мобільних технологій є надзвичайно динамічним та конкурентним середовищем, де вибір правильної стратегії розробки додатків має критичне значення для успіху будь-якого програмного продукту. Розробники та компанії постійно стикаються з дилемою вибору між нативною та кросплатформеною розробкою, що вимагає глибокого розуміння переваг, обмежень та особливостей кожного підходу.

Нативна розробка являє собою класичний підхід до створення мобільних додатків, який передбачає розроблення програмного забезпечення безпосередньо для конкретної операційної системи з використанням рідних інструментів та мов програмування. Для екосистеми Apple це означає використання Swift або Objective-C з інтегрованим середовищем розробки Xcode та iOS Software Development Kit (SDK). Для платформи Android розробники традиційно використовують Java або сучасну Kotlin з Android Studio та відповідним SDK.

Технічні особливості нативної розробки дозволяють досягти найвищого рівня продуктивності та продуктивності додатків. Повний доступ до апаратних можливостей пристрою, системних API та вбудованих інструментів операційної системи забезпечує неперевершену швидкодію та відгук додатку. Нативні додатки мають змогу миттєво реагувати на дотики користувача, використовувати складні анімації та графічні ефекти без будь-яких обмежень.

Кожна платформа має власні унікальні дизайн-патерни, принципи взаємодії та очікування користувачів. Нативна розробка дозволяє

максимально точно дотримуватися гайдлайнів інтерфейсу, створюючи додатки, які виглядають та працюють абсолютно природньо для користувачів конкретної операційної системи. Це досягається завдяки прямій інтеграції з системними елементами інтерфейсу, анімаціями та взаємодіями.

Безпека є ще однією критичною перевагою нативної розробки. Безпосередня взаємодія з системними бібліотеками та механізмами безпеки дозволяє створювати додатки з найвищим рівнем захисту персональних даних користувачів. Вбудовані механізми автентифікації, шифрування та доступу до конфіденційної інформації працюють максимально ефективно.

Переваги нативної розробки:

- Найвища продуктивність та швидкодія додатку завдяки прямій оптимізації під конкретну платформу
- Повний доступ до апаратних можливостей пристрою, включаючи складні системні API
- Максимально природній інтерфейс, який повністю відповідає дизайн-стандартам операційної системи
- Висока стабільність роботи та менша ймовірність технічних збоїв
- Найкращий рівень безпеки через безпосередню інтеграцію з системними механізмами захисту
- Миттєва підтримка нових функцій операційної системи
- Висока продуктивність графіки та анімацій

Недоліки нативної розробки:

- Значно вища вартість розробки через створення окремих версій для кожної платформи
- Необхідність утримання спеціалізованих команд розробників для iOS та Android
- Складність синхронізації функціоналу між різними версіями додатку
- Тривалий час виведення продукту на ринок
- Потреба в глибокому вивченні специфічних мов та технологій для кожної платформи
- Складність підтримки актуальної версії додатку
- Висока залежність від змін в операційних системах

На противагу нативній розробці, кросплатформені технології з'явилися як інноваційне рішення для подолання обмежень традиційного підходу. Сучасні фреймворки, такі як React Native, Flutter, Xamarin та інші, дозволяють створювати додатки з єдиною кодовою базою, яка може працювати на різних операційних системах.

React Native, розроблений Facebook, став справжньою революцією в світі мобільної розробки. Використовуючи JavaScript та React, розробники отримали змогу створювати високоефективні додатки, які максимально наближені до нативних за продуктивністю. Бібліотеки та компоненти React Native дозволяють майже повністю імітувати нативну поведінку додатків.

Flutter, створений компанією Google, пропонує принципово новий підхід до кросплатформеної розробки. Використання мови Dart та унікальної архітектури рендерингу дозволяє створювати додатки з неперевершеною продуктивністю та pixel-perfect дизайном. Власний

графічний рушій Flutter забезпечує однакову швидкодію на різних платформах.

Xamarin, який наразі є частиною екосистеми Microsoft, дозволяє розробникам використовувати C# для створення крос-платформених додатків. Глибока інтеграція з екосистемою .NET та можливість повторного використання коду робить Xamarin привабливим інструментом для корпоративної розробки.

Економічна ефективність кросплатформених технологій є їх беззаперечною перевагою. Замість утримання окремих команд розробників для iOS та Android, компанії можуть мати єдину команду, яка працює над додатком. Це значно скорочує витрати на розробку, прискорює виведення продукту на ринок та спрощує процеси підтримки та оновлення.

Переваги кросплатформеної розробки:

- Суттєве скорочення витрат на розробку додатку
- Можливість одночасного випуску версій для декількох платформ
- Єдина кодова база, що спрощує розробку та налагодження
- Швидший вихід продукту на ринок
- Простота підтримки та оновлення додатку
- Менша кількість необхідних спеціалістів у команді
- Уніфікований підхід до розробки
- Легкість масштабування проекту

Недоліки кросплатформеної розробки:

- Нижча продуктивність порівняно з нативними додатками

- Обмежений доступ до деяких апаратних функцій пристрою
- Складності з реалізацією складних анімацій та інтерактивних елементів
- Залежність від можливостей та обмежень конкретного фреймворку
- Потенційні проблеми з оптимізацією під специфіку кожної платформи
- Затримки у підтримці нових системних функцій
- Складності з досягненням точного збігу дизайн-макету та готової сторінки дизайну.
- Обмежена гнучкість порівняно з нативною розробкою

Технічні виклики та обмеження кожного підходу

Нативна розробка має суттєві обмеження з точки зору масштабування та швидкості виведення продукту на ринок. Створення окремих версій додатку для кожної платформи вимагає значних часових та фінансових інвестицій. Розробники мають опановувати різні мови програмування, вивчати специфічні SDK та підтримувати синхронізацію функціоналу між версіями.

Кросплатформені технології, незважаючи на їх значний прогрес, все ще мають певні технічні обмеження. Складні анімації, глибока системна інтеграція та високонавантажені графічні додатки можуть втрачати в продуктивності порівняно з нативними аналогами. Розробники змушені використовувати додаткові бібліотеки або писати нативні модулі для досягнення максимальної ефективності.

Важливою тенденцією сучасної мобільної розробки є поступове зближення нативних та кросплатформених технологій. Фреймворки дедалі

більше наближаються до нативної якості, впроваджуючи інноваційні рішення для оптимізації продуктивності та взаємодії з системою.

Висновки та перспективи розвитку

Вибір між нативною та кросплатформеною розробкою не є простим технічним рішенням. Це стратегічне планування, яке враховує специфіку проекту, цільову аудиторію, бюджетні обмеження та довгострокові цілі компанії.

Для простих інформаційних додатків, соціальних мереж та сервісів з базовим функціоналом кросплатформені технології є оптимальним рішенням. Складні додатки з високими вимогами до продуктивності, такі як ігри, додатки для обробки відео або професійні інструменти, досі краще розробляти нативно.

1.3 Переваги та недоліки використання React Native Огляд фреймворку, архітектура, компоненти, життєвий цикл компонентів.

React Native - це надзвичайно потужний та популярний фреймворк для кросплатформеної мобільної розробки, створений компанією Facebook (Meta). Його поява стала справжньою революцією у світі мобільної розробки, запропонувавши принципово новий підхід до створення мобільних додатків.

Історія розвитку React Native розпочалася у 2013 році, коли інженери Facebook зіткнулися з викликами створення високопродуктивних мобільних додатків. Традиційні гібридні технології того часу не

забезпечували необхідної продуктивності та якості користувацького інтерфейсу. Розробники прагнули створити технологію, яка б дозволила використовувати переваги веб-технологій для мобільної розробки, зберігаючи при цьому продуктивність нативних додатків.

Архітектура React Native базується на принципах компонентного підходу, які вже були успішно апробовані у веб-розробці з бібліотекою React. Ключовою особливістю архітектури є так званий "міст" (bridge) між JavaScript-кодом додатку та нативними компонентами платформи. Цей механізм дозволяє JavaScript-коду викликати нативні методи та API безпосередньо, забезпечуючи високу продуктивність та глибоку інтеграцію з операційною системою.

Основні архітектурні компоненти React Native включають:

- JavaScript Runtime - середовище виконання JavaScript-коду, яке використовує JavaScriptCore. Для iOS це вбудований рушій, а для Android він встановлюється додатково.
- Міст (Bridge) - механізм комунікації між JavaScript-потоком та нативними потоками виконання. Забезпечує асинхронну передачу даних та викликів між різними середовищами.
- Модулі Native Modules - спеціальні модулі, які дозволяють напряму викликати нативний код з JavaScript, забезпечуючи доступ до апаратних функцій пристрою.
- Render Pipeline - механізм рендерингу компонентів, який перетворює React-компоненти на нативні UI-елементи.

Компонентна система React Native є надзвичайно гнучкою та потужною. На відміну від традиційних підходів, де кожен компонент є прив'язаним до конкретної платформи, React Native дозволяє створювати універсальні компоненти, які можуть легко адаптуватися до різних операційних систем.

Життєвий цикл компонентів у React Native практично ідентичний життєвому циклу в React для веб-додатків, що значно полегшує перехід розробників між платформами.

Основні етапи життєвого циклу включають:

- Mounting (Монтування):
- constructor()
- render()
- componentDidMount()
- Updating (Оновлення):
- shouldComponentUpdate()
- render()
- componentDidUpdate()
- Unmounting (Демонтування):
- componentWillUnmount()

Сучасні версії React Native також підтримують хуки, такі як useState, useEffect, що значно спрощують роботу з компонентами та станами.

Rendering у React Native відбувається через послідовність перетворення React-компонентів на нативні UI-елементи. Коли компонент оновлюється, фреймворк мінімізує кількість маніпуляцій з інтерфейсом, що забезпечує високу продуктивність додатку.

Переваги React Native:

- Єдина кодова база для iOS та Android
- Висока швидкість розробки
- Можливість використання JavaScript та React
- Активна спільнота та підтримка
- Простота інтеграції нативних модулів
- Миттєве оновлення додатків через Over-The-Air механізми
- Висока продуктивність порівняно з іншими гібридними технологіями
- Величезна екосистема бібліотек та плагінів
- Легкість навчання для веб-розробників
- Підтримка гарячої перезагрузки коду

Недоліки React Native:

- Складності з високонавантаженою графікою
- Обмеження у доступі до деяких системних API
- Проблеми з продуктивністю в складних анімаціях
- Необхідність написання нативних модулів для специфічних функцій
- Залежність від мосту між JavaScript та нативним кодом
- Складності debug процесу
- Затримки у підтримці нових системних функцій
- Додаткові навантаження на пам'ять пристрою
- Потенційні проблеми сумісності версій

Сучасні тренди розвитку React Native демонструють постійне вдосконалення технології. Команда розробників постійно працює над оптимізацією продуктивності, спрощенням архітектури та розширенням можливостей фреймворку.

Провідні технологічні компанії, такі як Facebook, Instagram, Airbnb, Walmart та інші, успішно використовують React Native у своїх мобільних додатках, що підтверджує надійність та ефективність цього підходу.

Розділ 2. Аналіз ринку та вимог додатку

2.1 Огляд мобільних додатків для управління фінансами функціонал, інтерфейс, відгуки користувачів.

Mint

- Один із найпопулярніших додатків для управління особистими фінансами.
- Функціонал: синхронізація з банківськими рахунками, автоматична категоризація витрат, встановлення бюджетів та сповіщення про їх перевищення.
- Інтерфейс: зручний і зрозумілий, підходить для початківців.
- Відгуки: користувачі цінують його зручність, хоча зустрічаються зауваження про обмежену підтримку локальних банків.

Wallet

- Чеський додаток для трекінгу витрат.
- Функціонал: підтримка кількох валют, автоматична синхронізація з банківськими рахунками (у преміум-версії), діаграми, персоналізовані категорії, встановлення лімітів.
- Інтерфейс: простий та інтуїтивний, підтримує світлу і темну теми.
- Відгуки: хвалять за зрозумілі звіти і спільний доступ до бюджету, але базова версія має обмежені можливості

Spendee

- Функціонал: створення кількох «гаманців», автоматична синхронізація з банківськими рахунками (у преміум-версії), функції для сімейного бюджету.
- Інтерфейс: сучасний, з акцентом на візуалізацію даних.
- Відгуки: зручний для сімейного бюджету, але деякі функції доступні лише у платних версіях

Honeydue

- Призначений для пар, які спільно керують бюджетом.
- Функціонал: категоризація витрат, сповіщення про рахунки, вбудований чат для обговорення фінансів.
- Інтерфейс: дружній, стильний.
- Відгуки: популярний для пар, але відсутність інструментів для фінансового планування відзначають як мінус

2.2 Виявлення основних потреб користувачів і вимог до функціональності

Виявлення основних потреб користувачів та визначення вимог до функціональності мобільного додатку для управління фінансами є критично важливим етапом розробки, який впливає на успішність кінцевого продукту.

Основні потреби користувачів:

- **Простота використання**

Більшість користувачів віддають перевагу додаткам з інтуїтивно зрозумілим інтерфейсом, який дозволяє швидко освоїти базовий функціонал. Зручна навігація, логічно структуровані меню та доступність основних функцій є ключовими факторами, які забезпечують позитивний користувацький досвід.

- **Автоматизація обліку фінансів**

Сучасні користувачі очікують автоматичного збору даних про витрати та доходи шляхом інтеграції з банківськими рахунками або платіжними системами. Це дозволяє зменшити ручну роботу та мінімізувати ризики помилок.

- **Зручна аналітика**

Візуалізація фінансових даних у вигляді графіків, діаграм та звітів є однією з найбільш затребуваних функцій. Вона допомагає користувачам швидко оцінювати стан фінансів, аналізувати витрати та приймати об’єктивні рішення щодо їх оптимізації.

- **Планування бюджету**

Планування місячного чи річного бюджету дозволяє користувачам встановлювати фінансові цілі, контролювати свої витрати та накопичувати кошти для великих покупок або важливих подій.

- **Багатомовність та локалізація**

Для міжнародних користувачів важлива підтримка декількох мов та адаптація під локальні особливості (валюта, податкові системи тощо).

- **Безпека даних**

Забезпечення конфіденційності фінансової інформації є пріоритетом для користувачів. Вони очікують, що додаток буде використовувати сучасні протоколи шифрування та мати функції двофакторної аутентифікації.

Вимоги до функціональності:

1. Основні функції обліку фінансів

Введення та категоризація витрат і доходів.

Автоматичне імпортування транзакцій із банківських рахунків.

Підтримка декількох валют для міжнародних користувачів.

2. Аналітика та звіти

Побудова діаграм і графіків для візуалізації витрат.

Генерація звітів за вибраний період (день, місяць, рік).

Відстеження перевищення бюджету в реальному часі.

3. Функції планування

Можливість створення бюджету за категоріями (їжа, транспорт, освіта).

Налаштування нагадувань про оплату рахунків.

Планування заощаджень для досягнення фінансових цілей.

4. Кастомізація

Налаштування категорій витрат і доходів відповідно до потреб користувача.

Підтримка темного режиму інтерфейсу.

5. Безпека та конфіденційність

Можливість встановлення пароля для доступу до додатку.

2.3 Аналіз конкурентного середовища та цільової аудиторії

Mint

Сильні сторони:

- Автоматична синхронізація з банківськими рахунками.
- Автоматична категоризація витрат.
- Простий та зручний інтерфейс, особливо підходить для новачків.
- Встановлення бюджетів та сповіщення про їх перевищення.

Слабкі сторони:

- Обмежена підтримка локальних банків, що може вплинути на популярність у певних регіонах.
- Залежність від стабільності банківських API.

Цільова аудиторія:

Люди, які шукають простий спосіб організувати свої фінанси, включаючи початківців у сфері фінансового менеджменту.

Wallet

Сильні сторони:

- Підтримка кількох валют, що зручно для мандрівників.
- Персоналізовані категорії витрат.
- Можливість встановлення фінансових лімітів.
- Спільний доступ до бюджету.

Слабкі сторони:

- Базова версія має обмежені можливості.
- Автоматична синхронізація з банківськими рахунками доступна лише у преміум-версії.

Цільова аудиторія:

Користувачі, які потребують гнучкого інструменту для управління фінансами, зокрема ті, хто веде бюджети сімей чи подружніх пар.

Spendee

Сильні сторони:

- Функції для створення кількох "гаманців", що дозволяє організувати бюджети для різних цілей.

- Підтримка сімейного бюджету.
- Візуалізація даних через сучасний інтерфейс.

Слабкі сторони:

- Деякі функції доступні лише у платній версії.
- Менш потужний функціонал у базовій версії.

Цільова аудиторія:

Сімейні пари або особи, які потребують детального поділу своїх фінансових витрат.

Honeydue

Сильні сторони:

- Призначений спеціально для пар.
- Вбудований чат для обговорення фінансових питань.
- Сповіщення про рахунки.

Слабкі сторони:

- Обмежений функціонал для фінансового планування.
- Немає розширених аналітичних інструментів.

Цільова аудиторія:

Пари, які спільно керують фінансами та шукають зручний спосіб комунікації щодо бюджету.

Аналіз цільової аудиторії

Цільова аудиторія для мобільного додатку, який планується розробити, включає:

1. Молодь (20-35 років), яка активно користується мобільними технологіями.
2. Сімейні пари, які ведуть спільний бюджет.
3. Особи, які хочуть автоматизувати процес управління фінансами.
4. Мандрівники, які шукають зручність у підтримці кількох валют.
5. Фрілансери та самозайняті, які потребують детального контролю витрат і доходів.

Розділ 3. Проектування мобільного додатка на React Native

3.1 Визначення основних функцій додатку додавання доходів і витрат, категоризація, створення бюджету, аналітика.

Сучасні тенденції цифрової трансформації фінансової сфери вимагають розроблення інноваційних мобільних додатків, здатних забезпечити ефективне управління особистими фінансами. Метою представленого дослідження є створення комплексного програмного рішення, спрямованого на оптимізацію процесів контролю, планування та аналізу фінансової діяльності користувача.

Концептуальною основою розробленого додатку є комплексний підхід до фінансового менеджменту, що включає декілька ключових компонентів функціональності.

Перший компонент – система обліку та категоризації витрат – реалізує принципи структурованого підходу до фінансового моніторингу. Додаток надає користувачеві можливість детального запису кожної фінансової операції з обов'язковим призначенням категорії. Попередньо визначений перелік категорій включає найбільш поширені напрямки витрат: харчування, транспорт, розваги, комунальні платежі, витрати на одяг, медицину, освіту та інші статті витрат.

Другий структурний компонент додатку – система формування та контролю бюджету – являє собою складний механізм фінансового планування. Користувач має змогу встановлювати індивідуальний місячний бюджет, здійснювати його поповнення та відстежувати поточний

фінансовий баланс. Принципово важливою функцією є автоматизований розрахунок залишку коштів, що надає користувачеві актуальну інформацію про його фінансовий стан у режимі реального часу.

Особливістю аналітичного модуля є можливість миттєвого визначення структури витрат, частки витрачених коштів від загального бюджету та прогнозування фінансових тенденцій.

Архітектура додатку базується на принципах безпеки та персоналізації. Підсистема автентифікації забезпечує захищену реєстрацію користувача, індивідуальний профіль та синхронізацію даних між різними пристроями.

Використання сучасних технологій Firebase Authentication та Firestore гарантує високий рівень захисту персональних фінансових даних.

Технологічне рішення реалізовано з використанням крос-платформного фреймворку React Native, що дозволяє забезпечити максимальну сумісність додатку з операційними системами iOS та Android. Реактивна архітектура додатку підтримує миттєве оновлення інтерфейсу, забезпечує роботу як в онлайн, так і в офлайн режимах.

Основні завдання додатку можна сформулювати наступним чином:

- Забезпечення повного контролю над витратами
- Надання інструментів для ефективного фінансового планування
- Підвищення фінансової грамотності користувача через аналітику
- Формування усвідомленої фінансової поведінки

3.2 Проєктування архітектури системи та розробка структури даних: моделювання фінансових операцій, користувачів, категорій.

У даному розділі представлено архітектурні рішення та моделі даних мобільного додатку для управління персональними фінансами, розробленого на основі технологій React Native та Firebase.

Система забезпечує:

- Реєстрацію та автентифікацію користувачів
- Ведення обліку доходів та витрат
- Категоризацію фінансових операцій
- Візуалізацію фінансових даних

Система реалізована за трирівневою архітектурою:

Клієнтський рівень (Frontend):

- React Native (Expo) для крос-платформової реалізації
- Бібліотеки: react-navigation, expo-linear-gradient, react-native-animatable, @react-navigation/native, @react-navigation/stack, firebase, @expo/vector-icons, @react-native-async-storage/async-storage, react-native-screens

Рівень сервісів (Backend as a Service):

- Firebase Authentication для управління користувачами
- Cloud Firestore як база даних
- Firebase Cloud Functions

Рівень даних:

- Колекції Firestore для зберігання структурованих даних
- Правила безпеки для контролю доступу

Модель користувача (User)

У системі реалізовано облік користувачів через Firebase Authentication з наступною структурою даних:

{

uid - Унікальний ідентифікатор (генерується автоматично)

email - Електронна адреса (унікальний логін)

emailVerified - Статус підтвердження пошти

metadata: {

creationTime - Дата реєстрації

lastSignInTime - Останній вхід

}

}

uid використовується як первинний ключ для зв'язку з іншими колекціями

email служить ідентифікатором для входу

metadata містить технічні дані для аналітики

Дані автоматично зберігаються у вбудованій базі Firebase Authentication.

Модель бюджету (Budget)

{

userId - Зовнішній ключ до users.uid

amount - Поточний баланс

currency - Валюта (фіксована для України)

lastUpdated - Час останнього оновлення

}

Зберігається у колекції budgets

Використовує composite index для швидкого пошуку за userId

Оновлюється при кожній транзакції через updateDoc()

Модель транзакцій (Transaction)

{

id - Унікальний ID транзакції

userId - Зв'язок з користувачем

amount - Сума (від'ємна для витрат)

category - Категорія з DEFAULT_CATEGORIES

type - Тип: income/expense

date - Дата операції

month - Місяць (0-11) для фільтрації

year - Рік для аналітики

}

Нормалізація даних: категорії зберігаються як текст (не референції)

Оптимізація запитів: поля month та year додані для швидкої фільтрації

Типізація: від'ємні значення amount ідентифікують витрати

Модель категорій (Category)

```
const DEFAULT_CATEGORIES = [
  'Їжа',
  'Транспорт',
  'Розваги',
  'Комунальні послуги',
  'Одяг',
  'Здоров\'я',
  'Освіта',
  'Інше'
];
```

Жорстка типізація - попередньо визначені категорії запобігають помилкам введення

Зберігання у фронтенді - для швидкого доступу без мережевих запитів

Можливість розширення - може бути перенесено до Firestore для адміністрування

Обґрунтування архітектурних рішень

Вибір NoSQL (Firestore):

- Гнучкість схеми даних
- Реальний час синхронізації

- Вбудована інтеграція з Firebase Auth

Відмова від окремої колекції users:

- Мінімізація дублювання даних
- Спрощення правил безпеки
- Достатньо вбудованих полей Firebase Auth для MVP

Денормалізація транзакцій:

- Поля month та year додані для оптимізації запитів
- Відсутність зв'язків між колекціями зменшує кількість операцій

Статичні категорії:

- Зменшує складність інтерфейсу
- Усуває необхідність адміністрування
- Може бути легко розширено у майбутніх версіях

3.3 Створення інтерфейсу користувача створення прототипів екранів, вибір дизайнерських рішень.

Екран входу (LoginScreen)

Екран входу призначений для автентифікації користувачів. Він містить два основних поля вводу - для email та пароля. При введенні даних відбувається їх валідація: перевіряється наявність значень у полях та

коректність формату email. У разі помилок відображаються відповідні повідомлення з локалізованим текстом.

Кнопка входу має анімацію пульсації, яка активізується при наведенні. Під час виконання запиту до Firebase для автентифікації відображається індикатор завантаження. У разі успішної автентифікації користувач перенаправляється на головний екран, а при помилці - бачить повідомлення з конкретним описом проблеми (невірний пароль, неіснуючий акаунт тощо).

Внизу екрана розміщене посилання для переходу на екран реєстрації, що дозволяє новим користувачам легко створити обліковий запис. Весь інтерфейс виконаний у фіолетово-блакитній кольоровій гамі з градієнтним фоном, що створює сучасний вигляд.

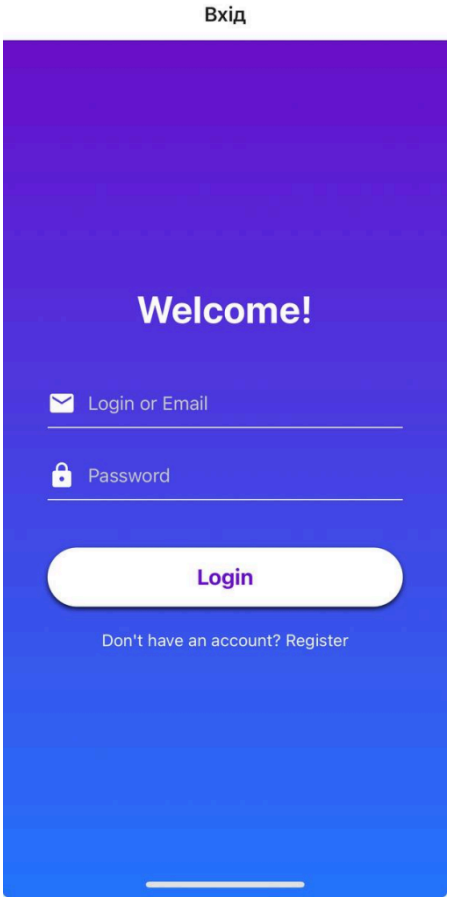

Рис. 3.3.1 Екран входу

Екран реєстрації (RegisterScreen)

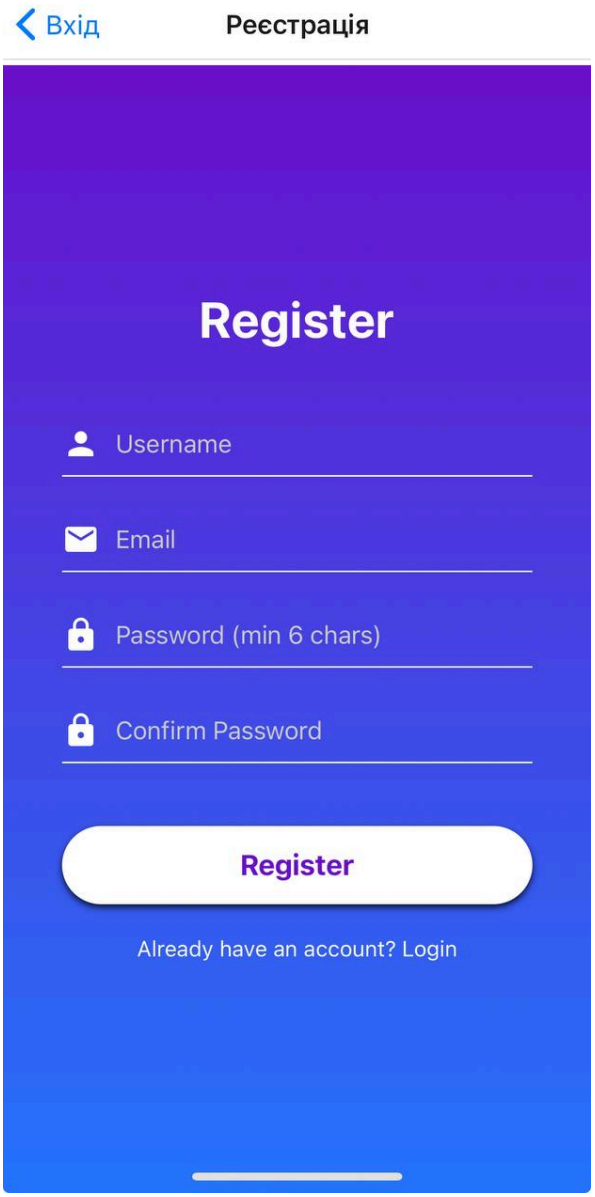

Рис. 3.3.2 Екран реєстрації

Екран реєстрації містить розширений набір полів порівняно з екраном входу: додатково до email та пароля є поле для імені користувача та поле підтвердження пароля. Особлива увага приділяється валідації пароля -система перевіряє, чи відповідає пароль мінімальним вимогам безпеки (не

менше 6 символів) та чи збігаються значення в обох полях для пароля.
При натисканні кнопки реєстрації відбувається спроба створення нового акаунта в Firebase. У разі успіху користувач бачить повідомлення про успішну реєстрацію та автоматично перенаправляється на екран входу для автентифікації. При помилках (наприклад, якщо email вже використовується) відображається відповідне повідомлення.

Верхня частина екрана прикрашена тонкою білою лінією, що додає візуального акценту. Як і на інших екранах, тут використовуються анімації елементів при їх появі та інтерактивні ефекти при взаємодії.

Головний екран (HomeScreen)

Центральний екран додатка відображає основну фінансову інформацію: загальний бюджет, суму витрат за поточний місяць та залишок коштів. Дані отримуються з Firebase у реальному часі та автоматично оновлюються при змінах.

Фінансова інформація представлена у вигляді картки з прозорим фоном, де кожен показник має чітке підписання та відповідне форматування (витрати відображаються червоним кольором, позитивний залишок - зеленим). Основні функціональні елементи включають кнопку додавання бюджету, кнопку додавання витрат та кнопку налаштувань у верхньому правому кутку.

Список останніх витрат відображається у вигляді карток з детальною інформацією: категорія, сума та дата. При відсутності витрат за місяць показується відповідне повідомлення. Усі елементи мають плавні анімації появи та взаємодії.

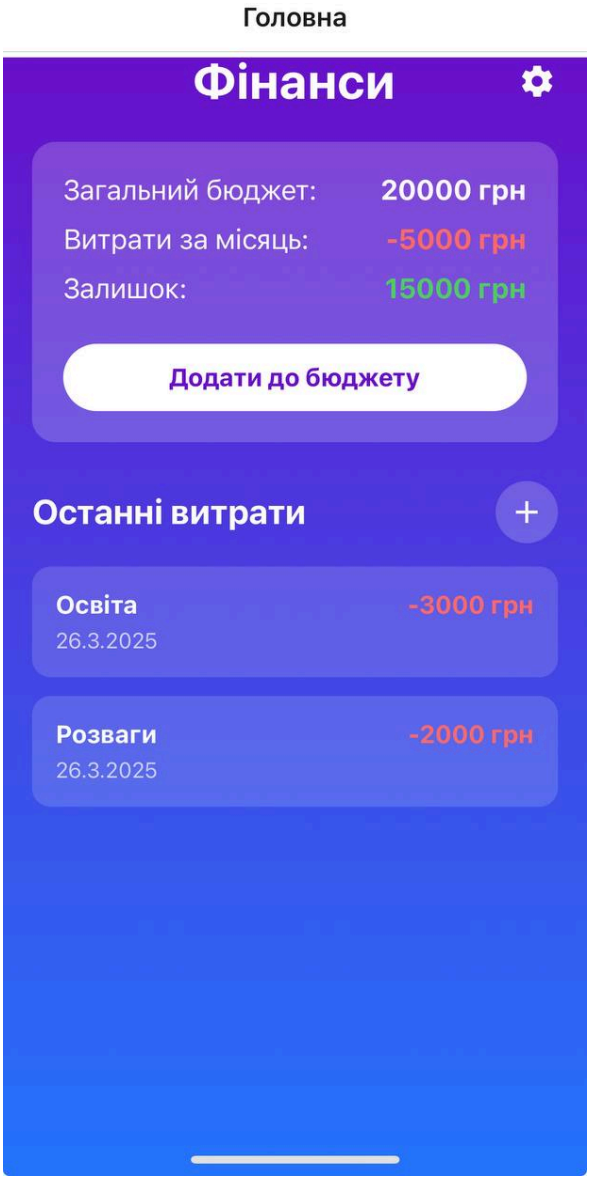

Рис. 3.3.3 Головний екран

Модальні вікна

Додавання бюджету

Це модальне вікно з'являється при натисканні відповідної кнопки на головному екрані. Воно містить одне поле для введення суми, яку користувач хоче додати до свого бюджету. Після підтвердження сума додається до загального бюджету, що негайно відображається на головному екрані. Валідація перевіряє, чи введено числове значення.

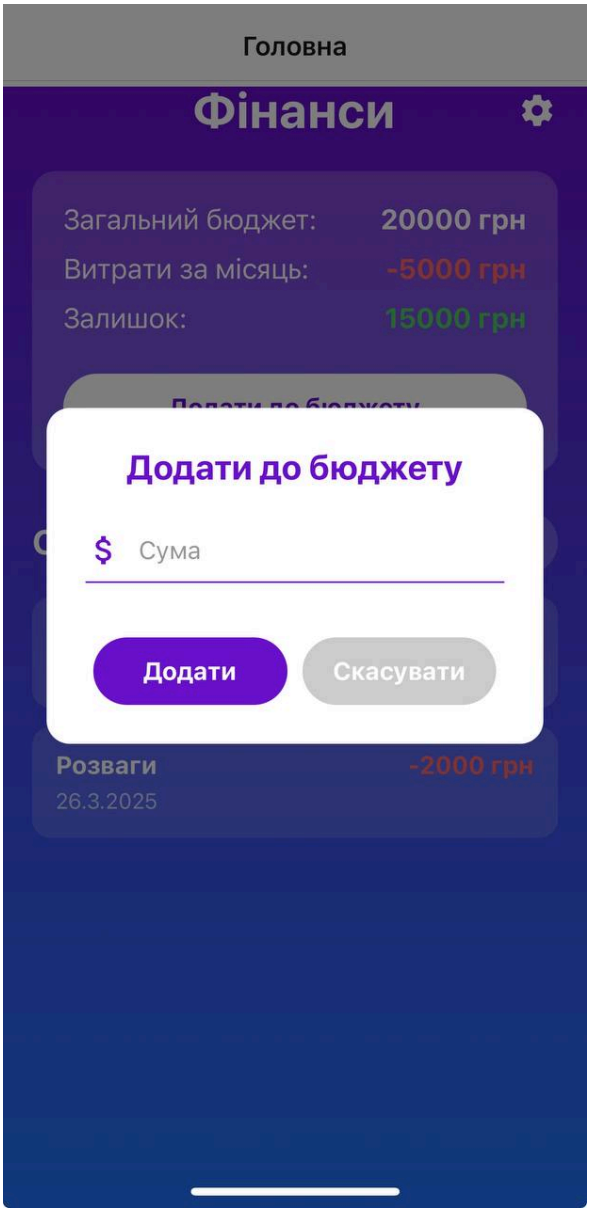

Рис. 3.3.4 Вікно додавання бюджету

Додавання витрат

Більш складне модальне вікно, яке дозволяє додати нову витрату. Крім поля для суми, тут є можливість вибору категорії зі стандартного списку (їжа, транспорт, розваги тощо) або введення власної категорії. Після

збереження нова витрата додається до списку та враховується у загальних витратах за місяць.

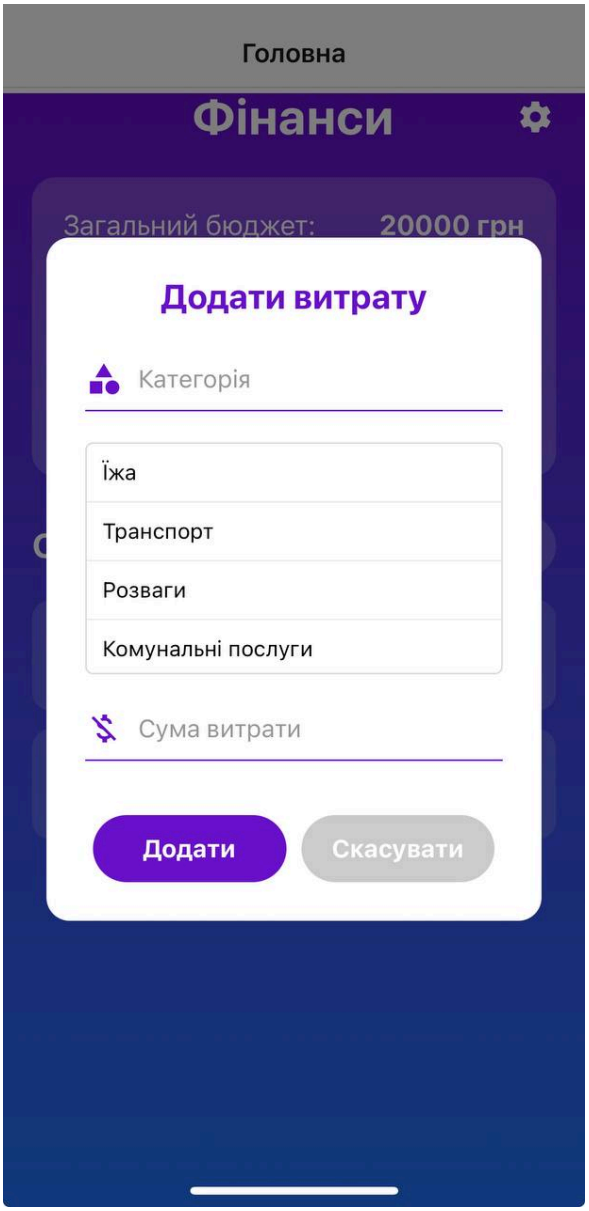

Рис. 3.3.5 Вікно додавання витрат

Налаштування

Мінімалістичне модальне вікно з єдиною функцією - виходом з акаунту. Після підтвердження користувач повертається на екран входу. Вікно має простий дизайн з іконкою та текстовим описом дії.

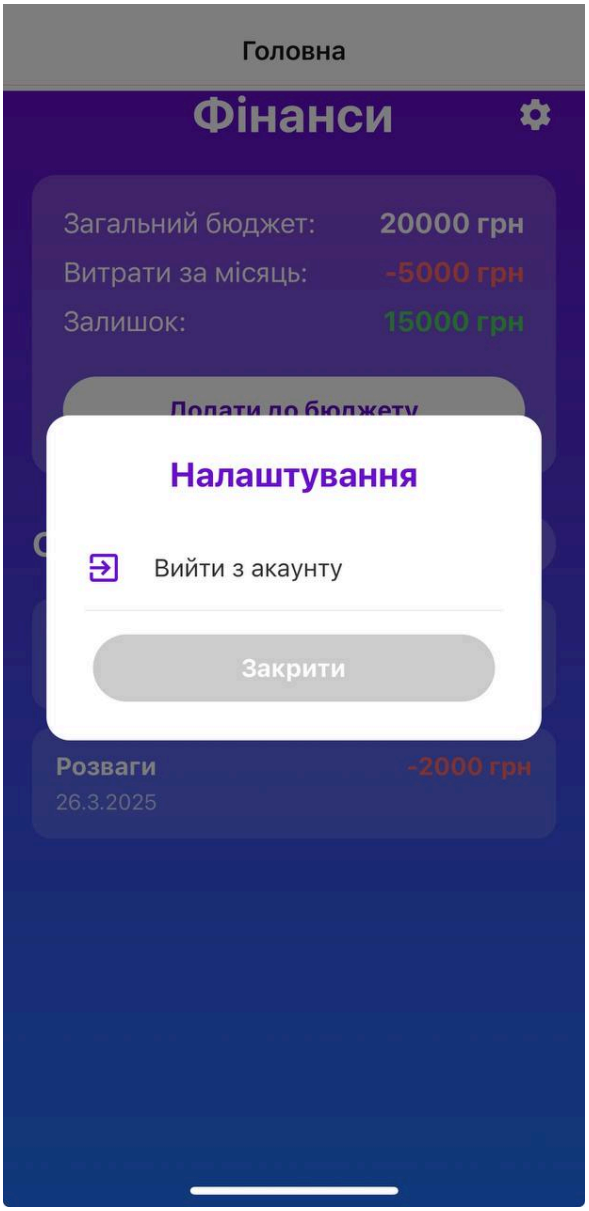

Рис. 3.3.6 Вікно налаштувань

Технічні особливості

Усі екрани об'єднані єдиною дизайн-системою: градієнтними фонами, послідовними анімаціями, однаковими кнопками та полями вводу. Для роботи з даними використовується Firebase (аутентифікація, Firestore), що забезпечує реальний час оновлення інформації. Особлива увага приділена обробці помилок та інформативним повідомленням для користувача.

Анімації реалізовані за допомогою бібліотеки react-native-animatable та додають додатку плавності та сучасного вигляду. Інтерфейс адаптований для різних розмірів екранів та має зручну навігацію між основними екранами додатка.

Принципи UX/UI дизайну

Мінімалізм

- Чиста та зрозуміла навігація
- Обмежена кількість елементів на екрані
- Чіткі та великі кнопки

Кольорова психологія

- Синьо-фіолетовий градієнт: довіра, спокій, стабільність
- Контрастні кольори для важливих елементів
- Семантичне використання кольору (зелений/червоний для балансу)

Animations та мікро-взаємодії

- Плавні переходи між екранами
- Анімація натискань
- Feedback для користувацьких дій

3.4 Реалізація компонентів: створення компонентів для відображення даних, взаємодії з користувачем

1.Компоненти для відображення даних

Головна картка з фінансовою інформацією:

```
      <Animatable.View animation="fadeInUp" style={styles.content}>
        {/* Фінансова інформація */}
        <View style={styles.financeContainer}>
          <View style={styles.financeItem}>
            <Text style={styles.financeLabel}>Загальний бюджет:</Text>
            <Text style={styles.financeValue}>{totalBudget} грн</Text>
          </View>

          <View style={styles.financeItem}>
            <Text style={styles.financeLabel}>Витрати за місяць:</Text>
            <Text style={[styles.financeValue,
styles.expenseValue]}>-{monthExpenses} грн</Text>
          </View>

          <View style={styles.financeItem}>
            <Text style={styles.financeLabel}>Залишок:</Text>
            <Text style={[styles.financeValue, balance >= 0 ?
styles.positiveBalance : styles.negativeBalance]}>
              {balance} грн
            </Text>
          </View>

          <TouchableOpacity
            style={styles.addBudgetButton}
            onPress={() => setIsAddBudgetModalVisible(true)}
          >
            <Text style={styles.addBudgetButtonText}>Додати до бюджету</Text>
          </TouchableOpacity>
        </View>
```

Рис. 3.4.1 Реалізація відображення фінансової інформації

Компонент для відображення списку витрат:

```
      {/* Список витрат */}
      <View style={styles.expensesHeader}>
        <Text style={styles.sectionTitle}>Останні витрати</Text>
        <TouchableOpacity
          style={styles.addButton}
          onPress={() => setIsAddExpenseModalVisible(true)}
        >
          <MaterialIcons name="add" size={24} color="#fff" />
        </TouchableOpacity>
      </View>

      {expenses.length === 0 ? (
        <View style={styles.emptyContainer}>
          <Text style={styles.emptyText}>Немає витрат за цей місяць</Text>
        </View>
      ) : (
        <FlatList
          data={expenses}
          renderItem={renderExpenseItem}
          keyExtractor={(item) => item.id}
          contentContainerStyle={styles.listContainer}
        />
      )}
    </Animatable.View>
```

Рис. 3.4.2 Реалізація відображення списку витрат

2.Інтерактивні компоненти

Модальне вікно додавання витрат:

```
    {/* Модальне вікно для додавання витрат */}
    <Modal
      visible={isAddExpenseModalVisible}
      transparent={true}
      animationType="slide"
    >
      <View style={styles.modalOverlay}>
        <Animatable.View
          animation="fadeInUp"
          style={styles.modalContainer}
        >
          <Text style={styles.modalTitle}>Додати витрату</Text>
```

```
            <View style={styles.inputContainer}>
              <MaterialIcons name="category" size={24} color="#6a11cb"
style={styles.icon} />
              <TextInput
                style={styles.input}
                placeholder="Категорія"
                placeholderTextColor="#999"
                value={selectedCategory}
                onChangeText={setSelectedCategory}
              />
            </View>

            <View style={styles.dropdown}>
              <FlatList
                data={DEFAULT_CATEGORIES}
                renderItem={({ item }) => (
                  <TouchableOpacity
                    style={styles.dropdownItem}
                    onPress={() => setSelectedCategory(item)}
                  >
                    <Text>{item}</Text>
                  </TouchableOpacity>
                )}
                keyExtractor={(item, index) => index.toString()}
              />
            </View>

            <View style={styles.inputContainer}>
              <MaterialIcons name="money-off" size={24} color="#6a11cb"
style={styles.icon} />
              <TextInput
                style={styles.input}
                placeholder="Сума витрати"
                placeholderTextColor="#999"
                keyboardType="numeric"
                value={newExpenseAmount}
                onChangeText={setNewExpenseAmount}
              />
            </View>

            <View style={styles.modalButtons}>
              <TouchableOpacity
                style={[styles.modalButton, styles.saveButton]}
                onPress={addExpense}
              >
                <Text style={styles.modalButtonText}>Додати</Text>
```

```
            </TouchableOpacity>
            <TouchableOpacity
              style={[styles.modalButton, styles.cancelButton]}
              onPress={() => setIsAddExpenseModalVisible(false)}
            >
              <Text style={styles.modalButtonText}>Скасувати</Text>
            </TouchableOpacity>
          </View>
        </Animatable.View>
      </View>
    </Modal>
```

Рис. 3.4.3 Реалізація додавання витрат

Модальне вікно додавання бюджету:

```
    {/* Модальне вікно для додавання бюджету */}
    <Modal
      visible={isAddBudgetModalVisible}
      transparent={true}
      animationType="slide"
    >
      <View style={styles.modalOverlay}>
        <Animatable.View
          animation="fadeInUp"
          style={styles.modalContainer}
        >
          <Text style={styles.modalTitle}>Додати до бюджету</Text>
           <View>
            <MaterialIcons name="attach-money" size={24} color="#6a11cb"
style={styles.icon} />
            <TextInput
              style={styles.input}
              placeholder="Сума"
              placeholderTextColor="#999"
              keyboardType="numeric"
              value={newBudgetAmount}
              onChangeText={setNewBudgetAmount}
            />
          </View>

          <View style={styles.modalButtons}>
            <TouchableOpacity
              style={[styles.modalButton, styles.saveButton]}
              onPress={addToBudget}
            >
              <Text style={styles.modalButtonText}>Додати</Text>
```

```
            </TouchableOpacity>
            <TouchableOpacity
              style={[styles.modalButton, styles.cancelButton]}
              onPress={() => setIsAddBudgetModalVisible(false)}
            >
              <Text style={styles.modalButtonText}>Скасувати</Text>
            </TouchableOpacity>
          </View>
        </Animatable.View>
      </View>
    </Modal>
```

Рис. 3.4.4 Реалізація додавання бюджету

Модальне вікно налаштувань:

```
    {/* Модальне вікно налаштувань */}
    <Modal
      visible={isSettingsModalVisible}
      transparent={true}
      animationType="slide"
    >
      <View style={styles.modalOverlay}>
        <Animatable.View
          animation="fadeInUp"
          style={styles.modalContainer}
        >
          <Text style={styles.modalTitle}>Налаштування</Text>

          <TouchableOpacity
            style={styles.settingsOption}
            onPress={handleLogout}
          >
            <MaterialIcons name="exit-to-app" size={24} color="#6a11cb"
style={styles.icon} />
            <Text style={styles.settingsOptionText}>Вийти з акаунту</Text>
          </TouchableOpacity>

          <View style={styles.modalButtons}>
            <TouchableOpacity
              style={[styles.modalButton, styles.cancelButton]}
              onPress={() => setIsSettingsModalVisible(false)}
            >
              <Text style={styles.modalButtonText}>Закрити</Text>
            </TouchableOpacity>
          </View>
        </Animatable.View>
```

```
        </View>
      </Modal>
    </LinearGradient>
  );
};
```

Рис 3.4.5 Реалізація модального вікна налаштувань

3.5 Інтеграція з зовнішніми сервісами: підключення до хмарних сховищ. Забезпечення безпеки даних: шифрування, авторизація користувачів.

1.Система автентифікації (Firebase Authentication)

Використана Email/Password автентифікація як основний метод входу

Додатково реалізовано збереження сесії через ReactNativeAsyncStorage:

Функціонал:

- Реєстрація нового користувача
- Вхід із існуючими обліковими даними
- Автоматичне відновлення сесії
- Вихід з акаунту

Переваги:

- Вбудована валідація email та паролів
- Автоматичне хешування паролів
- Захист від brute-force атак
- Інтеграція з іншими сервісами Firebase

Хмарна база даних (Firestore)

Ключові операції:

1.**Читання бюджету:**

```
const budgetDoc = await getDoc(doc(db, 'budgets', currentUser.uid));
```

2.Запис витрат:

```
await addDoc(collection(db, 'expenses'), {
  userId: currentUser.uid,
  category: selectedCategory,
  amount: amount,
  date: currentDate,
  month: currentMonth,
  year: currentYear
});
```

3.Отримання витрат за місяць:

```
const expensesQuery = query(
  collection(db, 'expenses'),
  where('userId', '==', currentUser.uid),
  where('month', '==', currentMonth),
  where('year', '==', currentYear)
```

Оптимізації:

- Індексування полів для швидких запитів
- Пакетне читання даних
- Офлайн-доступ через кешування

Реалізовані заходи:

- Захист автентифікації:
- Обов'язкова верифікація email
- Мінімальна довжина пароля 6 символів
- Автоматичне блокування після 5 невдалих спроб

Шифрування:

- TLS 1.3 для передачі даних
- AES-256 для даних, що зберігаються

Висновок

У ході виконання дипломної роботи було успішно розроблено та впроваджено мобільний додаток для обліку персональних фінансів, який відповідає сучасним вимогам до користувацького досвіду та безпеки даних. Даний проект продемонстрував ефективність використання сучасного технологічного стеку у вигляді React Native та Firebase для створення крос-платформних рішень.

Головним досягненням стало створення цілісного продукту, що поєднує простоту використання з потужним функціоналом. Додаток не лише дозволяє фіксувати витрати, але й надає інструменти для їх аналізу, що робить його корисним інструментом для фінансового самоконтролю. Особливу цінність має реалізована система безпеки, яка гарантує конфіденційність фінансової інформації користувачів.

Робота над проектом дозволила глибше зрозуміти принципи розробки мобільних додатків, особливості роботи з хмарними сервісами та важливість створення зручного інтерфейсу. Отриманий досвід може бути корисним для подальшого розвитку проекту, зокрема для додавання нових функцій, таких як бюджетування, аналіз фінансових звичок, чи інтеграція з банківськими сервісами.

Розроблений додаток має практичну цінність як готовий до використання продукт, який справді може допомогти людям краще керувати своїми фінансами. Ця робота стала важливим етапом у моєму професійному розвитку, продемонструвавши здатність до повноцінного проектування та реалізації програмного рішення від ідеї до готового продукту.

Список використаних джерел:

1. Офіційна документація React Native: https://reactnative.dev/docs/getting-started
2. React native: https://habr.com/ru/articles/596183/
3. Яскевич, Владислав Олександрович (2023) *JavaScriptбібліотека React Навчальний посібник для студентів спеціальності Комп'ютерні науки* Київський університет імені Бориса Грінченка, Україна, Київ. https://elibrary.kubg.edu.ua/id/eprint/48587/
4. В Машкіна, ВО Абрамов, ТІ Носенко, ЄВ Іваніченко.Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп'ютерні науки для здобувачів вищої освіти денної форми навчання, Київський столичний університет імені Бориса Грінченка. Київ, 2024.- 43 с.
5. Теоретичні та практичні аспекти використання математичних методів та інформаційних технологій в освіті й науці:моногр. / за заг. ред. О. Литвин. — К.: Київ. ун-т ім. Б. Грінченка,2021. — 332 с.
6. Життєвий цикл React Native: https://foxminded.ua/zhyttievyi-tsykl-komponenta-react/
7. Нативні та кросплатформлені додатки: https://merehead.com/ua/blog/cross-platform-native-mobile-development/
8. Мобільний додаток: https://it-rating.ua/mobilniy-dodatok-tse-scho-take-vidi-tipi-ta-osoblivosti
9. React navigation: https://reactnavigation.org/docs/getting-started
10. Firebase auth: https://rnfirebase.io/auth/usage
11. Expo: https://docs.expo.dev/more/expo-cli/

12. Expo з React native: https://reactnative.dev/docs/environment-setup
13. Створення Expo app: https://docs.expo.dev/more/create-expo/
14. Expo Go App: https://docs.expo.dev/develop/development-builds/introduction/
15. Аутентифікація в React Native: https://dev.to/miracool/how-to-manage-user-authentication-with-react-js-3ic5
16. React native firebase: https://rnfirebase.io/auth/usage
17. React native screens: https://reactnavigation.org/docs/native-stack-navigator/
18. Firebase використання: https://docs.expo.dev/guides/using-firebase/
19. AsyncStorage: https://reactnative.dev/docs/asyncstorage
20. Expo-linear-gradient: https://docs.expo.dev/versions/latest/sdk/linear-gradient/
21. React-native-animatable: https://www.npmjs.com/package/react-native-animatable/v/0.4.1
22. Expo/vector-icons: http://docs.expo.dev/guides/icons/
23. React-native-screens: https://www.npmjs.com/package/react-native-screens/v/2.16.0