Міністерство освіти і науки України

Луцький національний технічний університет
(повне найменування закладу вищої освіти)

Факультет комп'ютерних та інформаційних технологій
(повне найменування факультету)

Кафедра комп'ютерної інженерії та кібербезпеки
(повне найменування кафедри)

КВАЛІФІКАЦІЙНА РОБОТА ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

КРОС-ПЛАТФОРМЕННИЙ ДОДАТОК ЗАСОБАМИ REACT NATIVE ДЛЯ АНАЛІЗУ ТА КОНТРОЛЮ ВИТРАТ

A CROSS-PLATFORM APPLICATION BY MEANS REACT NATIVE FOR ANALYSIS AND COST CONTROL

спеціальність 123 Комп'ютерна інженерія
(шифр і назва спеціальності)

освітня програма Комп'ютерна інженерія
(назва освітньої програми)

Виконав: здобувач вищої освіти
групи КІ-41
Присняк Владислав Святославович

(підпис)

Керівник:
к.т.н., доцент
Пех Петро Антонович

(підпис)

Кваліфікаційну роботу
допущено до захисту
«____» червня 2023 р.
Гарант освітньої програми:
к.т.н., доцент
Лавренчук Світлана Василівна

(підпис)

Луцьк – 2023 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерної інженерії та кібербезпеки*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *123 Комп'ютерна інженерія*
Освітня програма: «*Комп'ютерна інженерія*»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Н. Чернящук
«_____» _____2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Присняку Владиславу Святославовичу
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Крос-платформенний додаток засобами React Native для аналізу та контролю витрат*

Керівник роботи: *Пех Петро Антонович, к.т.н., доцент*

затверджені наказом закладу вищої освіти від «28» грудня 2022 р. №982/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи 01.06.2023 р.

3. Вихідні дані до роботи *Джерелом розробки є науково-технічна література та публікації в періодичних виданнях з даного питання, опубліковані зарубіжні та вітчизняні роботи в даній області, різні інтернет-ресурси технічного спрямування*

4. Зміст пояснювальної записки (перелік питань, що потрібно розробити):
Вступ
Загальні відомості про розробку мобільного ПЗ
Проектування структури мобільного додатку
Розробка мобільного додатку
Висновки

5. Перелік графічного (ілюстративного) матеріалу:
Рисунки інтерфейсів користувача, схеми моделей бази даних, архітектурні діаграми, лістинги коду

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Загальні відомості про розробку мобільного ПЗ	*Пех П.А.*		
Проектування структури мобільного додатку	*Пех П.А.*		
Розробка мобільного додатку	*Пех П.А.*		
Висновки	*Пех П.А.*		

7. Дата видачі завдання 01.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1.	*Обґрунтування теми*	05.11.2022	Виконано
2.	*Огляд літератури на тему розробки ПЗ*	04.12.2022	Виконано
3.	*Дослідження загальних відомостей про розробку мобільного ПЗ*	14.01.2023	Виконано
4.	*Проектування структури мобільного додатку*	05.02.2023	Виконано
5.	*Розробка мобільного додатку*	25.03.2023	Виконано
6.	*Висновки та пропозиції*	02.04.2023	Виконано
7.	*Формування списку використаних джерел*	09.04.2023	Виконано
8.	*Нормоконтроль*	20.05.2023	Виконано
9.	*Інструментальна перевірка на академічний плагіат*	30.05.2023	Виконано
10.	*Представлення кваліфікаційної роботи бакалавра до захисту*	06.06.2023	Виконано

Здобувач вищої освіти _____ (Присняк В.С.)
(підпис) (прізвище, ініціали)

Керівник кваліфікаційної роботи _____ (Пех П.А.)
(підпис) (прізвище, ініціали)

АНОТАЦІЯ

Присняк В.С. Крос-платформенний додаток засобами React Native для аналізу та контролю витрат. Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерна інженерія» спеціальності 123 Комп'ютерна інженерія. Луцький національний технічний університет. Луцьк, 2023.

Кваліфікаційна робота складається з вступу, трьох розділів, висновків, списку використаних джерел, чотирьох додатків.

У першому розділі проведено аналіз стану проблеми, включаючи порівняльний аналіз інструментів для контролю витрат і доходів. Також були вибрані засоби розробки, зокрема база даних Firebase, JavaScript, фреймворк React Native, IDE WebStorm та платформа для розробки React Native додатків Expo.

У другому розділі було проведено проектування структури мобільного додатку: обґрунтовано обрані технології та описано принципи взаємодії з базою даних, представлена схема, створена модель бази даних.

У третьому розділі здійснено безпосередню розробку мобільного додатку: створено інтерфейс користувача, розроблено логіку та принципи робити додатку, проведено заходи для захисту даних, проведене тестування та налагодження системи та надано інструкцію користувача з використання розробленого додатку.

Об'єкт дослідження – сучасні технології розробки мобільних додатків.

Предмет дослідження – розробка мобільного додатка з використанням React Native, включаючи проектування структури додатку, забезпечення взаємодії з базою даних, розробку інтерфейсу користувача, способи захисту даних, тестування та налагодження.

Мета роботи полягає в розробці крос-платформенного додатку з використанням React Native для аналізу та контролю витрат, який забезпечить

аналіз витрат і доходів та наддасть користувачеві засоби для ефективного контролю та управління фінансами.

Ключові слова: React Native, аналіз витрат, контроль витрат, крос-платформенний додаток, JavaScript, IDE WebStorm, база даних Firebase.

ANNOTATION

Prysniak V.S. A cross-platform application by means React Native for analysis and cost control. Manuscript.

Bachelor's qualification work of the "Computer Engineering" educational program, specialization 123 Computer Engineering. Lutsk National Technical University. Lutsk, 2023.

The qualification work consists of an introduction, three sections, conclusions, a list of references, and four appendices.

The first section includes an analysis of the problem's current state, including a comparative analysis of tools for expense and income control. The chosen development tools are Firebase (database), JavaScript, React Native framework, WebStorm IDE, and Expo as the platform for developing React Native applications.

In the second section, the design of the mobile application's structure is carried out, justifying the chosen technologies and describing the principles of interacting with the database. A schema is presented, and a database model is created.

The third section focuses on the direct development of the mobile application, including the creation of the user interface, development of the application's logic and principles, implementation of data protection measures, testing, and debugging of the system. Additionally, a user guide for the developed application is provided.

Object of research – cross-platform application using React Native for expense analysis and control.

Subject of research – development of a mobile application using React Native, including application structure design, database interaction, user interface development, data protection methods, testing, and debugging.

The aim of the work is to develop a cross-platform application using React Native for expense analysis and control, providing users with tools for efficient financial management and control of expenses and income.

Keywords: React Native, expense analysis, expense control, cross-platform application, JavaScript, WebStorm IDE, Firebase database.

ЗМІСТ

ВСТУП

Актуальність теми. У сучасному світі, де мобільні пристрої стали невід'ємною частиною нашого повсякденного життя, виникає потреба в ефективних інструментах для аналізу та контролю наших витрат. Контроль фінансових ресурсів стає дедалі важливішим завданням для багатьох людей, а особливо для тих, хто бажає зберігати фінансову стабільність і планувати свої витрати належним чином.

Мета роботи. Мета кваліфікаційного проекту полягає у розробці мобільного додатку для ведення журналу витрат з використанням React Native. Додаток повинен бути простим у використанні та мати зручний інтерфейс користувача. Основною функцією додатку є можливість ведення журналу витрат і доходів та перегляду звіту про витрати за певний період часу.

Об'єкт дослідження – сучасні технології розробки мобільних додатків.

Предмет дослідження - розробка мобільного додатку засобами React Native.

Завдання, які необхідно виконати:

– спроектувати структуру мобільного додатку.

– розробити мобільний додаток з використанням React Native, що буде простим у використанні та матиме зручний інтерфейс користувача.

– розробити способи захисту даних в мобільному додатку та базі даних.

– провести тестування додатку та виправити помилки, які були виявлені під час тестування.

РОЗДІЛ 1
ЗАГАЛЬНІ ВІДОМОСТІ ПРО РОЗРОБКУ МОБІЛЬНОГО ПЗ

1.1 Аналіз стану проблеми

Додаток для ведення журналу витрат і доходів знадобиться кожній людині, яка має необхідність контролювати та аналізувати власні витрати і доходи. Це можуть бути як особи, які ведуть бізнес, студенти або просто ті, хто хоче більш ефективно керувати власними фінансами.

Для розробки додатку для ведення журналу витрат за допомогою React Native необхідно мати наступне програмне забезпечення:

– операційну систему Windows, Linux або macOS;

– Node.js (версія 10 або вище);

– менеджер пакетів Yarn або npm;

– редактор коду (наприклад, Visual Studio Code або WebStorm).

Додаток для ведення журналу витрат може працювати на будь-якому Android та iOS пристрої, що відповідає системним вимогам.

Додаток для аналізу та контролю витрат і доходів повинен зберігати та обробляти інформацію про фінансові операції користувача. Інформація повинна включати дату витрат, суму, опис, категорію тощо, зберігатися у базі даних та бути доступною для зручного перегляду.

Функціональний опис вихідного об’єкту (додаток для ведення журналу витрат та доходів повинен мати наступний функціонал):

– додавання нових операцій;

– редагування та видалення існуючих операцій;

– перегляд списку операцій та звітів за період;

– пошук та фільтрація операцій за різними параметрами (наприклад, за датою, категорією, описом тощо);

– можливість налаштування категорій операцій;

– автоматичне обчислення сумарних витрат або доходів за вказаний період.

Створення додатку для ведення журналу витрат за допомогою React Native є досить доцільним з точки зору практичності та зручності використання. Завдяки використанню React Native, додаток може розроблятися кросплатформенно, тобто бути запущений на різних пристроях та операційних системах. Це значно зменшує витрати на створення і пришвидшує час розробки.

1.2 Порівняльний аналіз інструментів для контролю витрат і доходів

Розробка додатку для аналізу та контролю витрат з використанням React Native є хорошим та ефективним варіантом, оскільки цей фреймворк дозволяє створювати мобільні додатки для iOS та Android на єдиній кодовій базі, що значно спрощує процес розробки та забезпечує швидке виконання проекту [1].

Перед тим, як перейти до обговорення існуючих аналогів, варто розглянути вибір методів вирішення завдання, що будуть використовуватися в процесі розробки додатку:

– React Native – це фреймворк від Facebook для створення мобільних додатків з використанням JavaScript та React. Його основна перевага полягає в тому, що додатки можуть бути розроблені за допомогою однієї кодової бази для двох основних мобільних платформ - iOS та Android. React Native також надає зручні інструменти для створення інтерактивних елементів у додатку, що є дуже важливим під час створення зручного та ефективного інтерфейсу [2].

– Firebase – це інструмент від Google, який дозволяє розробникам створювати та розгортати мобільні додатки швидко та ефективно. Firebase забезпечує безпеку даних, зручний API та інші необхідні інструменти. У цьому проекті я буду використовувати Firebase Realtime Database – noSQL базу даних від Google [3].

Існуючі аналогічні програми, та їх переваги та недоліки:

– Mint – це додаток для обліку бюджету та контролювання витрат, який доступний для iOS та Android. Він має зручний інтерфейс та можливість імпортувати дані про транзакції з банківських рахунків. Mint також надає

корисну аналітику, яка дозволяє відслідковувати розподіл витрат за категоріями. Однак, Mint (рис. 1.1) не підтримує можливість ведення журналу витрат за кожним днем, тобто не надає можливість детально відслідковувати витрати щодня.

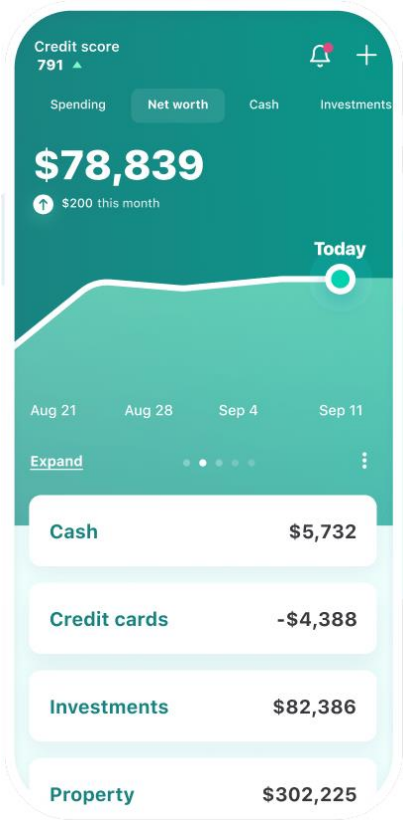

Рисунок 1.1 –Зображення головного меню додатку Mint

– PocketGuard – це ще один додаток для ведення бюджету та контролювання витрат, який також доступний для iOS та Android. PocketGuard має зручний інтерфейс та можливість імпортувати дані про транзакції з банківських рахунків. Він також надає корисну аналітику та можливість налаштувати бюджет для кожної категорії витрат. Однак, PocketGuard (рис. 1.2) не надає можливість детально відслідковувати витрати за кожним днем та не підтримує можливість вручну додавати транзакції.

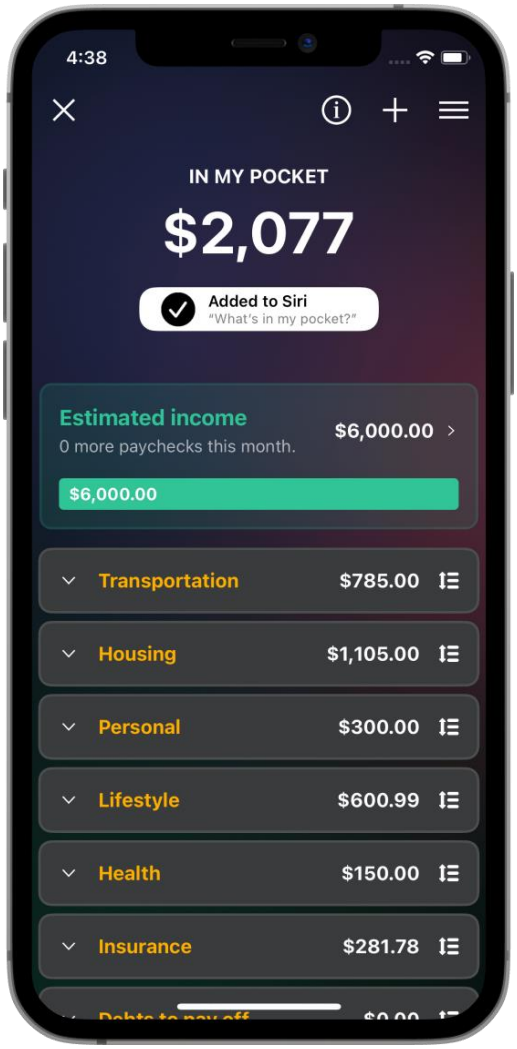

Рисунок 1.2 –Зображення головного меню додатку PocketGuard

– YNAB (You Need A Budget) – це додаток для ведення бюджету та контролювання витрат, який також доступний для iOS та Android. YNAB має дуже детальні налаштування та можливість вести журнал витрат за кожним днем. Він також надає корисну аналітику та можливість налаштувати бюджет для кожної категорії витрат. Однак, YNAB (рис. 1.3) вимагає оплачувати щомісячну підписку.

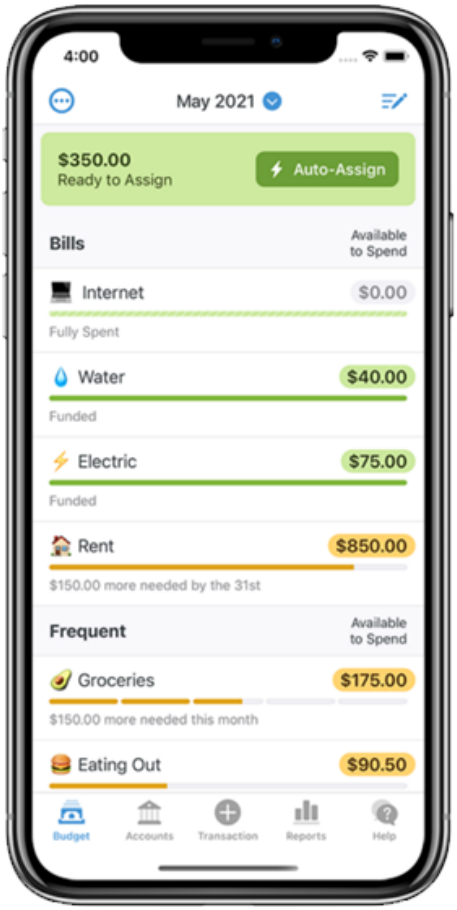

Рисунок 1.3 –Зображення головного меню додатку YNAB

1.3 Вибір засобів розробки

Для розробки додатку для ведення журналу витрат на платформі мобільних пристроїв можна розглянути різні засоби програмування та баз даних. Нижче наведено деякі з можливих варіантів.

Мови програмування:

– JavaScript – мова програмування, яка використовується для створення веб-додатків, а також мобільних додатків за допомогою фреймворка React Native [4, 5].

Бази даних:

– SQLite – легка та компактна база даних, яка добре підходить для мобільних додатків, оскільки забезпечує швидкий доступ до даних і не потребує великої кількості ресурсів. React Native має вбудовану підтримку SQLite.

– Firebase – хмарна платформа, яка надає інструменти для розробки мобільних додатків, зокрема базу даних [6]. Firebase може бути корисним вибором, оскільки дозволяє швидко налаштувати серверну частину додатку.

Апаратно-програмний комплекс:

– розробка мобільних додатків за допомогою React Native може вимагати мінімальних вимог до апаратного забезпечення, оскільки React Native використовує вбудовані компоненти мобільної платформи.

Якщо буде використана база даних Firebase, то потрібне забезпечення для роботи з хмарними сервісами.

React Native – це фреймворк для розробки мобільних додатків, який дозволяє створювати кросплатформні додатки для Android та iOS з використанням JavaScript. Порівняно з розробкою на нативних мовах програмування (Java/Kotlin для Android, та Swift для iOS), React Native має ряд переваг та недоліків [7].

Переваги React Native:

– кросплатформність: з використанням React Native можна створювати додатки для обох платформ Android та iOS, що зменшує час та затрати на розробку додатку.

– швидкість розробки: з використанням React Native можна створювати компоненти для додатку з використанням готових бібліотек та компонентів.

– менша складність: React Native дозволяє розробникам використовувати знайомий для них JavaScript, що зменшує час та затрати на навчання та розробку.

– легкість підтримки: з використанням React Native можна швидко виправляти помилки та оновлювати додаток на обох платформах.

Недоліки React Native:

– обмежені можливості: React Native може не мати певних можливостей, які доступні на нативних мовах програмування.

– відставання від оновлень: React Native може відставати від оновлень платформ Android та iOS, що може призвести до проблем з сумісністю.

Отже, використання React Native для розробки мобільних додатків має як переваги, так і недоліки. Вибір між React Native та нативними мовами програмування залежить від специфіки проекту та його вимог до швидкості, доступності та складності розробки. Для вимог мого проекту, React Native – чудовий вибір.

Для взаємодії з API можна використати бібліотеку Axios або Fetch, які дозволяють отримувати дані з сервера за допомогою запитів HTTP [8].

Для забезпечення зручного та інтуїтивно зрозумілого інтерфейсу користувача можна використати бібліотеку React Native Elements. Вона містить готові компоненти для створення кнопок, текстових полів, списків, діалогових вікон та інших елементів інтерфейсу, що дозволяє значно зменшити час на розробку та підтримку додатку [10].

Для реалізації функціоналу журналу витрат в додатку можна використати базу даних SQLite. Це легкий та швидкий реляційний додатковий двигун бази даних, який може зберігати дані локально на пристрої користувача. SQLite добре підходить для мобільних додатків, оскільки не вимагає великих ресурсів та не потребує сервера.

З іншої сторони, Firebase Realtime Database теж є хорошим вибором для реалізації бази даних в додатку. Вона працює у режимі реального часу та забезпечує швидку та надійну передачу даних між пристроями. Firebase Realtime Database має простий API, який дозволяє легко взаємодіяти з базою даних.

Firebase також надає рішення для автентифікації користувачів, що дозволить забезпечити безпеку та захист даних. Firebase Authentication має декілька методів автентифікації, таких як електронна пошта та пароль, телефонний номер, аутентифікація з використанням соціальних мереж та інші, має можливість забезпечення підтримки для хмарного збереження файлів та інші рішення, такі як, наприклад, Firebase Cloud Messaging, які дозволяють відправляти повідомлення на пристрої користувача.

Firebase Realtime Database та SQLite – це дві різні системи баз даних, які можна використовувати в розробці мобільних додатків.

Firebase Realtime Database є базою даних в реальному часі, яка дозволяє зберігати та синхронізувати дані між різними клієнтами в реальному часі. Для взаємодії з Realtime Database можна використати спеціальну бібліотеку Firebase, яка дозволяє здійснювати запити до бази даних та слідкувати за змінами в даних. Однією з головних переваг Firebase Realtime Database є можливість миттєво показувати зміни в даних на всіх підключених пристроях. Крім того, Firebase Realtime Database має досить великий безкоштовний ліміт на обсяг даних та швидкий час відповіді сервера.

SQLite – це легка та швидка реляційна база даних, яка використовується для зберігання даних локально на пристрої. SQLite має досить простий синтаксис запитів та може працювати з різними типами даних. Однією з головних переваг SQLite є швидкість та ефективність, оскільки вона не потребує підключення до Інтернету. Проте, для доступу до даних, розробникам необхідно вручну писати SQL-запити, що може бути складним для початківців. Також, для створення та збереження резервної копії даних необхідно створювати власні рішення, на відміну від Firebase.

У таблицях 1.1 та 1.2 наведені порівняння баз даних Firebase Realtime Database та SQLite.

Таблиця 1.1 – Порівняння архітектур баз даних Firebase та SQLite

Функції	Firebase Realtime DB	SQLite
Тип бази даних	NoSQL (зберігання даних у вигляді JSON)	SQL (зберігання даних у вигляді таблиць)
Синхронізація даних	Реальний час (real-time)	Локально на пристрої
Доступність даних	Завжди онлайн	Онлайн або офлайн

Продовження таблиці 1.1

Масштабованість	Висока, можливість легко масштабувати	Обмежена залежно від обсягу даних

Швидкість доступу	Висока швидкість доступу до даних в режимі реального часу	Висока швидкість доступу до локальних даних
Крос-платформеність	Підтримується на різних платформах (Android, iOS, Web, Unity)	Підтримується на різних платформах (Android, iOS) та настільних операційних системах (Windows, Linux, MacOS)
Мінімальні вимоги до серверів	Не потрібно власного серверу або бази даних	Потрібен власний сервер або база даних

Таблиця 1.2 – Порівняння можливостей баз даних Firebase та SQLite

Firebase Realtime Database	SQLite
Ідеально підходить для додатків, які вимагають оновлення даних в реальному часі	Підходить для стандартних мобільних додатків з невеликим обсягом даних
Є хмарним рішенням, тому не потрібен власний сервер для зберігання даних	Потребує встановлення на мобільному пристрої та налаштування з'єднання з базою даних
Може бути використаний для розробки додатків, які мають багато користувачів	Підходить для мобільних додатків з низьким рівнем одночасних звернень до бази даних
Не потрібно писати власний код для взаємодії з базою даних	Потребує написання SQL-запитів та власного коду для взаємодії з базою даних
Може бути використаний для зберігання будь-якого типу даних	Підтримує зберігання тільки структурованих даних
Підтримує відслідковування змін, що дозволяє легко оновлювати дані	Не має вбудованих засобів відслідковування змін

Продовження таблиці 1.2

Можливість використовувати Firebase Realtime Database API для взаємодії з даними	Потребує використання сторонніх бібліотек та API для взаємодії з базою даних
Підтримує інтеграцію з іншими сервісами Firebase	Не має вбудованих інструментів для інтеграції з іншими сервісами

Отже, Firebase Realtime Database є кращим вибором, якщо потрібно створити мобільний додаток, який має багато користувачів і потребує оновлення даних в реальному часі. Firebase Realtime Database є хмарним рішенням, тому не потрібно власного сервера для зберігання даних, і він підходить для зберігання будь-якого типу даних.

Перевагою Firebase Realtime Database є зручність в роботі та можливість миттєвої синхронізації даних між декількома користувачами, що використовують додаток. Вона також підтримує аутентифікацію користувачів та забезпечує більш високу захищеність даних у порівнянні з SQLite.

Після аналізу можна зробити висновок про використання таких засобів: для розробки об'єкту проектування буде використана мова програмування JavaScript та фреймворк React Native, оскільки вони дозволяють швидко та ефективно розробляти мобільні додатки для різних платформ. Апаратно-програмний комплекс буде мінімальним, оскільки React Native використовує вбудовані компоненти мобільної платформи. Для збереження даних буде використано Firebase Realtime DB та AsyncStorage (для офлайн зберігання даних). AsyncStorage – це локальне сховище даних у React Native, яке зберігає дані в асинхронному режимі (тобто без блокування основного потоку).

Також, варто згадати, що під час розробки буде використаний такий інструмент, як Expo. Expo – це надбудова над React Native, яка дозволяє швидко створювати та запускати додатки без необхідності у налаштуванні середовища розробки.

Принцип розробки React Native додатків з використанням Expo (рис. 1.4) полягає в створенні проекту за допомогою командного рядка expo-cli, який

автоматично генерує початковий код та налаштовує середовище для розробки. Потім можна виконувати зміни у коді та переглядати результат на симуляторі або пристрої після запуску команди «expo start» [10].

Для запуску розроблених додатків з використанням Expo Go (мобільний додаток-клієнт, через який і відбувається запуск додатку на етапі розробки) потрібно завантажити його на мобільний пристрій і сканувати QR-код, який буде доступний після запуску команди «expo start» [11]. При кожному оновленні коду достатньо лише перезапустити програму Expo Go, щоб отримати нову версію програми.

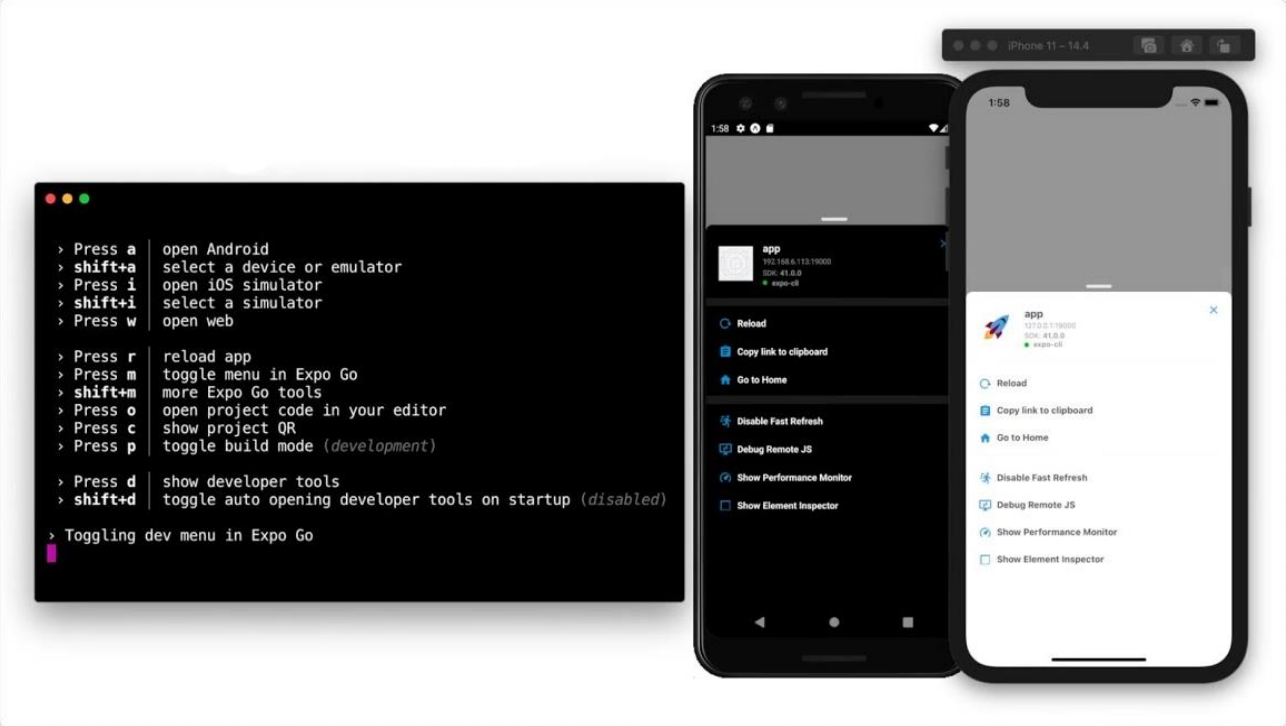

Рисунок 1.4 – Інтерфейс терміналу Expo та мобільного клієнту Expo Go

Додатково, варто відзначити, що перед початком розробки, макет об'єкту проектування був створений в Figma (рис. 1.5). Figma – це онлайн-інструмент для дизайну і прототипування веб-сайтів та мобільних додатків, що дозволяє створювати високоякісні макети і прототипи з використанням вбудованих компонентів та елементів дизайну.

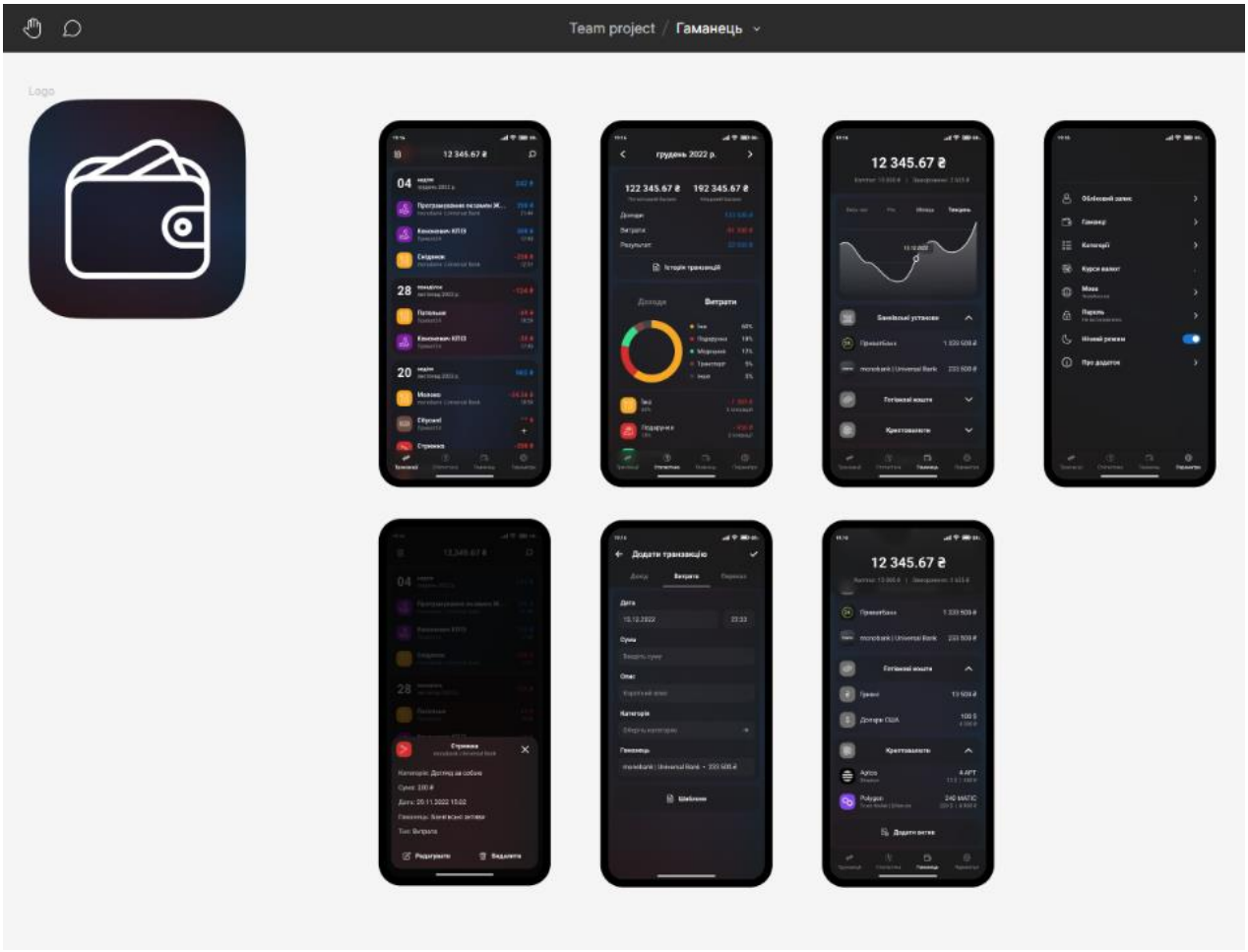

Рисунок 1.5 – Макет додатку у Figma

Створення макету в Figma перед розробкою додатку дозволяє зробити попередню оцінку необхідного функціоналу та інтерфейсу, проаналізувати зручність та зрозумілість розташування елементів і компонентів. Це дозволяє виявити та виправити можливі недоліки та проблеми, що виникають перед початком розробки, що зменшує кількість помилок та забезпечує більш швидку та ефективну розробку додатку.

Розробка мобільного додатку буде здійснюватися у середовищі JetBrains WebStorm (рис. 1.6).

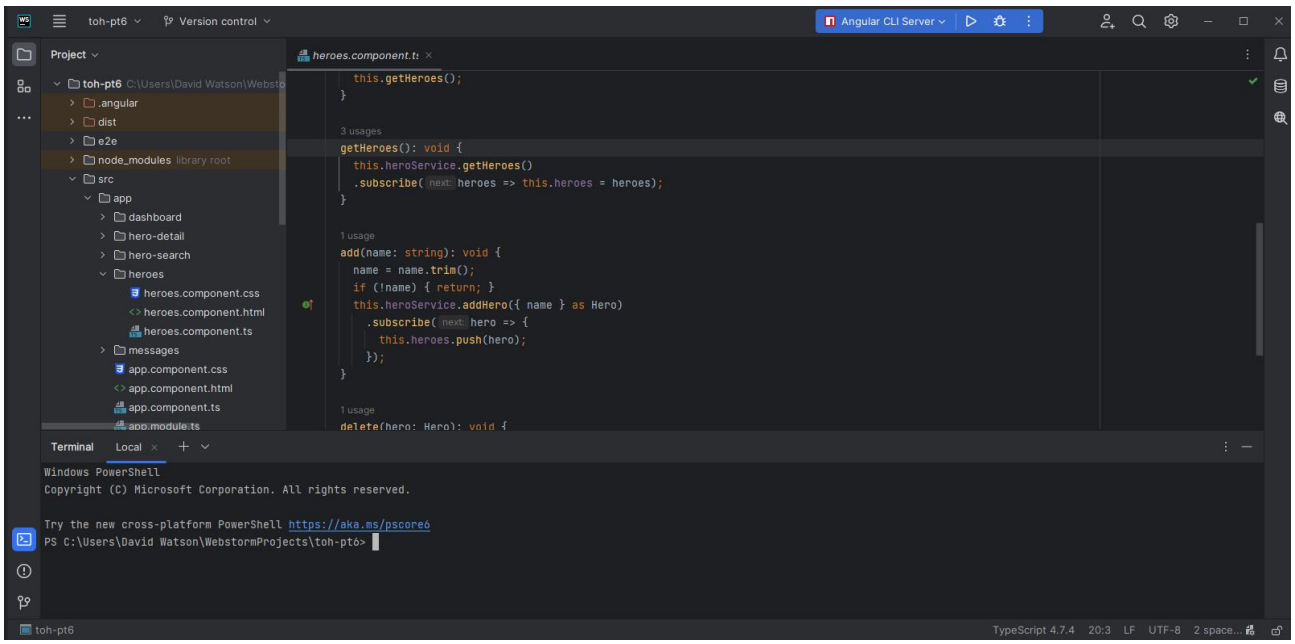

Рисунок 1.6 – Інтерфейс IDE JetBrains WebStorm

WebStorm – це інтегроване середовище розробки (IDE) для JavaScript, що надає широкий спектр функцій та інструментів для покращення продуктивності розробника [12]. WebStorm можна використовувати в тому числі і для розробки мобільних додатків на базі React Native. В IDE доступний повний набір інструментарію, необхідного для ефективної розробки: автодоповнення коду, перевірка синтаксису, вбудований debugger і т.д [13].

РОЗДІЛ 2
ПРОЕКТУВАННЯ СТРУКТУРИ МОБІЛЬНОГО ДОДАТКУ

2.1 Обґрунтування обраних технологій

При розробці додатку для ведення журналу витрат за допомогою React Native було вибрано такі технології та засоби:

– React Native – це крос-платформний фреймворк для розробки мобільних додатків, який дозволяє використовувати JavaScript в якості мови програмування.

– Firebase – це платформа для розробки мобільних та веб-додатків, яка дозволяє зберігати та синхронізувати дані між різними пристроями, а також забезпечує зручний доступ до багатьох сервісів, таких як аутентифікація, зберігання даних та аналітика.

Обрана схема розробки додатку заснована на використанні React Native та Firebase, що дозволяє зменшити час та вартість розробки, полегшити підтримку та забезпечити зручний доступ до додаткових сервісів Firebase. Крім того, React Native є високопродуктивною технологією, оскільки використовує нативні компоненти для рендерингу інтерфейсу користувача.

У порівнянні з іншими технологіями розробки гібридних мобільних додатків, такими як, наприклад, Apache Cordova, React Native має більш високу продуктивність і набагато більш наближений до нативної швидкості виконання додатків. Крім того, React Native має велику та активну спільноту розробників, яка постійно підтримує та розвиває дану технологію.

Для розробки додатку для ведення журналу витрат з використанням React Native було обрано підхід, який базується на використанні мінімального набору зовнішніх засобів, тобто орієнтуванні на стандартний склад технологій та бібліотек, які надає ця технологія. Це дає змогу зменшити залежність від сторонніх інструментів та забезпечити високу стабільність та продуктивність додатку [14].

Отже, React Native був обраний через свої наступні переваги:

– швидкість розробки. Завдяки використанню однієї мови програмування (JavaScript) для обох платформ, розробка додатку відбувається швидше порівняно з розробкою на нативних мовах програмування.

– висока переносимість. Код, написаний з використанням React Native, може бути використаний для створення додатків для інших платформ, або після невеликого рефакторингу, навіть для створення веб-версії. Все це значно зменшує час та витрати на розробку та підтримку.

– гнучкість. React Native дозволяє легко змінювати та оновлювати функціональність додатку, що дозволяє швидко реагувати на зміни вимог користувачів.

– широкі можливості налаштування. React Native дозволяє розробникам розширювати функціональність за допомогою багатьох зовнішніх бібліотек та компонентів.

Для реалізації додатку буде використано ряд додаткових бібліотек та засобів, таких як, наприклад, React Navigation для створення навігації в додатку. Також буде використано засіб Expo для розгортання та тестування додатку під час розробки.

2.2 Забезпечення взаємодії з БД

Для доступу до бази даних у даному проекті використовується набір для розробників Firebase SDK, який надає простий та зручний інтерфейс для роботи з Firebase Realtime DB. SDK дозволяє взаємодіяти з базою даних за допомогою JavaScript API, що спрощує роботу з даними та забезпечує інтеграцію з React Native.

Для використання Firebase Realtime Database у React Native додатках потрібно встановити та налаштувати Firebase SDK. Найпоширенішим вибором є використання пакету react-native-firebase, який надає набір інструментів для взаємодії з Firebase у React Native додатках.

Основні кроки для використання Firebase Realtime Database у React Native додатках:

– встановлення залежностей. Необхідно додати react-native-firebase до залежностей вашого проекту за допомогою менеджера пакетів, такого як npm або yarn. Так як у цьому проекті використовується менеджер пакетів npm, встановлення залежностей буде відбуватися командою npm install react-native-firebase.

– налаштування Firebase проекту. Далі необхідно створити проект у консолі Firebase (https://console.firebase.google.com/) та отримати файл конфігурації, який містить ключі доступу до бази даних.

– ініціалізація Firebase SDK. У React Native додатку потрібно ініціалізувати Firebase SDK з використанням конфігураційного файлу (рис. 2.1).

```
import firebase from 'react-native-firebase';

firebase.initializeApp({
  // ваша конфігурація Firebase
});
```

Рисунок 2.1 – Ініціалізація Firebase

– отримання посилання на базу даних. За допомогою Firebase SDK необхідно отримати посилання на базу даних: const database = firebase.database().

– операції з базою даних. Після ініціалізації можна легко використовувати методи Firebase SDK для збереження, читання та оновлення даних у реальному часі (рис. 2.2, рис. 2.3).

```
// Запис даних
database.ref('users/1').set({
  name: 'John',
  age: 25,
});

// Читання даних
database.ref('users/1').once('value')
  .then(snapshot => {
    const user = snapshot.val();
    console.log(user);
  });
```

Рисунок 2.2 – Запис та читання даних у Firebase

```
// Оновлення даних
database.ref('users/1').update({
  age: 26,
});

// Видалення даних
database.ref('users/1').remove();
```

Рисунок 2.3 – Оновлення та видалення даних у Firebase

Як можна побачити, за допомогою JavaScript як мови програмування можна дуже легко та ефективно взаємодіяти з Firebase Realtime Database, використовуючи методи Firebase SDK, які надаються для взаємодії з базою даних. JavaScript є мовою програмування, що добре підтримується Firebase, і вона дозволяє виконувати різні операції з базою даних, такі як збереження, читання, оновлення та видалення даних, за допомогою вбудованих методів Firebase SDK.

Firebase SDK також включає можливості автентифікації користувачів, що полегшує захист даних та контроль доступу до них. На рисунку 2.4 зображена схема, що демонструє прицнип взаємодії користувача з базою даних.

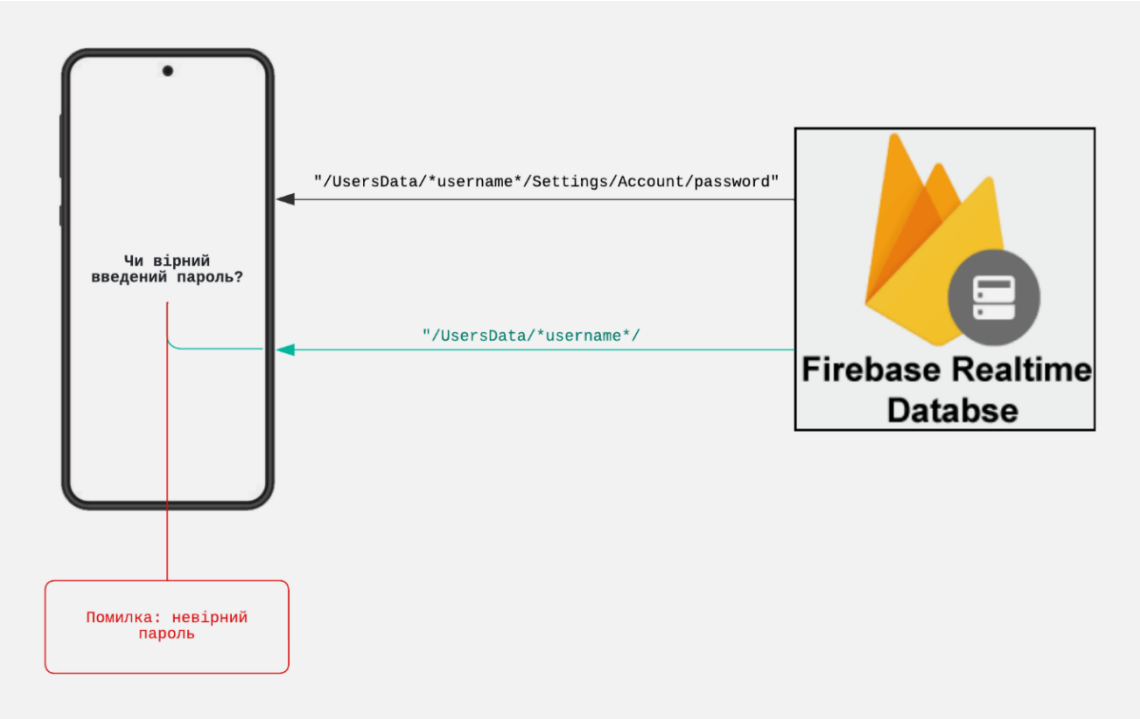

Рисунок 2.4 – Схема автентифікації та авторизації

Під час автентифікації, додаток шифрує введений користувачем пароль, та отримує з бази даних правильний пароль, теж в зашифрованому вигляді. Програма порівнює хеші паролів, та при їх збігу, проводить авторизацію, тобто, отримує з бази даних ще один пакет даних, але вже з даними користувача. У коді за це відповідає функція processLogin() (рис. 2.5).

```
const processLogin = () => {
        const auth = getAuth();
        try {
            signInAnonymously(auth)
                .then(() => {
                    const dataRef = ref(db, "/UsersData/" + login + '/Settings/Account/password');
                    get(dataRef)
                        .then(async (snapshot) => {
                            const data = snapshot.val()
                            try {
                                if (data === null) {
                                    setErrorMsg("Користувач відсутній")
                                } else {
                                    let decrypted = CryptoJS.AES.decrypt(data, 'WALLET').toString(CryptoJS.enc.Utf8);
                                    if (decrypted !== password) setErrorMsg("Невірний пароль")
                                    if (decrypted === password) {
                                        await authFunc();
                                        hide()
                                    }
                                }
                            } catch (e) {
                                console.log(e)
                            }
                        })
                        .catch((error) => {
                            const errorCode = error.code;
                            const errorMessage = error.message;
                            console.log(errorCode, errorMessage)
                        });
                })
                .catch((error) => {
                    const errorCode = error.code;
                    const errorMessage = error.message;
                });
        } catch (e) {
        }
}
```

Рисунок 2.5 – Функція processLogin()

2.3 Створення моделі бази даних

Інформаційна модель бази даних (БД) – це опис даних та взаємозв'язків між ними, який відображає структуру та зміст даних, які зберігаються в БД. Інформаційна модель може бути представлена у вигляді схеми БД, яка включає таблиці, стовпці, ключі та зв'язки між ними. Інформаційна модель бази даних базується на Firebase Realtime DB, яка є NoSQL базою даних, тобто, дані зберігаються в JSON форматі, що пропонує гнучкість та масштабованість для реалізації проекту.

Логічна модель бази даних (БД) – це абстрактне представлення структури даних і відносин між ними та обмежень, що накладаються на дані в базі даних. Логічна модель БД відображає спосіб організації даних, але не залежить від конкретної системи керування базами даних (СКБД) або фізичної реалізації. Це дозволяє фокусуватися на сутностях, атрибутах та відносинах, що представляють бізнес-процеси, без врахування технічних аспектів зберігання та обробки даних. Логічна модель БД зображена за допомогою ERD-діаграми. ERD-діаграма (рис. 2.6) для цієї бази даних представлена у вигляді деревоподібної структури, де кореневим елементом є колекція UserData, що містить одного або необмежену кількість користувачів (Username), внутрішні елементи якого - це різні властивості користувача.

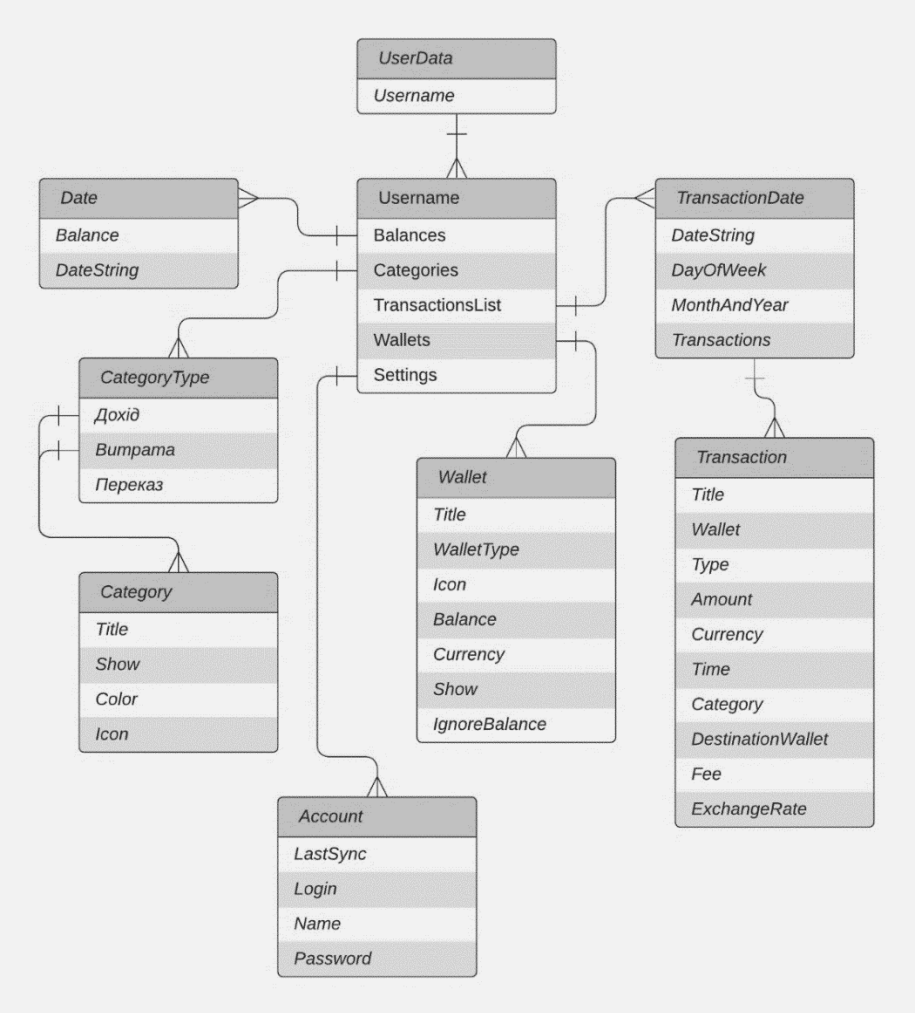

Рисунок 2.6 – ERD діаграма бази даних

Фізична структура БД (рис. 2.7) складається з колекції UserData, яка містить дані користувачів, такі як баланси, категорії, налаштування, список транзакцій та гаманці.

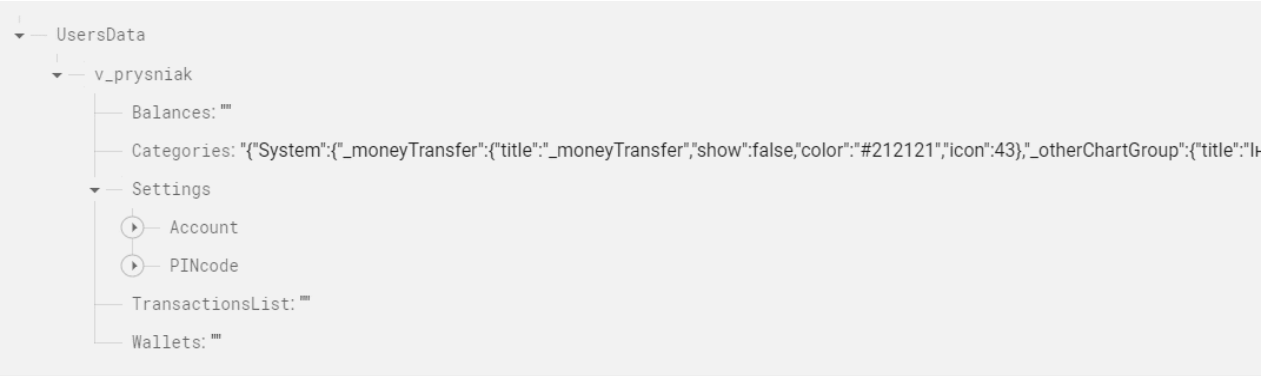

Рисунок 2.7 – Фізична структура БД

Firebase SDK для React Native надає ряд функцій, які дозволяють взаємодіяти з Firebase Realtime Database:

– firebase.database() – інстанс бази даних Firebase;

– .ref(path) – отримання посилання на шлях до вузла бази даних;

– .on(eventType, callback) – підписка на події (зміни) вузла бази даних та їх обробка за допомогою зворотного виклику (callback);

– .off(eventType, callback) – скасування попередньої підписки на події вузла бази даних;

– .set(data) – запис нових або оновлення існуючих значень за заданим шляхом до вузла бази даних;

– .push(data) – додавання нового елемента до списку значень за заданим шляхом до вузла бази даних;

– .remove() – видалення значення за заданим шляхом до вузла бази даних.

Ці методи можна використовувати для читання/запису/оновлення даних та забезпечення синхронізації між БД та клієнтською частиною проєкту.

Сучасні системи керування базами даних можна класифікувати за типом моделі даних, яка підтримується в системі. Основні типи СКБД включають реляційні (наприклад, MySQL, MS SQL Server, Oracle Database, DB2), NoSQL

(наприклад, MongoDB, Firebase Realtime DB, Couchbase) та NewSQL (наприклад, CockroachDB, Google Spanner). Оскільки логічна модель даних та бізнес-логіка обробки вимагають гнучкості та масштабованості, для реалізації даного проекту було обрано NoSQL базу даних Firebase Realtime DB.

Firebase Realtime DB є хмарною NoSQL базою даних, яка забезпечує синхронізацію даних в реальному часі між користувачами та пристроями. Вона пропонує простоту використання, автоматичне масштабування та високу доступність. Технічні характеристики Firebase Realtime DB включають JSON-сховище даних, автоматичне масштабування, відмінні можливості безпеки та підтримку офлайн режиму.

Для збереження даних локально, безпосередньо на пристрої користувача, для доступу в режимі офлайн, використовуватиметься AsyncStorage (вбудована локальна пам’ять в React Native) (рис. 2.8).

```
import { AsyncStorage } from 'react-native';

// Запис значення
async function saveData() {
  try {
    await AsyncStorage.setItem('key', 'value');
  } catch (error) {
    console.log(error);
  }
}

// Отримання значення
async function getData() {
  try {
    const value = await AsyncStorage.getItem('key');
    if (value !== null) {
      console.log(value);
    }
  } catch (error) {
    console.log(error);
  }
}
```

Рисунок 2.8 – Приклад роботи з AsyncStorage

Для роботи з ним, його необхідно імпортувати з бібліотеки “react-native”. Зберігати дані можна за допомогою методу setItem(key: string, value: string), а читати – getItem(key: string). Додатково є метод removeItem(key: string) для вилучення запису.

РОЗДІЛ 3
РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Створення інтерфейсу користувача

Структурно, додаток має 4 основних екрани та додаткові вікна. Графічно, повна структура додатку зображена на рисунку 3.1.

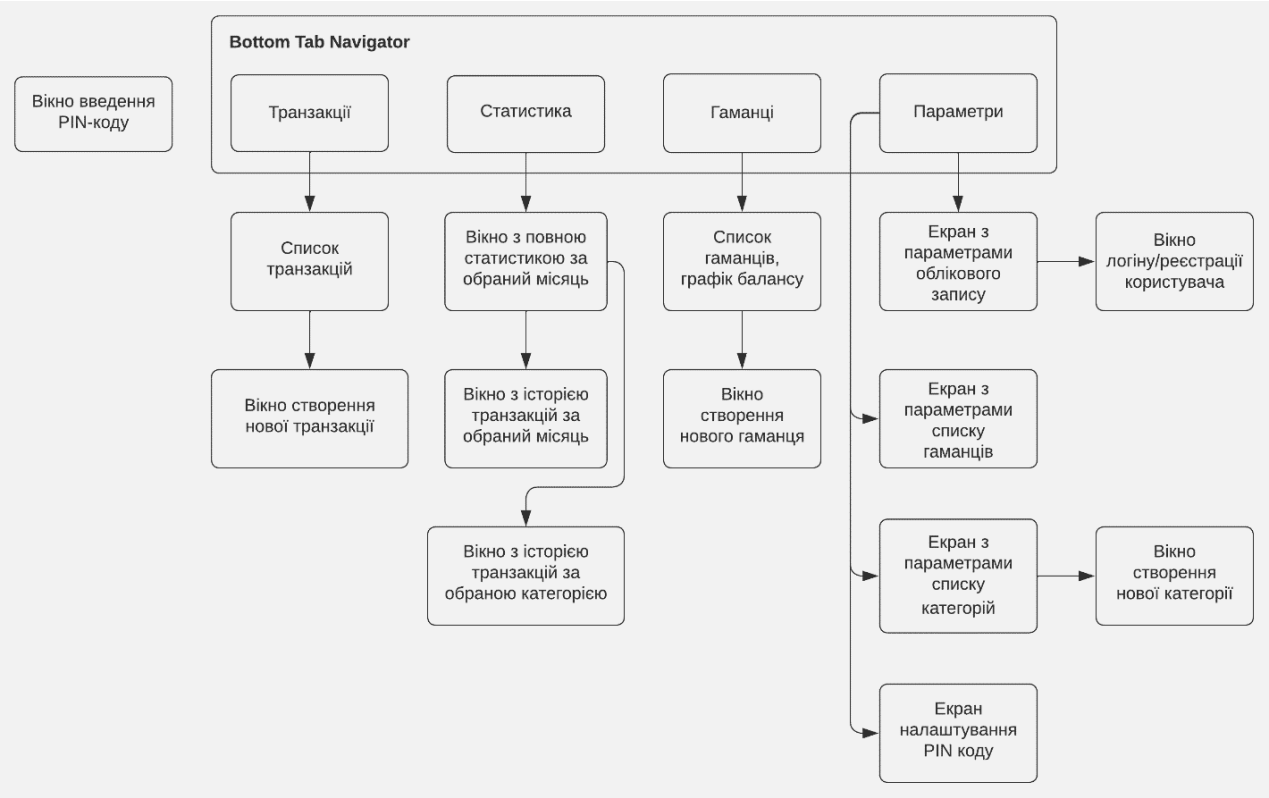

Рисунок 3.1 – Графічне зображення структури додатку

На рисунку 3.2 представлений JSX код, який відповідає за створення одного з ключових елементів інтерфейсу користувача – нажньої панелі навігації (Bottom Tab Navigator). Цей код створює константу Tab, яка містить функцію createBottomTabNavigator(), яка дозволяє створити нижню навігаційну панель з вкладками для мобільного додатку. Далі експортується компонент BottomTabs, який є функціональним компонентом, що повертає розмітку, яка містить Tab.Navigator з використанням screenOptions зі стилями styles.TabNavigator [15].

```
const Tab = createBottomTabNavigator()

export default BottomTabs = () => {
    return (
        <Tab.Navigator screenOptions={styles.TabNavigator}>
            <Tab.Screen name="Транзакції" component={Transactions} options=
{styles.tabBarOptions} />
            <Tab.Screen name="Статистика" component={Stats} options=
{styles.tabBarOptions} />
            <Tab.Screen name="Гаманці" component={Wallet} options=
{styles.tabBarOptions} />
            <Tab.Screen name="Параметри" component={Settings} options=
{styles.tabBarOptions} />
        </Tab.Navigator>
    )
}

export default function App() {
    NavigationBar.setBackgroundColorAsync("#1A191F");

    return (
        <NavigationContainer>
            <BottomTabs/>
            <StatusBar backgroundColor={"#1A191F"} barStyle={"light-content"}/>
        </NavigationContainer>
    );
}
```

Рисунок 3.2 – Код для створення нижньої панелі навігації

Компонент BottomTabs використовується з пакетом react-navigation для створення нижньої навігаційної панелі з вкладками. Він є функціональним компонентом, який повертає розмітку з використанням Tab.Navigator та Tab.Screen. Конкретно у цьому випадку, BottomTabs містить чотири екрани: «Транзакції» (додаток А) (рис. 3.3), «Статистика» (додаток Б) (рис. 3.4), «Гаманці» (додаток В) (рис. 3.5) і «Параметри» (додаток Г) (рис. 3.6). Кожен екран представлений в Tab.Screen з властивостями name (назва), component (компонент, який буде показуватись на цьому екрані) і options (настройки таб-бару). Все це розміщено всередині NavigatorContainer, що обгортає BottomTabs і StatusBar, та який повертається функцією App – головною функцією в React Native. Саме її вміст відображається під час запуску додатку [16].

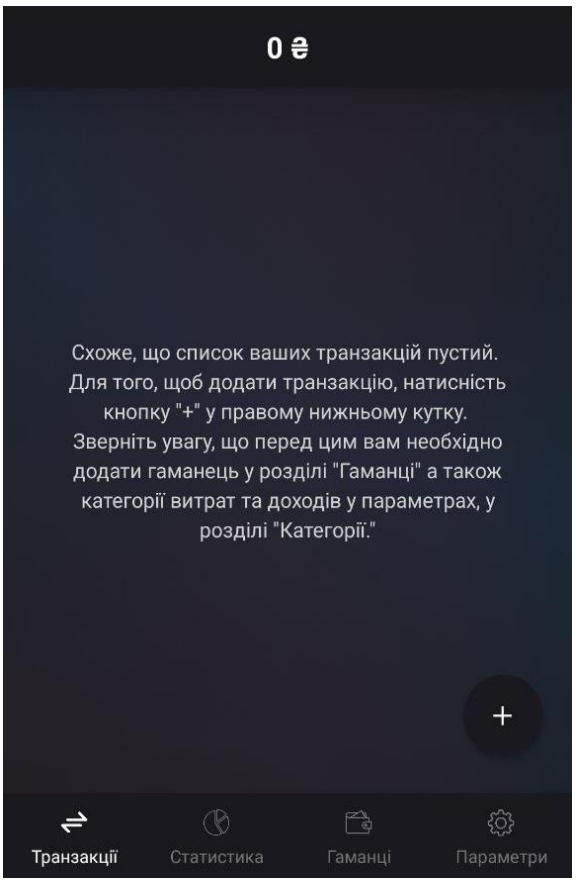

Рисунок 3.3 – Екран «Транзакції»

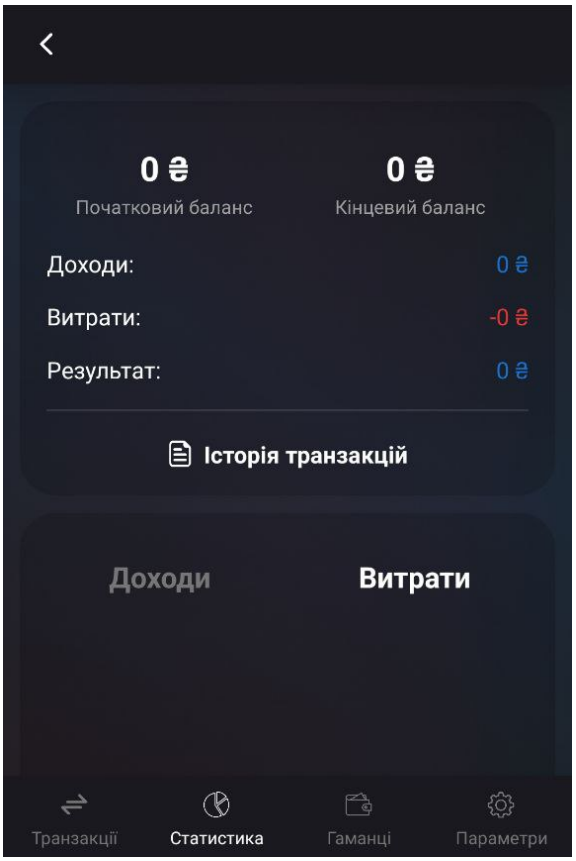

Рисунок 3.4 – Екран «Статистика»

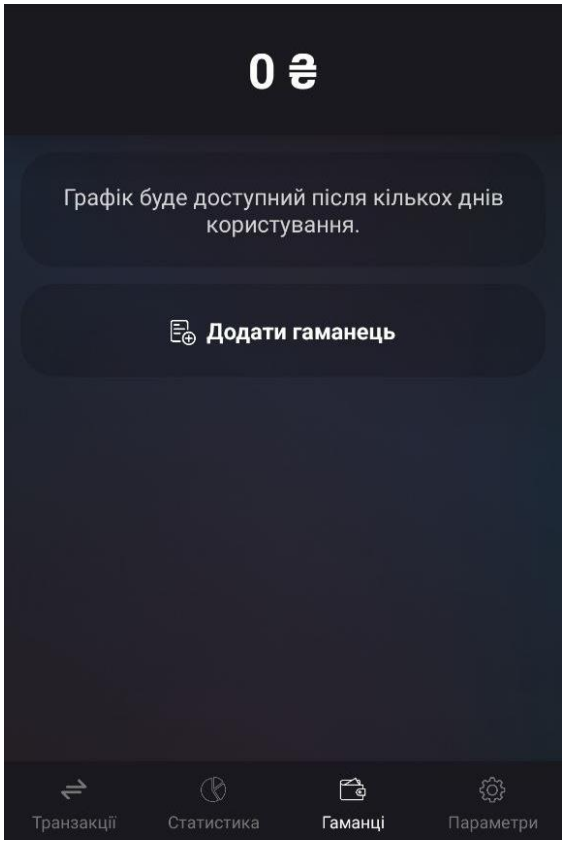

Рисунок 3.5 – Екран «Гаманці»

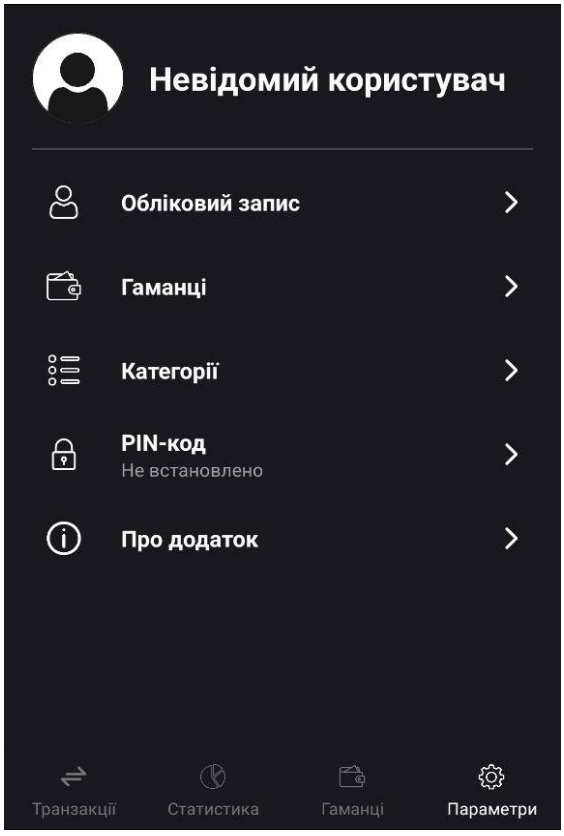

Рисунок 3.6 – Екран «Параметри»

Загалом, більшість компонентів додатку мають схожу між собою JSX структуру (рисунок 3.7). Загальний синтаксис компоненту в React Native складається з двох основних частин: декларації та рендерингу [17].

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';

const MyComponent = () => {
    return (
      <View>
        <Text>Hello World!</Text>
      </View>
    );
}

export default MyComponent;
```

Рисунок 3.7 – Загальний синтаксис компоненту у React Native

У цьому прикладі ми імпортуємо необхідні модулі та створюємо функцію MyComponent, яка повертає JSX-елемент, який буде виведений на екран.

Потім можна імпортувати цей компонент у батьківський файл та викликати його за допомогою `<MyComponent />`. Це загальний шаблон для створення компонента. Залежно від потреб проекту можуть бути додані різноманітні функції, розширення і т.д.

В основному, компоненти використовують практично ті ж самі засоби. Ось основні з них: функції useState(), useEffect(), AsyncStorage.getItem(), AsyncStorage.setItem() та useIsFocused(), а також інші засоби, такі Modal компонент з "react-native-modal" бібліотеки, компонент TouchableOpacity для створення кнопок, об'єкт Dimensions для отримання розміру екрану пристрою, та власноруч створені функції та скрипти, як наприклад функція monthsFormated для форматування масиву даних про місяць у вигляд строкових значень, функція fillEmptyDates для заповнення порожнього переліку дат [18]. Також використовується імпорт файлу colorPalette.js з палітрою кольорів та categoriesIcons.js з набором іконок категорій.

Список стандартних елементів, що зазвичай використовуються при створенні компоненту у React Native:

– View – це основний контейнер для розміщення інших елементів.

– Text – використовується для виведення тексту на екран.

– Image – дозволяє вставляти зображення на екран.

– TextInput – дозволяє користувачам вводити текст.

– Button – натискання кнопки запускає певний код або функцію.

– ScrollView/FlatList – дозволяють прокручувати багатострінковий контент по вертикалі або горизонталі.

Це лише кілька прикладів, оскільки можна створювати свої власні компоненти та складати їх з цих стандартних елементів.

Під час запуску, за допомогою функції _retrieveData() (рис. 3.8) відбувається завантаження даних з AsyncStorage, які зберігаються там у форматі JSON.

```
const _retrieveData = async () => {
        try {
            let data = {}
            data = JSON.parse(await AsyncStorage.getItem('TransactionsList'));
            if (data == null) {
                await AsyncStorage.setItem('TransactionsList', JSON.stringify({}))
                data = JSON.parse(await AsyncStorage.getItem('TransactionsList'));
            }
            saveData(data)
            try {
                let data = JSON.parse(await AsyncStorage.getItem('Categories'))
                setCategories(data)
            } catch (error) {
                console.error(error)
            }
            if (!pinLoginScreen) {
                try {
                    let data = JSON.parse(await AsyncStorage.getItem('Settings'))
                    showPinCode(data.PINcode.set)
                    setPinCode(data.PINcode.value)
                } catch (error) {
                    console.error(error)
                }
                setPinLoginScreen(true)
            }
        } catch (error) {
            console.error(error)
        }
    }
}
```

Рисунок 3.8 – Функція для отримання даних

Цей код виконує ряд дій для отримання та збереження даних з використанням AsyncStorage у React Native додатку. Спочатку функція _retrieveData намагається отримати дані з ключем 'TransactionsList' з AsyncStorage. Якщо дані існують, вони розпаковуються з формату JSON за допомогою JSON.parse і присвоюються змінній data [19]. Якщо значення data є null (тобто дані з ключем 'TransactionsList' ще не були збережені), то функція збереже порожній об'єкт {} з ключем 'TransactionsList' у AsyncStorage за допомогою AsyncStorage.setItem. Після цього викликається функція saveData з параметром data для подальшої обробки отриманих даних.

Далі функція спробує отримати дані з ключем 'Categories' з AsyncStorage та розпакувати їх з формату JSON. Отримані дані присвоюються змінній data і передаються у функцію setCategories для подальшої обробки.

Аналогічно, з налаштуваннями користувача: функція спробує отримати дані з ключем 'Settings' з AsyncStorage та розпакувати їх з формату JSON. Отримані дані присвоюються змінній data. Якщо значення pinLoginScreen є false, функція спробує отримати значення пін-коду з отриманих даних data. Булеве значення наявності пін-коду передається у функцію showPinCode для подальшого відображення пін-коду на екрані. Текстове значення пін-коду також присвоюється змінній pinCode. Наостанок, значення pinLoginScreen встановлюється на true для переходу на екран пін-коду [20].

Цей код використовує хук useEffect (рис. 3.9) для визначення побічних ефектів. Використовується залежність [isFocused], яка означає, що код буде виконуватись лише при зміні значення isFocused (коли екран знаходиться безпосередньо у полі зору).

```
React.useEffect(() => {
        if (isFocused) {
                getCurrentDate()
                checkStorageKeys()
                onRefresh();
        }
}, [isFocused]);
```

Рисунок 3.9 – Хук useEffect

У даному випадку, коли компонент, до якого відноситься цей код, отримує фокус, будуть виконані функції getCurrentDate, checkStorageKeys та onRefresh (рис. 3.10).

```
const onRefresh = async () => {
        setRefreshing(true);
        await _retrieveData()
        await getBalance()
        await saveBalance()
        await fillBalancesData()
        setRefreshing(false);
};
```

Рисунок 3.10 – Функція onRefresh

Як вже було згадано у розділі 2, для доступу до Firebase Realtime Database, у Firebase SDK використовується API ключ. У коді його можна побачити один раз, у файлі конфігурації dbConfig.js, у якому ініціалізується база даних (рис. 3.11).

```
import firebase from "firebase/compat";
import {getDatabase} from "firebase/database"
import {getStorage} from "firebase/storage"

const firebaseConfig = {
    apiKey: "***",
    authDomain: "***",
    projectId: "***",
    storageBucket: "***",
    messagingSenderId: "***",
    appId: "***",
    databaseURL: '***/'
};

if (firebase.apps.length === 0) {
    firebase.initializeApp(firebaseConfig)
}


const db = getDatabase()
const storage = getStorage()
export {db, storage}
```

Рисунок 3.11 – Ініціалізація Firebase Realtime DB

3.2 Способи захисту даних

Загалом принцип захисту даних у додатку полягає у шифруванні пароля за допомогою CryptoJS перед його збереженням до бази даних, а також перевірці правильності пароля при авторизації, перед отриманням користувачем доступу до своїх даних (рис. 3.12) [21].

```
const auth = getAuth();
        try {
            signInAnonymously(auth)
                .then(() => {
                    const dataRef = ref(db, "/UsersData/" + login +
'/Settings/Account/password');
                    get(dataRef)
                        .then(async (snapshot) => {
                            const data = snapshot.val()
                            try {
                                if (data === null) {
                                    setErrorMsg("Користувач відсутній")
                                } else {
                                    let decrypted = CryptoJS.AES.decrypt(data,
'WALLET').toString(CryptoJS.enc.Utf8);
                                    if (decrypted !== password) setErrorMsg("Невірний пароль")
                                    if (decrypted === password) {
                                        await authFunc();
                                        hide()
                                    }
                                }
                            } catch (e) {
                                console.log(e)
                            }
                        })
                        .catch((error) => {
                            const errorCode = error.code;
                            const errorMessage = error.message;
                            console.log(errorCode, errorMessage)
                        });
                })
                .catch((error) => {
                    const errorCode = error.code;
                    const errorMessage = error.message;
                });
        } catch (e) {
        }
}
```

Рисунок 3.12 – Лістинг коду, що відповідає за процес автентифікації

Цей код шифрує введений пароль користувача засобами бібліотеки CryptoJS, використовуючи заданий ключ. При спробі зареєструватися в додатку, якщо введені паролі співпадають, код перевіряє наявність користувацьких даних у базі даних Firebase. Якщо такі дані є, показує помилку (showUserError(true)), інакше реєструє нового користувача та приховує форму реєстрації (register()

hide()). Якщо ж введені паролі не збігаються - показуються повідомлення про помилку (showPasswordError(true)) [22].

У додатку також присутня можливості встановлення PIN-коду. Це буде запобігати несанкціонованому доступу до даних. Реалізований PIN-код у вигляді екрану з клавіатурою, що показується при запуску додатку, порівнюючи введений код зі збереженим і пам’яті пристрою (рис. 3.13).

Рисунок 3.13 – Екран введення PIN-коду

3.3 Тестування та налагодження

Так як React Native є фреймворком для розробки мобільних додатків, який використовує JavaScript та React, то для тестування та налагодження програми, створеної за допомогою React Native, можна використовувати наступні методи:

– unit-тести – це тести, які перевіряють окремі частини коду (функції або компоненти) ізольовано від інших частин програми.

– інтеграційні тести – це тести, яким проводяться перевірки працездатності системи у зведеному вигляді.

– ui-тести – це автоматизоване тестування користувацького інтерфейсу програми.

– налагодження (debugging): процес пошуку помилок у коді і їх локалізація.

Для полегшення процесу налагодження можна скористатися Reactotron (рис. 3.14) – це набір інструментів для розробників React Native / Redux, призначений для спостереження за станом програми під час розробки. Він дає змогу швидко налагодити ваш код завдяки своїм функціональним можливостям:

– Logging – запис подій і станів програми.

– Networking – відстеження запитів до серверів.

– Async Storage – читання/запис даних зі сховища Async Storage е.

– Custom Actions – створення подій та обробників подій.

– Benchmarking – вимірювання часу виконання певних операцій.

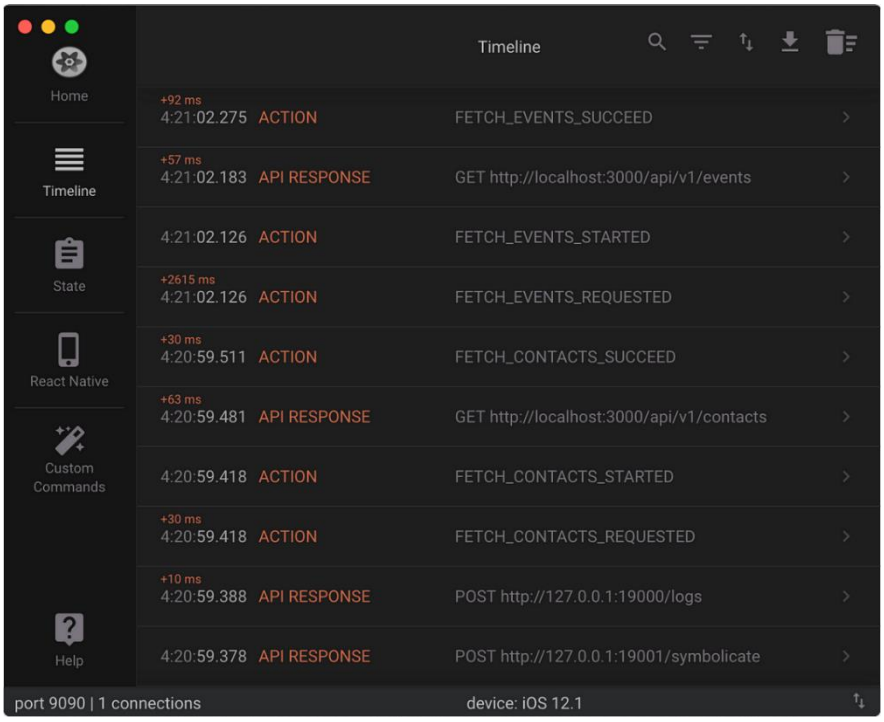

Рисунок 3.14 – Інтерфейс Reactotron

Загальний правильний план тестування та налагодження програми, створеної за допомогою React Native, включає в себе наступні етапи: написання unit-тестів для окремих частин коду, інтеграційне тестування працездатності

зведеного вигляду, UI-тестування користувацького інтерфейсу додатку та виправлення помилок [23].

3.4 Інструкція користувача з використання розробленого додатку

Створений додаток буде розповсюджуватися у вигляді .apk файлу, так як для завантаження додатку в Google Play необхідно створювати платний аккаунт розробника. Для того, щоб повноцінно почати користуватися додатком, користувачу необхідно виконати наступні кроки:

1. Завантажити .apk файл додатку на свій пристрій.

2. У разі потреби, увімкнути можливість установки стороннього ПЗ на пристрої (Налаштування > Безпека > Невідомі джерела).

3. Встановити додаток, натискаючи на файл або за допомогою будь-якої програми керування файлами.

4. При першому запуску, надати необхідні права доступу до функціоналу вашого пристрою.

5. Налаштувати обліковий запис користувача (рис. 3.15, рис. 3.16, рис. 3.17, рис. 3.18): створити новий або авторизуватися.

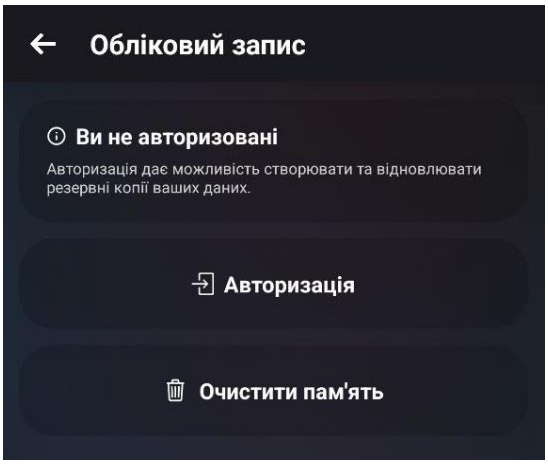

Рисунок 3.15 – Екран «Обліковий запис»

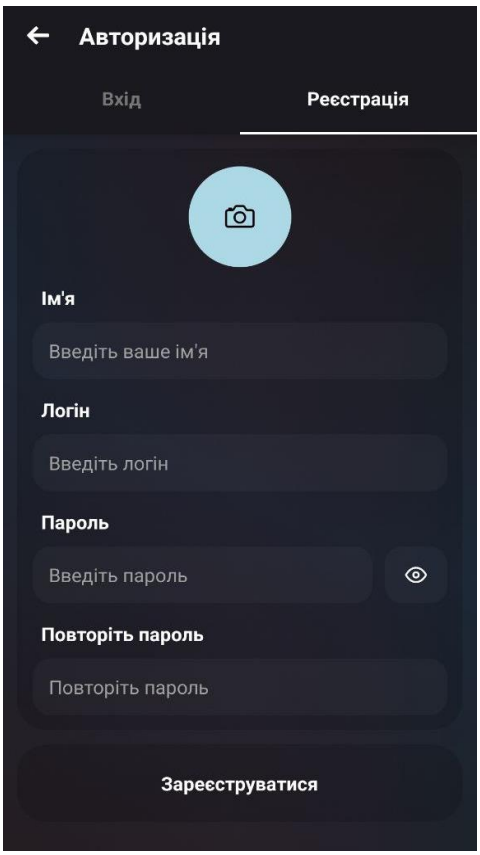

Рисунок 3.16 – Екран реєстрації користувача

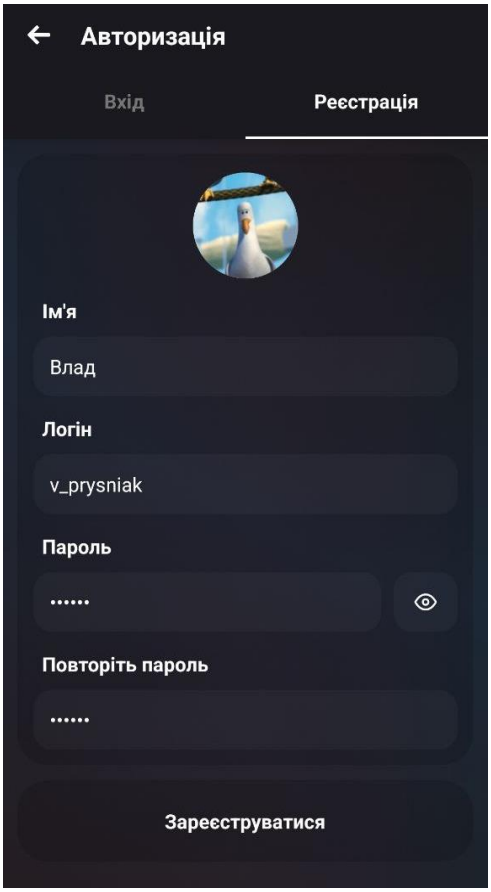

Рисунок 3.17 – Вигляд заповненого облікового запису користувача

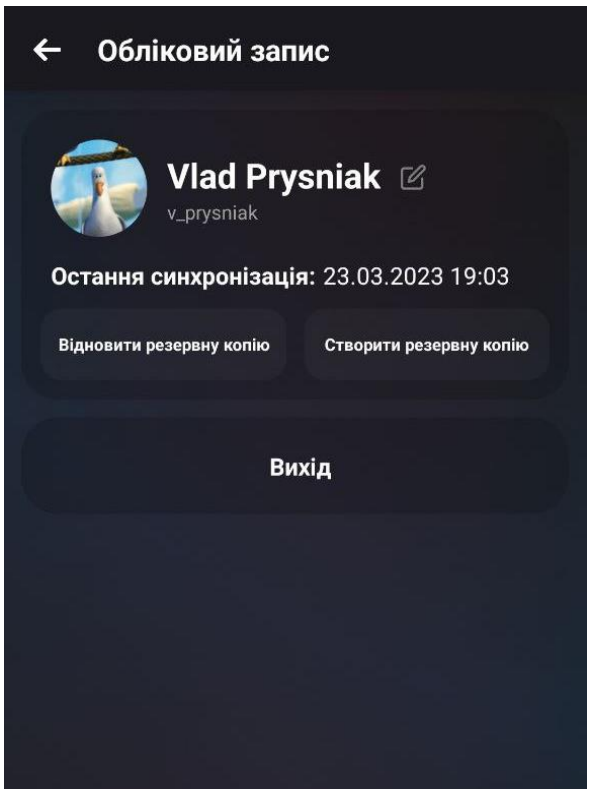

Рисунок 3.18 – Вигляд облікового запису після авторизації

6. Створити новий гаманець, обрати його тип, дані про баланс, а також обрати іконку (рис. 3.19).

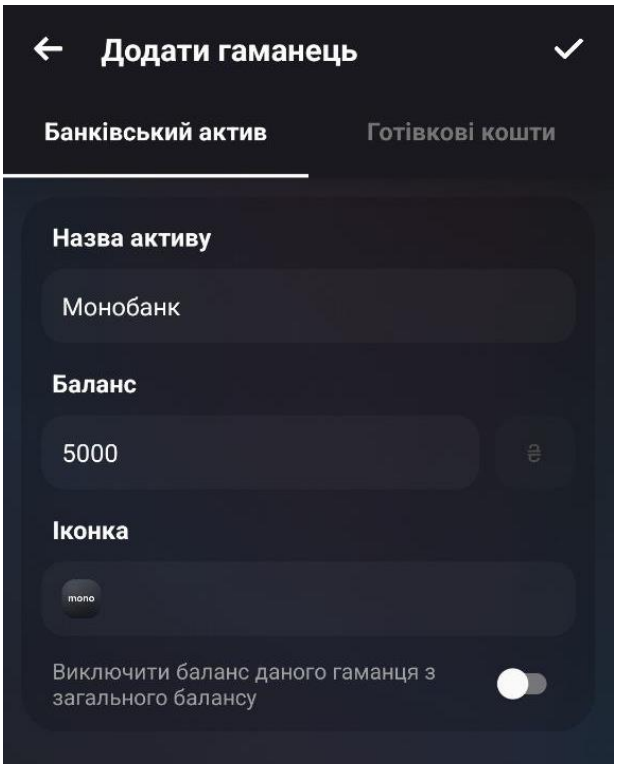

Рисунок 3.19 – Екран створення нового гаманця

7. Для додавання нового запису на екрані «Транзакції» натиснути кнопку "+" у правому нижньому кутку екрану.

8. Обрати тип операції та заповнити усі необхідні поля (рис. 3.20).

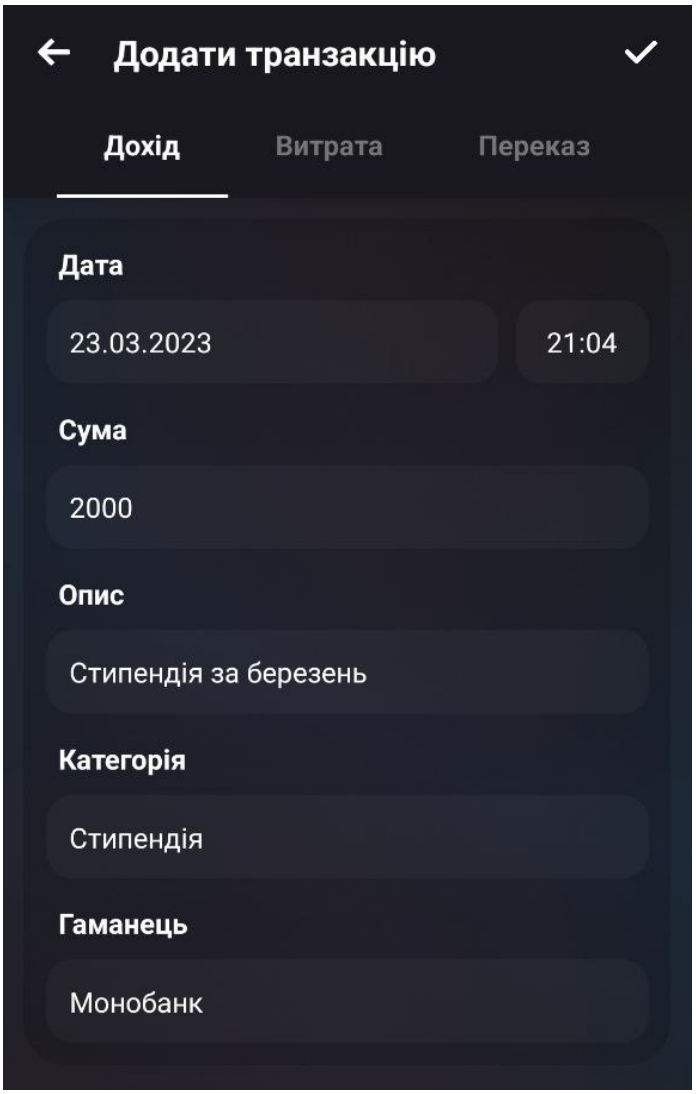

Рисунок 3.20 – Екран створення нового запису

9. Наступним кроком є завершення формування запису шляхом натискання кнопки "Готово". Створений запис тепер відображається у списку транзакцій (рис. 3.21).

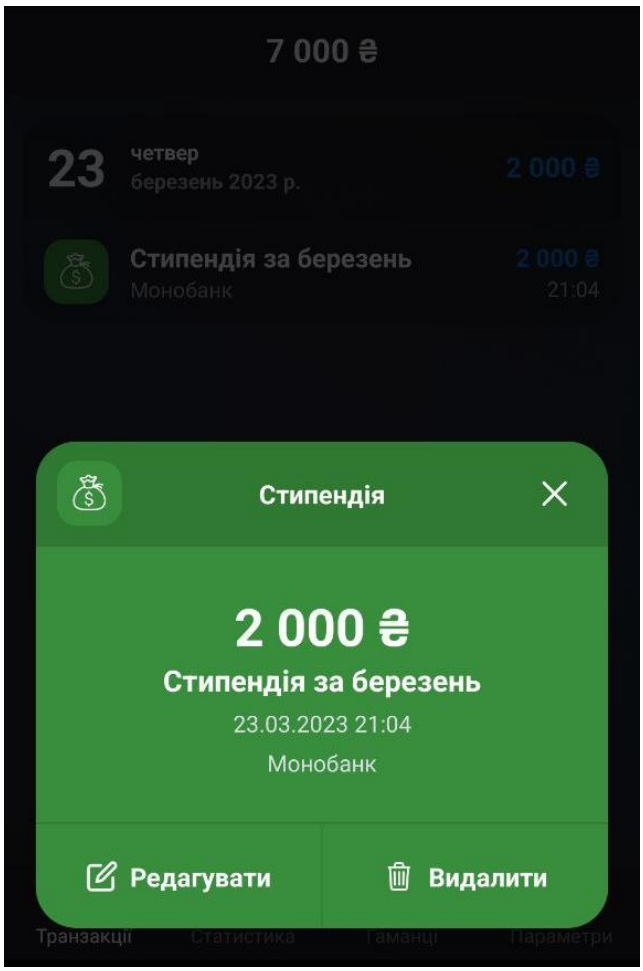

Рисунок 3.21 – Новостворений запис про дохід

Як можна побачити, додаток має зручний інтерфейс користувача та інтуїтивно зрозумілу логіку взаємодії, а тому його можна легко вважати зручним інструментом для аналізу власних витрат та доходів.

ВИСНОВКИ

У даній кваліфікаційній роботі була проведена розробка крос-платформенного мобільного додатку з використанням React Native для аналізу та контролю витрат. Для досягнення цієї мети були використані наступні засоби: база даних Firebase, JavaScript, фреймворк React Native, IDE WebStorm та платформа для розробки React Native додатків Expo.

У роботі був проведений аналіз стану проблеми, включаючи порівняльний аналіз інструментів для контролю витрат і доходів. Також були обрані необхідні засоби розробки, що відповідають вимогам проекту, обґрунтовані обрані технології та описані принципи взаємодії з базою даних, та була представлена схема та створена модель бази даних, що використовується в додатку.

Під час розробки була спроектована структура додатку, створений інтерфейс користувача, розроблена логіка та принципи роботи додатку. Також були виконані заходи для захисту даних, проведено тестування та налагодження системи, та надана інструкція користувача з використання розробленого додатку.

Отже, можна зробити висновок, що використання React Native та інших сучасних технологій дозволяє легко, швидко та ефективно створювати крос-платформні додатки різних рівнів складності. В результаті виконання кваліфікаційної роботи були досягнуті поставлені цілі та отримані практичні результати, які можуть бути використані для подальшого розвитку та вдосконалення мобільного додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is a Native Mobile App Development? Mdevelopers. URL: https://mdevelopers.com/blog/what-is-a-native-mobile-app-development. (дата звернення: 04.01.2023)

2. Мобільні додатки на React Native. Arttimeweb. URL: https://arttimeweb.com/rozrobka-mobilnyh-dodatkiv/mobilni-dodatky-na-react-native/ (дата звернення: 07.01.2023).

3. Документація Firebase. Firebase by Google. URL: https://firebase.google.com/docs. (дата звернення: 15.01.2023).

4. Документація JavaScript. Developer.mozilla.org. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript. (дата звернення: 19.01.2023).

5. Як створити мобільний додаток за допомогою JS. Шлях React Native. URL: https://ab-soft.net/uk/yak-stvoriti-mob%D1%96lnij-dodatok-za-dopomogoyu-js-shlyah-react-native/ (дата звернення: 04.02.2023).

6. Firebase: Аналітика для мобільних додатків. URL: https://brander.ua/technologies/firebase (дата звернення: 09.02.2023).

7. Заполовський М. Й., Петровська І. Ю., Мітяєв А. С. Розроблення та дослідження мобільного додатку на основі фреймворку React Native. Проблеми інформатизації : матеріали міжнар. наук. конф., м. Київ, 10 груд. 2021 р. Харків, 2021. С. 23.

8. Вербівський Д. С., Поліщук М., Ярмоленко Т. Графічні можливості JavaScript. Інформаційно-комунікаційні технології в освіті та науці. 2021. № 1. С. 132–135.

9. React ‘ Native Internals. React Native Guide. URL: https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html. (дата звернення: 12.02.2023)

10. Документація Expo. Expo.dev. URL: https://docs.expo.dev/. (дата звернення: 18.02.2023).

11. React Native Init vs Expo 2022: What Are the Differences? Fulcrum. URL: https://fulcrum.rocks/blog/react-native-init-vs-expo. (дата звернення: 25.02.2023)

12. Getting started with WebStorm. JetBrains. URL: https://www.jetbrains.com/help/webstorm/getting-started-with-webstorm.html. (дата звернення: 02.03.2023).

13. Top Languages Used in 2022. Github. URL: https://octoverse.github.com/2022/top-programming-languages. (дата звернення: 05.03.2023)

14. Розробка мобільних додатків на React Native. Brander. URL: https://brander.ua/what-we-offer/application-development/rozrobka-mobilnykh-dodatkiv-na-react-native (дата звернення: 10.03.2023).

15. Документація React Native. Introduction – React Native. URL: https://reactnative.dev/docs/getting-started. (дата звернення: 15.03.2023).

16. React Lifecycle of Components. GeeksForGeeks. URL: https://www.geeksforgeeks.org/reactjs-lifecycle-components/. (дата звернення: 21.03.2023)

17. React Native - що це і як його освоїти? Початок роботи з бібліотекою React Native. URL: https://apeirondb.com/ua/blog/it-tehnologii-react-native (дата звернення: 28.03.2023).

18. Неділько І. М. Розробка мобільного фітнес застосунку засобами React Native та Node.js : кваліфікаційна робота магістра спеціальності 121 "Інженерія програмного забезпечення" / наук. керівник А. Г. Кривохата. Запоріжжя : ЗНУ, 2022. 57 с.

19. Розробка мобільних додатків на React Native. Artjoker. URL: https://artjoker.ua/uslugi/razrabotka-mobilnykh-prilozheniy-na-react-native/ (дата звернення: 04.04.2023).

20. Кунденко П.Р Дослідження ефективності фреймворку React Native при розробці мобільних додатків : кваліфікаційна робота магістра спеціальності 122 "Комп’ютерні науки" / наук. керівник Л. І. Мещеряков. Дніпро: Національний технічний університет «Дніпровська політехніка», 2022. 119 с.

21. Глущук А. І., Баленко О. І. JavaScript як мова розробки мобільних додатків. Проблеми інформатики та моделювання. Харків, 2022. 1 с.

22. Потапенко Ю. Чи безпечні додатки на React Native у порівнянні з нативними. DOU. URL: https://dou.ua/forums/topic/34677/ (дата звернення: 05.05.2023).

23. Бавикін Д. О., Сологуб О. І., Тодоріко Є. С. Фрейморк React Native як засіб для розробки кроссплатформенних додатків. Сучасні інформаційні технології : Матеріали дев'ятої міжнар. наук. конф. студентів та молодих вчен., м. Одеса. Одеса, 2019. С. 53–54.

ДОДАТКИ

Додаток А

Лістинг коду файлу Transactions.js (екран «Транзакції»)

```
import React from "react";
import {View, ImageBackground, StyleSheet, Dimensions, FlatList,
TouchableOpacity, RefreshControl, Text} from "react-native";
import TransactionsHeader from
"../components/transactionsScreen/TransactionsScreenHeader";
import TransactionCard from "../components/transactionsScreen/TransactionCard"
import AsyncStorage from "@react-native-async-storage/async-storage";
import FAB from "../components/transactionsScreen/FAB"
import {useIsFocused} from "@react-navigation/native";
import {monthsFormated} from "../data/calendarData";
import {fillEmptyDates} from "../components/Tools";
import PINcodeScreen from "./PINcode";
import colors from "../data/colorPalette";
import categoriesIcons from "../assets/icons/categories/categoriesIcons";

const Transactions = () => {
    const [isShownPINcode, showPinCode] = React.useState(false)
    const [pinLoginScreen, setPinLoginScreen] = React.useState(false)
    const [pinCode, setPinCode] = React.useState("")
    const [currentDate, setDate] = React.useState("")
    const [currentDayOfWeek, setDayOfWeek] = React.useState()
    const [transactionsData, setTransactionsData] = React.useState()
    const [refreshing, setRefreshing] = React.useState(false);
    const [balance, setBalance] = React.useState("");
    const [categories, setCategories] = React.useState("")
    const [storageCheck, setStorageChecked] = React.useState(false)
    const isFocused = useIsFocused();
    const addZero = (str) => str < 10 ? "0" + str : str

    const _retrieveData = async () => {
        //await AsyncStorage.clear()
        try {
            let data = {}
            data = JSON.parse(await AsyncStorage.getItem('TransactionsList'));
            //console.log(JSON.stringify(data))
            if (data == null) {
                await AsyncStorage.setItem('TransactionsList',
JSON.stringify({}))
                data = JSON.parse(await
AsyncStorage.getItem('TransactionsList'));
            }
            saveData(data)
            try {
                let data = JSON.parse(await AsyncStorage.getItem('Categories'))
                setCategories(data)
            } catch (error) {
                console.error(error)
            }
            if (!pinLoginScreen) {
                try {
                    let data = JSON.parse(await
AsyncStorage.getItem('Settings'))
                    showPinCode(data.PINcode.set)
                    setPinCode(data.PINcode.value)
                } catch (error) {
                    console.error(error)
                }
                setPinLoginScreen(true)
```

```
            }
        } catch (error) {
            console.error(error)
        }
    };

    const saveData = async (data) => {
        let newData = Object.entries(data)
            .map(item => {
                return item[1]
            })
            .sort((objA, objB) => {
                let dateAArr = objA.date.split(".")
                let dateA = new Date(dateAArr[2] + "-" + dateAArr[1] + "-" +
dateAArr[0])
                let dateBArr = objB.date.split(".")
                let dateB = new Date(dateBArr[2] + "-" + dateBArr[1] + "-" +
dateBArr[0])
                return Number(dateB) - Number(dateA)
            })
        setTransactionsData(newData)
    }

    const removeTransaction = async (transaction, date) => {
        let data = JSON.parse(await AsyncStorage.getItem('TransactionsList'));
        data[date].transactions = data[date].transactions.filter(value => {
            return JSON.stringify(value) != JSON.stringify(transaction)
        })
        if (data[date].transactions.length == 0) {
            delete data[date]
        }
        await AsyncStorage.setItem('TransactionsList', JSON.stringify(data))
        try {
            let data = JSON.parse(await AsyncStorage.getItem('Wallets'))
            if (transaction.type === "Витрата") data[transaction.wallet].balance
= +data[transaction.wallet].balance + +transaction.amount
            if (transaction.type === "Дохід") data[transaction.wallet].balance =
+data[transaction.wallet].balance - +transaction.amount
            if (transaction.type === "_moneyTransfer") {
                data[transaction.wallet].balance =
Number(+data[transaction.wallet].balance + +transaction.amount).toFixed(2)
                data[transaction.destinationWallet].balance =
Number(+data[transaction.destinationWallet].balance -
+transaction.amount).toFixed(2)
            }
            await AsyncStorage.setItem("Wallets", JSON.stringify(data))
        } catch (error) {
            console.error(error)
        }

        try {
            if (date !== currentDate) {
                let balancesData = JSON.parse(await
AsyncStorage.getItem('Balances'))
                let datesList = fillEmptyDates(date, currentDate)
                datesList.pop()
                datesList.map(listDate => {
                    if (listDate in balancesData) {
                        if (transaction.type == "Витрата")
balancesData[listDate].balance = +balancesData[listDate].balance +
+transaction.amount
                        if (transaction.type == "Дохід")
balancesData[listDate].balance = +balancesData[listDate].balance -
+transaction.amount
```

```
                    }
                })
                await AsyncStorage.setItem("Balances",
JSON.stringify(balancesData))
            }
        } catch (error) {
            console.error(error)
        }

        onRefresh()
    }

    const checkStorageKeys = async () => {
        if (JSON.parse(await AsyncStorage.getItem('TransactionsList')) == null)
{
            await AsyncStorage.setItem('TransactionsList', JSON.stringify({}))
        } else {
            //console.log(transactionsData)
        }

        if (JSON.parse(await AsyncStorage.getItem('Wallets')) == null) {
            await AsyncStorage.setItem('Wallets', JSON.stringify({}))
        } else {
            //console.log(walletsData)
        }

        if (JSON.parse(await AsyncStorage.getItem('Settings')) == null) {
            let obj = {
                "Account": {
                    "name": null,
                    "login": null,
                    "password": null,
                    "lastSync": null
                },
                "PINcode": {
                    "set": false,
                    "value": ""
                }
            }
            await AsyncStorage.setItem('Settings', JSON.stringify(obj))
        } else {
            //console.log(walletsData)
        }

        let balancesList = JSON.parse(await AsyncStorage.getItem('Balances'))
        setStorageChecked(true)
    }

    const fillBalancesData = async () => {
        let balancesList = JSON.parse(await AsyncStorage.getItem('Balances'))
        let balancesListResult = {}
        try {
            let balancesListArray = Object.entries(balancesList).filter(([key,
value]) => key != "").sort((objA, objB) => {
                let dateAArr = objA[0].split(".")
                let dateA = new Date(dateAArr[2] + "-" + dateAArr[1] + "-" +
dateAArr[0])
                let dateBArr = objB[0].split(".")
                let dateB = new Date(dateBArr[2] + "-" + dateBArr[1] + "-" +
dateBArr[0])
                return Number(dateA) - Number(dateB)
            })
            let tmpObj = {}
```

```
            let datesList = fillEmptyDates(balancesListArray[0][0],
balancesListArray[balancesListArray.length - 1][0])
            datesList.map(date => {
                if (balancesList[date] != undefined) {
                    balancesListResult[date] = balancesList[date]
                    tmpObj = balancesList[date]
                } else {
                    balancesListResult[date] = {
                        balance: tmpObj.balance,
                        date: date.split(".")[0] + " " +
monthsFormated[date.split(".")[1]] + " " + date.split(".")[2] + " p."
                    }
                }
            })
            try {
                await AsyncStorage.setItem('Balances',
JSON.stringify(balancesListResult))
            } catch (err) {
                console.log(err)
            }
        } catch (err) {
            console.log(err)
        }
    }

    const printStorageData = async () => {
        AsyncStorage.getAllKeys((err, keys) => {
            AsyncStorage.multiGet(keys, (error, stores) => {
                stores.map((result, i, store) => {
                    console.log({[store[i][0]]: store[i][1]});
                    return true;
                });
            });
        });
        //await AsyncStorage.clear()
        //await AsyncStorage.setItem("Balances", JSON.stringify({}))
    }

    const getBalance = async () => {
        let currentBalance = 0
        try {
            let data = JSON.parse(await AsyncStorage.getItem('Wallets'))
            Object.entries(data).map(wallet => {
                if (wallet[1].show == true && wallet[1].ignoreBalance == false)
currentBalance += +wallet[1].balance
            })
            setBalance(currentBalance)
        } catch (error) {
            console.error(error)
        }
    }

    const saveBalance = async () => {
        try {
            let data = JSON.parse(await AsyncStorage.getItem('Balances'))
            data[currentDate] = {
                balance: balance,
                date: currentDate.split(".")[0] + " " +
monthsFormated[currentDate.split(".")[1]] + " " + currentDate.split(".")[2] + "
p."
            }
            await AsyncStorage.setItem("Balances", JSON.stringify(data))
        } catch (error) {
            console.error(error)
```

```
        }
    }

    const getCurrentDate = () => {
        let dateObj = new Date()
        let day = addZero(dateObj.getDate())
        let month = addZero(dateObj.getMonth() + 1)
        let year = dateObj.getFullYear()
        let hours = addZero(dateObj.getHours())
        let minutes = addZero(dateObj.getMinutes())
        let date = day + '.' + month + '.' + year;
        let time = hours + ':' + minutes
        let dayOfWeek = dateObj.getDay()
        setDate(date)
        setDayOfWeek(dayOfWeek)
        //return [date, time, dayOfWeek]
    }

    React.useEffect(() => {
        getCurrentDate()
    }, []);

    const onRefresh = async () => {
        setRefreshing(true);
        await _retrieveData()
        await getBalance()
        await saveBalance()
        await fillBalancesData()
        setRefreshing(false);
    };

    React.useEffect(() => {
        if (isFocused) {
            getCurrentDate()
            checkStorageKeys()
            onRefresh();
        }
    }, [isFocused]);

    const deleteItem = async (date) => {
        let data = JSON.parse(await AsyncStorage.getItem('TransactionsList'));
        delete data[date]
        await AsyncStorage.setItem('TransactionsList', JSON.stringify(data))
        onRefresh()
    }

    if (storageCheck) {
        if (categories != "") {
            return (
                <View>
                    <ImageBackground
source={require("../assets/backgrounds/mainBG.png")} resizeMode="cover"
style={{height: "100%"}}>
                        <TransactionsHeader printStorageData={printStorageData}
balance={balance}/>
                        {transactionsData.length != 0 && <FlatList
                            data={transactionsData}
                            style={styles.list}
                            ListFooterComponent={<View style={{height: 1, width:
'100%', marginTop: 12}}></View>}
                            refreshControl={<RefreshControl
refreshing={refreshing} onRefresh={onRefresh}/>}
                            renderItem={({item}) => (
```

```
                                <TransactionCard data={item}
removeTransaction={removeTransaction} refreshList={onRefresh}
categories={categories}/>
                            )}
                        />}
                        {transactionsData.length == 0 &&
                            <View style={{flex: 1, justifyContent: "center",
marginHorizontal: "10%"}}>
                                <Text style={{
                                    color: "lightgray",
                                    textAlign: "center",
                                    fontSize: 14,
                                    lineHeight: 20
                                }}>{`Схоже, що список ваших транзакцій пустий.
\nДля того, щоб додати транзакцію, натисність кнопку "+" у правому нижньому
кутку. \nЗверніть увагу, що перед цим вам необхідно додати гаманець у розділі
"Гаманці" а також категорії витрат та доходів у параметрах, у розділі
"Категорії."`}</Text>
                                </View>}
                    </ImageBackground>
                    <FAB refreshList={onRefresh}/>
                    <PINcodeScreen show={isShownPINcode}
setShowState={showPinCode} initPinCode={pinCode}/>
                </View>
            )
        }
    }
}

export default Transactions

const styles = StyleSheet.create({
    screen: {
        backgroundColor: '#242424',
        height: Dimensions.get('window').height,
    },
    list: {
        height: "auto"
    },
    fab: {
        color: 'black'
    }
});
```

Додаток Б

Лістинг коду файлу Stats.js (екран «Статистика»)

```
import React from "react";
import { ImageBackground, ScrollView, StyleSheet, Text, View, RefreshControl }
from "react-native";
import StatsHeader from "../components/statsScreen/StatsScreenHeader.js";
import BalancesCard from "../components/statsScreen/BalancesCard.js";
import StatsCategories from "../components/statsScreen/StatsCategories.js";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { useIsFocused } from "@react-navigation/native";

export default Stats = () => {
    const [refreshing, setRefreshing] = React.useState(false);
    const [allTransactionsData, setAllTransactionsData] = React.useState([])
    const [transactionsData, setTransactionsData] = React.useState([])
    const [monthsList, setMonthList] = React.useState([])
    const [selectedMonth, setMonth] = React.useState(0)
    const [categories, setCategories] = React.useState("")
    const isFocused = useIsFocused();

    const monthsListSet = new Set()
    _retrieveData = async () => {
        try {
            let data = {}
            data = JSON.parse(await AsyncStorage.getItem('TransactionsList'));
            if (data == null) {
                await AsyncStorage.setItem('TransactionsList',
JSON.stringify({}))
                data = JSON.parse(await
AsyncStorage.getItem('TransactionsList'));
            }
            let newData = Object.entries(data)
                .map(item => {
                    monthsListSet.add(item[1].monthAndYear)
                    return item[1]
                })
                .sort((objA, objB) => {
                    let dateAArr = objA.date.split(".")
                    let dateA = new Date(dateAArr[2] + "-" + dateAArr[1] + "-" +
dateAArr[0])
                    let dateBArr = objB.date.split(".")
                    let dateB = new Date(dateBArr[2] + "-" + dateBArr[1] + "-" +
dateBArr[0])
                    return Number(dateB) - Number(dateA)
                })
            setAllTransactionsData(newData)
            const monthsList = [...monthsListSet];
            setMonthList(monthsList)
            try {
                let data = JSON.parse(await AsyncStorage.getItem('Categories'))
                setCategories(data)
            } catch (error) { console.error(error) }
        } catch (error) {
            console.error(error)
        }
    };

    const onRefresh = async () => {
        setRefreshing(true);
        await _retrieveData()
```

```
        setRefreshing(false);
    }

    React.useEffect(() => {
        if (isFocused) {
            onRefresh();
        }
    }, [isFocused]);

    return (
        <ImageBackground source={require("../assets/backgrounds/mainBG.png")}
resizeMode="cover" style={{ flex: 1 }}>
            <StatsHeader selectedMonth={selectedMonth} monthsList={monthsList}
setMonth={setMonth} />
            <ScrollView
                refreshControl={<RefreshControl refreshing={refreshing}
onRefresh={onRefresh} />}
            >
                {!refreshing && <View>
                    <BalancesCard
                        allTransactionsData={allTransactionsData}
                        selectedMonth={selectedMonth}
                        monthsList={monthsList}
                        refreshScreen={onRefresh} />
                    <StatsCategories
                        allTransactionsData={allTransactionsData}
                        selectedMonth={selectedMonth}
                        monthsList={monthsList}
                        refreshScreen={onRefresh}
                        categories={categories}
                    />
                </View>
                }
            </ScrollView>
        </ImageBackground>
    )
}

const styles = StyleSheet.create({
    image: {
        flex: 1,
        justifyContent: "center",
        alignItems: 'center'
    },
    text: {
        color: "white",
        fontSize: 42,
        lineHeight: 84,
        fontWeight: "bold",
        textAlign: "center",
        backgroundColor: "#000000c0"
    }
});
```

Додаток В

Лістинг коду файлу Wallet.js (екран «Гаманці»)

```
import React from "react";
import { ImageBackground, View, Text, RefreshControl, TouchableOpacity,
ScrollView, StyleSheet, Image } from "react-native";
import WalletHeader from "../components/walletScreen/WalletScreenHeader";
import GraphCard from "../components/walletScreen/GraphCard";
import WalletCategory from "../components/walletScreen/WalletCategory";
import AsyncStorage from "@react-native-async-storage/async-storage";
import { useIsFocused } from "@react-navigation/native";
import AddWalletPopup from "../components/walletScreen/AddWalletPopup";
import { monthsFormated } from "../data/calendarData"

const AddAssetBtn = (props) => {
    const [showModal, setShowModal] = React.useState(false)
    const hideModal = () => {
        setShowModal(false)
    }

    return (
        <TouchableOpacity style={[styles.card, { marginBottom: 12 }]}
onPress={() => {
            setShowModal(true)
        }}>
            <AddWalletPopup
                show={showModal}
                hide={hideModal}
                refreshList={props.refreshList}
                mode="add"
                currentDate={props.currentDate}
                currentDayOfWeek={props.currentDayOfWeek}
            />
            <Image style={{ height: 18, width: 18, marginTop: 3 }}
source={require("../assets/icons/system/addAsset.png")} />
            <Text style={{ color: "white", fontWeight: "bold", fontSize: 16,
marginLeft: 7.5 }}>Додати гаманець</Text>
        </TouchableOpacity>
    )
}

const Wallet = () => {
    const [currentDate, setDate] = React.useState("")
    const [currentDayOfWeek, setDayOfWeek] = React.useState()
    const [walletsList, setWallets] = React.useState({})
    const [balance, changeBalance] = React.useState(0)
    const [defaultBalance, setDefaultBalance] = React.useState(0)
    const [scrollState, toggleScroll] = React.useState(true)
    const [refreshing, setRefreshing] = React.useState(false);
    const isFocused = useIsFocused();

    let UAHtoUSD = 39.8
    const addZero = (str) => str < 10 ? "0" + str : str

    _retrieveData = async () => {
        let bankActives = []
        let cashActives = []
        let currentBalance = 0
        try {
            let data = JSON.parse(await AsyncStorage.getItem('Wallets'))
            Object.entries(data).map(wallet => {
```

```
                if (wallet[1].show == true) {
                    wallet[1].walletType == "Банківський актив" ?
bankActives.push(wallet[1]) : cashActives.push(wallet[1])
                    if (wallet[1].ignoreBalance == false) currentBalance +=
+wallet[1].balance
                }
            })
            let walletsObj = {
                ["Банківські активи"]: bankActives,
                ["Готівкові кошти"]: cashActives,
            }
            setWallets(walletsObj)
            changeBalance(currentBalance)
            setDefaultBalance(currentBalance)
        } catch (error) {
            console.error(error)
        }
    };

    const saveBalance = async () => {
        try {
            let data = JSON.parse(await AsyncStorage.getItem('Balances'))
            data[currentDate] = {
                balance: balance,
                date: currentDate.split(".")[0] + " " +
monthsFormated[currentDate.split(".")[1]] + " " + currentDate.split(".")[2] + "
p."
            }
            await AsyncStorage.setItem("Balances", JSON.stringify(data))
        } catch (error) {
            console.error(error)
        }
    }

    const getCurrentDate = () => {
        let dateObj = new Date()
        let day = addZero(dateObj.getDate())
        let month = addZero(dateObj.getMonth() + 1)
        let year = dateObj.getFullYear()
        let hours = addZero(dateObj.getHours())
        let minutes = addZero(dateObj.getMinutes())
        let date = day + '.' + month + '.' + year;
        let time = hours + ':' + minutes
        let dayOfWeek = dateObj.getDay()
        setDate(date)
        setDayOfWeek(dayOfWeek)
        //return [date, time, dayOfWeek]
    }

    const onRefresh = async () => {
        setRefreshing(true);
        await _retrieveData()
        await saveBalance()
        setRefreshing(false);
    }

    React.useEffect(() => {
        getCurrentDate()
    }, [])

    React.useEffect(() => {
        if (isFocused) {
            onRefresh();
        }
```

```
    }, [isFocused]);

    return (
        <ImageBackground source={require("../assets/backgrounds/mainBG.png")}
resizeMode="cover" style={{ flex: 1 }}>
            <WalletHeader balance={balance} />
            <ScrollView scrollEnabled={scrollState}
refreshControl={<RefreshControl refreshing={refreshing} onRefresh={onRefresh}
/>}>
                <GraphCard
                    balance={balance}
                    changeBalance={changeBalance}
                    balanceDataObj={balanceDataObj}
                    toggleScroll={toggleScroll}
                    defaultBalance={defaultBalance}
                    currentDate={currentDate}
                    refreshTrigger={refreshing}
                />
                {Object.entries(walletsList).map((category) => {
                    return <WalletCategory
                        title={category[0]}
                        data={category[1]}
                        exRate={UAHtoUSD}
                        key={category[0]}
                        onRefresh={onRefresh}
                        currentDate={currentDate}
                        currentDayOfWeek={currentDayOfWeek}
                    />
                })}
                <AddAssetBtn refreshList={onRefresh} mode="add"
currentDate={currentDate} currentDayOfWeek={currentDayOfWeek} />
            </ScrollView>
        </ImageBackground>
    )
}

export default Wallet

const styles = StyleSheet.create({
    card: {
        flexDirection: "row",
        backgroundColor: "rgba(0, 0, 0, 0.15)",
        marginTop: 12,
        marginHorizontal: '3%',
        borderRadius: 25,
        width: '94%',
        height: 65,
        justifyContent: "center",
        alignItems: "center",
    },
});
```

Додаток Г

Лістинг коду файлу Settings.js (екран «Параметри»)

```
import React, {useState} from "react";
import {ImageBackground, View, Text, StyleSheet, Image, TouchableOpacity,
FlatList, Switch, Button,} from "react-native";
import WalletsList from "../components/settingsScreen/WalletsList";
import CategoriesList from "../components/settingsScreen/CategoriesList";
import PINCodeSettings from "../components/settingsScreen/PINCodeSettings";
import AsyncStorage from "@react-native-async-storage/async-storage";
import Modal from "react-native-modal";
import AccountSettings from "../components/settingsScreen/AccountSettings";

const AboutMenu = (props) => {

    return (
        <Modal
            animationIn="fadeIn"
            animationOut="fadeOut"
            isVisible={props.show}
            backdropOpacity={0.75}
            useNativeDriver={true}
            useNativeDriverForBackdrop={true}
            onBackButtonPress={props.hide}
            onBackdropPress={props.hide}>
            <View style={styles.deleteModal}>
                <Text style={{fontSize: 18, color: "white", fontWeight: "bold",
marginHorizontal: "6%", marginVertical: "5%",}}>Гаманець 1.0</Text>
            </View>
        </Modal>
    )
}

const MenuItem = (props) => {
    const [descriptionData, setDescription] = React.useState("")
    let description = props.data.description !== undefined;
    let switchBtn = props.data.switch !== undefined;
    const [isEnabled, setIsEnabled] = useState(props.data.switch);
    const toggleSwitch = () => setIsEnabled(!isEnabled);
    const pass = () => {
    }

    const getDescription = async () => {
        if (props.data.description === "*PINCODE_SETTINGS*") {
            try {
                let data = JSON.parse(await AsyncStorage.getItem("Settings"))
                if (data.PINcode.set) setDescription("Встановлено")
                if (!data.PINcode.set) setDescription("Не встановлено")
            } catch (e) {
            }
        }
    }

    React.useEffect(() => {
        if (description) getDescription()
    }, [props.showState])

    return (
        <TouchableOpacity onPress={() => {
            switchBtn ? toggleSwitch : pass
            !switchBtn ? props.setShowState(!props.showState) : pass
```

```
        }} style={styles.menuItem}>
            <Image style={styles.menuItemIcon} source={props.data.icon}/>
            <View style={styles.menuItemTitleBlock}>
                <Text style={styles.menuItemTitle}>{props.data.title}</Text>
                {description && <Text
style={styles.menuItemDescription}>{descriptionData}</Text>}
            </View>
            {!switchBtn && <Image style={styles.menuItemArrow}
source={require("../assets/icons/arrowRight.png")}/>}
            {switchBtn && <Switch
                trackColor={{false: '#767577', true: '#1976D2'}}
                thumbColor={"white"}
                ios_backgroundColor="#3e3e3e"
                onValueChange={toggleSwitch}
                value={isEnabled}
            />}
        </TouchableOpacity>
    )
}

const Settings = () => {
    const [showAccountSettings, setShowAccountSettings] = React.useState(false)
    const [showWalletsList, setShowWalletsList] = React.useState(false)
    const [showCategoriesList, setShowCategoriesList] = React.useState(false)
    const [showPINCodeSettings, setShowPINCodeSettings] = React.useState(false)
    const [showAboutMenu, setShowAboutMenu] = React.useState(false)
    const [name, setName] = React.useState(null)
    const [login, setLogin] = React.useState(null)
    const [image, setImage] = React.useState(null);

    const retrieveData = async () => {
        try {
            let data = JSON.parse(await
AsyncStorage.getItem("Settings"))["Account"]
            let img = JSON.parse(await AsyncStorage.getItem("AccountPhoto"))
            setName(data.name)
            setLogin(data.login)
            setImage(img)
        } catch (e) {
        }
    }

    React.useEffect(() => {
        retrieveData()
    }, [])

    React.useEffect(() => {
        retrieveData()
    }, [showAccountSettings])

    return (
        <View style={styles.container}>
            <View style={styles.contentBG}>
                <View style={styles.contentBlock}>
                    <View style={styles.header}>
                        {image !== null && <Image style={styles.profilePhoto}
source={{uri: image}}/>}
                        {image === null && <Image style={styles.profilePhoto}
source={require("../assets/icons/user.png")}/>}
                        <View style={styles.account}>
                            {name !== null && <Text
style={styles.accountName}>{name}</Text>}
                            {login !== null && <Text
style={styles.accountLogin}>{login}</Text>}
```

```
                            {login === null && <Text
style={styles.accountName}>Невідомий користувач</Text>}
                        </View>
                    </View>
                    <View style={styles.divider}></View>
                    <View style={styles.menu}>
                        <MenuItem data={menuItems[0]}
showState={showAccountSettings} setShowState={setShowAccountSettings}/>
                        <MenuItem data={menuItems[1]}
showState={showWalletsList} setShowState={setShowWalletsList}/>
                        <MenuItem data={menuItems[2]}
showState={showCategoriesList} setShowState={setShowCategoriesList}/>
                        <MenuItem data={menuItems[5]}
showState={showPINCodeSettings} setShowState={setShowPINCodeSettings}/>
                        <MenuItem data={menuItems[7]} showState={showAboutMenu}
setShowState={setShowAboutMenu}/>
                    </View>
                </View>
            </View>
            {showWalletsList &&
                <WalletsList
                    show={showWalletsList}
                    hide={() => {
                        setShowWalletsList(!showWalletsList)
                    }}
                />}
            {showCategoriesList &&
                <CategoriesList
                    show={showCategoriesList}
                    hide={() => {
                        setShowCategoriesList(!showCategoriesList)
                    }}
                />}
            {showPINCodeSettings &&
                <PINCodeSettings
                    show={showPINCodeSettings}
                    hide={() => {
                        setShowPINCodeSettings(!showPINCodeSettings)
                    }}
                />}
            {showAboutMenu &&
                <AboutMenu
                    show={showAboutMenu}
                    hide={() => {
                        setShowAboutMenu((!showAboutMenu))
                    }}
                />}
            {showAccountSettings &&
                <AccountSettings
                    show={showAccountSettings}
                    hide={() => {
                        setShowAccountSettings((!showAccountSettings))
                    }}
                />}
        </View>
    )
}

export default Settings

const styles = StyleSheet.create({
    container: {
        flex: 1,
    },
```

```
image: {
    flex: 1,
    justifyContent: "center",
    alignItems: 'center'
},
contentBG: {
    height: '100%',
    width: '100%',
    backgroundColor: 'rgba(26, 25, 31, 1)'
},
contentBlock: {
    //backgroundColor: 'rgba(255, 255, 255, 0.2)',
    marginHorizontal: '5%',
    marginVertical: '5%',
},
header: {
    flexDirection: "row"
},
profilePhoto: {
    height: 65,
    width: 65,
    borderRadius: 100,
    backgroundColor: "lightgray"
},
account: {
    marginHorizontal: "5%",
    justifyContent: 'center'
},
accountName: {
    color: "white",
    fontSize: 23,
    fontWeight: "bold"
},
accountLogin: {
    color: "#A1A1A1",
    fontSize: 13
},
divider: {
    marginTop: "5%",
    backgroundColor: "rgba(255,255,255,0.25)",
    height: 1
},
menu: {
    marginTop: "2%"
},
menuItem: {
    //backgroundColor: "rgba(255,255,255,0.15)",
    height: 60,
    flexDirection: "row",
    alignItems: "center"
},
menuItemIcon: {
    height: 24,
    width: 24,
    marginHorizontal: "3%"
},
menuItemTitleBlock: {
    marginLeft: "5%",
    flex: 1
},
menuItemTitle: {
    color: "white",
    fontSize: 16,
    fontWeight: "bold",
```

```
    },
    menuItemDescription: {
        color: "#A1A1A1",
        fontSize: 13,
    },
    menuItemArrow: {
        height: 15,
        width: 9.5,
        marginRight: "3%"
    },
    deleteModal: {
        backgroundColor: '#1A191F',
    },

});

const menuItems = [
    {
        title: "Обліковий запис",
        icon: require("../assets/icons//system/settings/profile.png"),

    },
    {
        title: "Гаманці",
        icon: require("../assets/icons//system/settings/wallet.png"),
    },
    {
        title: "Категорії",
        icon: require("../assets/icons//system/settings/categories.png"),
    },
    {
        title: "Курси валют",
        icon: require("../assets/icons//system/settings/rates.png"),
    },
    {
        title: "Мова",
        icon: require("../assets/icons//system/settings/language.png"),
        description: "Українська"
    },
    {
        title: "PIN-код",
        icon: require("../assets/icons//system/settings/password.png"),
        description: "*PINCODE_SETTINGS*"
    },
    {
        title: "Нічний режим",
        icon: require("../assets/icons//system/settings/nightMode.png"),
        switch: true
    },
    {
        title: "Про додаток",
        icon: require("../assets/icons//system/settings/about.png"),
    },
]
```