Technical Development Guidelines: Kounta Mobile App

Document Version: 1.1
Date: July 14, 2025
Project: Kounta - Personal Financial Management Mobile App

1. Introduction & Project Goal

This document provides the complete technical specification for building the Minimum Viable Product (MVP) of the **Kounta** mobile application. The primary goal is to develop a **local-first, offline-capable** personal finance app for Android and iOS that allows a single user to manage their finances effectively.

The developer's objective is to follow these guidelines precisely to ensure the final product is secure, scalable, and aligns with the established brand and architectural vision.

2. Core Architecture: Local-First with a Repository Pattern

The application must be built using a **local-first** architecture.

- **Offline Capability:** All data is stored and managed on the user's device. The app must be 100% functional without an internet connection (except for the initial authentication and backup/restore process).
- **ORM-Powered Data Layer:** We will use the WatermelonDB ORM to manage all data interactions. This provides a type-safe, reactive layer on top of the local SQLite database.
- **Scalability via Repository Pattern:** This is a **critical** architectural requirement. All business logic must interact with a "Repository" layer. The Repository will, in turn, use WatermelonDB to execute data operations. This isolates the data source and will allow for a seamless transition to a backend API in the future.

[UI Components] -> [Business Logic/Hooks] -> [Repository] -> [WatermelonDB ORM] -> [SQLite Database]

3. Technology Stack

The following technologies and libraries must be used:

Category	Technology/Library	Version/Notes
Framework	**React Native with Expo**	Use the latest stable Expo

		SDK.
UI Components	**React Native Paper**	For a complete Material Design component library.
ORM & Database	**WatermelonDB**	Set up with the SQLite adapter. This is the primary way to interact with data.
Navigation	**React Navigation**	For all screen and navigation logic.
Authentication	**expo-auth-session / expo-google-app-auth**	For Google Sign-In.
Cloud Backup	**expo-google-drive-api-upload-and-download**	Or a similar library for Google Drive file operations.
Icons	**react-native-vector-icons/MaterialCommunityIcons**	Included with React Native Paper.

4. Database Schema & Models

The database schema will be defined as **WatermelonDB Models**. The ORM will be responsible for creating and migrating the underlying SQLite tables. All columns defined below should be implemented as fields in their respective models.

Account Model

- name (string)
- type (string)
- currency (string)
- opening_balance (number)
- created_at (number, timestamp)

Category Model

- name (string)
- type (string: "Income" or "Expense")
- created_at (number, timestamp)

Asset Model

- name (string)
- type (string)
- currency (string)

- value (number)
- created_at (number, timestamp)

Liability Model

- name (string)
- type (string)
- currency (string)
- total_amount (number)
- current_balance (number)
- created_at (number, timestamp)

Transaction Model

- description (string)
- amount (number)
- type (string: "Income" or "Expense")
- date (number, timestamp)
- **Relations:**
  - @Relation('accounts', 'account_id') account
  - @Relation('categories', 'category_id') category
  - @Relation('assets', 'asset_id') asset (optional)
  - @Relation('liabilities', 'liability_id') liability (optional)

5. Feature Implementation Guide

5.1. Onboarding & Setup

- On first launch, guide the user through setting up their primary currency.
- Use a WatermelonDB "batch" operation to pre-populate the database with default Category models.

5.2. Authentication & Backup

- Implement Google Sign-In. The app must request the https://www.googleapis.com/auth/drive.file scope.
- **Backup:** The backup function must locate the kounta.db file (and any related -wal, -shm files) used by WatermelonDB and upload them to Google Drive.
- **Restore:** The restore function must download the database files, replace the local ones, and restart the app.

5.3. CRUD Operations

- All Create, Read, Update, and Delete operations must be performed via WatermelonDB model methods (e.g., database.get('accounts').create(...)).
- UI components displaying lists of data (e.g., a list of accounts) should use the

@watermelondb/withObservables HOC or related hooks to be reactive.

5.4. Transaction Management

- The "Add Transaction" form will create a new Transaction model instance.
- Dropdowns for relations (account, category, etc.) will be populated by querying the respective model collections.
- All database changes for a single transaction (creating the transaction, updating an account balance) must be wrapped in a single database.write() block to ensure atomicity.

5.5. Dashboard & Reporting

- Dashboard widgets will be powered by WatermelonDB queries (Q.where(...), Q.sum(...), etc.).
- Due to WatermelonDB's reactive nature, the dashboard figures should update in real-time as new transactions are added.

6. UI & Styling Implementation

- Refer to the **"Brand Guidelines: Kounta"** document for all visual design.
- Use the **React Native Paper** <Provider> component at the root of the app.
- Create a custom theme file that defines the Green (#005248) and Amber (#B57B00) color palettes for both light and dark modes, as specified in the guidelines.
- The device's theme (light/dark) should be respected automatically.
- Configure the theme to use **Poppins** for headings and **Inter** for body text.