Project Requirements Document: Personal Accountant

1. Project Overview 🎯

PAccountant is a personal finance engine designed to track a user's total financial health. The system moves beyond simple expense tracking to provide a holistic view of "Net Worth" by managing liquid cash, physical assets, debts, and money lent to others.

2. Core User Journeys 👤

A. Wealth Setup

- **Manage Accounts:** Create and update liquid accounts (e.g., Bank, Mobile Money, Cash) to track available spending power.
- **Track Assets:** List high-value items (e.g., Land, Vehicles, Equipment) that contribute to total wealth but aren't liquid cash.
- **Track Liabilities:** Record formal debts or loans owed to others (e.g., Bank Loans, Mortgages).

B. Money Movement

- **Log Transactions:** Record daily **Income** and **Expenses**.
- **Categorize Spending:** Tag every movement with a category (e.g., Food, Transport, Salary) to understand where money goes.
- **Monitor Budgets:** Set monthly spending limits on specific categories to prevent overspending.

C. Lending & IOUs

- **Record Loans Given:** Track money lent to friends or family. In this system, money lent is treated as an **Asset** (Value you still own).
- **Record Repayments:** Log when people pay you back. This should reduce their debt to you and increase your account balance.

3. Logical Requirements (The "Rules") 🧠

1. The Atomic Balance Rule

Whenever a transaction is recorded, the linked **Account** balance must update immediately and automatically. No transaction can exist without an account.

2. The Net Worth Formula

The system must be able to calculate and display a user's total Net Worth at any time using this logic:

Total Value = (Sum of all Accounts) + (Sum of all Assets) + (Sum of all Loans Given)

Net Worth = Total Value - (Sum of all Liabilities)

3. The Double-Entry Constraint

- When a **Loan Given** is recorded, cash must leave an **Account** and the **Loan Asset** balance must increase.
- When a **Debt** is repaid, cash must leave an **Account** and the **Liability** balance must decrease.

4. The Budget Guardrail

The system must calculate total spending for the current month per category. If a transaction causes spending to exceed the defined **Budget**, a warning or flag must be triggered.

4. Technical Constraints 🏗️

- **Architecture:** Must follow **Clean Architecture** principles (Domain, Application, Infrastructure, UI).
- **UI:** Built using **Blazor** (Server or WebAssembly).
- **State:** The system must handle real-time balance updates in the UI when transactions are added.

5. Bonus Features (Phase 2) 🌟

- **Recurring Bills:** Implement a "Bill Rule" system to automate the prediction of monthly obligations.
- **Sinking Funds (Envelopes):** Allow users to "virtually" set aside money within an account for a specific future purpose (e.g., "Holiday Fund").
- **Multi-Currency:** Allow accounts and assets to be held in different currencies with a consolidated total in a base currency.

6. Success Metrics ✅

1. User can see their **Net Worth** update in real-time.
2. All **IOUs** are tracked clearly as assets.
3. No account balance falls out of sync with its transaction history.