PRD: The "Debt Buckets" Workflow & Architecture Refactor

Role: Lead Architect
Target System: Paydown Pilot (Next.js/FastAPI/PostgreSQL)
Objective: Refactor the "Add Account" flow to support "Balance Buckets" (Split APRs), specifically enabling the UK "Statement Wizard" manual entry process.

1. Context & Problem Statement

Currently, the app assumes a credit card has a single balance and a single APR. In reality, especially in the UK, a user's credit card balance is a "stack" of different debts (e.g., £2,000 Balance Transfer at 0%, £500 Purchases at 24.9%, £100 Cash Advance at 39.9%).

- **The Problem:** If we average these rates, the Math Brain optimizes incorrectly. It might pay off a 0% debt too early or fail to attack a toxic Cash Advance balance.
- **The Solution:** We must allow users to input "Buckets" of debt. The backend will then "virtualize" these buckets into separate debts for the optimizer, while grouping them visually for the user.

---

2. Database Schema Refactor (PostgreSQL/Drizzle)

We need to move from a flat accounts table to a relational structure that supports buckets.

Task: Modify schema.ts (or equivalent) to introduce debt_buckets.

2.1 Table Structure

The debt_accounts table remains the "Parent" (representing the physical card). A new

debt_buckets table holds the segments.

debt_accounts (Parent)

- id: UUID (PK)
- user_id: UUID (FK)
- lender_name: String (e.g., "Barclaycard Platinum")
- currency: String (GBP/USD)
- min_payment_amount: Integer (Cents/Pence) - *The mandate from the monthly statement.*
- next_payment_due_date: Date
- is_manual_entry: Boolean (True for UK flow)

debt_buckets (Children)

- id: UUID (PK)
- account_id: UUID (FK -> debt_accounts)
- name: String (e.g., "Purchases", "0% Balance Transfer")
- balance: Integer (Cents/Pence)
- apr: Float (e.g., 24.9)
- is_promo: Boolean
- promo_expiry_date: Date (Nullable) - *Critical for optimization.*

---

3. Frontend UX: The "Statement Wizard" (Add Account Flow)

We are replacing the simple modal with a **3-Step Wizard** for the manual entry flow.

Step 1: The "Headline" Data

- **Input:** Lender Name (Searchable Dropdown).
- **Input:** Currency (Defaults to User Preference).
- **Input:** Total Statement Balance (Big Input).
- **Input:** Next Payment Due Date (Date Picker).
- **Input:** Minimum Payment Due (Money Input).

Step 2: The "Split" Logic (The Pivot)

- **Visual:** Show a diagram of a credit card statement's "Allocation of Payments" table.
- **Question:** "Look at your statement. Is your balance split into different interest rates?"
- **Interaction:**
  - **Toggle Switch:** Single Rate vs Multiple Rates (Buckets).
  - *If Single Rate:* Show one APR input field.
  - *If Multiple Rates:* Proceed to Step 3 (The Bucket Builder).

Step 3: The Bucket Builder (Dynamic Form)

- **UI Component:** A dynamic list/table where users add rows.
- **Row Fields:**
  1. **Type:** Dropdown [Purchases, Balance Transfer, Money Transfer, Cash Advance].
  2. **Amount:** Money Input.
  3. **Interest Rate (APR):** % Input.
  4. **Expiry Date:** Date Picker (Conditional: Only show if APR is 0% or Type is Balance Transfer).
- **Real-Time Validation:**
  - Display a live sum of the bucket amounts: Sum: £2,450.
  - Compare it to the "Total Balance" from Step 1 (£2,500).
  - If they don't match, show a warning: *"These rows add up to £2,450, but your total is £2,500. We will add a 'Miscellaneous' bucket for the remaining £50."*

---

4. Backend Logic & Optimizer Handoff

This is the most critical logic. We must transform the user's "Hierarchical" view into the Solver's "Flat" view.

4.1 The "Virtualization" Pipeline

When the user saves the account, the backend creates the Parent and Children records. When the **Optimizer (Math Brain)** requests data, we must run a flatten_accounts service:

Input (Database):

- Account: Barclaycard (Min Pay: £50)
  - Bucket A: £2,000 @ 0% (Expires Dec 2025)
  - Bucket B: £500 @ 24.9% (Standard)

Output (To Solver Engine):
The solver receives a list of DebtAccount objects. We treat each bucket as a separate account but link them via a group_id.

1. **Virtual Account 1 (Barclaycard - Promo):**
   - balance: 200000
   - apr: 0.0
   - promo_end: "2025-12-01"
   - min_payment: 0 (We allocate the *Parent* min payment to the highest APR bucket first, or proportionally. *Start simple: 0*).
   - group_id: "barclay_uuid"
2. **Virtual Account 2 (Barclaycard - Standard):**
   - balance: 50000
   - apr: 24.9
   - min_payment: 5000 (The full £50 min payment from the statement is enforced here to ensure the card stays current).
   - group_id: "barclay_uuid"

4.2 The "Avalanche" Logic Update

The Optimizer's logic (solver_engine.py) needs to be updated to respect these buckets:

1. **Targeting:** It must target **Virtual Account 2** (24.9%) with all excess budget.
2. **Ignorance:** It must pay only the minimums (or zero if covered by the parent group min payment) on **Virtual Account 1** (0%) until the promo expiry nears.
3. **The Time-Bomb:** On 2025-12-01, Virtual Account 1's APR conceptually "switches" to the standard rate. The optimizer must plan to clear it *before* that date if the strategy is "Avoid Interest Traps."

---

5. Implementation Instructions (For Replit Agent)

System Prompt:
"You are implementing the 'Debt Buckets' feature. This refactors the manual 'Add Account'

form into a multi-step wizard and changes the database schema to support split-interest balances."

Step-by-Step Execution Plan:

1. **Schema Migration:**
   - Update shared/schema.ts to include the debt_buckets table definition.
   - Establish the relationship: debt_accounts has many debt_buckets.
   - Run the migration.
2. **API Update:**
   - Update server/routes.ts (specifically POST /api/accounts).
   - It must now accept a JSON body containing buckets: [].
   - Logic: Transactionally create the debt_account, then iterate and create debt_buckets.
3. **Frontend - Component Creation:**
   - Create components/add-account-wizard/step-basics.tsx (Lender, Total Bal, Min Pay).
   - Create components/add-account-wizard/step-buckets.tsx (The dynamic table).
   - Create components/add-account-wizard/bucket-row.tsx (Individual row with type/amount/apr).
4. **Frontend - Integration:**
   - Replace the existing AddAccountDialog content with this new Wizard flow.
   - Implement the validation logic (Bucket Sum == Total Balance).
5. **Solver Bridge:**
   - Modify solver_engine.py (or the Python script call in routes.ts) to query debt_buckets.
   - If an account has buckets, pass the *buckets* to the solver as individual debts.
   - If an account has no buckets (single rate), pass the account as one debt.

---

6. Acceptance Criteria

1. **User can add a "Standard" card:** User enters Balance £1000, APR 20%. -> System saves 1 Account + 1 "Default" Bucket.
2. **User can add a "Complex" card:** User enters Balance £3000. Splits it into £2000 (0% Promo) and £1000 (Standard).
3. **Data Persistence:** Refreshing the page retains the bucket breakdown.
4. **Portfolio View:** The main dashboard shows the *Total* balance (£3000), but clicking "Details" reveals the breakdown (The Stack).
5. **Solver Output:** The generated plan prioritizes paying off the £1000 Standard debt *before* the £2000 Promo debt.