Product Requirements Document: "Find My Budget" Module

- **Module:** `Find My Budget` (Plaid Integration)
- **Author:** Credit Card Optimiser App Developer (Lead Architect)
- **Status: Approved for Development**

1. Module Definition

- **One-Sentence Idea:** To securely connect to a user's bank accounts via Plaid, analyze their transaction history to identify income and recurring expenses, and present a suggested monthly debt paydown budget for user confirmation .
- **Problem Solved:** This feature addresses two primary user pain points:
    1. **Lack of Awareness:** Users often don't know their true disposable income or where their money is going.
    2. **Friction & Error:** Manual budget entry is tedious, error-prone, and a major obstacle to completing the app's onboarding flow.

2. Architectural Fit: The "Two-Brain" Doctrine

This feature's design is a critical implementation of our "Two-Brain" architectural doctrine.

- **The "Language Brain" (Claude 4.5 Sonnet)** : Its role is "Budget Analyzer". It will perform the complex *reasoning and categorization* task of analyzing raw transaction descriptions (e.g., "PAYROLL ACME," "Netflix.com") and sorting them into structured categories (Income, Essential Bills, Subscriptions, Discretionary) . It is explicitly **forbidden** from performing math other than summing the categories it identifies.
- **The Backend (FastAPI):** Our backend will perform all simple, deterministic arithmetic. It will take the categorized sums from the Language Brain (e.g., `{"income": X, "essential_spending": Y}`) and calculate the final budget figures (`Current Budget = X - Y - ...`).
- **The "Math Brain" (OR-Tools)**: This engine is unchanged. It will only receive the final, *user-confirmed* budget number (e.g., `50000` cents) as one of its inputs for the main optimization problem.

This strict separation ensures that complex language reasoning (Claude) is kept separate from deterministic financial calculation (FastAPI backend and OR-Tools solver).

3. Detailed User Journey

This describes the complete, step-by-step user flow for this new feature.

1. **Initiation:** The user clicks a new **"Find My Budget"** button located on the main dashboard and the "Budget" page.
2. **Consent:** A modal appears with the following text. The user must click "Agree & Connect" to proceed.
   "I authorize Paydown Pilot to securely connect to my bank via Plaid. I agree to a one-time analysis of my transaction history to identify income and expenses to suggest a budget. My raw transaction data will not be stored."
3. **Plaid Link (Frontend):** The Plaid Link modal opens. The user selects their bank, authenticates, and grants permission.
4. **Token Exchange (Backend):** The frontend receives a `public_token` from Plaid and sends it to our backend. The backend exchanges this for a permanent `access_token` and `item_id`, which are securely stored.
5. **Analysis (Backend & Language Brain):**
   - A loading state ("Analyzing your transactions...") is shown to the user.
   - The backend fetches transactions from Plaid's `/transactions/get` product.
   - The raw transaction list is sent to the **Language Brain (Claude)** for categorization.
   - Claude returns a structured JSON object with categorized totals (see section 4.3).
   - The backend receives this JSON and performs the final, simple calculations to determine `Identified Monthly Net Income`, `Identified Disposable Income`, and `Identified Current Budget`.
6. **Presentation (Frontend):** The user is shown a new summary view with the calculated figures:
   - **Identified Monthly Net Income:** $X,XXX
   - **Identified Disposable Income:** $Y,YYY
   - **Identified Current Budget:** $Z,ZZZ
   - Below this, a button is displayed: **"Increase My Budget"**.
   - The user can also choose to accept the `Identified Current Budget` as-is.
7. **Budget Increase Flow (Optional):**
   - If the user clicks "Increase My Budget," they are taken to a new page.
   - This page displays two categories populated from the Language Brain's analysis:
     1. **Category 1: Media & Subscriptions:** A list of all identified recurring subscriptions, each with a checkbox (e.g., `[ ] Netflix - $15.99`). The user ticks the boxes for subscriptions they can cancel.
     2. **Category 2: Discretionary Spending:** A list of spending categories, each with a text input (e.g., `Restaurants/Bars: $285.00 - "How much could you reduce this by each month?" [ $____ ]`).
   - At the bottom, a grand total of potential savings is shown, along with a button: **"Add to Current Budget"**.
8. **Confirmation & Integration:**

    - When the user confirms, the backend calculates the **`Identified Potential Budget`** (`Current Budget + Total Savings`).
    - Both `Identified Current Budget` and `Identified Potential Budget` are saved to the user's profile in the database.
    - The user is returned to the "Budget" page, where they can now select *which* budget (Current or Potential) to use as the input for their debt paydown plan.
    - The app will show a comparison of the payoff plan based on the *other* budget (e.g., "By using your Potential Budget, you'll pay off your debt X months faster and save $Y in interest compared to your Current Budget.").

4. Core Feature Breakdown by Component

4.1.

Frontend (React/Next.js)

- **New UI Components:**
    - `PlaidLinkButton`: A component that holds the "Find My Budget" button. It handles the logic for fetching the `link_token` from our backend and initializing the Plaid Link flow.
    - `PlaidConsentModal`: The modal displaying the specific consent text (see 3.2).
    - `BudgetAnalysisLoadingView`: A full-screen or modal loading state (e.g., "Analyzing transactions, this may take a moment...").
    - `BudgetSummaryView`: The view displaying the calculated `Net Income`, `Disposable Income`, and `Current Budget`.
    - `BudgetIncreaseView`: The new page for the "Increase My Budget" flow, including:
        - `SubscriptionCategoryList`: Renders items from `non_essential_subscriptions` with checkboxes.
        - `DiscretionaryCategoryList`: Renders items from `non_essential_discretionary_categories` with decimal input fields.
    - **Error Handling:** New toasts/alerts are required for Plaid connection failures (e.g., "Connection to your bank failed. Please try again.").

4.2.

Backend (FastAPI)

- **New API Endpoints:**
    - `POST /plaid/create_link_token`:
        - **Purpose:** For the frontend to initiate the Plaid Link flow.
        - **Action:** Calls Plaid's `/link/token/create` API and returns the `link_token` to the frontend.
    - `POST /plaid/exchange_public_token`:

        - **Purpose:** To securely exchange the one-time `public_token` for a permanent `access_token`.
        - **Body (Pydantic Schema):** `{ "public_token": str }`
        - **Action:** Calls Plaid's `/item/public_token/exchange`. Encrypts the received `access_token` and stores it and the `item_id` in the `plaid_items` table, linked to the `user_id`.
    - `GET /budget/analyze_transactions`:
        - **Purpose:** The main endpoint to trigger the entire budget analysis flow.
        - **Action:**
            1. Retrieves the user's encrypted `access_token` from the database.
            2. Decrypts the token and fetches transactions from Plaid's `/transactions/get` (e.g., 90-120 days).
            3. Sends the cleaned transaction list to the Language Brain (Claude) with the prompt defined in 4.3.
            4. Receives the structured JSON from Claude.
            5. Performs the final calculations for `Net Income`, `Disposable Income`, and `Current Budget`.
            6. Returns the final analysis object (including the categorized lists for the "Increase" page) to the frontend.
    - `POST /budget/user_budget`:
        - **Purpose:** To save the user's final, confirmed budget(s) after the analysis flow.
        - **Body (Pydantic Schema):** `{ "current_budget_cents": int, "potential_budget_cents": Optional[int] }`
        - **Action:** Updates the `user_profile` table with these new values.

4.3.

Language Brain (Claude 4.5 Sonnet)

- **Role:** Budget Analyzer

System Prompt:
You are an expert financial analyst for an app called Paydown Pilot. Your sole job is to analyze a JSON list of bank transactions and categorize them into income, essential costs, and non-essential spending.

You must adhere to these HARD constraints:
1. Your ONLY output must be a single, valid, minified JSON object. Do not include any text, apologies, or explanations before or after the JSON.
2. You MUST NOT perform any math or suggest a budget. Your job is to categorize transactions and sum the totals *per category*.
3. All monetary values in your output JSON MUST be in positive integer cents.
4. 'identified_monthly_net_income_cents' should be the monthly total of all identified income/payroll deposits.

5. 'identified_essential_expenses_total_cents' should be the monthly total of all essential, non-discretionary costs (e.g., Rent, Mortgage, Utilities, Groceries, Insurance, Car Payments).
6. 'non_essential_subscriptions' is a list of *recurring* non-essential charges (e.g., Netflix, Spotify, Gym).
7. 'non_essential_discretionary_categories' is a list of *variable* spending categories (e.g., Restaurants, Shopping).

You will be given a list of transactions. You must return a JSON object matching this exact format:

```
{
  "identified_monthly_net_income_cents": <int>,
  "identified_essential_expenses_total_cents": <int>,
  "non_essential_subscriptions": [
    {"name": "<string>", "amount_cents": <int>},
    ...
  ],
  "non_essential_discretionary_categories": [
    {"category": "<string>", "total_cents": <int>},
    ...
  ]
}
```

-

Few-Shot Example Input:
JSON
```
[
  {"description": "PAYROLL ACME INC", "amount": -2500.00, "date": "2025-11-01"},
  {"description": "Shell", "amount": 45.20, "date": "2025-11-01"},
  {"description": "Netflix.com", "amount": 15.99, "date": "2025-11-02"},
  {"description": "CHIPOTLE 123", "amount": 22.50, "date": "2025-11-03"},
  {"description": "EAGLE REALTY RENT", "amount": 850.00, "date": "2025-11-05"},
  {"description": "Spotify", "amount": 9.99, "date": "2025-11-06"},
  {"description": "PAYROLL ACME INC", "amount": -2500.00, "date": "2025-11-15"},
  {"description": "WHOLE FOODS", "amount": 180.40, "date": "2025-11-16"},
  {"description": "CHIPOTLE 567", "amount": 18.00, "date": "2025-11-20"}
]
```

-

Example Output (Based on above):
JSON
```
{"identified_monthly_net_income_cents":500000,"identified_essential_expenses_total_cents":107560,"non_essential_subscriptions":[{"name":"Netflix.com","amount_cents":1599},{"name":"Spotify","amount_cents":999}],"non_essential_discretionary_categories":[{"category":"Restaurants/Bars","total_cents":4050}]}
```

•

4.4.

Database (PostgreSQL)

- **New Table: `plaid_items`**
  - `id` (uuid, primary key)
  - `user_id` (uuid, foreign key to `auth.users`, non-nullable)
  - `access_token` (text, non-nullable, **must be field-level encrypted**)
  - `item_id` (text, non-nullable, unique)
  - `created_at` (timestampz, default now())
- **Schema Changes: `user_profile` table**
  - **Add Column:** `current_budget_cents` (integer, nullable)
  - **Add Column:** `potential_budget_cents` (integer, nullable)
  - *(Note: The existing `monthly_budget_cents` column will be populated from one of these new columns when the user confirms their selection.)*

5. Security & Data Handling (CRITICAL)

This is the most critical section. Non-compliance is not an option.

- **Plaid Token Storage:** All Plaid `access_token`s will be stored in the `plaid_items` table. The `access_token` column **must** be encrypted at rest using field-level encryption (e.g., `pgsodium`). Encryption keys will be managed by a dedicated KMS (e.g., AWS KMS or HashiCorp Vault) and **never** stored in the codebase.
- **Transaction Data Handling:** The raw transaction data fetched from Plaid will **never** be stored in our PostgreSQL database. It will be fetched from Plaid, processed entirely in-memory by the FastAPI backend and the Language Brain, and then immediately discarded upon successful analysis. Only the final, *analyzed summary* (the JSON structure defined in 4.3) will be retained.
- **User Consent:** The **exact** consent language in section 3.2 must be presented to and accepted by the user *before* the Plaid Link flow is initiated.

6. Future Ideas (V2)

- **Household Budgets:** The feature will be expanded to support household budgets for users with shared or combined accounts. This will leverage Plaid's `/identity/get` product to create and link profiles for each household member and their respective financial data.