This is a comprehensive **Product Requirements Document (PRD)** designed specifically to prompt an AI Agent (like Replit's Agent) to build **Resolve 2.0**.

It translates high-level architectural concepts into specific code instructions, file structures, and logic flows compatible with your existing FastAPI/Next.js/Supabase stack.

Product Requirements Document: Resolve 2.0 – Agentic Financial Intelligence Extension

1. Executive Summary

Objective: Upgrade "Resolve" from a deterministic debt optimizer into a Neuro-Symbolic Financial Intelligence Platform.
Core Concept: Implement a "Two-Brain Architecture":

1. **The Math Brain (Existing):** Deterministic, safe, constraint-based solver (OR-Tools) for debt planning.
2. **The Agentic Brain (New):** Probabilistic, context-aware AI (LangGraph + Claude) for transaction enrichment, subscription detection, and "Sherlock Holmes" style investigation using external context (Email, Calendar, Macro-events).

Success Criteria:

- Automatically distinguish between Service vs. Subscription (e.g., Uber Ride vs. Uber One).
- Enrich transactions with "Reasoning Traces" explaining *why* a category was chosen.
- Connect to User Email (via Nylas) to parse receipts for route/item details.
- Connect to Macro-Events (via PredictHQ) to explain spending anomalies.

---

2. Technical Architecture & Stack

2.1 The Two-Brain Separation

- **The Bouncer (Pydantic):** Data moving from the Agentic Brain to the Math Brain must pass strict Pydantic validation. Agents cannot touch the financial ledger directly; they must propose changes which are validated.
- **Orchestrator (FastAPI):** The Python backend (/server) acts as the central hub, dispatching tasks to LangGraph agents asynchronously.

2.2 New Services & Libraries

- **Orchestration:** langgraph, langchain-anthropic
- **LLM:** Claude Sonnet 4.5 (via Anthropic API)
- **Email/Context:** Nylas (Universal Email/Calendar API)
- **Search/Research:** Serper (Google Search API for Agents)
- **Events:** PredictHQ (Macro-event intelligence)
- **OCR:** Mindee (Receipt parsing)

---

3. Database Schema Extensions (Supabase/PostgreSQL)

The Replit Agent must apply the following schema changes via SQL migrations.

3.1 New Table: subscription_catalog

Acts as the "Master Spreadsheet" for subscription intelligence.

SQL

```sql
CREATE TABLE subscription_catalog (
  id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  merchant_name TEXT NOT NULL,        -- e.g., "Uber"
  product_name TEXT NOT NULL,         -- e.g., "Uber One"
  cost_pattern NUMERIC,               -- e.g., 5.99
  currency TEXT DEFAULT 'GBP',
  recurrence_period TEXT,             -- e.g., "Monthly"
  is_verified BOOLEAN DEFAULT FALSE,   -- TRUE if verified by human or high-confidence AI
  created_at TIMESTAMPTZ DEFAULT NOW(),
  updated_at TIMESTAMPTZ DEFAULT NOW()
);
CREATE INDEX idx_subscription_merchant ON subscription_catalog(merchant_name);
```

3.2 Update Table: transactions

Add fields for the "Reasoning Trace" and Context.

SQL

```sql
ALTER TABLE transactions
ADD COLUMN is_subscription BOOLEAN DEFAULT FALSE,
ADD COLUMN subscription_id UUID REFERENCES subscription_catalog(id),
ADD COLUMN reasoning_trace JSONB, -- Stores the "Why" (e.g., "Matched via Email Receipt")
ADD COLUMN context_data JSONB,    -- Stores external data (e.g., "Taylor Swift Concert")
ADD COLUMN ai_confidence_score FLOAT;
```

---

4. Feature Implementation Modules

Module A: The Subscription Detective (Agentic Research)

Goal: Distinguish "Uber One" from "Uber Ride".

Logic Flow:

1. **Ingest:** Transaction arrives (Uber * Pending, £5.99).
2. **Lookup:** Check subscription_catalog for Merchant=Uber AND Amount=5.99.
3. **Hit:** If found, tag is_subscription = true, link subscription_id.
4. **Miss (The Agentic Step):**
   - Trigger **LangGraph Research Agent**.
   - **Tool:** Serper (Google Search).
   - **Prompt:** *"Search for subscription tiers for merchant 'Uber' in currency 'GBP'. Return JSON of plan names and costs."*
   - **Action:** Agent finds "Uber One matches £5.99".
   - **Write:** Agent inserts row into subscription_catalog.
   - **Tag:** Update transaction categorization.

Implementation Details:

- Create server/agents/subscription_agent.py.
- Use LangGraph to define a state machine: CheckDB -> SearchWeb -> UpdateDB.

Module B: The Context Hunter (Universal Email Connector)

Goal: Link bank charges to email receipts for Level-3 data (Route/Items).

Logic Flow:

1. **Trigger:** Ambiguous transaction (e.g., Uber, £45.50 - High for a normal ride).
2. **Connect:** Backend calls **Nylas API** (User must authorize Email Read scope).

3. **Query:** Search user inbox: from:uber.com AND subject:"receipt" AND after:{txn_date-1day} AND before:{txn_date+1day}.
4. **Parse:**
   - If Body is HTML: Use LLM/BeautifulSoup to extract "Pickup", "Dropoff".
   - If Attachment (PDF/Img): Send to **Mindee API**.
5. **Enrich:** Save result to transactions.context_data ({"route_start": "Euston", "route_end": "Wembley"}).

Implementation Details:

- Create server/services/email_context.py.
- Add /api/connect-email route for Nylas OAuth flow.

Module C: The Sherlock Layer (Macro Events)

Goal: Explain anomalies using real-world events.

Logic Flow:

1. **Input:** Transaction has context_data showing destination "Wembley Stadium" + Timestamp.
2. **Query:** Call **PredictHQ API**.
   - location: Wembley Coordinates.
   - active.gte: Transaction Timestamp.
   - category: Concerts, Sports.
3. **Match:** API returns "Taylor Swift Eras Tour".
4. **Reason:** AI updates reasoning_trace: "High transport cost justified by Event Travel (Taylor Swift)."

---

5. API Interface & Data Contracts

5.1 Backend Routes (server/routes/enrichment.ts or .py)

POST /api/enrich/transaction

- **Input:** Transaction Object.
- **Process:** Async trigger of Subscription Detective + Context Hunter.
- **Output:** 202 Accepted (Processing happens in background).

GET /api/transactions/{id}/trace

- **Returns:**
  JSON
  ```json
  {
    "category": "Event Travel",
  ```

```
 "confidence": 0.95,
 "trace": [
   {"step": "Bank Data", "detail": "Merchant: Uber, Amount: £45.50"},
   {"step": "Subscription Check", "detail": "Negative. Does not match Uber One (£5.99)"},
   {"step": "Email Context", "detail": "Found Receipt. Route: Home -> Wembley"},
   {"step": "Sherlock Layer", "detail": "Detected Macro Event: Taylor Swift Concert at Wembley"}
 ]
}
```

---

6. Step-by-Step Implementation Plan (For Replit Agent)

Instructions to Replit: Execute these phases sequentially. Do not proceed to the next phase until the current one compiles and tests pass.

Phase 1: Infrastructure & Database

1. Install dependencies: langgraph, langchain, nylas, mindee, google-search-results (Serper).
2. Run the SQL migration (Section 3.1 & 3.2) to update Supabase.
3. Update server/db.ts (or Python equivalent) to reflect new schema in ORM types.

Phase 2: The Subscription Detective (LangGraph)

1. Create server/agents/tools.py: Implement search_subscription_costs(merchant_name) using Serper API.
2. Create server/agents/graph.py: Define the LangGraph workflow.
   - Node 1: check_catalog.
   - Node 2: web_research.
   - Node 3: update_catalog.
3. Expose this via a Python service function classify_subscription(transaction).

Phase 3: Email Integration (Nylas)

1. Setup Nylas SDK in server/services/nylas_client.ts.
2. Implement fetch_receipts(query_params) function.
3. Implement parse_receipt_content(email_body) using a lightweight LLM call or Regex as a fallback.

Phase 4: Frontend "Reasoning" UI

1. Modify client/src/components/account-timeline.tsx.

2. Add a "Sparkle" icon next to enriched transactions.
3. On hover/click, show a Popover displaying the reasoning_trace JSON in a human-readable Timeline format (Shadcn Card or Accordion).
4. Add a visual badge for "Subscription" vs "One-off".

Phase 5: The Feedback Loop

1. Create an API endpoint /api/transactions/{id}/correct.
2. If a user corrects "Subscription" to "One-off", the backend must update the subscription_catalog to prevent future errors (e.g., mark that price point as "Invalid" for that merchant).

---

7. Critical Safety & The "Iron Wall" (Crucial)

- **Privacy First:** Email content is *never* stored in the DB permanently. Only extracted metadata (Route, Merchant) is saved to context_data.
- **Math Isolation:** The Agentic Brain (Python/LangGraph) *cannot* alter Debt Repayment Plans directly. It only updates Transaction Metadata. The OR-Tools solver recalculates plans based on the new metadata.
- **Rate Limits:** Cache PredictHQ and Serper results to avoid API cost overruns.

7.1 Separation of Concerns (Transactions vs. Liabilities)

- **The AI (Agentic Brain)** analyzes **Cash Flow** (Transactions). It determines: *"You spent £50 on a credit card bill."*
- **The Solver (Math Brain)** analyzes **Liabilities** (Debts). It determines: *"You should pay £50 to your credit card to minimize interest."*

7.2 The "No-Create" Rule

- **Constraint:** The Agentic System is **STRICTLY FORBIDDEN** from creating, updating, or deleting `debt_accounts` in the database.
- **Behavior:**
  - If the AI sees a transaction like "VIRGIN MONEY 5432", it classifies it as `Category: Debt Repayment`.
  - It **DOES NOT** create a "Virgin Money" debt account.
  - It **DOES NOT** feed this transaction into the OR-Tools solver as a new constraint.
  - It **ONLY** excludes this amount from "Disposable Income" calculations so the Budget Engine knows that money is gone.

7.3 User Authority

- **Debt Accounts:** Can *only* be added via the `POST /api/debts` endpoint, triggered by the User Interface (Manual Entry).
- **Reasoning:** A bank transaction tells us *what you paid*, not *what you owe*. Only the user knows the total balance and interest rate.

-

8. User Experience (UX)

- **The "Magic" Moment:** A user connects their bank + email. Suddenly, a generic "Uber" charge transforms into "Uber: Ride to Taylor Swift Concert" with a "Subscription: No" badge.
- **Transparency:** The user can always click "Why?" to see the logic. If the AI is wrong, the user corrects it, and the Agent learns.

Product Requirements Document: Part 2 – Categorization Engine & Frontend Experience

1. The "Resolve Master Taxonomy" (Categorization Logic)

Problem: Raw bank categories (e.g., "Eating Places", "Taxicabs") are too granular and often wrong.

Solution: Implement a 3-Layer Mapping System. We do not rely solely on the AI. We map raw data to a strict, user-friendly list.

1.1 The Master Categories (User-Friendly)

These are the *only* categories the user sees.

1. **Bills & Utilities:** (Energy, Water, Council Tax, Broadband)
2. **Subscriptions:** (Netflix, Gym, App Store - *Verified by Subscription Detective*)
3. **Transport:** (Train, Uber, Bus, Fuel)
4. **Groceries:** (Supermarkets, Bakeries)
5. **Eating Out:** (Restaurants, Fast Food, Coffee Shops)
6. **Shopping:** (Amazon, Clothing, Electronics)
7. **Entertainment:** (Cinema, Events, Betting)
8. **Health & Wellbeing:** (Pharmacy, Doctors, Hairdressers)

9. **Transfers:** (Internal movement, Credit Card payments - *Excluded from budget*)
10. **Income:** (Salary, Dividends, Refunds)
11. **Uncategorized:** (Fallback)

1.2 The Logic Hierarchy (Backend category-mapping.ts)

The mapping logic must follow this strict order of operations (Priority 1 > Priority 4):

- **Priority 1: The "Ghost" Check (Deterministic)**
  - IF transaction_links exists OR regex matches "Internal Transfer" $\rightarrow$ **Category: Transfers**.
- **Priority 2: The Subscription Catalog (Exact Match)**
  - IF Merchant exists in subscription_catalog $\rightarrow$ **Category: Subscriptions** (or Bills for utilities).
- **Priority 3: The Context Hunter (Evidence Based)**
  - IF context_data contains "Event Travel" $\rightarrow$ **Category: Transport** (Context: Event).
- **Priority 4: Ntropy Mapping (Fallback)**
  - Map Ntropy labels to Master Taxonomy:
    - taxicabs, railways $\rightarrow$ **Transport**.
    - restaurants, bars $\rightarrow$ **Eating Out**.
    - groceries $\rightarrow$ **Groceries**.

---

2. Frontend UX/UI Specifications (React/Shadcn)

2.1 The Transaction Card (account-timeline.tsx)

Redesign the list item to be information-dense but clean.

- **Left Icon:**
  - **Primary:** Merchant Logo (from logo URL in DB).
  - **Fallback:** Category Icon (e.g., 🛒 for Groceries) on a colored background.
- **Main Text:**
  - **Top Line:** Merchant Name (Cleaned).
  - **Bottom Line:** The "Detected Context" (e.g., *"Uber One Subscription"* or *"Ride to Wembley"*).
- **Right Side:**
  - **Amount:** Red for outgoing, Green for incoming.
  - **Badge:** The Master Category (e.g., [Transport]).
- **Interactions:**
  - **Click:** Opens "Transaction Details" Drawer.
  - **Hover on Context:** Shows "Reasoning Trace" Popover.

2.2 New Views & Filters

A. The "Merchant Deep Dive" (New View)

- **Trigger:** Clicking a transaction's Merchant Name.
- **Display:**
  - **Header:** Merchant Logo + Total Spent (All Time/This Year).
  - **Chart:** Monthly spending trend for *just* this merchant.
  - **List:** Filtered history of all transactions for this merchant.
  - **Insight:** "You spend an average of £45/month here."

B. The "Category Explorer" (Filter View)

- **UI:** Horizontal scrollable pills at the top of the timeline: `[All] [Groceries] [Transport] [Bills] ...`
- **Logic:** Clicking `[Transport]` filters the list via client-side state (or DB query for pagination).
- **Summary Card:** When filtered, show a card at the top: *"Total Transport spending this month: £145.50"*.

C. The "Recurring View" (Subscription Monitor)

- **New Page:** `/subscriptions`
- **Content:** A table of only transactions tagged `is_subscription=TRUE` or `is_recurring=TRUE`.
- **Columns:** Merchant, Cost, Frequency, "Next Due Date" (Calculated).

---

3. Algorithmic Overview Figures (The Math)

Crucial Rule: Dashboard totals must be calculated via **SQL Aggregation**, never AI summarization.

3.1 The "Spend" Tile Logic

Formula:

SQL

```
SELECT SUM(amount)
FROM transactions
WHERE
  user_id = :current_user
  AND date >= :start_of_month
  AND amount < 0          -- Outgoing only
  AND is_excluded = FALSE    -- IMPORTANT: Ignore internal transfers/ghost pairs
  AND category != 'Debt Repayment' -- Optional: Separate debt from spending
```

-

3.2 The "Income" Tile Logic

Formula:

```
SQL
SELECT SUM(amount)

FROM transactions

WHERE

  user_id = :current_user

  AND date >= :start_of_month

  AND amount > 0

  AND is_excluded = FALSE    -- Ignore refunds/internal transfers
```

-

3.3 The "Bills vs. Discretionary" Split

- **Fixed Costs:** Sum of Bills, Subscriptions, Debt Repayment.
- **Discretionary:** Sum of Eating Out, Shopping, Entertainment.
- **Why:** This allows the user to see *"I spent £2,000, but £1,200 was fixed bills. I only controlled £800."*

---

4. Implementation Steps for Replit

1. **Step 1 (Backend):** Update server/services/category-mapping.ts with the **1.2 Logic Hierarchy**. Ensure it strictly outputs one of the **1.1 Master Categories**.
2. **Step 2 (Database):** Create a database view or API endpoint GET /api/stats/monthly that runs the SQL queries in **3.1 & 3.2**. Do not calculate this on the frontend to ensure accuracy.
3. **Step 3 (Frontend):**
   - Refactor TransactionRow component to support the Logo + Context layout.
   - Add the CategoryFilter component (Pills).
   - Create the MerchantDetailsModal component.
4. **Step 4 (Validation):** Compare the "Dashboard Total" against the sum of the transaction list manually to ensure is_excluded logic works.