Master Development Guide: The "Smart Budget Detective" (TrueLayer + Ntropy)

Role: You are the Lead Backend Architect.
Objective: Implement a robust transaction enrichment pipeline that acts as a middleware between TrueLayer (Raw Data) and our Budget Logic.
Core Philosophy: "Clean Input, Deterministic Output." We do not trust raw bank strings. We trust enriched, labeled data.

1. The Data Flow Architecture

You are building a pipeline that flows in one direction. Do not mix these stages.

Code snippet

```
graph TD
    A[TrueLayer API] -->|Raw JSON| B(Normalization Service)
    B -->|Cleaned Object| C[Ntropy SDK]
    C -->|Enriched JSON| D{Budget Logic Engine}
    D -->|Label: Loan/BNPL| E[Detected Debts List]
    D -->|Label: Subscription| F[Recurring Expenses List]
    D -->|Label: Unknown| G[Discretionary Spend]
    E --> H[User Confirmation UI]
```

2. Implementation Steps (Execution Order)

Phase 1: The Normalization Service (Source)

Goal: Extract raw data from TrueLayer and sanitize it before it touches any other logic.

- **TrueLayer Input:** You will receive a JSON payload from the /transactions endpoint.
- **Critical Handling Rules:**
    - **Amounts:** TrueLayer sends negative numbers for spending (e.g., -45.00). You **MUST** convert these to **absolute (positive) floats** for Ntropy.
    - **Direction:** You must derive incoming vs outgoing based on the TrueLayer

transaction_type (DEBIT/CREDIT) or the sign of the amount.
    - **Dates:** Truncate ISO timestamps (2023-10-27T10:00:00Z) to simple strings ("2023-10-27").

Phase 2: The Enrichment Integration (Ntropy)

Goal: Use the Ntropy Python SDK to turn "dumb" strings into "smart" objects.

- **SDK Setup:** Initialize ntropy_sdk.SDK with the API key.
- **The Mapper Function:** Create a function map_truelayer_to_ntropy(tl_tx) that returns an NtropyTransaction object.
    - *Input:* Your normalized TrueLayer dict.
    - *Output:* ntropy_sdk.Transaction object.
    - *Mandatory Fields:* amount (positive), entry_type, description, date, iso_currency_code, transaction_id.
    - *Crucial Context:* You **MUST** pass account_holder_id. Use a hashed version of the user's ID. This allows Ntropy to detect **recurrence** (e.g., "This user pays Netflix every 30 days"). Without this, the "Fixed Cost" detection will fail.

Phase 3: The "Triage Nurse" Logic (The Budget Engine)

Goal: Interpret the Ntropy results to classify the user's spending.
Do not just save the data. You must write a Classifier Function that sorts transactions into three internal buckets based on Ntropy's labels and recurrence fields.

- **Bucket A: "Potential Debt" (The Target)**
    - *Triggers:* Ntropy labels contain loan, mortgage, finance, bnpl, credit card, overdraft.
    - *Action:* Flag for the **"Debt Trap" User UI**.
    - *Budget Impact:* **Excluded** from "Fixed Costs" initially (to prevent double counting).
- **Bucket B: "Fixed Living Costs"**
    - *Triggers:* Ntropy labels contain utilities, insurance, subscription. **OR** recurrence is detected but it is *not* a debt.
    - *Action:* Auto-add to "Monthly Bills."
    - *Budget Impact:* Deducted from Income.
- **Bucket C: "Discretionary"**
    - *Triggers:* food, shopping, entertainment (that are not recurring).
    - *Budget Impact:* Counts towards "Safe to Spend."

Phase 4: Storage Strategy

- **Database Schema:** Create a transaction_cache table.
    - Store the ntropy_transaction_id, merchant_clean_name, logo_url, is_recurring, and the original_raw_description.
    - **Why?** So we never pay to enrich the same transaction twice. Check this cache before calling the API.

3. User Experience (UX) Requirements

The logic must drive the UI. The user should never see raw JSON or bank codes.

1. **The "Debt Confirmation" Modal:**
   - Present the items from **Bucket A** to the user.
   - *Copy:* "We found these recurring payments to **[Merchant Name]**. Are these debts you want to pay off?"
   - *Visuals:* Use the logo and website URL from Ntropy to display a recognizable icon next to the name.
2. **The "Recurrence" Insight:**
   - If Ntropy says a payment is recurring, display a small "cycle" icon next to it.
   - *Tooltip:* "Billed monthly on the [Day]th."

4. Replit Agent Specific Instructions

- **Language:** Python (FastAPI) for backend; React for Frontend.
- **Validation:** Use **Pydantic** for all data ingress. Define a TrueLayerIngestModel and an NtropyOutputModel to ensure type safety.
- **Error Handling:** If Ntropy fails or times out, fallback to a "Raw Mode" where we just display the description and ask the user to classify it manually. **Do not block the app.**
- **Security:** Never log the raw API keys. Use environment variables (os.environ).

Final Output Requirement:
Generate the EnrichmentService class in Python that handles the full lifecycle: ingest -> convert -> enrich -> classify.