Coding Assignment: Bank Transaction Categorization API

Design and implement a small, well-structured API service in **Python** that **manages and classifies bank transactions into categories** (e.g., “Amazon” → “Shopping”). The goal is to evaluate your **API design**, **problem solving**, and **product thinking**—not machine learning sophistication.

What You’ll Build

Create a service that:

1. **Manages users** with CRUD APIs.
2. **Manages merchants** (mandatory entity) with CRUD APIs.
3. **Manages transactions** with CRUD APIs.
4. **Classifies** transactions into categories using rules, merchant lookups, and heuristics.
5. **Filters & queries** transactions by user, category, date range, amount range, etc.
6. **Explains** why a category was chosen (top signals).

Implementation must be in **Python** (any framework such as FastAPI, Flask, or Django REST).

Functional Requirements

Entities

- **User** – represents the account holder.
- **Merchant** – mandatory lookup entity with aliases, MCCs, and default categories.
- **Transaction** – represents a bank transaction, always linked to a merchant.

Endpoints (minimum)

Users

- `POST /users` – Create a new user.
- `GET /users/{id}` – Retrieve user details.
- `PUT /users/{id}` – Update user details.

- `DELETE /users/{id}` – Delete a user.
- `GET /users` – List users (with optional filters: name, email).

Merchants

- `POST /merchants` – Create a merchant.
- `GET /merchants/{id}` – Retrieve merchant details.
- `PUT /merchants/{id}` – Update merchant details.
- `DELETE /merchants/{id}` – Delete a merchant.
- `GET /merchants` – List merchants with filters (by alias, MCC, etc.).

Transactions

- `POST /transactions` – Create a transaction (must include merchant_id).
- `GET /transactions/{id}` – Retrieve transaction details.
- `PUT /transactions/{id}` – Update a transaction.
- `DELETE /transactions/{id}` – Delete a transaction.
- `GET /transactions` – List transactions with filters:
    - By user_id
    - By merchant_id
    - By category
    - By date range
    - By amount range

Classification

- `POST /classify` – Classify one transaction (without persisting).
- `POST /classify/bulk` – Classify many transactions at once (without persisting).

Health

- `GET /health` – Liveness/readiness signal.

Request/Response Contracts

Transaction (input)

```
None
{
  "id": "txn_123",
  "user_id": "user_42",
```

```
  "merchant_id": "m_amazon",
  "posted_at": "2025-06-03T12:34:56Z",
  "amount": -129.99,
  "currency": "USD",
  "raw_description": "AMAZON Mktp US*2A3BD",
  "mcc": "5942",
  "channel": "ecom",
  "geo": {"country": "US"},
  "account_id": "acc_001"
}
```

Classification (output)

```
None
{
  "transaction_id": "txn_123",
  "category": "Shopping > Online Marketplace",
  "confidence": 0.93,
  "why": [
    "Matched alias 'AMZN Mktp' → Amazon",
    "MCC 5942 aligns with Shopping"
  ],
  "alternatives": [
    {"category": "Subscriptions > Digital Services",
"confidence": 0.71},
    {"category": "Shopping > General Retail", "confidence": 0.64}
  ]
}
```

Required Logic

1. **Normalization & Enrichment** – clean description, extract merchant candidates, MCC, and channel.
2. **Candidate Generation** – from merchant KB, MCC maps, regex rules, and semantic similarity.

3. **Scoring** – use weighted blend of signals:

```
None
FinalScore = w1*MerchantMatch + w2*SemanticSimilarity +
w3*RuleConfidence
```

4. **Re-ranking & Diversity** – avoid duplicate categories.
5. **Fallbacks** – MCC map or Uncategorized if low confidence.
6. **Explainability** – always provide `why[]` in output.

Data Model (suggested)

User

- `user_id`, `name`, `email`, `created_at`

Merchant

- `merchant_id`, `display_name`, `aliases[]`, `typical_mccs[]`, `default_category`

Transaction

- `id`, `user_id`, `merchant_id`, `posted_at`, `amount`, `currency`, `raw_description`, `normalized_description`, `mcc`, `channel`, `geo{}`

Category Taxonomy (sample)

- Shopping: Online Marketplace, General Retail, Electronics, Apparel, Home & Furniture
- Food & Drink: Restaurant, Fast Food, Coffee Shop, Grocery
- Transport: Rideshare, Public Transit, Fuel
- Subscriptions: Digital Services, Streaming, Software
- Bills & Utilities: Electricity, Water, Internet/Mobile
- Cash & ATM: Withdrawal, Deposit
- Transfers: Internal, External
- Fees & Charges: Bank Fee, Interest

- Travel: Airline, Hotel, Short-term Rental
- Healthcare: Pharmacy, Medical Services

Examples

Raw Description	MCC	Channel	Expected Category	Notes
"AMAZON Mktp US*2A3BD"	5942	ecom	Shopping > Online Marketplace	Alias & MCC match
"AMAZON PRIME*12MO"	4899	ecom	Subscriptions > Digital Services	Recurring pattern
"UBER *TRIP HELP.UBER.COM"	4121	ecom	Transport > Rideshare	Regex + MCC
"STARBUCKS #04210"	5814	pos	Food & Drink > Coffee Shop	
"SAFEWAY #2207"	5411	pos	Food & Drink > Grocery	
"DELTA AIR 0061234567890"	4511	ecom	Travel > Airline	Ticketing reference
"APPLE.COM/BILL"	5734	ecom	Subscriptions > Digital Services	Recurring biller
"CHASE OVERDRAFT FEE"	6012	bank	Fees & Charges > Bank Fee	Override MCC
"ATM CASH WD 123 MAIN ST"	6011	atm	Cash & ATM > Withdrawal	Suppress merchant semantics
"VENMO PAYMENT 1234"	4829	ecom	Transfers > External	P2P transfer
"CVS/PHARMACY #1234"	5912	pos	Healthcare > Pharmacy	

Performance & Concurrency Approach (required)

Provide a brief write-up (bulleted is fine) describing how you would handle throughput and latency under load:

- **Bulk classification strategy:** how `/classify/bulk` processes large inputs (e.g., streaming reads/writes, chunking/batching, memory bounds).
- **Concurrency model:** sync vs **async** (FastAPI/uvicorn workers), and/or **multiprocessing** for CPU-bound regex/MCC parsing; when to use each.
- **Backpressure & timeouts:** request limits, per-batch size, graceful degradation for oversized batches.
- **Parallel I/O:** DB access patterns (pooled connections, N+1 avoidance, pagination) and caching of merchant/MCC lookups.
- **Profiling & complexity:** expected Big-O hot spots, profiling tools you'd use (e.g., cProfile) and what you'd optimize first.
- **Observability:** minimal metrics you'd expose (throughput, p95 latency for classify endpoints, error rates) and how you'd test under load.

Stretch Goals (Optional)

- **Advanced search queries** (multiple filters combined).
- **Pagination and sorting** on `GET /transactions`.
- **Export endpoint** (`GET /transactions/export`) to CSV.
- **Role-based access** for admin vs user.
- **Background jobs** for async normalization.
- **Semantic similarity** (optional; rules-first is primary).

Additional Requirement

- Provide a short description of the **approach you would use to ensure concurrency and performance**. For example: handling large bulk classifications (100k+), async request processing, multiprocessing, streaming vs batch trade-offs, and scaling strategies.