Mini Ledger with Idempotent Transfers & Clean Architecture

Objective

Build a small but production-oriented ledger service that supports multiple currencies, manages merchant balances, and processes transfers between merchants with strict idempotency guarantees, correct behavior under concurrency, and transfer fees that must be handled consistently.

Functional Requirements

1. Data Model (Persistent Storage Required)

You design the data model yourself. It must persist data across restarts.
The system must support:

- merchants
- balances in multiple currencies
- transfers between merchants
- an idempotency mechanism that prevents a transfer from being executed more than once
- transfer fees that must be calculated and applied to each transfer

The internal structure is up to you.
You choose how fees are calculated, stored, and handled.

---

2. Async Requirement (Mandatory)

The service must be implemented using asynchronous Python (`async / await`).
This applies to:

- the web framework
- request handling
- database/storage operations (if supported by chosen tools)
- internal logic where async is appropriate

You may choose any async framework or libraries (for example: FastAPI, Aiohttp).

API Requirements

Use any Python web framework.

Create Merchant — `POST /merchants`

Creates a merchant with an initial balance.

Example request:

- `{`
- `  "merchant_name": "alice_store",`
- `  "currency": "BTC",`
- `  "initial_balance": "0.00001"`
- `}`

Get Merchant — `GET /merchants/{merchant_name}`

Returns merchant details, including balances.

Get Merchant Balance — `GET /merchants/{merchant_name}/balance`

Returns merchant balance(s).

Execute Transfer — `POST /transfers`

Executes a transfer between merchants.

- The idempotency key must be provided via the `Idempotency-Key` HTTP header.
- The fee must be applied as part of the transfer.

Example header:

- `Idempotency-Key: abc-123-xyz`

Example body:

- `{`
- `  "from_merchant": "alice_store",`
- `  "to_merchant": "bob_shop",`
- `  "currency": "BTC",`
- `  "amount": "0.000005"`
- `}`

Required behavior:

- deduct the transfer amount and the fee from the sender
- credit the receiver with the transfer amount
- do not allow negative balances
- reject transfers cleanly if funds are insufficient (amount + fee)
- repeated requests with the same `Idempotency-Key` must be idempotent and return the same result

---

List Transfers — `GET /transfers?from=&to=&currency=`

Returns a filtered list of transfers.

Non-Functional Requirements

1. Architecture

The project should demonstrate clear separation between:

- business logic
- persistence
- HTTP/transport layer

The structure is up to you.
Document decisions in the README.

---

2. Logging

Logging must be present.
You choose what to log and how.

---

3. Error Handling

The service must:

- use appropriate HTTP status codes
- return structured machine-readable error responses
- explicitly handle expected error scenarios, including:
- invalid or missing merchant

- insufficient funds (including fee)
- invalid input
- missing or invalid idempotency key
- fee calculation or validation issues

The exact error format is up to you.

---

4. Concurrency & Consistency

Balances must never become negative under concurrent operations.
Any approach is acceptable, including:

- transactions
- locking
- optimistic/pessimistic strategies
- other concurrency control models

Document the chosen approach in the README, including handling of amount + fee under concurrency.

---

5. Code Quality

- maintainable project structure
- meaningful naming
- type hints where appropriate
- no business logic inside route handlers
- no mixing persistence with domain logic

6. Tests

Use pytest or similar.
Tests must cover:

- idempotency behavior (including fee handling)
- insufficient funds scenarios
- main API flows
- at least one concurrency-related scenario

Simulated concurrency is acceptable.

7. Documentation

The README must include:

- how to run the project
- example API requests
- architecture overview
- idempotency design
- concurrency strategy
- multi-currency handling
- fee calculation and handling
- potential improvements