UATP Coding Challenge: External Payment Processor Integration & Normalization Service

Estimated Duration: 2–3 hours
Tech Stack: .NET/C#, EF, SQL Server

Scenario

Your team is building a service that ingests payment transaction data from multiple external providers (e.g., PayPal, Trustly, BitPay), normalizes the data, stores it internally, and exposes it via an internal API for other systems to consume.

Each provider uses a slightly different format, but you need to unify it into a common internal structure.

Objectives

1. Design the Normalized Data Model

Define a database schema for a normalized PaymentTransaction entity:

- TransactionId
- ProviderName (e.g., " PayPal ", " Trustly ")
- Amount
- Currency
- Status (enum: Pending, Completed, Failed)
- Timestamp (UTC)
- PayerEmail
- PaymentMethod (e.g., "CreditCard", "ACH", "Wallet")

You may create provider-specific models or mapping classes as needed.

---

2. Implement API Endpoints

- POST /ingest/{providerName}
  Accepts raw transaction payloads from a specific provider

    - Maps incoming data to your normalized format
    - Stores the transaction in the database
    - Returns a success or error message
- GET /transactions
  Returns all normalized transactions with optional filters:
    - ProviderName
    - Status
    - Date range (from/to)
- GET /summary
  Returns:
    - Total number of transactions
    - Total volume per provider
    - Breakdown of statuses (Pending, Completed, Failed)

---

3. Unit Testing

- Write unit tests for reconciliation logic and core service components
- Use xUnit or NUnit

---

4. Required Features

- Swagger / OpenAPI documentation
- Input validation and basic error handling
- Clean separation between mapping, business logic, and data access
- Unit tests for normalization logic and ingestion service

---

5. (Optional Bonus)

- Simulate webhook-like behavior where providers "push" transaction payloads
- Handle idempotency (e.g., don't insert the same transaction twice)

- Include lightweight authentication or API key header

Evaluation Criteria:

- Correctness – Is provider data properly mapped and stored in a normalized format?
- Database Design – Is the schema adaptable and future-proof for new providers?
- Code Quality – Is the logic modular, testable, and well-organized?
- Unit Tests – Are the key components (e.g., mapping logic) tested?
- Scalability – Would this design handle new providers or more transaction volume easily?
- Security – Are basic validation and safe parsing practices in place?
- Documentation – Is Swagger complete? Is the README clear and helpful?

Deliverables

- Source code (GitHub or zip)
- README with:
    - Setup instructions
    - Example provider payloads
    - Sample queries for filters
    - Notes on design and assumptions