Accelo Take-Home Assignment - Multi-Currency Invoices API

Overview

At Accelo, we're working on adding **multi-currency support** to our platform. In this exercise, you'll design and implement a backend service to support invoice creation and viewing in multiple currencies. You'll model the data, structure the API endpoints, and implement the core logic required for working with base and foreign currencies.

We want to see how you approach a real-world engineering challenge from system and API design to handling edge cases and data integrity.

Please don't worry about production-readiness or full coverage — we're more interested in your problem-solving and design choices.

Scenario

Accelo invoices are usually created in the base currency configured for the account. With the introduction of multi-currency, an invoice can now be created in a **foreign currency**, with an exchange rate to convert to the base currency. For example, if a company based in Australia (base currency AUD) creates an invoice in USD for $1,000 using a rate of 1 USD = 1.52 AUD, the system should store both the original amount and the calculated amount_in_base_currency = 1,520 AUD. These details are used for reporting, financial records, and payment summaries.

You are tasked with designing and implementing a system to:

- Store invoices in both base and foreign currencies
- Store exchange rates (with timestamp)
- Ensure accurate financial records without recalculating values once finalized

Requirements

Business Logic

- An **invoice** belongs to a company and has:
    - Title
    - Amount (in foreign currency)
    - Currency code (e.g., USD, PHP, INR)
    - Base currency (e.g., AUD)
    - Exchange rate used

    - Exchange rate timestamp
    - Amount in base currency (auto-calculated and stored)
- A **company** has:
    - ID
    - Name
    - Base currency (e.g., AUD)
- An **exchange rate** has:
    - From currency
    - To currency
    - Rate (e.g., 1 USD = 1.52 AUD)
    - Timestamp
    - Source (e.g., Manual, Live)

API Endpoints

Companies

```
Unset
GET    /companies          # Obtain the details of all companies
GET    /company/:id        # Obtain the details of a company
POST   /company            # Create a company with base currency
POST   /company/:id        # Update name or base currency
```

Exchange Rates

```
Unset
GET    /rates?from=USD&to=AUD
POST   /rates              # Add or update an exchange rate
```

Invoices

```
Unset
GET    /invoices         # Obtain the details of all invoices
GET    /invoice/:id      # Obtain the details of an invoice
POST   /invoice          # Create an invoice in a foreign currency
```

Objectives

- Design the **database schema.**
- Implement the **API endpoints.**

- Structure the logic for the API endpoints and relevant business logic in well defined modules.
- Include basic **validation and error handling.**
- Handle **calculation and storage** of `amount_in_base_currency` using the exchange rate.
- Support **historical accuracy**: once an invoice is sent to the customer, its exchange rate and base amount must not change, even if future rates do.
- Unit tests for your business logic.

Deliverables

1. Source code (in your preferred language) in a folder structure you would use for a real project.
2. Database schema (SQL or Object Relational model or ERP diagrams)
3. Setup instructions (how to run locally)
4. Sample `curl` or Postman requests with expected JSON responses for:
    - Creating a company
    - Creating an exchange rate
    - Creating an invoice
    - Viewing an invoice with all calculated fields

Notes

- No authentication is needed.
- No frontend is needed.
- Feel free to use any database (e.g., MySQL, PostgreSQL, MariaDB).
- Use any backend language/framework (e.g., Python, Perl, Java, PHP).
- Focus on **clarity, correctness, and maintainability** over completeness.
- You can use https://docs.openexchangerates.org/reference/latest-json in order to obtain the latest exchange rates.

Bonus (Not Required)

Feel free to go above and beyond with any of the following:

- Using the GraphQL schema and API to display the data.
- Handling cases like invalid currency codes or missing exchange rate.
- Design thoughts around using REST vs GraphQL.
- Playwright (or similar) automation tests for the API endpoints.