MyCurrency : Technical Design Document

Author: Adam Abidi

Project: MyCurrency Backend (Django + DRF)

1. Introduction

MyCurrency is a Django REST API for retrieving, storing, and converting currency exchange rates from
multiple external providers. The platform is designed for extensibility, clean separation of concerns, and
good performance, using the Adapter Design Pattern to isolate provider-specific logic.

It supports:
- Multiple configurable providers
- Provider priority and automatic failover
- Historical and real-time rate retrieval
- Asynchronous historical loading using asyncio
- Database caching of fetched rates
- Multi-currency conversion
- Admin UI for quick conversions
- OpenAPI documentation (Swagger)
- Handling multiple dates × multiple target currencies (bulk processing)

2. Technical Requirements

- Python 3.11+
- Django 5.x
- Django REST Framework
- drf-spectacular
- SQLite (default)
- asyncio

Setup Instructions

1. Clone the project

* git clone https://github.com/AdamAbidi/MyCurrency.git
* cd MyCurrency

2. Install dependencies

pip install -r requirements.txt

3. Run the server

python manage.py runserver

Admin Access

* URL: http://localhost:8000/admin/
* Username: admin
* Password: admin

Database already includes:

- **Providers**
- **Currencies**
- **Exchange rates**

API Documentation

http://localhost:8000/api/v1/docs/

Admin Converter View

http://localhost:8000/admin/converter/

3. Architecture Overview

Two main Django apps:

currency/
- Models: Currency, CurrencyExchangeRate
- Views, async & sync services, validators

provider/
- Model: Provider
- Adapters under adapters/
- Dynamic adapter registry
- Provider-specific error types

4. Data Models

- ### Currency

code (CharField) : ISO currency code (e.g., *USD*, *EUR*). Primary identifier.

name (CharField) : Human-readable currency name (e.g., *US Dollar*).

symbol (CharField) : Optional symbol used for display (e.g., *$*, *€*).

- ### Provider

name (CharField) – Display name of the provider (e.g., *Currency Beacon*).

adapter_key (CharField) – Unique identifier used to look up the adapter class dynamically.

priority (IntegerField) – Defines failover order; the system tries lower numbers first.

is_active (BooleanField) – Enables or disables the provider.

api_url (CharField, optional) – Base URL only for URL-based providers; other provider types can ignore this.

- **CurrencyExchangeRate**

source_currency (ForeignKey → Currency) – The “from” currency.

exchanged_currency (ForeignKey → Currency) – The “to” currency.

valuation_date (DateField) – The date for which the rate applies.

rate_value (DecimalField) – The numerical exchange rate for the pair.

5. Adapter Design Pattern

Each provider implements a custom adapter extending BaseProviderAdapter.
Adapters define: get_exchange_rate_data(base, quote, date)
Provider types may be HTTP APIs, SDK-based, file-based, or mocks.
Dynamic registry maps adapter_key → import path.

6. Exchange Rate Retrieval Logic

1. Look up DB first.
2. If missing: iterate active providers by priority.
3. Load adapter dynamically.
4. Try fetching; if success → save to DB.
5. If all providers fail → domain error.

Special case: identical currencies → rate = 1.0.

7. Data Storage Strategy

- Only missing rates stored in DB.
- Async loader processes multiple dates × multiple currencies efficiently.

8. Asynchronous Historical Loading

Built with asyncio for concurrency.
Each date spawns async tasks per target currency.
Each task checks DB → fetches → saves.

9. REST API Overview

Currencies:

- GET /api/v1/currencies/
- GET /api/v1/currencies/{id}/

Exchange Rates:

- GET /api/v1/exchange-rates/history/
- GET /api/v1/exchange-rates/history-async/
- GET /api/v1/exchange-rates/convert/
- GET /api/v1/exchange-rates/convert-multi/

Providers:

CRUD

- Toggle active
- Set priority

Extras:

- Admin converter
- Swagger docs

10. Test Coverage

Unit tests include:

- adapter registry
- provider failover
- caching
- retrieval logic
- conversion
- async workflow
- provider priority validation

11. Future Improvements

- Celery + Redis background ingestion
- Save provider reference on each saved rate
- Optional caching layer
- Additional adapter types
- Authentication
- Integration tests