Python Backend Test

Introduction

MyCurrency will be a web platform that allows users to calculate currency exchange rates.

MyCurrency will use an external provider CurrencyBeacon to retrieve and store daily currency rates. As MyCurrency is a flexible platform, it has to be designed to use multiple currency data providers. So, it won't only work with CurrencyBeacon but also with any other provider that might have different APIs for retrieving currency exchange rates.

You can easily create a free account at https://currencybeacon.com/register
Documentation is available at: https://currencybeacon.com/api-documentation

MyCurrency will work with EUR, **CHF**, **USD**, and **GBP** as available currencies.

Functionalities

Create a Django project with **Python 3.11** / **Django 4 or 5** that stores currencies and daily currency exchange rates. The application should also expose a REST API that would be eventually used by a Frontend or Mobile application.

Exchange rates data

Our goal is to feed data with a Django model **CurrencyExchangeRate** with data from different sources. Here there is a draft:

```python
class CurrencyExchangeRate(Model):
       source_currency  =  models.ForeignKey(Currency,related_name='exchanges',
on_delete=models.CASCADE)
    exchanged_currency = models.ForeignKey(Currency,on_delete=models.CASCADE)
    valuation_date = models.DateField(db_index=True)
           rate_value   =   models.DecimalField(db_index=True,decimal_places=6,
max_digits=18)

class Currency(ProtectedModel):
    code = models.CharField(max_length=3, unique=True)
    name = models.CharField(max_length=20, db_index=True)
    symbol = models.CharField(max_length=10)
```

The currency rates data retrieval must be built following the **Adapter Design Pattern** so that the rest of the platform (e.g., views) remains unaware of the specific functionality of the individual currency exchange rate data providers.

The way that this data retrieval procedure will be called will meet this method signature:

```python
def get_exchange_rate_data(source_currency,exchanged_currency, valuation_date, provider)
```

For the task, we ask you to at least implement two exchange rates providers:

- **CurrencyBeacon**: Driver that integrates with CurrencyBeacon.
- **Mock**: The mock provider should be capable of generating random mock data.

Exchange rates API

Create a REST / JSON API with Django Rest Framework with the following functionalities:

Service	Description	Parameters	Expected response
Currency rates list	Service to retrieve a List of currency rates for a specific time period	- source_currency - date from - date to	A time series list of rate values for each available Currency
Convert amount	Service that calculates (latest) amount in a currency exchanged into a different currency (currency converter)	- source_currency - amount - exchanged_currency	An object containing at least the rate value between source and exchange currencies.
Currency CRUD	Rest service to manage Currencies		CRUD of currencies

The platform will return data from its backend if available; otherwise, it will request the data from the providers. Implement the optimal logic to enhance the performance and functionality of retrieving exchange rates.

As we aim to build a resilient and robust portal, consider that it may integrate with various third-party providers. The choice of provider will depend on a "priority" criteria assigned to each provider within the platform. Additionally, providers can be activated or deactivated dynamically at runtime.

- **Change Provider priority.** At any time you can change the Provider priority, to make another one the “default” data source.

- **Providers have to be pluggable.** Platform providers can be activated or deactivated dynamically at runtime.
- **Provider services have to be resilient.** If one of the providers “fails” or gets no rates, the platform has to try with the next one in order of priority.

Exchange Rate Evolution backoffice / admin

Create a small back office site (using Django admin would ease this task) with the following functionalities:

- Converter View. Provide an online version of the Currency "converter" where it could be possible to set a source currency and multiple target currencies.

Note: Ensure the new section is accessible through the Django admin interface.

Async task to load historical data

Historical data loading can process thousands of exchange rates. We want you to implement an asynchronous method to load this historical data as quickly and efficiently as possible, minimizing resource consumption.
What do you think is a better option: concurrency or parallelism?

Provide historical data for the test

We want you to provide a mechanism to ingest real-ish exchange rate data for testing purposes.

API versioning/scope (optional)

Keep in mind the possibility that MyCurrency API could have different API versions or scopes.

Notes

- Save the code in any GIT repository platform you want (Github, Bitbucket, Gitlab…) and send us the link when it's ready.
- Provide a Postman collection for your API.
- Add a readme.txt/rst/md file with a brief description/instructions to set up the project.
- Follow good practices or guidelines you would use in a real project.
- Think about code readability, structure, flexibility, reusability, and clear naming.
- Let us know about any improvements that could be made.