Backend Test

V1.3

Introduction

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

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

You can easily create a free account at https://fixer.io/
Documentation for fixer.io is available at: http://data.fixer.io/api/latest

ExoCurrency will work with EUR, CHF, USD and GBP currencies.

Functionalities

Create a **Django project with Python 3 / Django 2.0** 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 with data a **Django** model CurrencyExchangeRate with data from different sources, we provide you a draft:

```
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)  # created is a datetime field, but we
could use it as valuation_date
   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 has to be built following an **"Adapter"** design pattern so that the rest of the platform (views, etc.) doesn't know about the specific functionality of each currency exchange rate data **Provider**. The way that this data retrieval procedure will be called will be a task with this signature:

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

Where provider could be built with the Providers that implement the following adapters:

- Fixer: Driver that integrates with fixer.io
- Mock: A mock driver with random data that can be used for testing, etc.

For building the Fixer adapter create a free account at https://fixer.io/
Documentation for fixer.io is available at: http://data.fixer.io/api/latest

Exchange rates API

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

- List of currency rates for a specific time period
  Parameters: date from / date to
- Calculate amount in a currency exchanged into a different currency
  Parameters: origin currency, amount, target currency
- Retrieve time-weighted rate of return for any given amount invested from a currency into another one from given date until today:
  Parameters: origin currency, amount, target currency, date invested
  Help: https://www.investopedia.com/terms/t/time-weightedror.asp

These views will remain in already stored data (**CurrencyExchangeRate Model**) if available, or call **get_exchange_rate_data** to retrieve data from the Provider.

The decission of which provider to use between those available, depends on which "order" has each **Provider.** *(hint: Provider is a Model!)*

Exchange Rate Evolution backoffice

Create a small backoffice where exchange rate evolution comparison between different currencies graphically. Have in mind:

- Must have a good performance to retrieve historical data for many currencies.
- The backoffice must populate mocked data (to avoid fixer provider restrictions) trying to avoid generating random data.
- You can change Provider order, to make another one the "default" data source.

Batch procedure to retrieve exchange rates (optional)

Design a procedure to implement exchange rate caching / storing to avoid calling an exchange rate provider each time API is called.

- This procedure will take data from a CSV (provide some simple example) or JSON file, parse it and store it into your models.

API versioning / scope (optional)

Have in mind the possibility that ExoCurrency 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
- Add a readme.txt/rst/md file with a brief description / instructions to setup the project
- Follow good practices 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 done