

Exo Currency

Pasos para lanzar el proyecto Django

Enlace al repositorio: <https://github.com/SorianoMarmol/exo-currency>.

Se recomienda usar virtualenv.

- Crear virtualenv
 - `sudo apt-get install python3-venv`
 - `python3 -m venv exo_investing`
- una vez creado, localizar el activate (bajo bin) y para activarlo y comenzar a trabajar
 - *Ejemplo:* `cd ~/virtualenvs/exo_investing/`
 - **source** `exo_investing/bin/activate`

Instalar dependencias

- `pip install -r requirements.txt`

Hay un warning que no afecta para poder trabajar, es el siguiente:

```
Building wheels for collected packages: unicode-slugify
Running setup.py bdist_wheel for unicode-slugify ... error
Complete output from command /home/rsoriano/exo_investing/bin/python3 -u -c "import setuptools, tokenize;__file__='/tmp/pip-build-suh28xwu/unicode-slugify/setup.py';exec(compile(getattr(tokenize, 'open', open)(__file__).read().replace('\r\n', '\n'), __file__, 'exec'))" bdist_wheel -d /tmp/tmpcf03ufmnpip-wheel- --python-tag cp35:
usage: -c [global_opts] cmd1 [cmd1_opts] [cmd2 [cmd2_opts] ...]
   or: -c --help [cmd1 cmd2 ...]
   or: -c --help-commands
   or: -c cmd --help
```

```
error: invalid command 'bdist_wheel'
```

```
-----
Failed building wheel for unicode-slugify
Running setup.py clean for unicode-slugify
```

No afecta para trabajar y si se vuelve a ejecutar el `pip install` no vuelve a aparecer

Migrar y crear super usuario

Dentro de exo-currency

- python manage.py migrate
- python manage.py collectstatic
- python manage.py createsuperuser

La migración nº 2 cargará las monedas de ejemplo. Se encuentran en settings (EXAMPLE_CURRENCIES)

Lanzar servidor, acceder al panel de administración, y dar de alta los Provider (uno por cada adapter)

- http://127.0.0.1:8000/admin/exchange_rates/provider/add/
- Ambos deben estar activos
- Se recomienda establecer en el orden primero el de Fixer

Select provider to change

ADD PROVIDER +

Search

Action: ----- Go 0 of 2 selected

<input type="checkbox"/>	SLUG	NAME	ORDER	ADAPTER	ACTIVE
<input type="checkbox"/>	mock	MOCK	8	exchange_rates.backends.mock.MockExchangeProviderBackend	<input checked="" type="checkbox"/>
<input type="checkbox"/>	fixer	FIXER	6	exchange_rates.backends.fixer.FixerExchangeProviderBackend	<input checked="" type="checkbox"/>

2 Providers

Save

Los provider se han podido cargar automáticamente por un problema con los árboles MPTT

Algunos settings a destacar

- FIXER_FORCE_EXCEPTION
 - Por defecto a False, permite forzar una excepción el Fixer para probar el siguiente Provider (poder probar como responde ante fallos, el orden, el mock, etc.)
- SHOW_PROVIDER
 - Devolverá el provider con los datos.
 - Poder ver la fuente de los datos almacenados
- RANDOM_RANGE_INIT / RANDOM_RANGE_END
 - Valores aleatorios para los Rates del mock

Cuando esté todo a punto, se puede acceder a http://127.0.0.1:8000/api/v1/currency_rates/ que debe devolver los datos del día.

Algunos detalles de implementación

- El proyecto denominado “exo_currency” tiene una aplicación “exchange_rates”
- Dentro de dicha aplicación se encuentra la `api_rest`, los backends/adaptadores y todas las funcionalidades.

Modelos:

- Se han usado los modelos de ejemplo para `Currency` y `CurrencyExchangeRate`.
 - En el caso de `CurrencyExchangeRate` se ha añadido el `Provider`
- `CurrencyExchangeRate` incluye una `property` para devolver los datos parseados de la manera adecuada
 - *Se podría parametrizar un mapeo en el settings para devolver los datos al gusto*
- Para `Provider`, he usado una herencia de `Category`, del módulo `django-categories` ya que cumple bastante bien
 - Requisitos:
 - Incluye campo `Orden` (y funciona bajo árboles *MPTT*)
 - Incluye campo “active”, que permitiría activar y desactivar al gusto.
 - Otras ventajas
 - Jerarquías
 - Dispone de sus propios campos y ordenaciones para mostrar las jerarquías
 - Posibilidad de asociar una imagen (thumbnail)
- Un `Provider` debe tener un `adapter` relacionado, de los que existen (`Mock` y `Fixer`). Es obligatorio y único.
 - En un settings se proporcionan los choices, aunque podía haberse puesto a mano
 - En este setting se indica la ruta de la clase del Adaptador, que se cargaría con el `get_connection` definido en el `__init__`

`get_exchange_rate_data`

- Se encuentra definida en el `utils` e incluye los parámetros indicados junto con un índice de `provider`
 - lo he implementado así para realizar llamadas recursivas, pudiendo probar con el siguiente `provider`
 - está todo documentado, si no puede obtener los datos de un `provider`, hace un `log.error`, obtiene el siguiente, y se llama de manera recursiva para probar con ese proveedor

- lanzará una excepción si no se puede obtener de ningún proveedor
- la función es bastante simple por que toda la lógica se encuentra en las clases de los adaptadores
 - En cada Adaptador se implementa “get_exchange_rate_data” que contiene las peculiaridades del conector (conectarse con fixer o generar datos aleatorios)

Adaptadores / backends

- Consta de una base (base.py) “BaseExchangeProviderBackend” y de las implementaciones de los adaptadores (fixer.py + mock.py)
- Uno de los métodos más importantes de la clase base es “get_rates”, que implementa la lógica de obtener los datos almacenados, o bien invocar a “get_exchange_rate_data” para obtenerlos del Provider
 - Para obtener el provider adecuado se tiene en cuenta el **orden**, que se puede cambiar en el panel de administración.

Api Rest:

- Se ha planteado un versionado. Ya que son posibles de varios tipos, se ha propuesto uno mediante URLs y división en subdirectorios

base (__init__)

```

1  from rest_framework import generics
2  from rest_framework.versioning import URLPathVersioning
3
4
5  class ExchangeRatesVersioning(URLPathVersioning):
6      default_version = 1
7      allowed_versions = 1
8      version_param = 'version'
9
10
11  class ResourceAPIRest(generics.GenericAPIView):
12      public_documentation = False
13      name = 'Servicio Web'
14      versioning_class = ExchangeRatesVersioning

```

urls, referenciando a versión

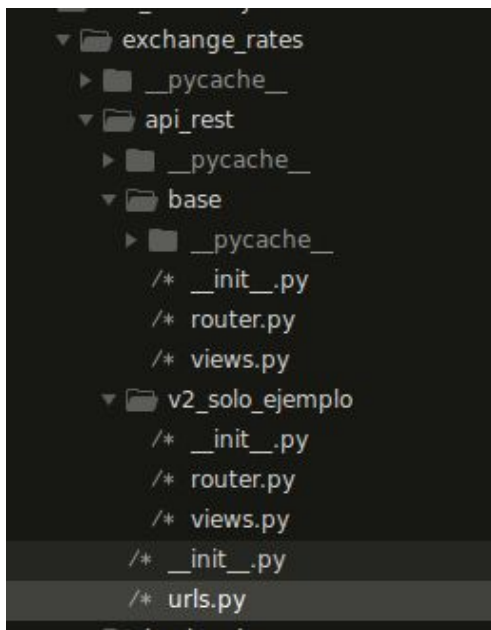
```

from django.urls import path, include

from .base.router import api_urlpatterns as api_v1
# solo ejemplo
# from .api.v2.router import api_urlpatterns as api_v2

urlpatterns = [
    # ^api/,
    path(r'v1/', include(api_v1)),
    # solo ejemplo
    # url(r'^v2/', include(api_v2, namespace="v2")),
]

```



Con la herencia es una buena aproximación

API REST - Exchange Rates

Como he comentado con anterioridad, se ha propuesto un versionado.

- **List of currency rates for a specific time period**

Parameters: date from / date to

Los parámetros (date_from / date_to) serían con formato "%Y-%m-%d", es decir AAAA-MM-DD.

- "?date_from=2019-07-22&date_to=2019-07-23"

En otros casos dará error (se podría programar fácilmente cualquier formato con una línea) pero he preferido dejar este para que cuadre con Fixer, además de invertir tiempo en otras cosas.

```
date = datetime.datetime.strptime(date, "FORMATO")
```

- http://127.0.0.1:8000/api/v1/currency_rates/
 - Las fechas tienen por defecto el día actual, por lo que devuelve resultados
- http://127.0.0.1:8000/api/v1/currency_rates/?date_from=2019-07-24
 - devuelve desde ese día hasta el actual
- http://127.0.0.1:8000/api/v1/currency_rates/?date_from=2019-07-22&date_to=2019-07-23
 - devuelve todos los días del rango

Es muy simple por que toda la lógica de comprobar si existen los datos o invocar al provider está en “get_rates”.

• Calculate amount in a currency exchanged into a different currency

Parameters: origin currency, amount, target currency

- Requiere los parámetros origin_currency, amount, target_currency
- Si falta alguno arroja un error simple.
- Si el usuario usa “,” en la cantidad, se reemplaza por “.” para que cuadre con Decimal
- Si no son valores correctos dará error (se puede controlar mucho mejor, ver si existe la moneda, etc)


Ejemplos

- http://127.0.0.1:8000/api/v1/calculate_amount/?origin_currency=EUR&target_currency=USD&amount=1.1
- http://127.0.0.1:8000/api/v1/calculate_amount/?origin_currency=CHF&target_currency=GBP&amount=1,256

En este caso también es simple, ya que en calculate_amount se invoca a get_rates, que tiene toda la lógica para obtenerlos (de la BD o del provider) de manera transparente.

Por ejemplo, tras la última prueba indicada, se observa que se han obtenido esos datos:

http://127.0.0.1:8000/admin/exchange_rates/currencyexchangerate/

	Swiss franc	Pound sterling	July 25, 2019	0.812317
---	-------------	----------------	---------------	----------

- **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>

- Requiere los parámetros origin_currency, amount, target_currency, date_invested
- Si falta alguno arroja un error simple.
- Si el usuario usa “,” en la cantidad, se reemplaza por “.” para que cuadre con Decimal
- Si no son valores correctos dará error (se puede controlar mucho mejor, ver si existe la moneda, etc)
- El parámetro de fecha date_invested es con formato "%Y-%m-%d", es decir AAAA-MM-DD.
 - date_invested=2019-07-23

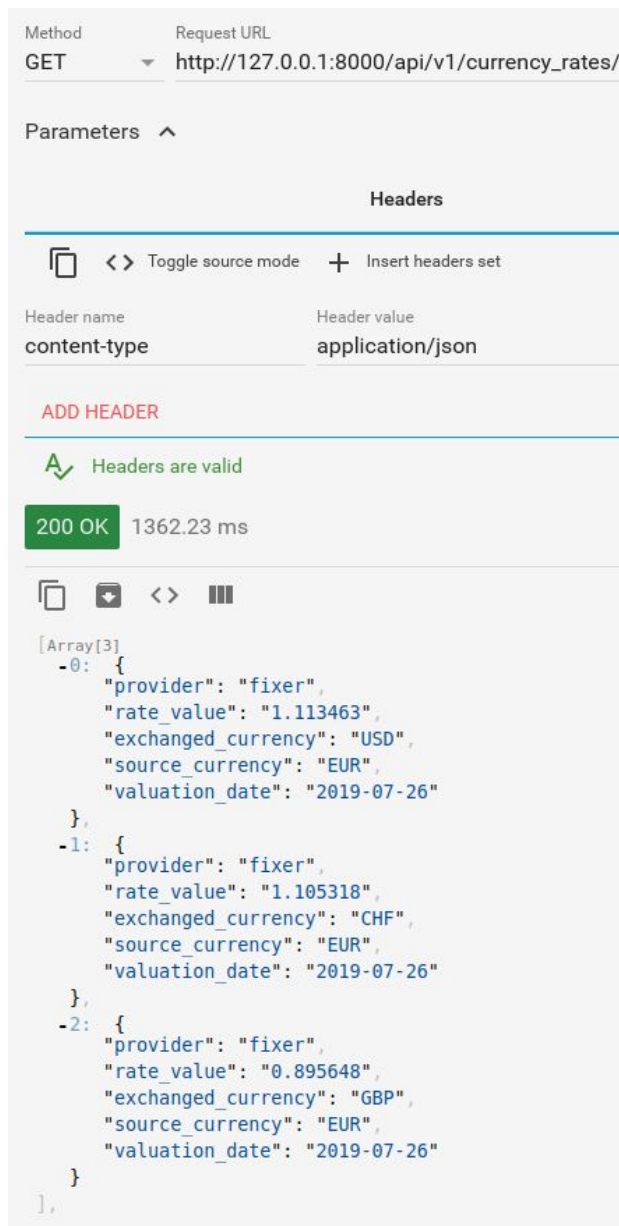
El Cálculo me ha costado bastante, espero esté correcto.

- http://127.0.0.1:8000/api/v1/TWR/?origin_currency=CHF&target_currency=GBP&amount=1000&date_invested=2019-07-22

Solo puedo comentar que obtengo el mismo resultado considerando un único periodo o cada día como un periodo.

En este caso también es simple (la api), ya que en calculate_time_weighted_rate se invoca a get_rates y se realizan los cálculos.

El json es válido e interpretable:



Batch procedure to retrieve exchange rates (optional)

He realizado un comando Django básico bajo la ruta típica de los comandos, en este caso, *exo_currency/exchange_rates/management/commands/load_exchange_rates_from_csv.py*

El hecho de cargar datos de un CSV es bastante sencillo (salvo en algunos casos por temas de codificación de caracteres) gracias al módulo “csv” y a métodos como “DictReader”, que tienen parametrizaciones como el delimitador (generalmente “,”).

Permite leer la cabecera (fieldnames) y luego recorrer las filas, obteniendo la información como si fuera un diccionario.

Si se coloca un diccionario en la parte superior para mapear los “nombres de campo” (modelo a introducir los datos) con las cabeceras, el script se puede adaptar más fácilmente a otras cabeceras.

Se acompaña con un ejemplo simple en el mismo directorio, por lo que podría invocarse así.

```
“exo_investing/exo_currency $ python manage.py load_exchange_rates_from_csv  
exchange_rates.csv --yes”
```

Se puede mejorar pero realiza su función. Si se realiza una exportación periódica de alguna fuente en algún directorio determinado, se podría establecer un cron o programar un

Exchange Rate Evolution backoffice

Este apartado es el que más trabajo tiene ya que no es solo el desarrollo, sino encontrar e integrar una buena librería de gráficas, obtener los datos...

Investigué varias librerías (entre otras)

- <https://plot.ly/python/figure-factory-subplots/>
- <https://www.fusioncharts.com/>
- <https://djangopackages.org/packages/p/django-chartit/>
 - <http://django-chartit.mrsenko.com/>
- <https://github.com/peopledoc/django-chartjs>

Finalmente opté por fusioncharts, ya que hay un tipo de gráfica que permite cosas, a mi entender, necesarias para este apartado:

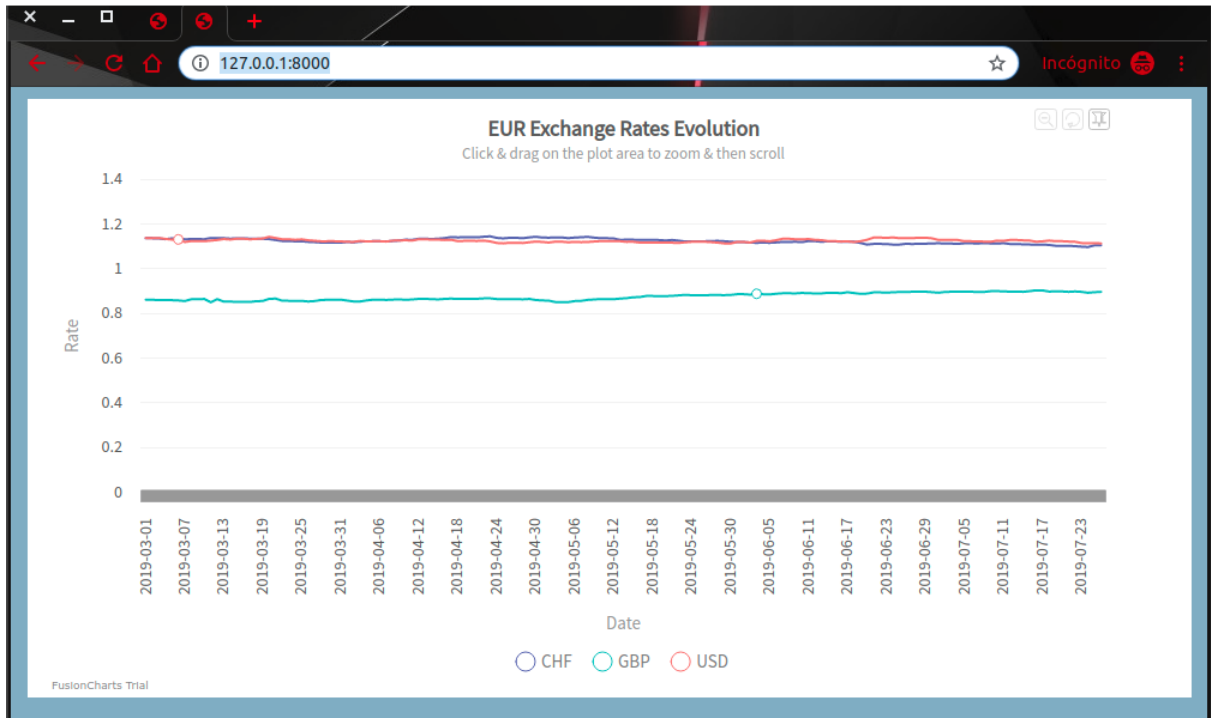
- Eliminar datos: puedes ver sólo la moneda (o monedas) que te interesen
- Zoom: se selecciona un rango y hace zoom, pudiendo desplazar
- Comparación: se selecciona un rango y se mantiene, pudiendo desplazar viendo los puntos marcados.

Se accede en la raíz: <http://127.0.0.1:8000/>

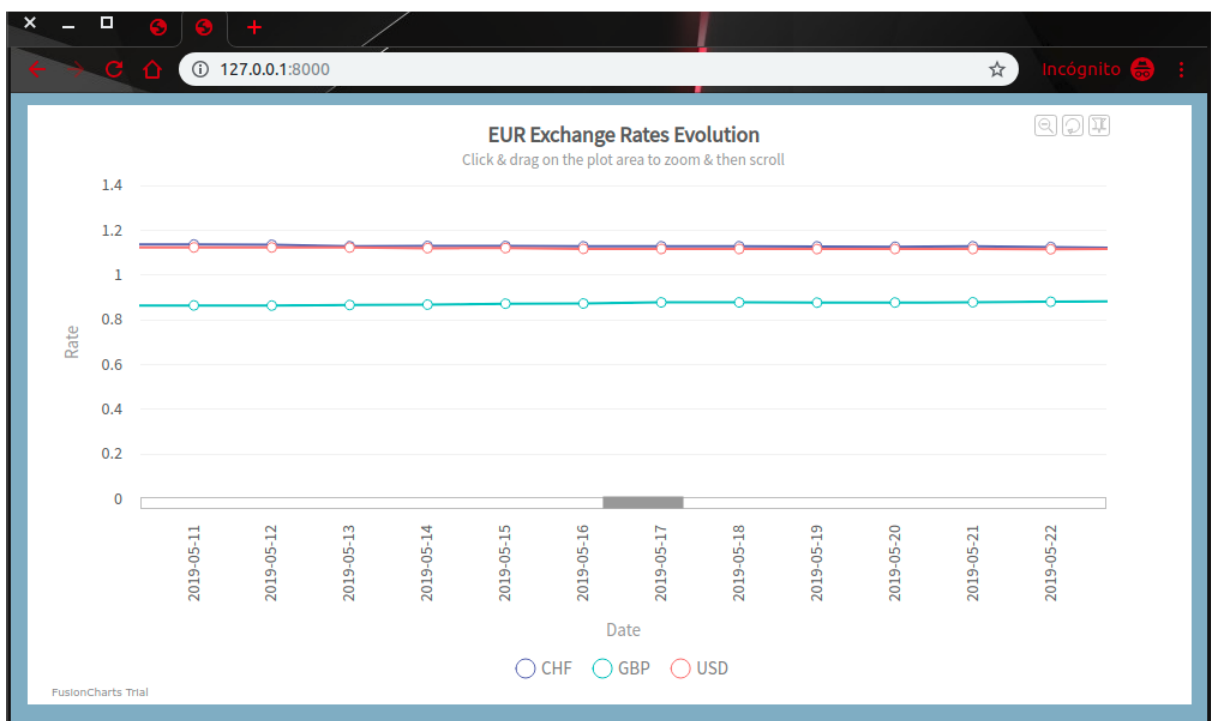
Este sería el resultado:

Con moneda base EUR

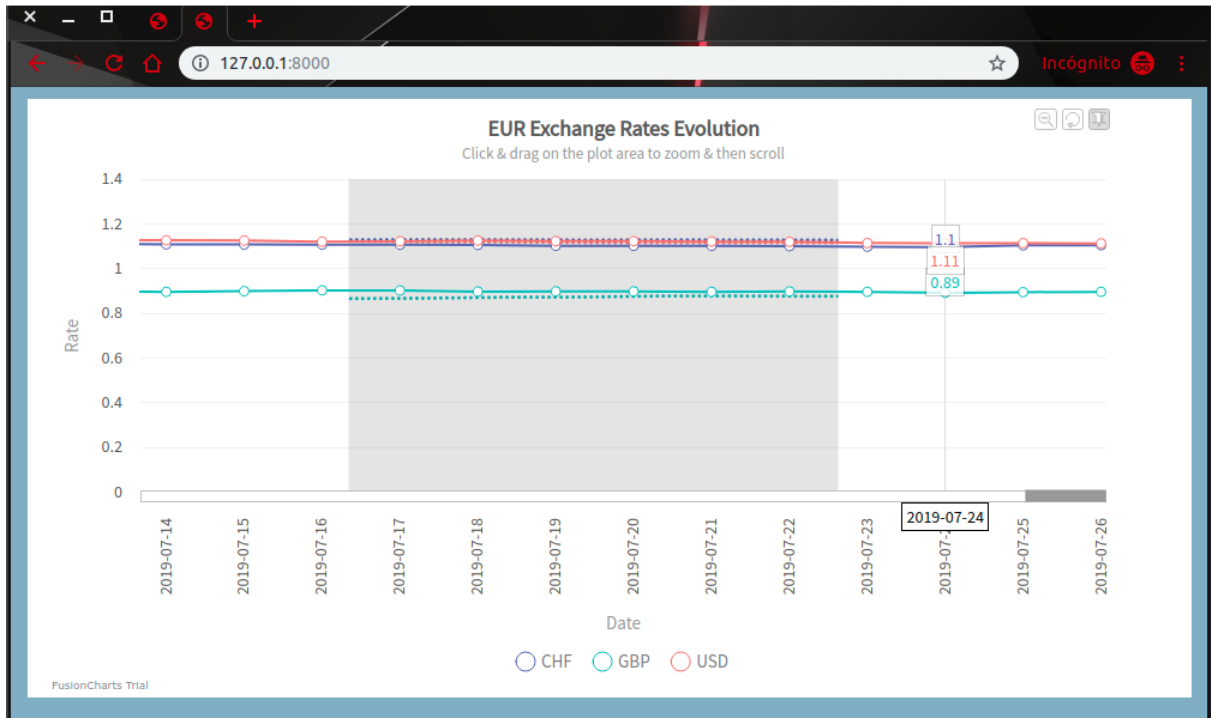
Visión general:



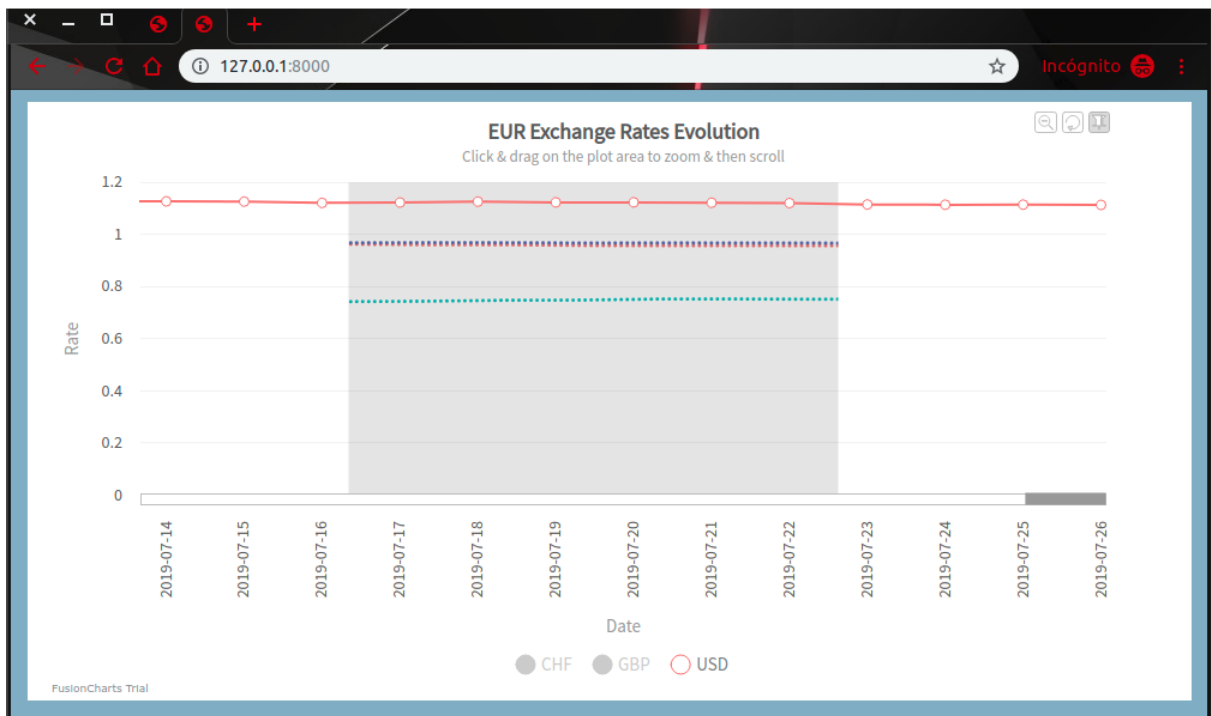
Haciendo zoom:



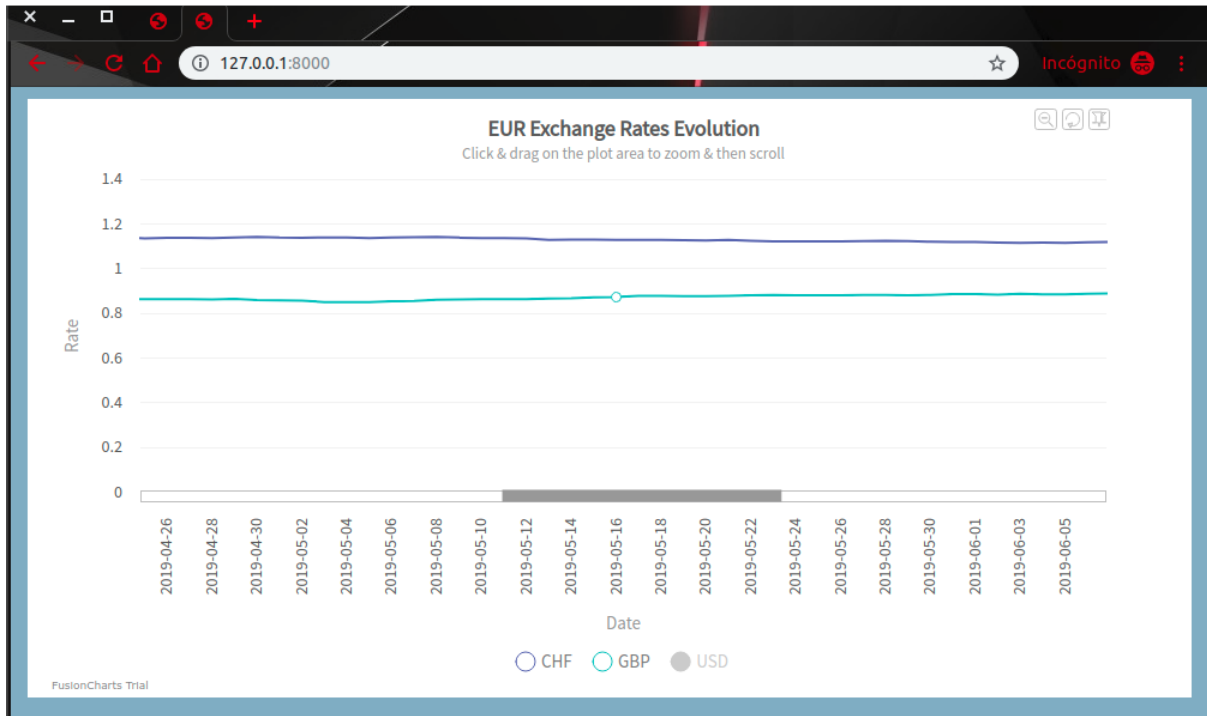
Comparando:



Ver solo una moneda + comparación:



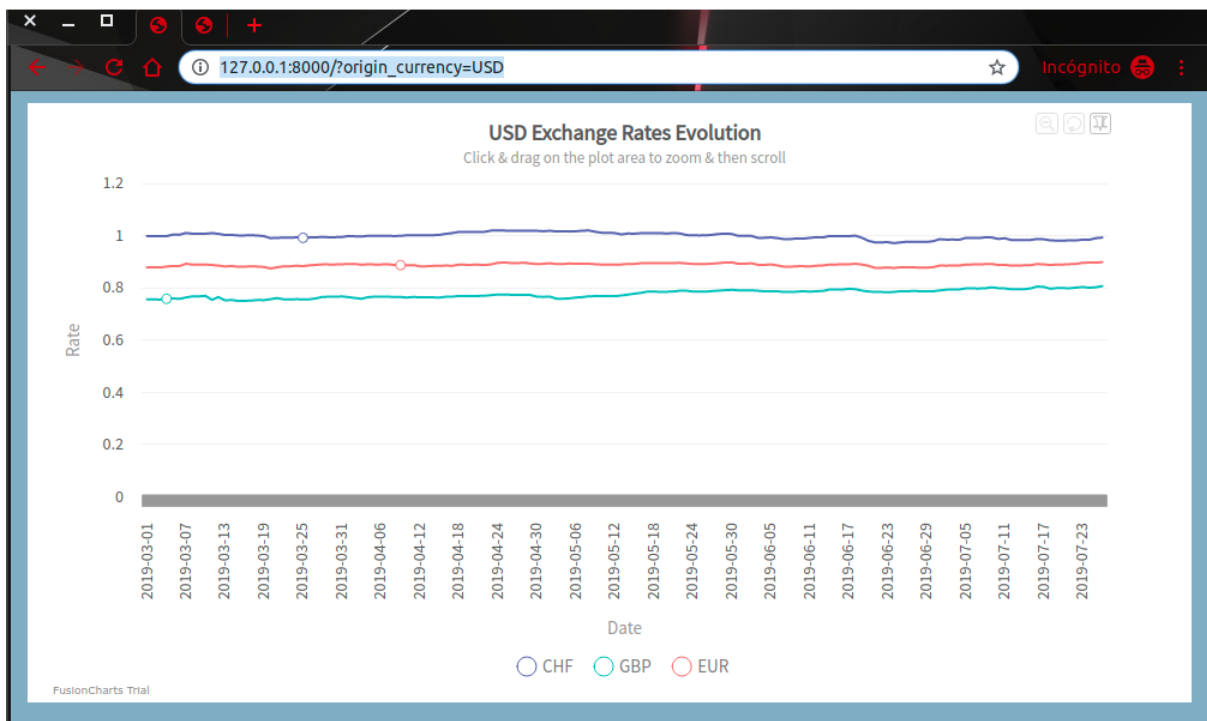
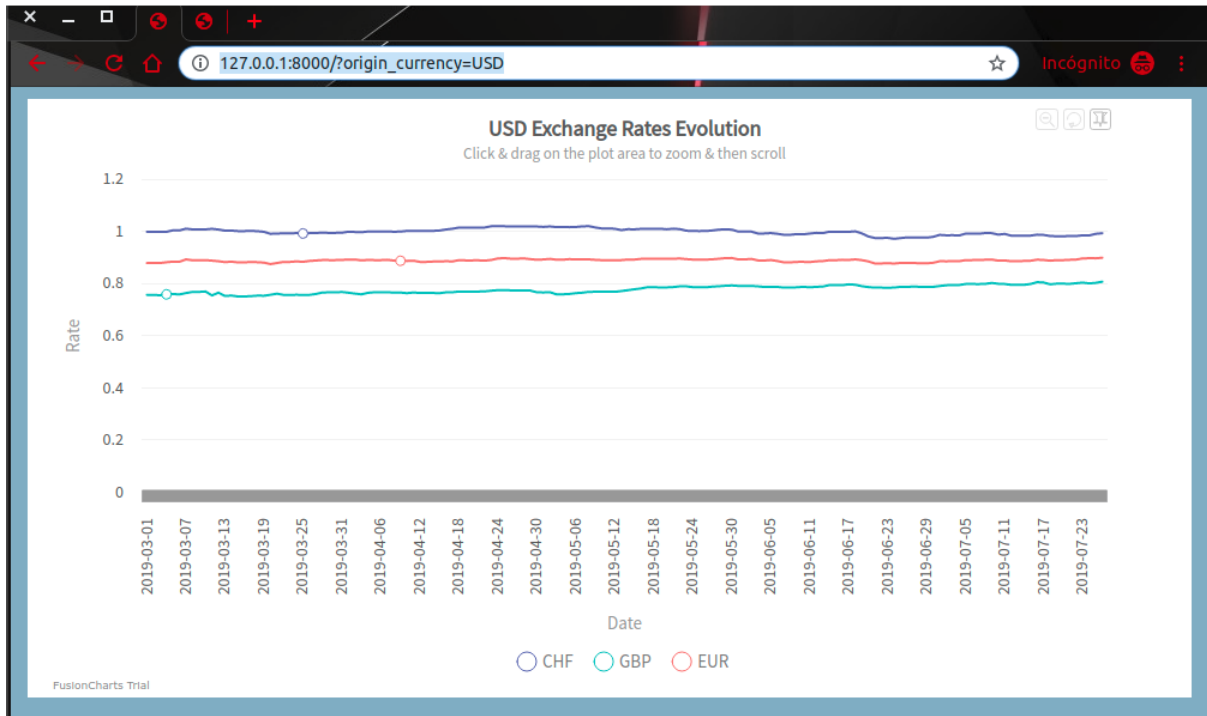
Las otras dos monedas:



Con moneda base USD

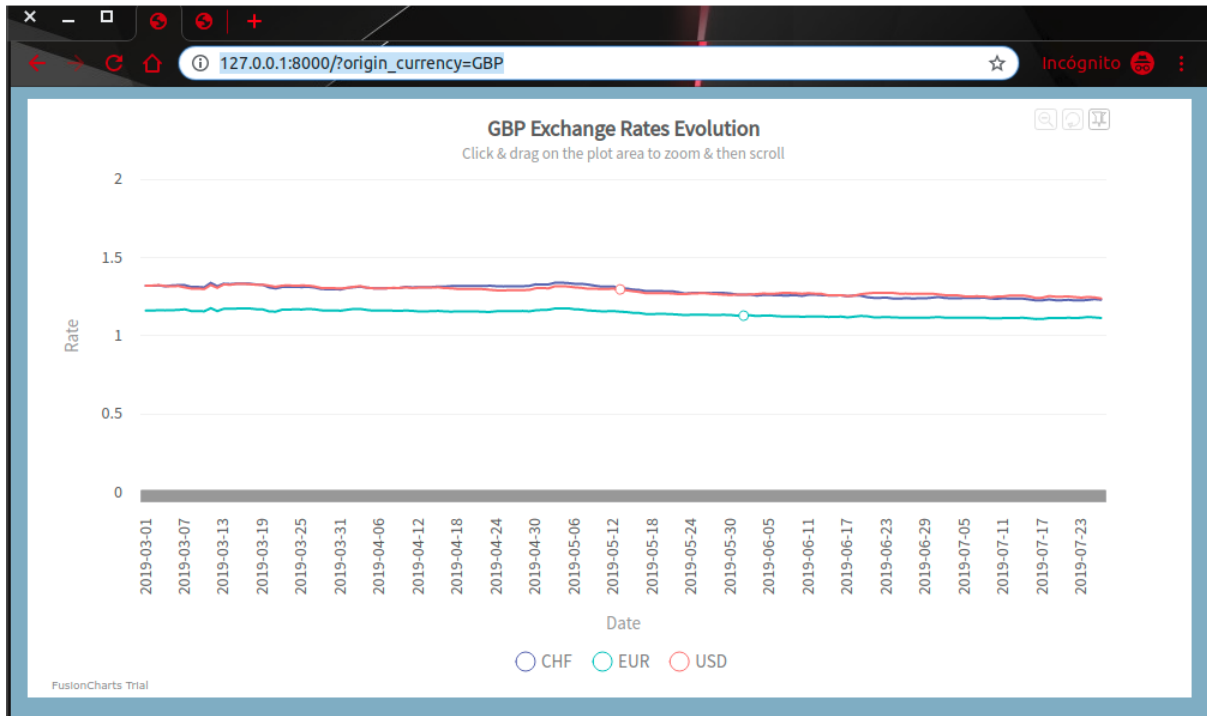
- Para cambiar la moneda base, se usa un parámetro GET
 - http://127.0.0.1:8000/?origin_currency=USD

El resultado es el mismo pero con la fuente USD. No voy a poner tantas capturas:



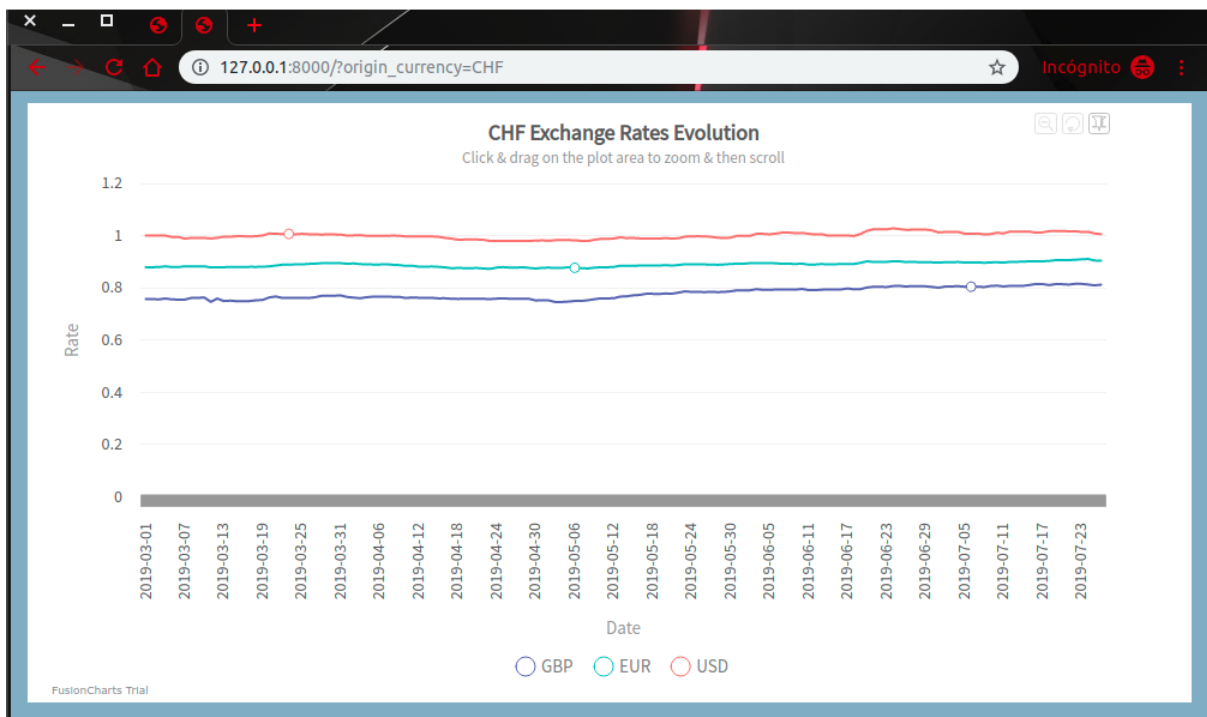
MONEDA BASE GBP:

http://127.0.0.1:8000/?origin_currency=GBP



MONEDA BASE CHF

http://127.0.0.1:8000/?origin_currency=CHF



Notas sobre este apartado:

- He incluido el `get_rates` con unas fechas. Lo ideal sería haber cargado los datos antes por lo que **la primera vez tarde mucho es normal si no están los datos cargados previamente.**
- No he podido dedicarle tanto tiempo como me gustaría a este apartado debido al tiempo total invertido en la prueba, me ha resultado imposible y han sido muchas horas.
 - Posibles mejoras
 - El parámetro `get` es funcional pero con un formulario quedaría mejor
 - Cargar los datos sin recargar vía `ajax` sería un buen punto
 - Poder seleccionar las fechas también
 - Como ya he comentado, separar la carga de datos de `fixer` del acceso a este apartado, haciendo unas consultas mucho más óptimas (como `select_related`, etc) de cara a obtener la información.
 - Seguramente muchas más mejoras