Documentación LowPriCEX

Agustín Borrego Díaz

David de los Santos Boix

Índice de contenidos

1. Descripción del proyecto 2
1.1 Introducción y contexto 2
1.2 Información técnica 2
1.3 Información funcional 3
2. Instalación 4
2.1 Instalación de Python 4
2.2 Instalación de paquetes necesarios 5
2.3 Configuración de la base de datos 5
2.4 Configuración del entorno virtual 6
2.5 Configuración de Django 7
2.6 Creando la estructura de la base de datos y ejecutando el proyecto 7
2.7 Poblando la base de datos 8
2.7.1 Usando el *dump* de la base de datos 8
2.7.2 Usando nuestra utilidad de carga de datos 8
2.7.3 Usando el *crawler* de datos 9
2.8 Ejecutando el *crawler* 9
2.8.1 Automatizando la ejecución del *crawler* 10

1. Descripción del proyecto

1.1 Introducción y contexto

El proyecto LowPriCEX surge con la idea de proporcionar a los compradores y vendedores habituales de productos tecnológicos de segunda mano, especialmente videojuegos, información histórica sobre el estado de gran cantidad de juegos de varias plataformas en las tiendas CEX distribuidas a lo largo de España.

Más concretamente, se ofrece para cada juego el precio actual de compra, venta e intercambio en las tiendas CEX, actualizado a diario. Además, permite consultar la evolución de los mismos a lo largo del tiempo, para que tengan aún más información a la hora de tomar una decisión acerca de comprar o vender videojuegos en esta cadena de tiendas.

Como valor añadido, se ha ampliado enormemente la información que se muestra para cada juego respecto a la que ofrece la página web de CEX gracias a la integración con IGDB (*Internet Games DataBase*). Gracias a este servicio, podemos ofrecer información adicional sobre cada juego como por ejemplo su fecha de salida, empresas desarrolladoras y comercializadoras, imágenes y vídeos, carátulas, etcétera.

Además, se ha desarrollado un sistema de recomendación basado en contenido que permite a los usuarios descubrir nuevos juegos que podrían ser de su interés, teniendo en cuenta información como las palabras clave asociadas a un juego, su género, desarrolladores y precios similares.

LowPriCEX se encuentra desplegado en https://lowpricex.info

1.2 Información técnica

El proyecto se ha desarrollado con Python 3.5.2, usando Django como framework de desarrollo web. El motor de base de datos usado es PostgreSQL, por lo que se ha debido configurar tanto Django como PostgreSQL para su integración mutua (la información detallada sobre ello se encuentra en el apartado de instalación del sistema).

Además, se ha usado Whoosh como herramienta de indexado para las descripciones de los juegos, integrándolo con Django-Haystack.

El *scrapping* de la web de CEX se ha hecho con Scrapy, ya que proporciona una mayor comodidad a la hora de implementar la tarea de procesado de juegos: gracias a las *pipelines* de Scrapy, cada juego recuperado de la página web de CEX se procesa individual y automáticamente.

La información detallada de los juegos se obtiene a partir de la API de IGDB, para la cual hemos obtenido una clave de acceso. El archivo que contiene nuestras claves de APIs no se encuentra en nuestro repositorio de código ya que planeamos hacerlo público en el futuro cercano, sin embargo, se ha incluido en el código entregado para que funcione correctamente.

La integración con IGDB planteó un problema, dado que los nombres de los juegos no coinciden exactamente en general con los de CEX. Aún más, ya que el *scrapping* se realiza en la versión española de CEX, algunos títulos están en español en CEX y en inglés en IGDB. Para solucionar este problema, se compara el índice de similaridad entre ambos y se considera que son el mismo juego si éste supera el 50%.

Si no se encuentra ningún juego con estas características, se intenta de nuevo la búsqueda traduciendo el título del juego a inglés gracias a la API de traducción de Yandex y se efectúa de nuevo la comparación.

El proyecto se ha desarrollado en Ubuntu 16.04 y se ha desplegado en el mismo sistema operativo en un VPS propiedad de David de los Santos, usando nginx como servidor web.

1.3 Información funcional

La página principal de la aplicación web muestra tres paneles informativos, cada uno de ellos indica los cinco juegos que más han bajado de precio (en el caso de venta) o que más han subido de precio (en el caso de compra o intercambio) respectivamente, informando además del precio anterior y el actual. La información de estos paneles se actualiza diariamente de forma automática.

Existe una vista de búsqueda simple, que permite filtrar los juegos según nombre y plataforma. Para cada juego se muestra el título, la plataforma, los precios actuales y un enlace a la vista detallada del juego. Los resultados se encuentran paginados con un máximo de 25 resultados por página. También existe una vista de búsqueda avanzada, que permite filtrar por parámetros más específicos como el desarrollador, la descripción del juego, etcétera.

Además, para cada juego existe una vista detallada en la que se muestran todos los atributos del juego, una galería de imágenes y vídeos del juego, una gráfica que muestra la evolución de los diferentes precios del juego a lo largo del tiempo y 5 juegos recomendados si el usuario está interesado en ese juego, usando un sistema de recomendación basado en contenido.

2. Instalación

En esta sección se detallarán los pasos necesarios para instalar el proyecto en un entorno Linux. El despliegue se realizará empleando el servidor de desarrollo proporcionado por Django, dado que los procesos necesarios para desplegar el proyecto en un servidor de producción quedan fuera del ámbito de esta documentación.

La instalación se realizará bajo un usuario con permisos sudo. Se propone que la carpeta raíz del proyecto se encuentre en el directorio *home* del usuario.

2.1 Instalación de Python

La versión de Python que se ha empleado es la 3.5.2. Se puede comprobar la versión instalada en nuestro sistema abriendo una consola de comandos y ejecutando el comando `python3 --version`, lo que mostrará la versión de Python de la que disponemos. Si ya es la versión 3.5.2, se puede pasar directamente al siguiente apartado.

Si no, debemos descargar e instalar Python 3.5.2. Podemos hacerlo con el comando

```
wget https://www.python.org/ftp/python/3.5.2/Python-3.5.2.tgz
```

Descomprimimos el archivo y accedemos a la carpeta creada:

```
tar -xzf Python-3.5.2.tgz
cd Python-3.5.2
```

Configuramos la instalación del código fuente

```
./configure
```

Y por último compilamos e instalamos la versión de Python

```
make && sudo make install
```

Este último paso podría demorarse dependiendo del sistema en cuestión. Una vez haya finalizado, podemos ejecutar de nuevo

```
python3 --version
```

Y, si la instalación ha sido correcta, veremos que la versión de Python es ahora 3.5.2.

Opcionalmente, podemos limpiar el directorio de los archivos descargados y generados:

```
rm -rf ../Python-3.5.2*
```

2.2 Instalación de paquetes necesarios

Una vez tenemos instalada la versión correcta de Python, debemos asegurarnos también de que se encuentran en el sistema los paquetes necesarios para la ejecución del proyecto. Actualizamos en primer lugar el gestor de paquetes:

```
sudo apt-get update
```

E instalamos los paquetes requeridos:

```
sudo apt-get install build-essential libssl-dev libffi-dev
python-dev python3-dev python3-pip
```

Instalamos también el motor de base de datos PostgreSQL, que se configurará en el siguiente apartado.

```
sudo apt-get install libpq-dev postgresql postgresql-
contrib
```

2.3 Configuración de la base de datos

A continuación, procederemos a configurar la base de datos del proyecto en PostgreSQL, creando un usuario y dándole permisos en la base de datos deseada.

En primer lugar, debemos cambiar al usuario del sistema *postgres* recién creado:

```
sudo su - postgres
```

Podremos observar que el usuario de nuestra consola ha cambiado del que estuviéramos usando anteriormente al usuario *postgres*, que es el administrador de la base de datos. Accedemos a la interfaz de línea de comandos de la base de datos ejecutando el siguiente comando:

```
psql
```

Creamos la base de datos que usará el proyecto. El nombre que se usará en esta configuración es *lowpricex_db*, pero se puede usar cualquier otro siempre que se cambie también en la configuración del proyecto en Django:

```
CREATE DATABASE lowpricex_db;
```

Ahora creamos el usuario que tendrá acceso a la base de datos. De igual forma, se puede usar cualquier otro nombre siempre que se configure adecuadamente el proyecto:

```
CREATE USER lowpricex_user WITH PASSWORD
'lowpricex_password';
```

Obviamente, la contraseña elegida es muy débil y podría (debería) cambiarse, pero sirve de ilustración para esta documentación. Especial atención a mantener las comillas simples que rodean la contraseña.

Sólo resta asignar al usuario los permisos para la base de datos:

```
GRANT ALL PRIVILEGES ON DATABASE lowpricex_db TO
lowpricex_user;
```

Podemos salir de la interfaz de PostgreSQL y volver a nuestro usuario original pulsando dos veces CTRL + D.

2.4 Configuración del entorno virtual

En Python, un entorno virtual es una carpeta que contiene todo lo necesario para la ejecución de un determinado proyecto, con la ventaja de que aísla las dependencias del proyecto de los paquetes del sistema o de otros proyectos. En el desarrollo, hemos usado esta funcionalidad para gestionar las dependencias de LowPriCEX.

En primer lugar, se debe actualizar el gestor de paquetes de Python, *pip*, para asegurarnos de que contamos con su última versión:

```
sudo pip3 install --upgrade pip
```

Instalamos ahora la utilidad de entornos virtuales

```
sudo pip3 install virtualenv
```

Creamos el entorno virtual en el que se ejecutará el proyecto. Se propone que la carpeta tenga como nombre *lowpricex_env* y se encuentre dentro del directorio raíz del proyecto, pero de nuevo, se puede cambiar esta configuración siempre que se mantenga la consistencia a lo largo de la instalación.

```
virtualenv -p python3 ~/LowPriCEX/lowpricex_env
```

Antes de activar el entorno virtual, se debe realizar un pequeño cambio en el mismo: con nuestro editor de texto de preferencia, editamos el archivo `~/LowPriCEX/lowpricex_env/bin/activate` y añadimos estas tres líneas al final del mismo:

```
export PYTHONPATH=$PYTHONPATH:~/LowPriCEX/lowpricex/

export
PYTHONPATH=$PYTHONPATH:~/LowPriCEX/lowpricex/lowpricex_scrapper/

export DJANGO_SETTINGS_MODULE=lowpricex.settings
```

Esto se hace para que las dependencias entre elementos internos del proyecto funcionen correctamente dentro del entorno virtual.

De nuevo, existe la asunción de que la carpeta raíz del proyecto se encuentra en el directorio *home*, por lo que las líneas anteriores deben modificarse convenientemente si esto no es así.

Ahora sí, procedemos a activar el entorno virtual:

```
source ~/LowPriCEX/lowpricex_env/bin/activate
```

Podremos observar que nuestro *prompt* ha cambiado para indicar que se encuentra activo el entorno virtual: aparece entre paréntesis su nombre a la izquierda.

Con el entorno virtual activo, podemos instalar las dependencias del proyecto:

```
pip3 install Django==1.10.4 django-haystack==2.5.1
psycopg2==2.6.2 requests==2.9.1 Scrapy==1.3.0 Whoosh==2.7.4
```

Hemos finalizado de configurar el entorno virtual del proyecto. Éste debe permanecer activado para los pasos posteriores.

2.5 Configuración de Django

Django en sí mismo no requiere una configuración excesiva para nuestro proyecto, simplemente adaptar la configuración de base de datos a la que se ha creado.

La configuración de la base de datos y la clave secreta de Django se han extraído del habitual archivo *settings.py*, ya que se planea publicar el proyecto desarrollado en GitHub. En su lugar, se han trasladado a un archivo *secret_settings.py* que está excluido del repositorio en GitHub pero que se incluye para mayor comodidad.

El archivo está ya adaptado a la configuración realizada, pero si se ha realizado algún cambio en el nombre de la base de datos, el usuario o la contraseña, debe actualizarse para que Django pueda conectarse correctamente con PostgreSQL.

2.6 Creando la estructura de la base de datos y ejecutando el proyecto

Nota: en el código entregado se incluye un archivo de *dump* de la base de datos ya poblada. Si se decide usarlo, este paso no es necesario. Ver 2.7.1.

Una vez hemos configurado Django para que se conecte con PostgreSQL, debemos crear la estructura de tablas de la base de datos. Esta tarea se puede realizar fácilmente gracias a la utilidad de migraciones de Django, basta con ejecutar estos dos comandos:

```
python3 ~/LowPriCEX/lowpricex/manage.py makemigrations lowpricex_app
```

```
python3 ~/LowPriCEX/lowpricex/manage.py migrate
```

Opcionalmente, podemos asegurarnos de que la base de datos está inicialmente limpia usando:

```
python3 ~/LowPriCEX/lowpricex/manage.py flush
```

El proyecto ya está listo para ser ejecutado, aunque la base de datos estará vacía por lo que no habrá datos para mostrar. En cualquier caso, se puede arrancar el servidor usando:

```
python3 ~/LowPriCEX/lowpricex/manage.py runserver
```

2.7 Poblando la base de datos

Existen tres formas de poblar inicialmente la base de datos de nuestro proyecto. A continuación se describe cada una de ellas.

2.7.1 Usando el *dump* de la base de datos

Dentro de la carpeta raíz de nuestro proyecto, se provee un archivo *database_dump.pgsql* que contiene la estructura de la base de datos, toda la información sobre juegos, y compañías y cierta cantidad de información histórica de precios de juegos. Esta es, con diferencia, la forma de poblado más rápida.

Para volcar los datos en la base de datos, basta con ejecutar el siguiente comando:

```
psql -U lowpricex_user -h localhost lowpricex_db <
~/LowPriCEX/database_dump.pgsql
```

Nótese que se ha usado el nombre del usuario en PostgreSQL y de la base de datos definidos anteriormente, por lo que si se han cambiado, el comando también debería cambiarse para reflejar esos cambios. Se nos pedirá una contraseña, por lo que deberemos introducir la del usuario que hayamos creado. Tras mostrar alguna información por pantalla, la base de datos estará poblada y lista para su uso.

2.7.2 Usando nuestra utilidad de carga de datos

Se proveen dentro del proyecto varios archivos CSV que contienen información sobre varias tablas de la base de datos, así como 10 archivos históricos correspondientes a la ejecución del *crawler* en 10 días consecutivos que pueden usarse para poblar la base de datos.

Por defecto, se usará el archivo de juegos correspondiente al día 22 de diciembre de 2016, pero este comportamiento puede modificarse cambiando el archivo que se desea cargar al final de

```
~/LowPriCEX/lowpricex/populateDatabase.py
```

Una vez se ha seleccionado el archivo histórico a cargar, se ejecuta el procedimiento de carga de datos de esta manera:

```
cd ~/LowPriCEX/lowpricex/

python3 manage.py loadData
```

Se procederá entonces a cargar los archivos CSV de información general y el archivo de información histórica deseado.

Este procedimiento puede demorarse varias horas, teniendo en cuenta que cada archivo histórico contiene aproximadamente 12.000 juegos, y cada uno de ellos requiere como mínimo una consulta de datos a IGDB, por lo que el tiempo medio de procesamiento de cada juego es de aproximadamente un segundo.

2.7.3 Usando el *crawler* de datos

Se puede usar el *crawler* para realizar una carga de datos inicial en la base de datos (internamente, el procedimiento anterior utiliza la *pipeline* del *crawler* para procesar los juegos), pero este método es aún más lento que el anterior dado que no cuenta con los datos de los archivos CSV (por ejemplo, compañías o *keywords*) por lo que añade aún más peticiones a IGDB.

Ver la sección 2.9 para obtener información sobre cómo ejecutar el *crawler*.

2.8 Cargando los detalles de los juegos

Una vez la base de datos ha sido poblada por cualquiera de los métodos anteriores, se debe cargar la descripción de los juegos del sistema y construir su índice con Django-Haystack. El proceso es muy sencillo: en primer lugar, se cargan los datos en la base de datos usando los siguientes comandos:

```
cd ~/LowPriCEX/lowpricex/

python3 manage.py loadDetails
```

Y una vez termina el proceso, se genera el índice de Haystack (que usa internamente Whoosh) con el siguiente comando:

```
python3 manage.py rebuild_index
```

2.9 Ejecutando el *crawler*

Nuestro *crawler* navega por las diferentes categorías de los juegos que ofrece CEX y obtiene la información básica de cada uno junto con los diferentes precios de venta, compra e intercambio. El procesamiento que realiza de cada juego es el siguiente:

Si el juego ya existe en la base de datos, comprueba si los precios han cambiado. Si es así, añade un nuevo histórico de precios para ese juego y lo actualiza. Si no, no realiza ninguna acción sobre el juego.

Si el juego no existe en la base de datos, realiza una petición a IGDB para obtener la información detallada del juego (imágenes, vídeos, desarrollador…) y lo almacena en la base de datos con una información histórica inicial.

Hay que tener en cuenta que la búsqueda en IGDB se realiza por el nombre del juego, y en la inmensa mayoría de los casos el nombre del juego en IGDB no es el mismo que en CEX, por lo que se sigue una aproximación probabilística para determinar si son o no el mismo juego.

Para ejecutar el *crawler*, se deben ejecutar los siguientes comandos:

```
cd ~/LowPriCEX/lowpricex/lowpricex_scrapper
scrapy crawl CexSpider
```

Empezará a aparecer por pantalla la información de cada juego procesado. La ejecución podrá demorarse más o menos dependiendo de cuántos juegos haya en ese momento en la base de datos.

2.9.1 Automatizando la ejecución del *crawler*

Lógicamente, es interesante que el proceso se ejecute cada día para registrar los cambios en los precios de los juegos y poder mostrar esa información al usuario. La automatización de la ejecución del *crawler* se puede conseguir a través de la utilidad *crontab* y un pequeño *script*.

Con nuestro editor de texto de preferencia, creamos el archivo *script_crawler.sh* en la carpeta *home* del usuario con el siguiente contenido:

```
#!/bin/bash
cd ~/LowPriCEX/lowpricex/lowpricex_scrapper
source ~/LowPriCEX/lowpricex_env/bin/activate
PATH=$PATH:/usr/local/bin
export PATH
scrapy crawl CexSpider > /dev/null
deactivate
```

Le otorgamos permisos de ejecución para el usuario:

```
chmod 744 ~/script_crawler.sh
```

Y añadimos una entrada en el cron del usuario:

```
crontab -e
```

Añadir la siguiente línea:

```
0 6 * * * ~/script_crawler.sh
```

De esta manera, nuestro *crawler* se ejecutará todos los días a las 6:00 AM.

Esta configuración ejecuta el *crawler* desechando su salida, pero es sencillo modificarla para guardar registros de cada ejecución diaria. Creamos la carpeta en la que se guardarán los registros, por ejemplo, *LowPriCEX-logs:*

```
mkdir ~/LowPriCEX-logs
```

Y modificamos la penúltima línea del script que hemos creado sustituyéndola por esta otra:

```
scrapy crawl CexSpider --logfile ~/LowPriCEX-logs/$(date
+%F).log --loglevel INFO
```

Esto creará archivos diarios de log del formato AAAA-MM-DD.log en la carpeta creada. Se puede modificar también el nivel mínimo de severidad de la información a mostrar (los posibles valores, de menor a mayor, son DEBUG, INFO, WARNING, ERROR y CRITICAL).