TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Prueba de concepto de un sistema automatizado de recolección y análisis de datos de compra y venta de viviendas en Amazon web Service

TITULACIÓN: Grado en Ingeniería de Sistemas de Telecomunicaciones

AUTOR: Roger Ortega Casals

DIRECTOR: Maria Dolores Royo Valles

FECHA: 05/09/2022

Título: Prueba de concepto de un sistema automatizado de recolección y análisis de datos de compra y venta de viviendas en Amazon web Service

Autor: Roger Ortega Casals

Director: Maria Dolores Royo Valles

Fecha: 05/09/2022

Resumen

Este trabajo se ha realizado por mi interés por unas técnicas utilizadas para recopilar información de páginas web. Estas técnicas tienen un nombre y se llama web scraping. Web scraping es utilizado para rastrear información en páginas web. Esta información luego puede ser utilizada para extraer conclusiones de los datos que de otra forma no se podrían extraer. Por ejemplo, se realiza web scraping para rastrear piezas de coche que ya no están en venta en portales de compra venta de segunda mano. Al haber poca oferta de estas piezas enseguida son vendidas. Con web scraping se puede recibir una notificación cuando una de estas piezas es puesta a la venta en uno de los portales.

El trabajo realizado es una prueba de concepto de web scraping con el framework Scrapy. Scrapy es un framework escrito en python utilizado para realizar bots que recopilan la información de sitios webs. Durante el trabajo se realiza un bot con Scrapy. Antes de realizar el proyecto desconocía que el trafico de bots en internet fuera tan elevado, alrededor del 66%. Es por este motivo que muchos sitios web no permiten que estos bots naveguen por sus páginas filtrando su tráfico. En el proyecto se explica que modificaciones se han realizado en el bot para que pueda sortear estas defensas de los sitios web. Este proyecto podría servir para ayudar a las paginas web a mejorar sus defensas contra el tráfico de bots que solo quieren recopilar información de sus páginas.

En el proyecto también se ha desplegado la aplicación en el proveedor de servicios web amazon web services. Amazon web service es uno de los mayores proveedores de servicios web del mundo. Hay muchas ofertas de trabajo que requieren conocimientos de este proveedor y me parecía interesante conocer cómo se podían desplegar aplicaciones en él.

Title: Proof of concept of an automated system for collecting and analyzing data of home sales on Amazon web service.

Author: Roger Ortega Casals

Director: Maria Dolores Royo Valles

Date: 05/09/2022

Abstract

This work has been done because of my interest in some techniques used to collect information from web pages. These techniques have a name and it is called web scraping. Web scraping is used to crawl information on web pages. This information can then be used to draw conclusions from the data that otherwise could not be drawn. For example, web scraping is used to track down car parts that are no longer for sale on used car parts portals. As these parts are in short supply, they are quickly sold. With web scraping it is possible to receive a notification when one of these parts is offered for sale on one of the portals.

The work done is a proof of concept of web scraping with the Scrapy framework. Scrapy is a framework written in Python used to create bots that collect information from websites. During the work a bot is made with Scrapy. Before the project I didn't know that the traffic of bots on the internet was so high, about 66%. It is for this reason that many websites do not allow these bots to browse their pages by filtering their traffic. The project explains what modifications have been made to the bot so that it can bypass these website defenses. This project could be used to help websites improve their defenses against traffic from bots that only want to collect information from their pages.

The project has also deployed the application on the web service provider amazon web services. Amazon web service is one of the largest web service providers in the world. There are many job offers that require knowledge of this provider and it seemed interesting to me to know how applications could be deployed on it.

ÍNDICE

INTRODUCCIÓN

Este proyecto trata de la realización de una prueba de concepto. Esta prueba de concepto trata sobre la extracción de información de páginas web de forma automatizada mediante software. El procesamiento de un gran volumen de datos para obtener información valiosa de ellos es un campo muy solicitado por la gran abundancia de datos presentes actualmente. En la última década las empresas disponen de una gran cantidad de información de diferentes fuentes como transacciones comerciales, vídeos, dispositivos inteligentes, equipo industrial, medios sociales y más. Este gran volumen de información no puede ser analizado por medios humanos, seria inabarcable. La información se analiza mediante un conjunto de métodos, llamados Big Data, que analizan y extraen sistemáticamente información que es demasiado grande o compleja para ser tratada por el procesamiento de datos tradicional. En este proyecto se va a tratar el paso previo al análisis de la información. Se extraerá información de las páginas web, esta actividad es llamada popularmente web scraping.

Web scraping son un conjunto de técnicas y métodos utilizados para extraer información de la web en formato legible para humanos, mediante software. Estas técnicas son utilizadas por empresas para recopilar información de páginas webs, como podría ser amazon, y con información valiosa para sus intereses. Luego esta información es filtrada y analizada en post procesos para extraer conclusiones. En este proyecto se utilizarán los métodos de web scraping para extraer información de los pisos en venta en portales donde se venden. Concretamente este proyecto se ha centrado en la recopilación de datos del portal www.idealista.com.

Para realizar web scraping se utilizará un framework diseñado para este propósito llamado Scrapy. Scrapy es un framework de scraping y crawling de código abierto, escrito en Python. Actualmente está mantenido por Scrapinghub Ltd., una empresa que ofrece productos y servicios de web scraping.

En este proyecto también se explicará cómo desplegar todo el software desarrollado en la infraestructura de amazon web service (AWS). De esta forma se mostrará cómo se desplegará en un entorno de producción de cualquier empresa. AWS es una de las ofertas internacionales más importantes de computación en la nube.

Este proyecto es de carácter de investigación donde se busca la mejor forma de obtener los datos mediante web scraping, en ningún caso los métodos aquí explicados serán utilizados de forma empresarial.

CAPTÍULO 1. SCRAPY

1.1. Introducción

Scrapy es el framework utilizado en este proyecto para obtener la información de las páginas web ¿Por qué se ha escogido Scrapy? Se ha escogido Scrapy porque ofrece a los desarrolladores programar sus propios bots, término que se explicará posteriormente, y es gratis, la mayoría de software de web scraping son programas de pago. Es un framework programado en python. Permite a los desarrolladores programar sus propios bots "arañas web" también permite la utilización de plugins que añaden funcionalidades o cambian el comportamiento del framework. También es posible modificar el propio código fuente sobrescribiendo sus funciones. No se ha encontrado ningún software o framework que permita este grado de libertad al desarrollo de los bots.

Cuando un software de un dispositivo electrónico, como puede ser un ordenador de sobremesa o un servidor, navega por internet para obtener información simulando un comportamiento humano es llamado araña web, web crawler o spider bot.

Estos bots son utilizados por ejemplo por google para indexar las páginas webs en su base de datos para que puedan ser encontradas mediante su navegador.

Muchos estudios determinan que gran parte del tráfico de internet es generado por bots. Uno de estos artículos es el realizado por la empresa Barracuda de seguridad informática (ver [1]). El articulo explica que dos terceras partes del tráfico en internet son bots. A los propietarios de páginas webs no les interesa que el 66% del tráfico que navega por sus webs sean bots. Porque son costos de computación que no les aportan ningún tipo de beneficio.

Todas las páginas webs tienen un archivo público llamado robots.txt. Este archivo determina que urls pueden ser accedidas y cuáles no por los bots. En este proyecto la araña no analizó este archivo y navegó por todas las urls deseadas de www.idealista.com.

1.2. Estructura de un proyecto de Scrapy

Todos los proyectos de Scrapy comparten la misma estructura. En el siguiente recuadro (Fig. 1.1) se muestra como es la estructura de ficheros de un proyecto, suponiendo que el proyecto es llamado tutorial.

```
tutorial/
    scrapy.cfg
    tutorial/
        __init__.py
        items.py
        middlewares.py
        pipelines.py
        settings.py
        spiders/
            __init__.py
```

Fig. 1.1 Estructura de carpetas por defecto de scrapy

En los siguientes apartados se analizarán los archivos y cuál es su función en el proyecto.

1.3. Archivos del proyecto

1.3.1. scrapy.cfg

Este archivo es utilizado para la configuración de scrapy cuando se despliega en un entorno de producción. La configuración que viene por defecto de este archivo no se ha modificado porque el entorno de producción AWS no lo requería.

1.3.2. __init__.py

Este archivo es utilizado por python para importar paquetes externos. Durante la elaboración del proyecto no ha sido necesaria la importación de paquetes exernos de python.

1.3.3. Middlewares.py

Este archivo permite modificar y crear comportamientos en el bot. Por ejemplo, puede modificar la creación de las peticiones http o modificar el comportamiento del bot dependiendo de la respuesta del servidor. Más adelante se explicará más profundamente las implicaciones de los middlewares en scrapy. Este archivo ha sido modificado en este proyecto.

1.3.4. Pipelnes.py

Este archivo es ejecutado por el bot después de extraer los datos de la página web. El archivo pipelines permite un postprocesado de esta información permitiendo:

- “Limpiar” los datos HTML
- Validar los datos
- Buscar duplicados
- Guardar los datos en una base de datos.

1.3.5. settings.py

Este es de los archivos más importantes del proyecto. En él se encuentran todas las variables que configuran todo el comportamiento del bot como:

- El tiempo de espera entre peticiones a las diferentes urls.
- El valor del campo user agents de las peticiones.
- El uso o no de cookies.
- El uso o no de cache http para las peticiones.
- La configuración de los middlewares
- La configuración de los pipelines
- El número de peticiones concurrentes posibles.

1.3.6. Spiders

Este es el directorio donde se encuentran los archivos de los bots o arañas. En los apartados posteriores se mostrará el archivo utilizado.

1.4. Arquitectura de Scrapy

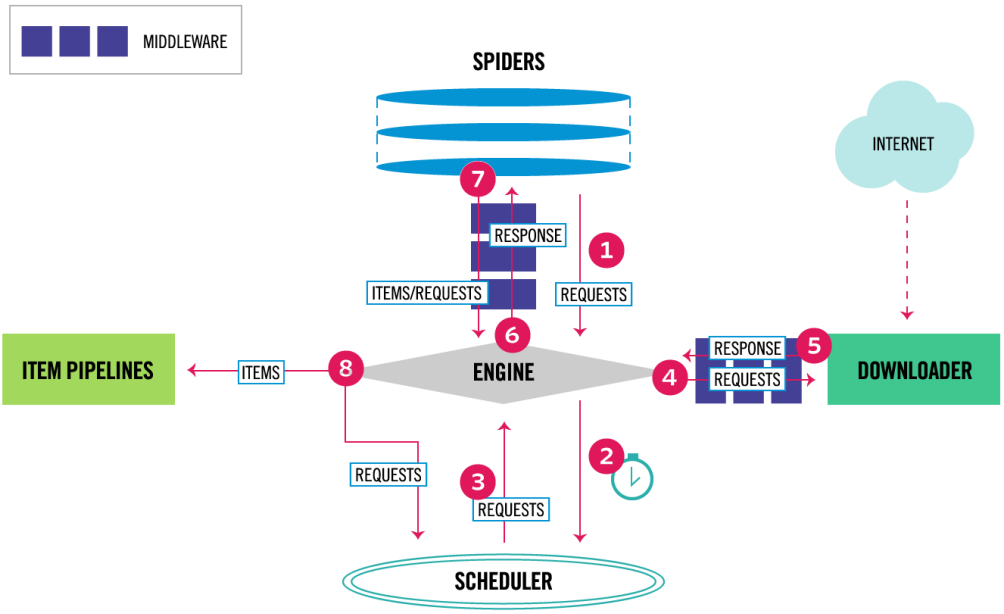


Fig. 1.2 Diagrama de la arquitectura de scrapy obtenido de la web oficial

1. El "engine" recibe las peticiones iniciales para extraer la información que el bot le indica.

2. El "engine" programa las peticiones en el scheduler y pregunta al bot la siguiente petición a realizar.

3. El "engine" recibe del scheduler las peticiones que han de ejecutarse en ese momento.

4. El "engine" envía las peticiones al Downloader pasando a través del Downloader Middlewares.

5. Una vez que la página termina de descargarse, el Donwloader genera una respuesta y la envía al engine, pasando a través de los Downloader Middlewares.

6. El engine recibe la Respuesta del downloader y la envía al bot para su procesamiento, pasando por el bot Middleware.

7. La bot procesa la Respuesta y devuelve los elementos procesados y las nuevas Solicitudes al engine, pasando por el bot Middleware.

8. El engine envía los elementos procesados al item pipeline, luego envía las solicitudes procesadas al scheduler y pide las posibles siguientes urls al bot.

9. El proceso se repite (desde el paso 1) hasta que no hay más urls en el scheduler.

1.5. Componentes de Scrapy

1.5.1. Engine

El engine se encarga de controlar el flujo de datos entre todos los componentes del sistema y de desencadenar eventos cuando se producen determinadas acciones.

1.5.2. Scheduler

El Scheduler recibe las solicitudes del engine y las pone en cola para más tarde darlas cuando el engine las necesita.

1.5.3. Downloader

El downloader es la parte responsable de descargarse las páginas web y dárselas al engine, que a su vez, se las da a los bots.

1.5.4. Spiders

Los bots son archivos programados por desarrolladores que se utilizan para extraer información de las páginas web descargadas.

1.5.5. Item pipelines

Los item pipelines son archivos programados por desarrolladores que se utilizan para procesar la información, una vez que ya ha sido extraída por los bots. Las

tareas típicas incluyen la limpieza, la validación y la persistencia de la información obtenida.

1.5.6. Middlewares de descarga

Los middlewares de descarga son componentes que hacen de intermediarios entre el engine y el donwloader. Cuando reciben peticiones del engine las procesan y las envían al Downloader y viceversa, cuando reciben respuestas del Downloader las procesan y las envían al engine.

Los middlewares de descarga son utilizados en los siguientes casos:

- Procesar solicitudes justo antes de que se envíen al Downloader.
- Cambiar las respuestas recibidas antes de enviarlas al bot.
- Enviar nuevas peticiones sin que el bot procese las respuestas.
- Enviar una respuesta al bot sin buscar una página web.
- Eliminar algunas peticiones del bot.

1.5.7. Middlewares de bot

Los middlewares de bot son componentes que hacen de intermediarios entre el bot y el engine. Son capaces de procesar tanto la entrada como la salida de los bots.

Los middlewares de bot son utilizados en los siguientes casos:

- Para modificar crear o eliminar la información extraída del bot.
- Para manejar las excepciones del bot.
- Para crear respuestas distintas dependiendo de la respuesta del bot

1.6. Middlewares en Scrapy

Scrapy utiliza unas técnicas de programación llamadas "hooking". Estas técnicas de programación se basan en la intercepción de las llamadas a funciones, eventos o mensajes entre componentes de software.

Toda la lógica de Scrapy va pasando entre componentes de software, desde que se crea una petición a una url hasta que se recibe la respuesta del servidor y se procesa la respuesta. Scrapy te permite crear una clase e implementarla entre dos procesos permitiendo así la creación de un comportamiento no esperado por el framework. Estas clases que permiten interceptar llamadas son los middlewares.

Como se ha explicado anteriormente, scrapy tiene dos tipos de middlewares, el download middleware y el spider middleware. El primero coje las peticiones justo antes de enviarse al servidor y las respuestas justo cuando llegan al engine. El segundo coje las peticiones justo después de que el bot las cree y coje las respuestas justo cuando entran en el bot.

Esta técnica de programación permite descargarse plugins a Scrapy, añadiendo funcionalidades que por defecto el framework no tiene. Un ejemplo sería la utilización de proxies o la rotación del valor del campo user_agents de las cabeceras de las peticiones.

A parte los desarrolladores también pueden crear sus propios middlewares. En este proyecto se ha realizado un middleware, más adelante se explicará en qué consiste.

Scrapy por defecto contiene una serie de middlewares, tanto download middleware como spider middleware. Estos middlewares pueden ser activados o no por los desarrolladores.

En la tabla (Tabla 1.1.) se muestran los download Middlewares que hay por defecto en Scrapy:

Tabla 1.1. Tabla de Middlewares de descarga de scrapy

Download Middleware	**Descripción**
CookiesMiddleware	Permite trabajar en webs donde las cookies son necesarias, igual que las sesiones de usuario.
DefaultHeadersMiddleware	Configura las cabeceras de las peticiones que vienen por defecto.

DownloadTimeoutMiddleware	Tiempo máximo de espera de una petición.
HttpAuthMiddleware	Permite la autenticación de todas las peticiones con basic auth.
HttpCacheMiddleware	Permite cache en todas las peticiones http.
HttpProxyMiddleware	Permite configurar un proxy por el que pasarán todas las peticiones.
RedirectMiddleware	Este middleware maneja la redirección de las solicitudes en función del estado de la respuesta.
MetaRefreshMiddleware	Este middleware maneja la redirección de peticiones basadas en la etiqueta html meta-refresco.
RetryMiddleware	Este middleware es utilizado para reintentar solicitudes fallidas que son potencialmente causadas por problemas temporales, como un tiempo de espera de la conexión o un error HTTP 500
RobotsTxtMiddleware	Este middleware filtra las peticiones prohibidas por la norma de exclusión de robots.txt.
UserAgentsMiddleware	Middleware que permite a las arañas anular el agente de usuario predeterminado.

En la tabla (Tabla 1.2.) se muestran los spider Middlewares que hay por defecto en Scrapy:

Tabla 1.2. Tabla de Spider middlewares de scrapy

Spider Middleware	**Descripción**
DepthMiddleware	El DepthMiddleware se utiliza para rastrear la profundidad de cada solicitud dentro de la página web que se está extrayendo.
OffsiteMiddleware	Filtra las respuestas HTTP erróneas para que los bots no tengan que lidiar con ellas, lo cual (la mayoría de las veces) impone una sobrecarga, consume más recursos y hace más compleja la lógica de los bots.
RefererMiddleware	Este middleware filtra todas las solicitudes cuyos nombres de host no están en el atributo allowed_domains del bot. También se permiten todos los subdominios de cualquier dominio de la lista.
UrlLenghtMiddleware	Filtra las solicitudes con URLs más largas que la propiedad URLLENGTH_LIMIT

CAPTÍULO 2. TECNOLOGIAS UTILIZADAS

A parte del framework de scrapy utilizado para crear el bot se han utilizado otras tecnologías para poder realizar esta tarea. Entre ellas estarían xpath. Utilizada para seleccionar la información requerida de la paginas web. Otra tecnología también seria python.

2.1. Xpath

Xpath es un lenguaje que permite mediante expresiones seleccionar partes de un fichero xml. Xpath fue definido por el consorcio W3C.
Las expresiones xpath serian como las queries sql para una base de datos sql. En vez de realizarse las queries a una base de datos se harían en un fichero xml.

En este proyecto es importante esta tecnología porque mediante estas expresiones el framework de scrapy selecciona la información necesaria de las páginas webs descargadas. Ha partir de un documento xml xpath lo analiza y genera un arbol de nodos. Este arbol representa el archivo xml con los elementos que aparecen en este representados en forma de nodos. Existen varios tipos de nodos:

- Nodo raiz
- Nodo elemento
- Nodo texto
- Nodo atributo
- Nodo comentario

2.2. Python

Python es un lenguaje de programación de alto nivel interpretado. En la actualidad es bastante popular por su sencilla sintaxis y su rápido aprendizaje comparado con otros lenguajes. Es el lenguaje de programación mas utilizado para los ámbitos de big data y machine learning.

Para elaborar este proyecto ha sido necesaria la elaboración de código de programación en este lenguaje. Esto ha ocurrido porque el framework utilizado esta escrito en este lenguaje y por consiguiente todo el código realizado esta escrito en python.

CAPTÍULO 3. CREACIÓN DEL PROYECTO EN UN ENTORNO LOCAL

Este proyecto primero se desarrolló en el entorno local de un ordenador propio y posteriormente se desplegó el proyecto a un servidor de amazon web service, simulando un entorno de producción.

3.1. Creación proyecto

Primero de todo se instaló Scrapy en un entorno local. Scrapy es una librería de python. En este proyecto se ha utilizado la versión de python 2.7 aunque en la web oficial de Scrapy se recomienda la versión 3.5.2+ la versión 2.7 se puede

utilizar igualmente. Hay varias formas de instalar Scrapy pero se ha optado por una de las más sencillas como se muestra en la figura siguiente (Fig. 2.1).

```
pip install Scrapy
```

Fig. 2.1 Comando de instalación del framework scrapy

Si en el entorno local no hubiese la librería pip se habría de descargar desde la página oficial de pip https://bootstrap.pypa.io/get-pip.py. Luego habría que ejecutar el siguiente comando (Fig. 2.2) en el directorio donde se encuentre el archivo.

```
python get-pip.py
```

Fig. 2.2 Comando para instalar la librería pip

Se empezó creando el proyecto inicial utilizando una plantilla de proyecto que provee el propio framework. Ejecutando el siguiente comando (Fig. 2.3) el framework crea una estructura de carpetas y archivos propios de scrapy, explicados anteriormente.

```
scrapy startproject tutorial
```

Fig. 2.3 Comando para instalar la librería pip

El comando anterior crea la estructura de carpetas y archivos, pero no genera el archivo que describe el bot. El bot como se ha explicado anteriormente es el software que navega a través de las páginas del sitio web indicado, extrayendo la información relevante para el proyecto. Este archivo se debe de colocar dentro de la carpeta Spiders que genera automáticamente el comando anterior. En este proyecto se utilizó como plantilla inicial el bot ofrecido en la página oficial de Scrapy mostrado en la siguiente figura (Fig. 2.4).

```
import scrapy


class QuotesSpider(scrapy.Spider):
    name = "quotes"
```

```
    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com/page/1/',
            'http://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, response):
        page = response.url.split("/")[-2]
        filename = 'quotes-%s.html' % page
        with open(filename, 'wb') as f:
            f.write(response.body)
        self.log('Saved file %s' % filename)
```

Fig. 2.4 Pantilla de archivo spider de la web de scrapy

A partir de este código de ejemplo se han hecho modificaciones para alcanzar los intereses del proyecto.

Como se ve en el código anterior hay creados dos métodos por defecto. start_requests y parse. Scrapy tiene varios métodos por defecto que se pueden utilizar en los bots. En este código se describen estos métodos que son suficientes para que un bot funcione y haga su labor.

parse(self, respuesta)

Este es el método predeterminado al que llama Scrapy para procesar las respuestas extraídas, cuando sus solicitudes no especifican otro método.

Este método debe devolver un objeto con un valor o varios.

start_requests(self)

Este método devuelve un objeto o varios con las urls que deben rastrearse por el bot. El método es llamado por scrapy cuando el bot es ejecutado. Scrapy solo llama una vez este método.

Con el código base proporcionado por Scrapy, del bot, el siguiente paso fue hacer que la araña extrajera la información de la página web deseada. En este caso es la información del portal de compra y venta de pisos https://www.idealista.com.

Se observó que esta página web recoge pisos de toda España. Pero en este proyecto sólo interesaban los pisos de Barcelona. Todos los pisos de Barcelona se encontraban en una url de este formato:

https://www.idealista.com/alquiler-viviendas/barcelona/{zona-barcelona}/{sub-zona-barcelona}/

Para recorrer todas las urls que contienen pisos de Barcelona el bot recorrió la url anterior sustituyendo los valores por los barrios de Barcelona.

Se hizo una primera prueba con el bot donde se puso en el array de urls la url de una zona de Barcelona con pisos:
https://www.idealista.com/alquiler-viviendas/barcelona/sant-marti/el-poblenou/

Dando como resultado el siguiente código (Fig. 2.5).

```
def start_requests(self):
        urls = ['https://www.idealista.com/alquiler-viviendas/barcelona/sant-marti/el-poblenou/']
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)
```

Fig. 2.5 Código que recorre una url

Una vez Scrapy se descarga una página el bot parsea el código html. Al parsear la web se genera un objecto al que se le pueden realizar queries. Así se pueden extraer los datos más interesantes del html.

Estas queries se realizan mediante xpath. XPath es un lenguaje que permite construir expresiones que recorren y procesan un documento XML.

En el siguiente código (Fig. 2.6) se muestra cómo se realiza en el proyecto. Response es el objeto que guarda toda la información del html y la variable QUERY es la query en formato xpath.

```
response.xpath(QUERY)
```

Fig. 2.6 Consulta para obtener datos requeridos

Para obtener la query en formato xpath se utilizó una extensión de google llamada ChroPath. ChroPath permite seleccionar elementos de las páginas web y como resultado muestra el xpath del elemento seleccionado, a parte de otras variables.

Los datos de los pisos que interesan en este proyecto son:

- El id del piso: para tenerlo identificado y no confundirlo con otros.
- Título del piso

- Precio
- Superficie

El primer paso fue seleccionar el xpath que compartían todos los pisos de una misma página. Realizar una query con este xpath da como resultado el array de pisos que contiene una página.
Para obtener este xpath se utilizó la extensión ChroPath. Este fue el xpath obtenido (Fig. 2.7).

```
//*[contains(concat( " ", @class, " " ), concat( " ", "item", " " ))] |
//*[contains(concat( " ", @class, " " ), concat( " ", "item_fade", " " ))] |
//*[contains(concat( " ", @class, " " ), concat(" ", "item-multimedia-container", "
"
))]
```

Fig. 2.7 Consulta para obtener todos los pisos de una pagina

El xpath anterior está seleccionando todas las partes, de las páginas web, iguales a la imagen inferior (Fig. 2.8). Por lo tanto al hacer una query con este xpath se seleccionan todos los pisos de las páginas.

Fig. 2.8 Imagen de piso en idealista.com

En el código del recuadro inferior (Fig. 2.9) se muestra cómo se recorren todos los pisos de una página. Se hace la query, antes mencionada, dando como resultado un array de pisos. Luego se recorren estos pisos con la expresión for. Como resultado cada vez que itera la expresión for la variable quote es un piso distinto.

```
for quote in response.xpath('//*[contains(concat( " ", @class, " " ), concat( " ",
"item", " " ))] | //*[contains(concat( " ", @class, " " ), concat( " ", "item_fade", " "
))] | //*[contains(concat( " ", @class, " " ), concat(" ", "item-multimedia-container", "
" ))]').getall():
```

Fig. 2.9 Código que itera en todos los pisos obtenidos

Una vez obtenidos cada uno de los pisos se ha obtenido la información específica que se requiere de cada uno de ellos, como el número de habitaciones, la superficie, etc...

En la fotografía inferior (Fig. 2.10) se muestra la zona en la página web donde se muestra la información que se quiere obtener, concretamente la superficie del inmueble. Para obtener los metros cuadrados del piso primero debemos obtener el xpath del elemento que engloba todos estos campos.

Fig. 2.10 Sección de una vivienda en idealista.com

Utilizando ChroPath obtenemos el xpath de la zona mostrada en la fotografía. ChroPath es una extensión de los navegadores chomium que facilita la obtención de los xpath de los elementos de una pagina web.

```
'//*[contains(concat( " ", @class, " " ), concat( " ", "item-detail", " " ))]'
```

Fig. 2.11 Xpath de una vivienda en idealista.com

Este xpath hace referencia el código html del recuadro de la fotografía superior (Fig. 2.11). La obtención del código html que referencia el xpath anterior se realiza mediante el código del recuadro inferior (Fig. 2.12).

```
info_items = Selector(text=quote).xpath('//*[contains(concat( " ", @class, " " ), concat(
" ", "item-detail", " " ))]').getall();
            for item in info_items:
```

Fig. 2.12 Query para obtener una vivienda en idealista.com

La variable info_items contiene toda la información del recuadro de la fotografía (Fig. 2.8). En el código del recuadro inferior (Fig. 2.13) se extrae la información de los metros cuadrados mediante su xpath. Su valor se guarda en la variable surface.

```
for item in info_items:
                if Selector(text=item).css('small::text').get() == 'm²':
                    surface = Selector(text=item).css('span::text').get()
```

Fig. 2.13 Bucle de listado de viviendas para obtener su superficie

Una vez obtenido el campo surface se han extraído el resto de campos que se han requerido. El campo surface era el campo más complejo de extraer. El resto de campos se extraen mediante un simple xpath. Mediante la expresión yield se retorna la información extraída al siguiente componente de Scrapy (Fig. 2.14).

```
yield {
                'id': Selector(text=quote).xpath('//article/@data-adid').get(),
                'title': Selector(text=quote).css('.item-link::text').get(),
                'price': Selector(text=quote).css('.h2-simulated::text').get(),
                'surface': surface
            }
```

Fig. 2.14 Creación de los datos guardados por cada vivienda

Todo el proceso, explicado anteriormente, se repite hasta que no hay más pisos que extraer en la página. En ese momento el software analiza si se puede realizar click a la página siguiente (Fig. 2.15). Si no es así el bot da por terminada la extracción de la información de ese distrito o zona.

Fig. 2.15 Imagen selección páginas de viviendas

Para poder hacer click al botón con el valor “siguiente” para cambiar de página se han realizado los siguientes pasos. Se ha encontrado el valor css de la flecha

de la derecha de la fotografía superior, mediante la extensión de chrome ChroPath. Se ha obtenido el elemento mediante css porque el valor del xpath era demasiado largo. Scrapy soporta ambos métodos para obtener elementos de las páginas, tanto xpath como css. En este caso la query es la del siguiente recuadro (Fig 2.16).

```
'.icon-arrow-right-after::attr(href)'
```

Fig. 2.16 Xpath del botón para cambiar de página de viviendas

En el código del proyecto se observa si este campo existe o no (Fig. 2.17). Si no existe significa que ya no hay más páginas para extraer pisos y el bot termina.

```
if next_page is not None:
```

Fig. 2.17 Condición para cambiar de pagina

Si la variable next_page no es nula se retorna una petición con esta url obtenida del botón siguiente (Fig. 2.18).

```
next_page = response.urljoin(next_page)
            yield Request(next_page, callback=self.parse)
```

Fig. 2.18 Query de obtención de la siguiente pagina

Todo el código que se ha comentado anteriormente forma el código del archivo del bot que recorre las páginas web mostrado en el siguiente recuadro (Fig. 2.19).

```
from scrapy import Request, Spider
from scrapy.selector import Selector
from scrapy.exporters import JsonItemExporter

class Scraper(Spider):
    name = 'scraper'
```

```
    rotate_user_agent = True


      def start_requests(self):
        urls = [
'https://www.idealista.com/alquiler-viviendas/barcelona/sant-marti/el-poblenou/'
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)


    def parse(self, response):

        print("URL: " + response.request.url)
               for quote in response.xpath('//*[contains(concat( " ", @class, " " ),
       concat( " ", "item", " " ))] |            //*[contains(concat( " ", @class, " "
       ), concat( " ", "item_fade", " " ))] | //*[contains(concat( " ", @class, " " ),
       concat(" ", "item-multimedia-container", " " ))]').getall():
                   info_items = Selector(text=quote).xpath('//*[contains(concat( " ",
       @class, " " ), concat( " ", "item-detail", " " ))]').getall();

            for item in info_items:
                if Selector(text=item).css('small::text').get() == 'm²':
                    surface = Selector(text=item).css('span::text').get()
            yield {
                'id': Selector(text=quote).xpath('//article/@data-adid').get(),
                'title': Selector(text=quote).css('.item-link::text').get(),
                'price': Selector(text=quote).css('.h2-simulated::text').get(),
                'surface': surface
            }

        next_page = response.css('.icon-arrow-right-after::attr(href)').get()
        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield Request(next_page, callback=self.parse)
```

Fig. 2.19 Código para obtener la información de todos los pisos de una ubicación en idelista.

Con la primera iteración del código se empezó a probar el bot.
Al principio no se descargaba la página apareciendo el siguiente error (Fig. 2.20).

```
ERROR: No response downloaded for: https://www.idealista.com/alquiler-
viviendas/barcelona/sant-marti/el-poblenou/
```

Fig. 2.20 Error al intentar descargar una página.

3.2. Pruebas scraping, mejora proyecto inicial

Después de realizar el bot, y probarlo, apareció un error al intentar descargarse la pagina que se quería analizar. Este fue un error inesperado. Después de buscar información por internet posibles soluciones al problema se encontró que una posible solución sería deshabilitando la lectura del archivo robots.txt

3.2.1. Archivo robots.txt

Esto es debido al archivo robots.txt. Este archivo lo tienen prácticamente todas las páginas webs. Especifica que urls de las páginas web no pueden ser navegadas por bots, es decir software.

Principalmente este archivo lo utilizan las páginas webs para que bots de google no analicen todas las urls de una página web. Así no se indexan en la base de datos de google paginas sin ninguna utilidad práctica para los usuarios. Este caso serían por ejemplo las páginas de contacto o las páginas de política de privacidad.

El comportamiento por defecto de scrapy es primero analizar este archivo y observar si la página que va a extraer scrapy lo permite robots.txt o no. En este caso el robots.txt de la página web idealista.com no permite la extracción de ninguna de sus páginas que contienen pisos.

Aunque los robots.txt indiquen que una página web no puede ser extraída scrapy permite extraerlas igualmente. Esto se consigue mediante la modificación de una de las propiedades del archivo settings.py, en concreto la variable ROBOTSTXT_OBEY. Por defecto esta variable está a True, pero en este proyecto será False, para poder extraer la información de idealista.com.

Una vez modificada esta propiedad el bot empezó a extraer información. Pero solo extrajo la información de las primeras 3 páginas. Luego apareció un error 403 forbidden (Fig. 2.21).

```
[scrapy] DEBUG: Crawled (403) <GET https://www.idealista.com/alquiler-
viviendas/barcelona/sant-marti/el-poblenou/pagina-10.htm> (referer: None)
```

Fig. 2.21 Error de prohibición al descargar una pagina

3.2.2. User agents

Este error aparece porque idealista.com tiene unas políticas internas de filtraje de las peticiones a sus servidores. Cada página web tiene las suyas y muchas no tienen. Los campos por los que usualmente se filtran las peticiones son las ips y el campo user agents de las peticiones http. Cuando un servidor detecta que algunas peticiones pueden ser creadas por un bot directamente prohíbe la navegación al servidor de esa ip.

El campo User Agents de las cabeceras de las peticiones contiene una cadena de caracteres. Este campo permite identificar a los servidores el tipo de aplicación, sistema operativo, proveedor de software y su versión de las peticiones que les llegan. Gracias a este campo un servidor puede saber qué tipo de dispositivo está intentando conectarse.

En nuestro proyecto al no tener ningún valor en el campo user agent el servidor detectó fácilmente que el bot no era un usuario.

Un ejemplo de user agent sería el del siguiente recuadro (Fig. 2.22)

```
Mozilla/5.0 (platform; rv:geckoversion) Gecko/geckotrail Firefox/firefoxversion
```

Fig. 2.22 Campo user agent de la cabecera de una peticcion http

- ***Mozilla/5.0*** Es el token general que indica que el navegador es compatible con Mozilla, es el más común en la mayoría de los navegadores actuales.
- ***platform*** Describe la plataforma nativa en la que el navegador se ejecuta (ejemplo. Windows, Mac, Linux o Android), y si es o no un teléfono móvil.
- **rv:*geckoversion*** indica la versión de Gecko(por ejemplo "17.0"). En los navegadores más recientes la versión de ***gecko*** *es la misma que la versión de* ***firefox***
- ***Gecko/geckotrail*** indica que el navegador está basado en Gecko.
- En escritorio ***geckotrail*** tiene la siguiente string fija "20100101"
- ***Firefox/firefoxversion*** indica que el navegador es Firefox, y muestra la versión (por ejemplo "17.0").

Al utilizar un user agent el número de páginas que se pudieron extraer fueron 30. Fueron más que anteriormente pero aun así no era el resultado esperado.

3.2.3. Middleware user agents

En este punto del proyecto se optó por una técnica muy utilizada que permite que los bots puedan realizar más peticiones sin ser detectados por los servidores. La técnica consiste en realizar las peticiones con user agents distintos en cada una de las peticiones. Esto se ha logrado seleccionando en cada una de las peticiones un user agent aleatorio de un listado de user agents previamente configurado.

Para poder implementar la técnica mencionada anteriormente se ha utilizado un middleware. Como se ha comentado en apartados anteriores un middleware es un software que permite cambiar el comportamiento de un bot. El tipo de middleware utilizado para variar el user agent de las peticiones es un downloader middleware, porque modifica las cabeceras de las peticiones justo cuando se envían.

Scrapy tiene un middleware por defecto para este propósito llamado UserAgentsMiddleware. Para poderlo utilizar hay que descargarlo primero como una librería de python (Fig. 2.23).

```
pip install scrapy-useragents
```

Fig. 2.23 Comando de instalación del middleware scrapy-useragents

Una vez instalada la librería se debe especificar a Scrapy que se está usando este middleware. También se ha de especificar en qué orden se debe de ejecutar en relación a los otros middlewares. Esto se configura en el fichero settings.py de la siguiente manera (Fig. 2.24).

```
DOWNLOADER_MIDDLEWARES={
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
    'scrapy_useragents.downloadermiddlewares.useragents.UserAgentsMiddleware': 500,}
```

Fig. 2.24 Configuración de middlewares en el proyecto

Con el código anterior se configura el bot para que utilice el middleware de user agents rotativos. También se desactiva el middleware de user agents que viene por defecto en el framework. Para desactivarlo hay que poner la prioridad a None. En cambio al middleware de user agents rotativo se le pone prioridad 500. Cuanto menor el número mayor la prioridad.
Cuanta más prioridad de un middleware antes se ejecuta respecto a los demás middlewares.

Una vez instalado el middleware de user agents falta especificar que user agents se deben enviar rotativamente. Para esto también se debe de modificar el fichero de settings.py añadiendo la variable USER_AGENTS con un listado de user agents aleatorios (Fig. 2.25).

```
USER_AGENTS = [
    ('Mozilla/5.0 (X11; Linux x86_64) '
     'AppleWebKit/537.36 (KHTML, like Gecko) '
     'Chrome/57.0.2987.110 '
     'Safari/537.36'),  # chrome
    ('Mozilla/5.0 (X11; Linux x86_64) '
     'AppleWebKit/537.36 (KHTML, like Gecko) '
     'Chrome/61.0.3163.79 '
     'Safari/537.36'),  # chrome
    ('Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:55.0) '
     'Gecko/20100101 '
     'Firefox/55.0')  # firefox]
```

Fig. 2.25 Listado de user agents utilizados en el proyecto

Para que el bot recorra todas las páginas de idealista se deben añadir todas las urls que se desean extraer en la variable url.

Para el proyecto se pensó que sería mucho mejor que la información extraída se guardase en carpetas en función del distrito al que pertenece, esta tarea la haría el bot. Cada carpeta tiene el nombre del distrito y en su interior carpetas especificando el rango de precios en los que hay los pisos. El rango de precios que se ha escogido es de 20000 euros a 20000 euros. Es decir, todos los pisos extraídos de Sants se guardan en la carpeta de Sants y están ordenados por precio, una carpeta de 0 a 20000 euros, de 20000 a 40000, así sucesivamente. Además, los nombres de los archivos extraídos tienen como nombre la fecha en la que se extrajeron.

Se ha escogido este método de organización porque si en un futuro se quisiese migrar toda esta información a una base de datos ya estaría ordenada y no habría que procesarla porque ya estaría bastante ordenada.

Para poder realizar esto el único método que se encontró fue a través de la línea de comandos. A través de la línea de comandos se le pueden enviar parámetros a la ejecución del bot. Entonces se pensó en enviar al bot el nombre del archivo y la url que debía extraerse a través del comando. Se ha hecho un script de windows para que ejecute en la cmd esta serie de comandos consecutivamente. El codigo del script se muestra en el siguiente recuadro (Fig. 2.26).

```
$basePath="https://www.idealista.com/venta-viviendas/barcelona/"
$neighborhood= @(
"eixample",
"ciutat-vella",
"sarria-sant-gervasi",
"eixample/la-dreta-de-l-eixample",
"sant-marti",
"sants-montjuic",
"gracia",
"ciutat-vella/el-raval",
"ciutat-vella/sant-pere-santa-caterina-i-la-ribera",
"sarria-sant-gervasi/sant-gervasi-galvany",
"les-corts",
"ciutat-vella/el-gotic",
"eixample/l-antiga-esquerra-de-l-eixample",
"playa-la-barceloneta",
"playa-somorrostro",
"gracia/vila-de-gracia",
"playa-sant-miquel",
"eixample/la-sagrada-familia",
"eixample/la-nova-esquerra-de-l-eixample",
"sarria-sant-gervasi/el-putxet-i-el-farro",
"playa-sant-sebastia",
"horta-guinardo",
"playa-de-la-barceloneta",
"ciutat-vella/la-barceloneta",
"les-corts/les-corts",
"sants-montjuic/el-poble-sec-parc-de-montjuic",
"eixample/sant-antoni",
"sarria-sant-gervasi/sant-gervasi-la-bonanova",
"sarria-sant-gervasi/sarria",
"eixample/el-fort-pienc",
"sants-montjuic/sants",
"sant-marti/el-poblenou",
"alquiler-viviendas/playa-el-bogatell",
"gracia/el-camp-d-en-grassot-i-gracia-nova",
"les-corts/la-maternitat-i-sant-ramon",
"playa-del-bogatell",
"sant-andreu",
"sant-marti/el-camp-de-l-arpa-del-clot",
"playa-la-nova-icaria",
"primera-linea-de-playa-de-la-barceloneta",
"sant-marti/diagonal-mar-i-el-front-maritim-del-poblenou",
"les-corts/pedralbes",
"playa-la-mar-bella",
"playa-de-la-nova-icaria",
"sarria-sant-gervasi/les-tres-torres",
"nou-barris",
"sants-montjuic/sants-badal",
"playa-la-nova-mar-bella",
```

```
"horta-guinardo/el-guinardo",
"sant-marti/el-parc-i-la-llacuna-del-poblenou",
"sants-montjuic/hostafrancs",
"gracia/vallcarca-i-els-penitents",
"horta-guinardo/el-baix-guinardo",
"playa-de-llevant-2",
"sant-marti/provencals-del-poblenou",
"gracia/la-salut",
"sant-marti/el-clot",
"sants-montjuic/la-bordeta",
"sant-marti/la-vila-olimpica-del-poblenou",
"sants-montjuic/la-font-de-la-guatlla",
"sant-andreu/sant-andreu",
"sant-marti/el-besos",
"sant-andreu/navas",
"primera-linea-de-playa-de-llevant-2",
"horta-guinardo/el-carmel",
"primera-linea-de-playa-de-la-nova-mar-bella",
"nou-barris/vilapicina-i-la-torre-llobeta",
"sant-andreu/la-sagrera",
"primera-linea-de-playa-de-la-mar-bella",
"sants-montjuic/la-marina-del-port",
"primera-linea-de-playa-del-bogatell",
"los-quince",
"horta-guinardo/horta",
"primera-linea-de-playa-de-la-nova-icaria",
"sant-andreu/el-congres-i-els-indians",
"horta-guinardo/la-teixonera",
"sant-marti/sant-marti-de-provencals",
"horta-guinardo/can-baro",
"gracia/el-coll",
"horta-guinardo/sant-genis-dels-agudells-montbau",
"nou-barris/porta",
"sarria-sant-gervasi/vallvidrera-el-tibidabo-i-les-planes",
"sant-marti/la-verneda-i-la-pau",
"primera-linea-de-playa-de-sant-sebasta",
"nou-barris/can-peguera-el-turo-de-la-peira",
"nou-barris/les-roquetes",
"nou-barris/la-prosperitat",
"horta-guinardo/la-font-d-en-fargues",
"nou-barris/ciutat-meridiana-torre-baro-vallbona",
"nou-barris/la-guineueta",
"nou-barris/verdun",
"horta-guinardo/la-vall-d-hebron-la-clota",
"sant-andreu/la-trinitat-vella",
"sant-andreu/el-bon-pastor",
"nou-barris/la-trinitat-nova",
"nou-barris/canyelles",
"sants-montjuic/zona-franca-port",

)
$filesName = Get-Date -Format s
$filesName = $filesName -replace "\:", 't'
for ($i=0;$i -lt $neighborhood.count; $i++){

for($j=0;$j -lt 10; $j++){

$highestPrice = $j * 20000
$lowerPrice = $j * 20000 + 20000
$urlToScrap = $basePath + $neighborhood[$i] + "con-precio-hasta_"+ $lowerPrice +",precio-
desde_"+ $highestPrice +"/"
$pathFile = "data/"+$neighborhood[$i]+"/"+$lowerPrice+"-
"+$highestPrice+"/"+$filesName+".json"

scrapy crawl scraper -o $pathFile -a url=$urlToScrap
```

```
}
}
```

Fig. 2.26 Código utilizado para recorrer todos los distritos de Barcelona por precio.

Una vez realizado este cambio y ejecutado el bot de nuevo se extrajeron 300 páginas del sitio web antes de ser expulsado con un error forbidden (error 403).

3.2.4. Scraping mediante proxies

Para poder extraer más pisos sin ser expulsado de la web se pensó otro método. Se pensó en realizar las peticiones mediante proxies.
Si alguna de las peticiones es rechazada por el servidor y las siguientes ya no son aceptadas por el servidor el bot cambiaria de proxy, por lo tanto de ip. Al cambiar de ip el servidor permite realizar peticiones porque no es capaz de relacionar las peticiones realizadas posteriormente, en la anterior ip, con la nueva. Hay diferentes middlewares de Scrapy para utilizar proxies. Se ha utilizado el middleware scrapy-rotating-proxies. Este middleware permite hacer las peticiones a través de un proxy y cuando Scrapy detecta un forbidden 403 automáticamente cambia de proxy.

En un entorno profesional los proxies son de pago. Como este proyecto no cuenta con presupuesto se optó por proxies gratuitos. En este proyecto se optó por utilizar proxies que se publican en listados de proxies en github de forma gratuita. La mayoría de los proxies no funcionan pero Scrapy intenta hacer las conexiones hasta que encuentra alguno que si lo hace.

Para instalar este middleware se descargó la librería con el siguiente comando (Fig. 2.27).

```
pip install scrapy-rotating-proxies
```

Fig. 2.27 Comando instalación del middleware scrapy-rotating-proxies

Luego se especificó en el archivo settings.py en la variable DOWNLOADER_MIDDLEWARE el orden de los middlewares (Fig. 2.28).

```
DOWNLOADER_MIDDLEWARES={
    'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
    'scrapy_useragents.downloadermiddlewares.useragents.UserAgentsMiddleware':      500,
    'rotating_proxies.middlewares.RotatingProxyMiddleware': 610,
    'rotating_proxies.middlewares.BanDetectionMiddleware': 620
}
```

Fig. 2.28 Configuración de middlewares en el proyecto

A parte se ha especificado la lista de urls de los proxies que se usarán. Se pueden especificar directamente en settings.py o en un fichero aparte. En este proyecto se ha hecho en un fichero aparte. Se ha hecho añadiendo la siguiente variable al fichero settings.py (Fig. 2.29).

```
ROTATING_PROXY_LIST = <PATH_DEL_ARCHIVO>
```

Fig. 2.29 Configuración para archivo externo con listado de proxies

El archivo como se ha explicado anteriormente contiene una lista de las ips de los proxies extraídos de github.

El fichero settings.py final es el siguiente (Fig. 2.30)

```
# -*- coding: utf-8 -*-

# Scrapy settings for idealista project
#
# For simplicity, this file contains only the most important settings by
# default. All the other settings are documented here:
#
#     http://doc.scrapy.org/en/latest/topics/settings.html
#

from .proxies import get_proxies


############################
# Main configuration
############################

BOT_NAME = 'idealista'

SPIDER_MODULES = ['idealista.spiders']
NEWSPIDER_MODULE = 'idealista.spiders'

DOWNLOADER_MIDDLEWARES = {
```

```
    #'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware': None,
    #'scrapy_useragents.downloadermiddlewares.useragents.UserAgentsMiddleware': 500,
    'idealista.middlewares.customMiddleware': 600,
    #'rotating_proxies.middlewares.RotatingProxyMiddleware': 610,
    'rotating_proxies.middlewares.BanDetectionMiddleware': 620
}
ROTATING_PROXY_BAN_POLICY = 'idealista.policy.MyPolicy'

DOWNLOAD_TIMEOUT = 3
DOWNLOAD_DELAY = 20
COOKIES_ENABLED = False

############################
# User agent configurarion
############################

USER_AGENTS = [
    ('Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/81.0.4044.113 Safari/537.36'),
    ('Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0')

    # Add more user agents which actually work nowadays
]

DEFAULT_REQUEST_HEADERS = {
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive',
    'Host': 'www.idealista.com',
    'Pragma': 'no-cache',
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/84.0.4147.125 Safari/537.36',
}

#########################
# Proxies configuration
#########################

RETRY_HTTP_CODES = [500, 502, 503, 504, 400, 403, 404, 408]

ROTATING_PROXY_PAGE_RETRY_TIMES = 99999999999 # TODO: is it possible to setup this
parameter with no limit?
ROTATING_PROXY_LIST_PATH  = 'C:\Users\ortegaro\Desktop\TFG\scraping\Scrapy-Idealista-
master\httpv2.txt'
```

Fig. 2.30 Archivo settings.py

Una vez configurado todo se ejecutó de nuevo el bot para que extrajera la información de las páginas web. En las listas de proxies que se obtienen de github están mezclados proxies que funcionan bien con proxies que no. La mayoría no funcionan, pero hay algunos que sí. El bot va probando los diferentes proxies hasta que encuentra uno que funciona. Este proceso puede durar hasta

15 minutos, entre que es expulsado de un proxy hasta que encuentra uno de nuevo que funcione.

Se dejó al bot extraer información durante seis horas y consiguió extraer la información de 10000 pisos. En ese momento se paró la ejecución del bot por motivos de seguridad.

3.2.5. Creación middleware

Llegados a este punto se quiso encontrar otra alternativa para extraer la información sin utilizar proxies.

Cuando el servidor de idealista rechaza las peticiones al cabo de 20 minutos permite realizarlas de nuevo. Se pensó en hacer esperar al bot 20 minutos y intentar realizar peticiones de nuevo. Desgraciadamente no hay ningún middleware ni proxy de Scrapy que permita hacer esto. Por este motivo se ha creado un middleware propio para esta tarea.

Scrapy permite crear middlewares propios. En ellos se utilizan métodos predefinidos por Scrapy que captan peticiones o respuestas en momentos distintos de la ejecución del bot.

```
import logging

from twisted.internet import defer
from twisted.internet.error import (
    ConnectError,
    ConnectionDone,
    ConnectionLost,
    ConnectionRefusedError,
    DNSLookupError,
    TCPTimedOutError,
    TimeoutError,
)
from twisted.web.client import ResponseFailed

from scrapy.exceptions import NotConfigured
from scrapy.utils.response import response_status_message
from scrapy.core.downloader.handlers.http11 import TunnelError
from scrapy.utils.python import global_object_name

logger = logging.getLogger(__name__)
from scrapy.downloadermiddlewares.retry import RetryMiddleware
import time

class customMiddleware (RetryMiddleware):
    def process_response(self, request, response, spider):
        if response.status in self.retry_http_codes:
            time.sleep(1200)
            reason = response_status_message(response.status)
            return self._retry(request, reason, spider) or response
```

```
        return response
```

Fig. 2.31 Middleware creado para esperar 20 minutos después de ser expulsado de la web

El código del recuadro superior (Fig. 2.29) es el middleware implementado en el proyecto. En este código se está utilizando una función interna de Scrapy, la función process_response. Esta función permite interceptar la respuesta del servidor al que se envían las peticiones. Esto permite que el bot espere un cierto tiempo antes de lanzar una nueva petición. En este caso se está cogiendo el campo response status de la respuesta. Este campo nos indica si la petición ha sido correcta, en este caso su valor sería 200, o ha habido algún tipo de problema, habría otros valores en este caso (códigos erróneos http). En el middleware se está observando el código de respuesta 403. Este código significa que el servidor ha rechazado la petición y no procesara más peticiones hasta 20 minutos más tarde. Por este motivo el valor del método time.sleep() es igual a 1200 segundos.

El valor retry_http_codes es una propiedad especificada en el settings.py donde se especifican los http codes por los cuales la araña debe de re intentar conectarse.

Una vez hechos todos estos cambios se lanzó de nuevo el bot. El bot descargó 10000 pisos, como en la prueba anterior con proxies, pero tardó 10 horas.

CAPTÍULO 4. EJECUCIÓN PROYECTO SCRAPY EN UN ENTORNO DE PRODUCCIÓN

Se han utilizado los servicios de computación en la nube de amazon para alojar el bot. Los servicios de computación en la nube de amazon se llaman (AWS) Amazon web services. Son una de las ofertas internacionales más importantes de la computación en la nube y compite directamente contra servicios como Microsoft Azure y Google Cloud Platform. Actualmente se requieren muchos profesionales cualificados con las certificaciones AWS ya que muchas empresas están demandando sus servicios.

AWS permite alojar sus servicios en 17 regiones geográficas. Estas son las regiones en las que se pueden alojar:

- EE.UU. Este: Norte de Virginia, Ohio
- EE.UU. Oeste: Norte de California, Oregón
- Asia Pacífico: Bombay, Seúl, Singapur, Sídney, Tokio
- Canadá: Central
- China: Pekín, Ningxia
- Europa: Fráncfort, Irlanda, Londres, París
- América del Sur: São Paulo

AWS ofrece una gran variedad de servicios en la nube. Entre ellos bases de datos, computación para machine learning, servicios serverless… y un sin fin de aplicaciones más. En este proyecto se va escoger la aplicación EC2. EC2 es un servicio que ofrece AWS que permite alquilar servidores virtuales por horas. Se alquila una máquina virtual con unas prestaciones escogidas por el usuario, tanto de memoria como de computación. También permite utilizar gran variedad de sistemas operativos a través de sus interfaces de servicios web, personalizarlos, gestionar permisos de acceso a la red y ejecutar tantos sistemas como uno desee. Este tipo de servicio supone un cambio en el modelo informático al proporcionar capacidad informática con tamaño modificable en la nube, pagando por la capacidad utilizada. En lugar de comprar o alquilar un determinado procesador para utilizarlo varios meses o años, en EC2 se alquila la capacidad por horas.

EC2 permite desplegar aplicaciones de forma escalable ya que si se necesitan más prestaciones por parte de los servidores con un par de clicks se puede migrar una aplicación a un servidor con más prestaciones. También permite la utilización de tantas instancias como uno quiera permitiendo también utilizar un load balancer, distribuyendo así el tráfico hacia las distintas instancias. Por este motivo EC2 (elastic compute cloud) es llamado elastic, por la elasticidad que ofrece en sus prestaciones.

También ofrece la posibilidad de alojar las aplicaciones en unas 17 localizaciones geográficas distintas (Fig. 3.1). Permitiendo así alojarlas en los lugares más interesantes para optimizar la latencia y altos niveles de redundancia.

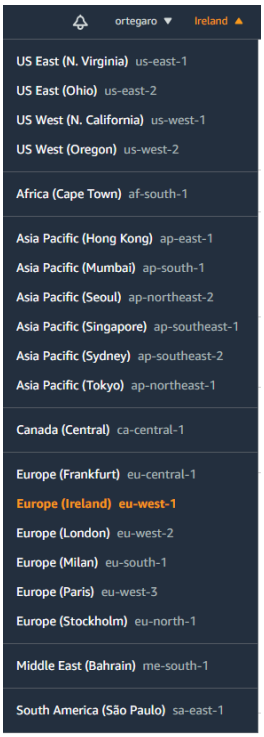


Fig. 3.1 Lista de localizaciones de AWS para ubicar una instancia EC2

4.1. Creación de Instancia EC2 para alojar aplicación

Una vez creada una cuenta de AWS se dispone de la posibilidad de utilizar cualquiera de los servicios que ofrece AWS. AWS permite el uso de alguna de sus tecnologías de forma gratis. Esto lo permite para que los usuarios puedan aprender a usarlas y familiarizarse con ellas durante un tiempo.

EC2 va a ser el servicio que se va a utilizar para desplegar la aplicación en un servidor. AWS ofrece las primeras 500 horas de una aplicación EC2 de tipo t2.micro de forma gratuita. Gracias a esta oferta se ha podido desplegar la aplicación en los servidores sin ningún tipo de coste.

El primer paso para crear una instancia EC2 es escoger una imagen del sistema operativo (Fig. 3.2). En este proyecto se ha escogido un sistema operativo Ubuntu 18.04 LTS - Bionic. Se ha escogido este sistema operativo porque los sistemas Linux son más estables que los sistemas de Windows. Por este motivo la mayoría de servidores utilizan SO basados en Linux.

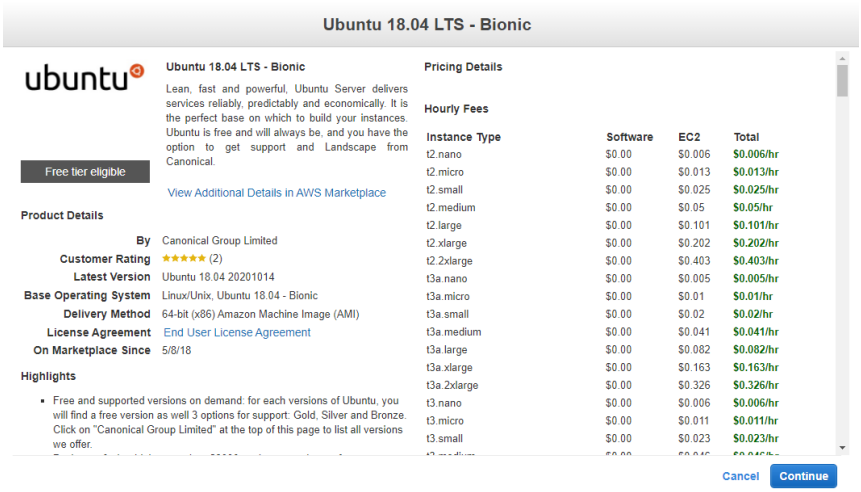
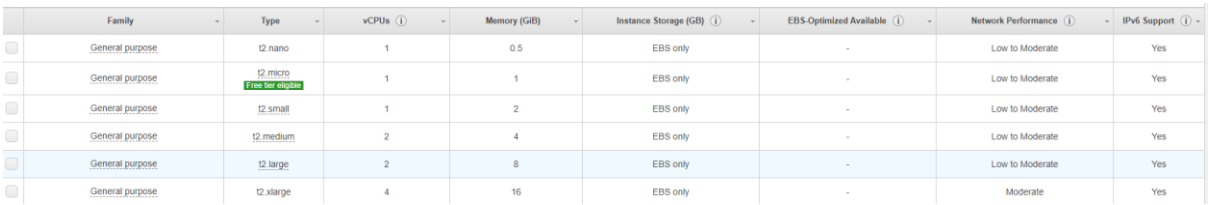


Fig. 3.2 Panel de descripción de la imagen del sistema operativo Ubuntu 18.04 LTS utilizada en el proyecto

Una vez escogido el sistema operativo se escogen las prestaciones de la máquina virtual. Se ha seleccionado un servidor de tipo t2.micro. Porque este servidor nos ofrece las 500 horas gratuitas comentadas anteriormente.



Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes

Fig. 3.3 Panel de selección de características de hardware de instancia EC2

En el siguiente paso se ha configurado la máquina virtual (Fig. 3.3). Se pueden seleccionar el número de instancias que se quieren lanzar, a la vez. En el proyecto se ha lanzado 1 pero se pueden lanzar hasta 10. En esta sección también se puede escoger la red de la instancia. Se pueden crear redes virtuales en las que solamente las ips de una red puedan acceder a la instancia. Se ha escogido la red por defecto, permite acceder a nuestra instancia desde cualquier ip.

También es posible escoger una subred dentro de la red, no se ha escogido ninguna. También se le puede asignar una ip a la instancia. En este proyecto no se escoge ninguna ip y se permite que la instancia obtenga su propia ip al crearse.

También se puede escoger un IAM role (Fig. 3.4). Un IAM role es un sistema de autentificación que permite configurar diferentes permisos para distintos usuarios. En este caso no se usará este sistema de autenticación. También se puede escoger la forma en que una instancia debería terminar si se produce un error. Puede terminar eliminando la instancia EC2 o solamente pausándola permitiendo reiniciarla posteriormente. En este caso se optará por pausarla. También se puede configurar que AWS pregunte cada vez que se elimine una instancia si realmente se quiere terminarla, se permite esta opción en la instancia. AWS también permite monitorizar el estado de las instancias al detalle aceptando la opción monitoring. De por sí AWS monitoriza las instancias pero si se necesita una monitorización más exhaustiva, porque es una aplicación crítica, AWS ofrece esta opción, es de pago.

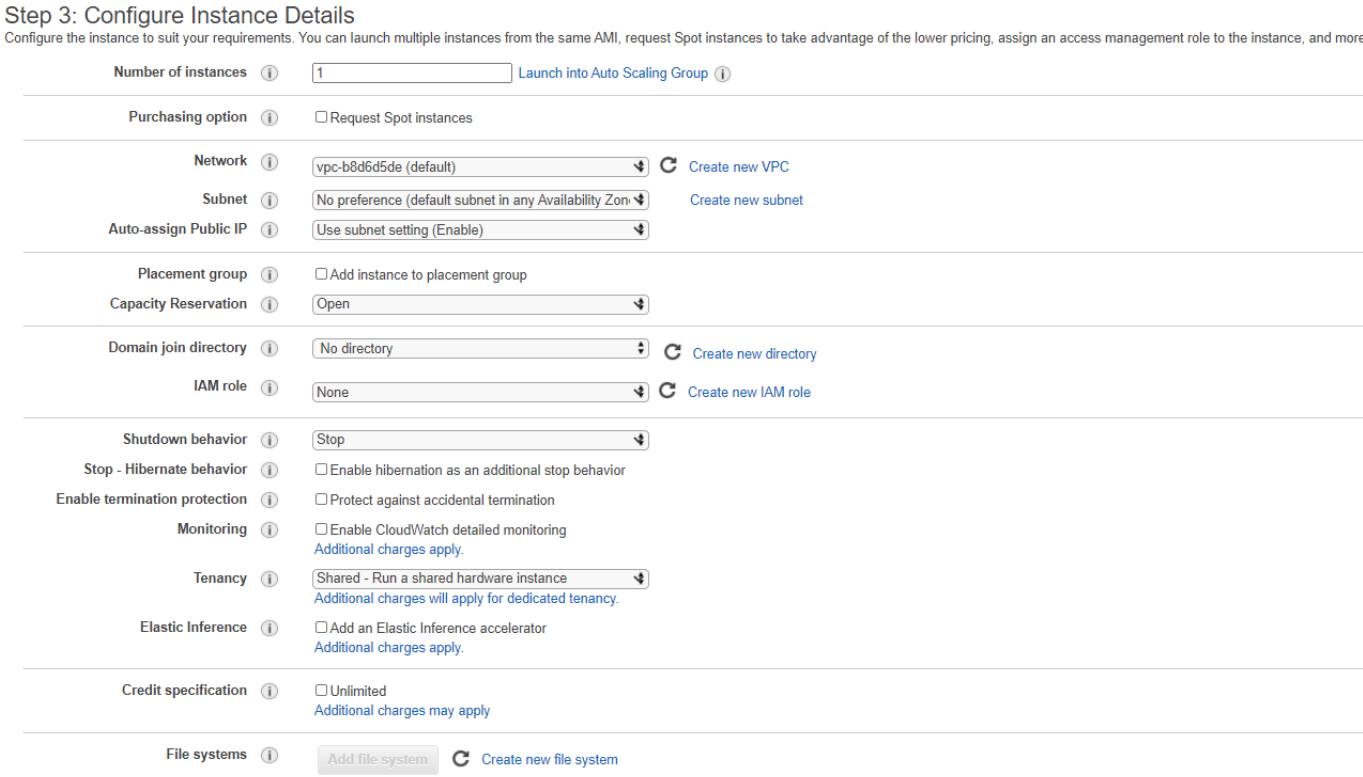


Fig. 3.4 Panel de configuración de detalles de una instancia EC2

4.1.1. Añadir almacenamiento

En el paso Agregar almacenamiento está seleccionada por defecto una memoria SSD de uso general de 8 GB (Fig. 3.5). (El tamaño máximo de volumen que podemos dar a un volumen de uso general es de 16 GB)

Se puede cambiar el tamaño de los volúmenes, agregar nuevos volúmenes, cambiar el tipo de volumen, etc.

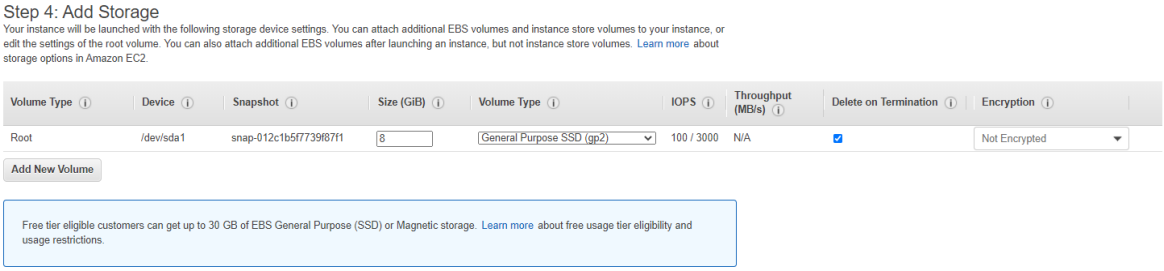


Fig. 3.5 Configuración del almacenamiento de la instancia EC2

4.1.2. Añadir etiquetas

En el paso 5 hay la posibilidad de añadir etiquetas a las instancias para asignarles un nombre y reconocerlas mejor (Fig. 3.6). No se puso ningún tag porque en el proyecto solo existe una instancia, no hay posibilidad de equivocarse con otras.

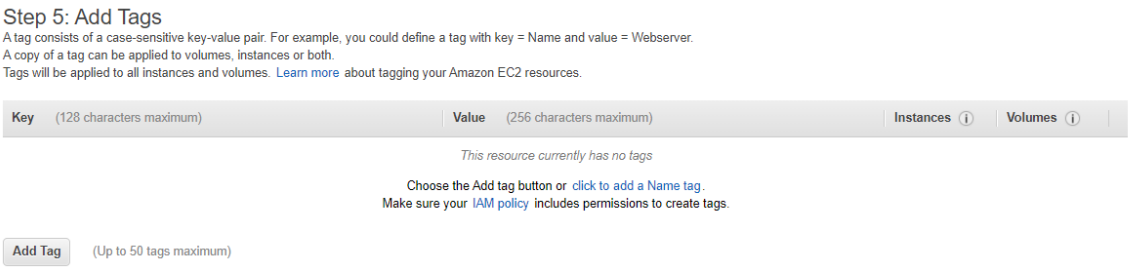


Fig. 3.6 Panel de configuración de etiquetas de instancia EC2

4.1.3. Configurar seguridad de grupo

En este siguiente paso se puede restringir el tráfico en los puertos de la instancia (Fig. 3.7). Este es un mecanismo de firewall adicional proporcionado por AWS, además del firewall del sistema operativo de su instancia. Se pueden definir las ips y los puertos abiertos.
En el proyecto se ha creado un nuevo grupo de seguridad. En este grupo se ha permitido el uso de los puertos ssh de http y https. Se ha dejado el puerto ssh

abierto porque se necesitaba conectividad con el servidor, para poder configurarlo posteriormente. También se han dejado los puertos http y https abiertos porque primordialmente son el tipo de peticiones que hará el bot.

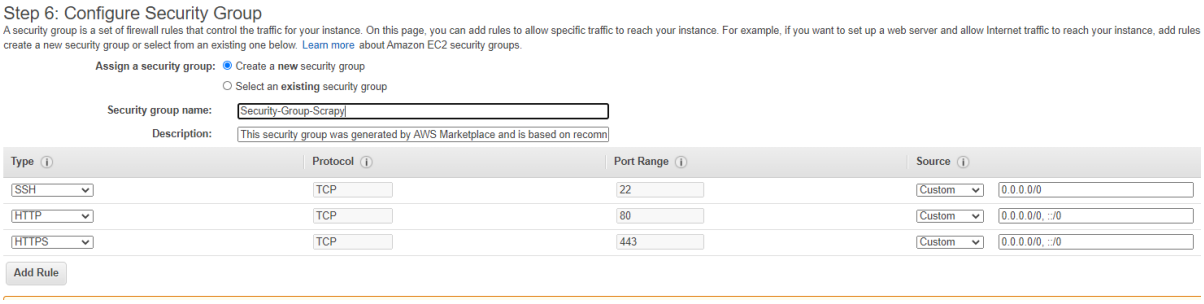


Fig. 3.7 Panel de configuración de puertos de la instancia EC2

También se podrían haber configurado un rango de ips por cada puerto pero se ha creído que no era necesario.
Una vez creada la instancia solo faltaba entrar en ella e instalar todo lo necesario para poder ejecutar el bot.

4.2. Configuración del servidor

Cuando se crea el servidor AWS da una clave privada. Mediante esta clave privada se puede acceder con una conexión ssh al servidor.
Para poder realizar la conexión al servidor se utilizó el programa MobaXtream (Fig. 3.8). Este programa permite manejar conexiones remotas de una manera más sencilla. MobaXterm proporciona todas las herramientas de red remotas importantes (SSH, X11, RDP, VNC, FTP, MOSH, ...) y comandos Unix (bash, ls, cat, sed, grep, awk, rsync, ...) para el escritorio de Windows. Por este motivo se ha escogido este programa.

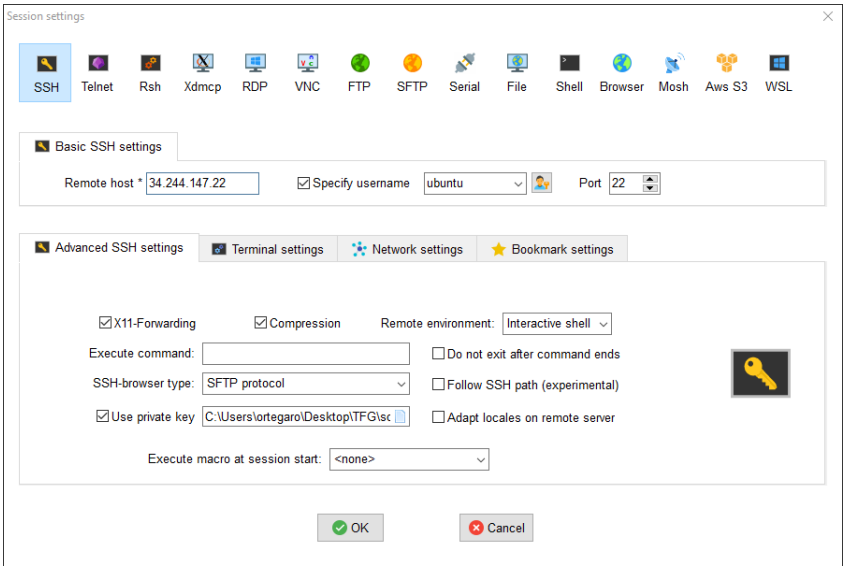


Fig. 3.8 Panel de configuración del protocolo ssh de MobaXtrem

Se configuró MobaXtream para conectarse con el servidor. Se especificó la ip del servidor, en este caso es 34.244.147.22, y se utilizó la llave privada que había dado AWS al crear la instancia. También se puso el usuario con el que se entró al servidor, el SO Ubuntu tiene por defecto el usuario ubuntu.

Una vez dentro del servidor se instaló python con el siguiente comando (Fig. 3.9).

```
sudo apt install python2
```

Fig. 3.9 Comando de instalación de python

Una vez instalado python se instaló la librería de python pip y la librería Scrapy, como se ha realizado anteriormente en la instalación del proyecto en un entorno local.

Luego se subió el proyecto de Scrapy que se había realizado en un entorno local al servidor. Para poder subirlo se utilizó el comando scp (Fig. 3.10).

Este comando permite copiar archivos almacenados en diferentes dispositivos de forma segura. Para subir el proyecto se utilizó este comando.

```
scp -i C:\Users\ortegaro\Desktop\TFG\scrapServerAwsKeys.pem -r
C:\Users\ortegaro\Desktop\TFG\scraping\tutorial
ubuntu@54.170.135.43:/home/ubuntu/TFG
```

Fig. 3.10 Comando scp utilizado para copiar elementos del entorno local al entorno remoto de la instacia EC2

Este comando indica con la opción -i que utilice una llave privada para la conexión. y que copie el contenido situado en la carpeta C:\Users\ortegaro\Desktop\TFG\scraping\tutorial en la carpeta home/ubuntu/TFG del servidor con el usuario ubuntu.
Después se ejecutó de nuevo el bot.

CAPTÍULO 5. CONCLUSIONES

En los objetivos de este trabajo figuraban crear un bot capaz de recopilar información de sitios web y un despliegue de la aplicación en el proveedor de servicios amazon web service.

El primer objetivo se ha podido realizar. Se ha creado un bot capaz de recopilar información de sitios web. Durante la implementación del bot se ha aprendido el funcionamiento de scrapy y como funciona internamente. Realizar el bot ha sido una tarea relativamente sencilla. La mayor dificultad del trabajo no ha sido realizar el bot sino encontrar las maneras de sortear las defensas de la pagina web para no ser rechazados y poder seguir recopilando información. Al comienzo del proyecto se desconocía que había políticas de filtraje de las peticiones http en los servidores para discriminar el tráfico de humanos y bots. Esto ha provocado que se hayan aprendido una serie de conceptos que se desconocían. La importancia del fichero robot.txt. Como los servidores filtran el tráfico de bots por el campo user agents de las cabeceras http. Como los servidores bloquean peticiones por ip. Gracias a la prueba y error y a la búsqueda de información se ha adquirido un conocimiento más profundo sobre las técnicas y herramientas del web scraping y del funcionamiento del filtraje de peticiones en los servidores.

Aun con los intentos del servidor web de filtrar las peticiones provenientes de bots se ha podido recopilar la información igualmente. Un método para captar este tipo de tráfico seria ejecutando un pequeño código de javascript en el lado del cliente. El resultado de la ejecución de este código seria enviado al servidor. Si el código no ha sido ejecutado correctamente significaría que el cliente no tiene habilitado javascript o que directamente no puede ejecutar código javascript. Esto seria un claro indicador de que el trafico esta siendo realizado por un bot.

La pagina web donde se han ejecutado las pruebas en el proyecto analizaba muy bien las peticiones desde el lado del servidor, pero desde el lado del cliente no se estaba realizando ningún tipo de filtraje. Si hubiese habido filtraje de peticiones desde el lado del cliente no se hubiese podido recopilar información de ningún piso con scrapy. Se tendría que haber implementado un bot con tecnologías con navegadores automatizados como es Selenium.

Respecto al segundo objetivo de desplegar la aplicación en los servicios de amazon web service ha sido cumplido también. Durante el despliegue de la aplicación se ha aprendido que es una instancia EC2 de amazon web service. Se ha aprendido a configurarla y ha desplegar la aplicación en una de estas instancias. La configuración de la instancia es relativamente fácil. Subir el proyecto del entorno local a la instancia EC2 también ha sido relativamente fácil. Se ha observado porque amzon web service es uno de lo mejores de proveedores de servicios en la nube. Con relativa facilidad se puede desplegar una aplicación en uno de sus servidores sin unos conocimientos muy profundos en configuración de servidores.

CAPTÍULO 6. BIBLIOGRAFÍA

[1] https://assets.barracuda.com/assets/docs/dms/Bot_Attacks_report_vol1_EN.pdf
https://scrapy.org/
https://pypi.org/project/scrapy-user-agents/
https://blog.scrapinghub.com/scrapy-proxy
https://es.wikipedia.org/wiki/Web_scraping
https://aws.amazon.com/es/
https://www.ticportal.es/temas/cloud-computing/amazon-web-services/amazon-ec2
https://www.datacamp.com/community/tutorials/making-web-crawlers-scrapy-python