

APLICACIÓN WEB DE COMPARACIÓN DE PRENDAS



VNiVERSiDAD
DE SALAMANCA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Septiembre 2023

RESUMEN

El comercio electrónico ha revolucionado el mundo de los negocios de desde que nació. Algunas de las ventajas de este relativamente nuevo modelo de negocio son la de reducción de costes, por no necesitar tienda física, el mayor alcance que proporciona, pues elimina las barreras geográficas, está abierto veinticuatro horas al día y siete días a la semana, etc. Las ventajas mencionadas, y muchas más, que caracterizan al comercio electrónico favorecen la competitividad y la democratización de las ventas, lo que da lugar a la aparición casi de forma diaria de una gran cantidad de empresas y productos distintos listos para competir en el mercado.

En 2021, en España, el 24% de las compras totales se hicieron a través de Internet y este porcentaje sólo se espera que suba en los años venideros. Ese mismo año, el negocio que más se benefició gracias al comercio electrónico fue el textil, que es el que más facturó a nivel nacional.

Este proyecto quiere dar respuesta a la gran cantidad de negocios y prendas que hay en el mercado textil para los jóvenes con el desarrollo de una aplicación web en la que se pueda ver una fácil comparación y clasificación de prendas en base a varios parámetros de tres de las marcas de ropa líderes en este sector: Zara, H&M y Carhartt WIP. Esta aplicación usará un algoritmo de búsqueda de prendas en los sitios web de dichas marcas y obtendrá datos de las prendas de forma periódica para posteriormente procesarlos y presentarlos de forma sencilla al usuario gracias a una interfaz web. Este modo de obtención de datos es llamado *web scraping* y también es usado por famosas webs como puede ser Rastreator o SkyScanner.

El proceso de construcción de esta aplicación se ha dividido en dos partes. La primera parte trata de un trabajo de investigación de las distintas singularidades de las webs utilizadas, en la que se han estudiado las estructuras de las páginas de los productos y las partes de dicha web en las que estos están situados. La segunda parte consiste en el desarrollo: se ha implementado el algoritmo de búsqueda de prendas aplicando todo el conocimiento obtenido de la primera fase para poder presentar al usuario de forma atractiva los datos obtenidos por el algoritmo de búsqueda mediante una aplicación web.

Las tecnologías utilizadas son el lenguaje Python, que se encuentra hoy a la cabeza de proyectos de este tipo, y dos *frameworks* que usan este lenguaje: Scrapy, un *framework* orientado a *web scraping* líder en rendimiento y Flask, un *framework* orientado a la creación de páginas webs.

Palabras clave

Comercio electrónico, prenda, sitio web, Zara, H&M, Carhartt WIP, *web scraping*, algoritmo de búsqueda, Python, Scrapy, Flask

ABSTRACT

The electronic commerce (from now on, ecommerce) has turned upside down the business world since the day it was born. Some of the advantages of this relatively new business model are the reduction of costs due to the lack of need of physical shops, the greater reach that it provides, because it deletes the geographical barriers, it is open twenty-four hours a day, seven days a week, etc. The advantages already mentioned and many more that define the ecommerce favour competitiveness and the democratisation of sales, which leads to the appearance on a daily basis of an enormous number of new companies and products ready to compete in the market.

In 2021, in Spain, the 24% of the total sales were committed through Internet and this percentage is only expected to rise on the upcoming years. That same year, the market that more benefit took from the ecommerce was the clothing market, which was the one that more profit made nationally.

This project wants to give an answer to the huge quantity of businesses and garments that are in the textile market for young people with the development of a web application where an easy comparison and classification of clothes can be made based on a few parameters of some of the most leading brands in this sector: Zara, H&M and Carhartt WIP. This application will use a search algorithm for clothes in those companies' websites and will get data from the clothes regularly to later process and present it easily to the users thanks to a web interface. This method of extracting the data is called web scraping and it is also used by famous websites like Rastreator or SkyScanner.

The process of building the application is divided in two phases. The first one consists of doing research of the singularities of each web, where the structures of the product's pages and the different parts of them have been studied. The second part consists of the development itself: the clothing search algorithm is implemented applying all the knowledge that has been taken from the research phase in order to be able to present to the user that extracted data attractively with a web application.

The technologies that have been used are the Python programming language, that nowadays leads this kind of projects, and two frameworks that use this language, Scrapy, a web-scraping-oriented framework leader in performance and Flask, a web-development-oriented framework.

Key words

Ecommerce, clothe, website, Zara, H&M, Carhartt WIP, web scraping, search algorithm, Python, Scrapy, Flask

ÍNDICE

1.	Introducción	1
1.1.	Contexto	1
1.2.	Estructura de la memoria	2
1.2.1.	Descripción de la memoria principal	2
1.2.2.	Descripción de anexos	3
2.	Objetivos del proyecto	5
2.1.	Objetivos generales.....	5
2.2.	Objetivos personales.....	6
3.	Conceptos teóricos	7
3.1.	API.....	7
3.1.1.	API REST	8
3.2.	Funcionamiento básico de un sitio web.....	8
3.2.1.	La URL	9
3.2.2.	El documento HTML	10
3.2.3.	Carga dinámica de contenido	10
3.2.4.	DOM, la representación en árbol	10
3.3.	Web Scraping.....	11
3.3.1.	Parsing	12
3.3.1.1.	XPath	12
3.3.1.2.	Expresiones regulares	15
3.3.2.	Crawling.....	15
3.3.2.1.	Archivo robots.txt	15
3.3.2.2.	Sitemaps.....	16
3.3.2.3.	<i>Crawling:</i> tipos y problemas.....	17
3.4.	Framework.....	20
3.4.1.	Scrapy	20
3.4.1.1.	Conceptos básicos	20
4.	Técnicas y Herramientas.....	25
4.1.	Herramientas y utilidades de desarrollo	25
4.1.1.	Visual Studio Code	25
4.1.2.	Scrapy	25
4.1.3.	MariaDB	26
4.1.4.	Flask.....	26
4.1.5.	Bootstrap.....	27

4.1.6.	Mozilla Firefox: herramienta para desarrolladores web	27
4.2.	Lenguajes empleados.....	29
4.2.1.	Python	29
4.2.2.	HTML y CSS	29
4.2.3.	SQL.....	30
4.3.	Otras herramientas	30
4.3.1.	Microsoft Word	30
5.	Aspectos Relevantes del Desarrollo	33
5.1.	Investigación.....	33
5.1.1.	Zara	34
5.1.2.	H&M.....	42
5.1.3.	Carhartt WIP	48
5.1.4.	Bershka	52
5.1.5.	Pull&Bear	52
5.2.	Desarrollo	53
5.2.1.	<i>Web Scraping</i>	53
5.2.1.1.	Ítems.....	54
5.2.1.2.	<i>Spider</i> de Zara	55
5.2.1.3.	<i>Spider</i> de H&M.....	58
5.2.1.4.	<i>Spider</i> de Carhartt WIP	61
5.2.1.5.	<i>Item Pipelines</i>	63
5.2.1.6.	Configuración	70
5.2.1.7.	<i>Middlewares</i>	74
5.2.1.8.	<i>Script</i> de lanzamiento de las arañas	74
5.2.2.	Aplicación web	75
5.2.2.1.	API Imágenes.....	76
5.2.2.2.	Uso de Bootstrap.....	77
5.2.2.3.	Distintas URLs y documentos HTML	77
6.	Trabajos Relacionados	83
6.1.	Lyst	83
6.2.	Fashiola.....	84
7.	Conclusiones y Líneas de Trabajo Futuro.....	87
7.1.	Conclusiones personales.....	87
7.2.	Conclusiones técnicas.....	88
7.3.	Líneas de trabajo futuras.....	88

8. Bibliografía	91
---------------------------	----

LISTA DE ILUSTRACIONES

Ilustración 1 - Explicación de una API	7
Ilustración 2 - Secuencia básica de la evolución de un documento HTML	9
Ilustración 3 - Herramienta Inspeccionar	10
Ilustración 4 - Representación en DOM	11
Ilustración 5 - Noticia ejemplo para XPath	12
Ilustración 6 - Ejemplo de código XML.....	13
Ilustración 7 - Cabecera HTTP por defecto de Scrapy	19
Ilustración 8 - Cabecera HTTP de Google Chrome desde un macOS.....	19
Ilustración 9 - Configuración de arañas en Scrapy	21
Ilustración 10 - Configuración de proyecto en Scrapy	22
Ilustración 11 - Arquitectura de Scrapy	23
Ilustración 12 - Pestaña Inspector de la herramienta para desarrolladores web de Mozilla Firefox	28
Ilustración 13 - Pestaña Console de la herramienta para desarrolladores web de Mozilla Firefox	28
Ilustración 14 - Pestaña Network de la herramienta para desarrolladores web de Mozilla Firefox	28
Ilustración 15 - Valores que puede tomar el campo de género mediante XPath en Zara	35
Ilustración 16 - Obtención de composición mediante la API de Zara	36
Ilustración 17 - API de Buscar en Zara	37
Ilustración 18 - Estructura de los sitemap de Zara	37
Ilustración 19 - Contenido de sitemap-es-es.xml.gz de Zara	38
Ilustración 20 - Resultados de "tipos_sitemaps.py" del 09-06-20.....	41
Ilustración 21 - Resultados de "tipos_sitemaps.py" del 17-06-2023.....	41
Ilustración 22 - Formato de composición en H&M.....	43
Ilustración 23 - Proceso de análisis de la API de carga dinámica de H&M	44
Ilustración 24 - Atributos del archivo JSON obtenido con la API de H&M	46
Ilustración 25 - Atributo category de los objetos del archivo JSON de la API de H&M	46
Ilustración 26 - Proceso de análisis de la API de Buscar de H&M	47
Ilustración 27 - Estructura de los sitemap de H&M	48
Ilustración 28 - Nombre artículos de Carhartt	49
Ilustración 29 - Estructura de los sitemap de Carhartt.....	50
Ilustración 30 - Contenido de es/sitemap-products.xml de Carhartt	50
Ilustración 31 - Formato URLs de es/sitemap-products.xml de Carhartt.....	51
Ilustración 32 - API de Buscar de Carhartt.....	51
Ilustración 33 - Clase derivada de Item en Scrapy utilizada	55
Ilustración 34 - Expresión regular de rechazo de URLs de productos que no son los elegidos.....	56
Ilustración 35 - Parsing de los enlaces tipo -p	56
Ilustración 36 - Parsing de los enlaces tipo -l.....	56
Ilustración 37 - Clase SitemapSpider de Zara	57
Ilustración 38 - Atributo sitemap_urls en Zara	57
Ilustración 39 - Método sitemap_filter() de Zara	57
Ilustración 40 - Redirección a cada tipo de enlace con su método gestor	58
Ilustración 41 - Paso de lista de IDs de productos de Zara al iniciarse la araña de Zara	58

Ilustración 42 - Expresión regular de rechazo de productos no deseados por nombre y categoría H&M	58
Ilustración 43 - Clase Spider de H&M	59
Ilustración 44- Atributo start_urls en H&M	59
Ilustración 45 - Preparación para llamar a la API JSON de H&M	59
Ilustración 46 - Llamada a la API JSON de H&M	60
Ilustración 47 - Parsing del archivo JSON de H&M	60
Ilustración 48 - Parsing del HTML de productos para añadir composición y descripción	61
Ilustración 49 - Expresión regular de rechazo de URLs de productos que no son los elegidos en Carhartt	61
Ilustración 50 - Parsing de los productos en Carhartt.....	62
Ilustración 51 - Artículo de Carhartt.....	62
Ilustración 52 - Clase SitemapSpider de Carhartt	63
Ilustración 53 - Atributo sitemap_urls en Carhartt	63
Ilustración 54 - Clase PipelineErroneos para filtrar productos con campos vacíos	64
Ilustración 55 - Clase PipelineNinos para filtrar productos de niños de Zara	64
Ilustración 56 - Tratamiento de nombres en la clase PipelineLimpieza	65
Ilustración 57 - Creación del ID en la clase PipelineLimpieza	66
Ilustración 58 - Tratamiento de la composición en la clase PipelineLimpieza	66
Ilustración 59 - Formatos de cadenas extraídas con XPath para el campo color	66
Ilustración 60 - Limpieza del campo color en clase PipelineLimpieza.....	67
Ilustración 61 - Estructura de directorios donde se almacenan las imágenes y miniaturas de los productos	69
Ilustración 62 - Asignación de rutas a las imágenes descargadas con la clase PipelineImagenes.....	69
Ilustración 63 - Inicialización y cierre del acceso a la base de datos en clase PipelineBaseDatos	70
Ilustración 64 - Inserción en la base de datos en clase PipelineBaseDatos	70
Ilustración 65 - Configuración de la araña de Zara	71
Ilustración 66 - Configuración de la araña de H&M	71
Ilustración 67 - Configuración de la araña de Carhartt WIP	71
Ilustración 68 - Configuración de cabeceras y User-Agent.....	71
Ilustración 69 - Configuración del orden de envío de las peticiones.....	72
Ilustración 70 - Configuración de seguimiento de robots.txt	72
Ilustración 71 - Configuración de ratio de envío de peticiones	72
Ilustración 72 - Configuración de cookies.....	72
Ilustración 73 - Configuración de orden Item Pipelines.....	73
Ilustración 74 - Configuración de orden de Downloader Middlewares	73
Ilustración 75 - Configuración de reintentos	73
Ilustración 76 - Configuración de almacenamiento de imágenes.....	74
Ilustración 77 - Configuración del log.....	74
Ilustración 78 - Obtención de las URLs de productos de Zara.....	75
Ilustración 79 - Ejecución de las arañas en el script.....	75
Ilustración 80 - API de cargado de imágenes de la aplicación web	76
Ilustración 81 - Visualización de plantilla de paginación de Bootstrap	77
Ilustración 82 - Código de CSS y JavaScript utilizado de Bootstrap	77

Ilustración 83 - Direcciones posibles de la página web.....	77
Ilustración 84 - Documentos HTML que puede devolver la página web	78
Ilustración 85 - Visualización de la página de market.html	78
Ilustración 86 - Obtención de los filtros seleccionados	78
Ilustración 87 - Obtención de los productos de la base de datos en base a los filtros seleccionados	79
Ilustración 88 - Ordenación de los productos en base a precio	79
Ilustración 89 - Visualización de la página de registro.html	80
Ilustración 90 - Visualización de la página de login.html	80
Ilustración 91 - Visualización de la página item.html	80
Ilustración 92 - Búsqueda de productos en Lyst.....	83
Ilustración 93 - Sistema de filtros en Lyst	84
Ilustración 94 - Productos en Lyst.....	84
Ilustración 95 - Búsqueda de productos en Fashiola	85
Ilustración 96 - Búsqueda de filtros en Fashiola	85
Ilustración 97 - Visualización de producto en Fashiola.....	85

LISTA DE TABLAS

Tabla 1 - Tabla de parámetros a obtener al hacer scraping	33
Tabla 2 - Scripts de investigación del sitemap de Zara	40
Tabla 3 - Campos que tiene cada ítem	54
Tabla 4 - Tabla de asignación de colores en PipelineLimpieza	68

1. INTRODUCCIÓN

1.1. Contexto

El nacimiento del comercio electrónico tiene lugar en la década de los 1920 en Estados Unidos con la venta por catálogo, que permitió, por primera vez en la historia, vender un producto sin necesidad de verlo físicamente ni de desplazarse al punto de venta. Este precedente sembró el germen de una tendencia que ha sido imparable y que hoy por hoy mueve a nivel mundial cerca de 6 billones de euros.

El sector textil, uno de los que más se han beneficiado del comercio online, fue valorado en 654.6 mil millones de dólares americanos en 2021. En este sector, en el podio se sitúan multinacionales como la española Inditex, que enmarca distintas marcas como Pull&Bear, Massimo Dutti o su marca insignia Zara. El conglomerado español tuvo de beneficio 2505 millones de euros, seguido entre otras por la marca sueca H&M con una ganancia de 624 millones de euros.

Por otro lado, gracias a Internet y a la gran cantidad de negocios que han florecido gracias a este, se han hecho necesarias herramientas para rastrear y comparar la ingente cantidad de ofertas y precios que hay en busca de la mejor opción. Ha habido sitios que han destacado en esta disciplina como SkyScanner¹, un comparador de vuelos o Rastreator² que a su vez compara seguros. Dichas marcas han logrado, gracias a un potente algoritmo de búsqueda por toda la web, estandarizar y comparar el acceso a una gran cantidad de información y presentarla de forma sencilla al cliente para que este tome decisiones de forma más eficaz.

El proceso de obtención información de estas páginas web se denomina *web scraping*³. Para ello, se usan programas o *scripts*⁴, llamados comúnmente *bots* en la jerga informática, que recorren los distintos enlaces de un sitio web o a través de una API⁵ (*Application Programming Interface*) y almacenando la información que se considere relevante. Al vivir en una sociedad donde cada vez la información y el tratamiento de datos es más importante, este campo de la informática puede proveer mucho valor, ya que un buen conjunto de datos obtenidos gracias a este método puede ayudar posteriormente a crear estadísticas y a su vez modelos de predicción y de tendencias útiles para un negocio. En la práctica es un proceso muy útil, pero a la vez costoso y complejo, pues cada web tiene distintas particularidades como puede ser la disposición de datos que otras webs pueden no tener o disponer en distintos formatos, tener mecanismos para evitar que programas las recorran o funcionar simplemente de forma distinta. Lo que da lugar a que

¹ <https://www.skyscanner.net/>

² <https://www.rastreator.com/>

³ El proceso de obtención automatizada de información presente en una o varias páginas web. Se cubrirá en profundidad en el apartado “3.3 Web Scraping”.

⁴ En español, guion, se trata de un archivo de texto que contiene instrucciones que son interpretadas línea a línea en tiempo real cuando se ejecuta.

⁵ En español, Interfaz de Programación de Aplicaciones, se explicará en el apartado “3.1 API”.

estandarizar la información recogida sea difícil. Todo esto hace que el *web scraping* sea una actividad demandada y muy presente en nuestros días.

Volviendo ahora a este proyecto, constará de dos partes. La primera, consiste en la investigación para obtener webs candidatas y las posibilidades que ofrecen las elegidas, en el cual se seleccionan los sitios web de los que se hará *web scraping* y se profundizará en el funcionamiento de las elegidas. Al hacer la selección de marcas se primará marcas de ropa para el público juvenil conocidas y que provean de cierta facilidad para la obtención de los datos de las prendas y que además muestren los siguientes parámetros: nombre, precio, descripción de producto, color, composición, sexo o género e imagen del producto. En la investigación de las resultantes se ahondará en el proceso de recorrer los productos y en la obtención de los parámetros buscados para cada prenda, que resulta distinto en cada página.

Las principales tecnologías usadas son el lenguaje Python, un lenguaje de *scripting* de tipado débil que se sitúa hoy en día a la vanguardia en muchos sectores dentro de los que se encuentra el *web scraping*. Además, se han utilizado dos *frameworks*⁶ que usan dicho lenguaje de programación: Scrapy que está especializado en *web scraping* y que se ha utilizado en el motor de búsqueda de las prendas y Flask para el desarrollo e implementación de la página web.

1.2. Estructura de la memoria

La estructura de la memoria se dividirá en dos apartados, la descripción de la memoria principal, que definirá la estructura de este mismo documento donde se darán los aspectos más importantes del proyecto y la descripción de los anexos, que son los documentos complementarios que tratarán aspectos más técnicos.

1.2.1. Descripción de la memoria principal

La memoria principal está dividida en los siguientes apartados:

- **Resumen y abstract:** se trata de un breve resumen para dar una idea general del proyecto. Está escrito en español y en inglés.
- **Introducción:** se presentan los aspectos más importantes del proyecto y se contextualiza en el marco en el que se ha desarrollado la aplicación. Además se muestra brevemente la estructura del trabajo.
- **Objetivos del proyecto:** se indican las metas a las que se quiere llegar con el proyecto. Son de diferentes tipos, funcionales, no funcionales y personales.
- **Conceptos teóricos:** se explican distintos aspectos relevantes sobre los que se sustenta el trabajo.
- **Técnicas y herramientas:** los distintos procesos y las herramientas utilizadas que han sido necesarias para llevar a cabo todo el trabajo.

⁶ En español es entorno o marco de trabajo y se trata de un conjunto de reglas y convenciones que se usan para desarrollar software de forma más eficiente y rápida.

- **Aspectos relevantes del desarrollo:** se destacan los puntos más relevantes que se han dado en el proceso de formación de este proyecto.
- **Trabajos relacionados:** se comparará el producto final con aplicaciones ya existentes.
- **Conclusiones y futuras líneas de trabajo:** se exponen las conclusiones obtenidas tras realizar el proyecto y cuáles podrían ser las principales vías de mejora.
- **Bibliografía:** se citan las fuentes usadas para la realización de este proyecto.

1.2.2. Descripción de anexos

Los anexos, que son documentos complementarios principalmente técnicos son los siguientes:

- **Anexo I – Plan de Proyecto del Software:** se trata la planificación temporal del proyecto, un aspecto clave de la Gestión de Proyectos, en el cual se elabora una programación temporal de las distintas actividades que conforman el proyecto.
- **Anexo II – Especificación de Requisitos del Software:** contiene los requisitos del sistema software a elaborar y cómo se relacionan además de la refinación de los requisitos en la etapa de análisis.
- **Anexo III – Especificación de Diseño:** continuación del Anexo II, tratando las etapas de Ingeniería del Software de análisis e implementación.
- **Anexo IV – Documentación Técnica de Programación:** este documento contiene toda la información técnica relevante al código fuente.
- **Anexo V – Manual de Usuario:** se detalla los puntos clave del sistema desde el punto de vista del usuario para que maneje correctamente la aplicación.

Introducción

2. OBJETIVOS DEL PROYECTO

El objetivo de este proyecto es el de realizar una prueba de concepto de una aplicación web que compare de forma efectiva y realista prendas de Zara, H&M y Carhartt WIP⁷ (*Work In Progress*) de forma que sea fácil seleccionar, filtrar y ordenar prendas en base a distintos parámetros y que de esta forma el usuario pueda tomar mejores elecciones a la hora de comprar prendas textiles.

A continuación se expondrán los distintos tipos de objetivos de este proyecto, los funcionales, los no funcionales y los personales.

2.1. Objetivos generales

- Investigación de las páginas web de Zara, H&M y Carhartt WIP para poder realizar un proceso de obtención de datos óptimo y correcto.

Este objetivo global puede refinarse en subobjetivos más concretos:

- Estudio de la estructura de la página web de los tres casos seleccionados.
- Identificación de similitudes y diferencias de la estructura de las páginas.
- Creación de *scripts* para obtener para obtención de información de las páginas.
- Realización de pruebas exhaustivas para comprobar la corrección de los datos obtenidos.

- Obtención de los catálogos de ropa de hombre y mujer de Zara, H&M y Carhartt WIP.

Este objetivo global puede subdividirse en objetivos pequeños más concretos:

- Obtención de todas las prendas de hombre y mujer del catálogo de Zara.
- Obtención de todas las prendas de hombre y mujer del catálogo de H&M.
- Obtención de todas las prendas de hombre y mujer del catálogo de Carhartt WIP.

- Búsqueda y ordenación de los resultados mostrados por la aplicación web según distintos criterios.

Este objetivo se subdivide en objetivos menores:

- Filtrado de resultados por tipo de prenda.
- Filtrado de resultados por color.
- Filtrado de resultados por género.

⁷ En español, Trabajo En Proceso, se refiere a la submarca Carhartt WIP de Carhartt, centrada en ropa más juvenil pero de muy alta calidad inspirada en el mundo de los grafitis y la cultura urbana.

- Filtrado de resultados por material de composición.
 - Filtrado de resultados por marca.
 - Ordenación de resultados de menor a mayor precio y viceversa.
 - Posibilidad de utilizar varios criterios de filtrado a la vez.
- Gestión de usuarios.

2.2. Objetivos personales

Las motivaciones personales han sido importantes a la hora de realizar este trabajo y son las siguientes. Aprender a utilizar un lenguaje puntero hoy en día como es Python de forma autodidacta, la realización de un proyecto realista íntegramente desde el principio y aprender y profundizar sobre la programación web y el funcionamiento de las mismas.

Estos objetivos han sido claves a la hora de elección del tema y las tecnologías usadas. Sobre todo, el primero, dado que en el mundo de la informática existen una enorme cantidad de contenidos gratuitos por todo Internet para poder aprender, el embarcarse en un proyecto de media escala de forma autodidacta con diferentes tecnologías nuevas es algo relevante para ponerse a prueba.

3. CONCEPTOS TEÓRICOS

En este apartado se expondrán conceptos claves para la realización de este proyecto. Se empezarán a tratar los conceptos de API y el funcionamiento básico de un sitio web, que son claves de entender para el tercer punto sobre el que gira el proyecto entero, el *web scraping*. Dentro de este apartado se tratarán las dos partes que lo componen, el *parsing* y el *crawling*, en las que se estudiarán con cierta profundidad los aspectos específicos de las mismas que se han usado en el proyecto. Por último, se tratará el concepto de *framework* y se centrará el foco en Scrapy y su funcionamiento, clave a la hora de la ejecución del *web scraping*.

3.1. API

El término API es una interfaz de programación de aplicaciones. Una interfaz es una capa que conecta dos sistemas y una API es una interfaz que conecta distintas aplicaciones para que comparten información. Suelen ser como una caja negra, se conoce su funcionalidad, pero no su funcionamiento interno. Un ejemplo de una especie de API serían los métodos públicos de una clase en Programación Orientada a Objetos, funciones conocidas de una clase que se pueden llamar para realizar una tarea.

Al igual que los métodos de las clases, según quién pueda usarla, una API puede ser pública, privada o funcionar bajo contrato de socios:

- Privada: solo se pueden usar internamente o por usuarios que han sido autorizados para ello. Suelen ser usadas en el ámbito de una empresa de forma interna.
- De socios: se comparten con socios empresariales específicos normalmente bajo pago de uso y normalmente si se da una colaboración entre empresas.
- Pública: todo el mundo tiene acceso a dicha API, lo que permite que terceros puedan desarrollar aplicaciones para interactuar con dicha API.



Ilustración 1 - Explicación de una API

3.1.1. API REST

Se trata de una API que utiliza la arquitectura para servicios web REST⁸ (*RepresEntational State Transfer*), que este a su vez utiliza el famoso protocolo HTTP⁹ (*HyperText Transfer Protocol*) para comunicarse ya que la gran mayoría de las veces los programas que hacen uso de las APIs se encuentran en puntos distintos de Internet.

Las operaciones más comunes del protocolo HTTP son las siguientes:

- GET: hace una petición de lectura de un recurso.
- POST: hace un envío de un dato al servidor. Los datos enviados van incluidos en el cuerpo de la petición.
- PUT: reemplaza el recurso existente en el servidor por el provisto en la petición.
- DELETE: se elimina el recurso especificado.

Al usar HTTP, las direcciones que tienen las APIs son como las que tienen las páginas webs. Un ejemplo será la dirección de la siguiente API de Spotify¹⁰ que permite obtener un objeto Playlist https://api.spotify.com/v1/users/{user_id}/playlists/{playlist_id} siendo *user_id* el identificador de usuario de Spotify y *playlist_id* el identificador de una lista de reproducción.

3.2. Funcionamiento básico de un sitio web

Para realizar *web scraping* de forma correcta es necesario primero conocer cómo funciona de forma básica una página web, qué ocurre desde que ponemos en el navegador www.zara.com, hasta que vemos la web en cuestión.

1. Se escribe la URL en el navegador, www.zara.com. Esta parte www.zara.com es la que se usa para encontrar el servidor anfitrión de dicha página web gracias a los servidores DNS¹¹ (*Domain Name System*).
2. El servidor de Zara, una vez recibe nuestra petición devuelve un documento HTML¹² (*HyperText Markup Language*) al navegador y puede que también ficheros CSS¹³ (*Cascade Style Sheet*) y scripts de JavaScript. El servidor también

⁸ En español, Estado de Transferencia Representacional, se trata de un estilo de arquitectura para sistemas web que utiliza como protocolo de transferencia de datos HTTP. Es la arquitectura más usada para el desarrollo de APIs hoy en día.

⁹ En español, Protocolo de Transferencia de Hipertexto, es un protocolo de la capa de aplicación para transmitir datos a través de la red. Sigue el modelo clásico de cliente-servidor.

¹⁰ <https://open.spotify.com/>

¹¹ En español, Sistema de Nombres de Dominio, es un sistema jerárquico de servidores especiales cuya función es la de traducir una URL a una IP de un servidor existente.

¹² En español, Lenguaje de Marcado de Hipertexto, se trata de un lenguaje para diseñar el contenido y la estructura de una página web de forma que cuando sea renderizado sea visible dicho contenido para los humanos.

¹³ En español, Hoja de Cascada de Estilo, es el lenguaje que se usa para estilizar un código HTML.

podría devolver otros formatos como XML¹⁴ (*eXtensible Markup Language*) o JSON¹⁵ (*JavaScript Object Notation*).

3. Si viene indicado en el HTML dichos *scripts* se ejecutan para distintas tareas como puede ser embellecer el documento. En este caso nos centraremos en la tarea de carga dinámica de contenido, que se realiza mediante peticiones a una API para cargar más contenido y añadirlo al HTML.
4. Este documento HTML con la información incluida de forma dinámica es transformado en una representación del mismo en forma de árbol, también llamado DOM¹⁶ (*Document Object Model*).
5. Esta representación se presenta por pantalla tal cuál la vemos nosotros.

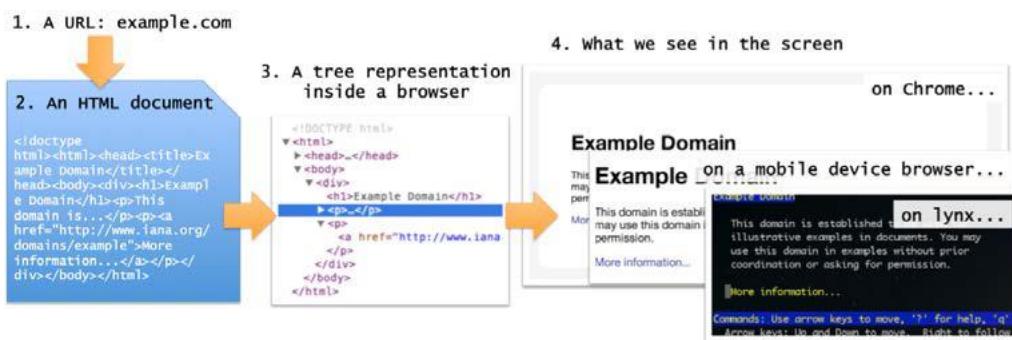


Ilustración 2 - Secuencia básica de la evolución de un documento HTML

3.2.1. La URL

La URL¹⁷ (*Uniform Resource Locator*) tiene dos partes principales, la primera se usa para localizar el servidor anfitrión de dicha página web con ayuda de los servidores DNS y se corresponde con una dirección IP¹⁸ (*Internet Protocol*) y la segunda para ver qué recurso se le pide a ese servidor, puede ser una imagen, un documento, etc. Dada la siguiente URL, <https://www.zara.com/es/es/pantalon-cargo-relaxed-fit-p05520310.html> la primera parte se corresponde con www.zara.com cuya IP equivalente es 23.58.159.186 por lo que la URL podría escribirse como <https://23.58.159.186/es/es/pantalon-cargo-relaxed-fit-p05520310.html>. En este caso la segunda parte se corresponde con un documento HTML que muestra al usuario un pantalón.

¹⁴ En español, Lenguaje Extensible de Marcado, es un lenguaje cuya finalidad es la de transmitir datos a través de Internet.

¹⁵ En español, Notación de Objetos de JavaScript, es una representación en texto plano de datos estructurados basados en la sintaxis de JavaScript.

¹⁶ En español, Modelo de Objetos del Documento, es una transformación en árbol de un documento en HTML. Se detallará con mayor profundidad en “3.2.4 DOM, la representación en árbol”.

¹⁷ En español, Localizador Uniforme de Recursos, es una dirección para localizar en Internet un recurso de forma única.

¹⁸ En español, Protocolo de Internet, se trata de un protocolo que especifica cómo localizar dispositivos en Internet gracias a unas direcciones llamadas direcciones IP.

3.2.2. El documento HTML

Dada la URL, el servidor reconoce el recurso que se pide, que en este caso se enfocará en documentos HTML, y se lo devuelve al navegador. Hoy en día, además de HTML se incluyen ficheros de muchos más formatos para complementar el contenido de este HTML y otras razones, pero se centrará la atención en el HTML y en archivos en JavaScript, estos se tratarán en el siguiente apartado. Un documento HTML es un archivo jerarquizado específico para los navegadores y especialmente hecho para que estos interpreten la información y la muestren de forma que los humanos la podamos ver.

3.2.3. Carga dinámica de contenido

JavaScript es un lenguaje de interpretación de contenido orientado a objetos que se suele usar para programación del lado del cliente, es decir, se ejecuta en el equipo del cliente, para hacer el contenido de una web más interactivo. Suele realizar distintas funciones, como puede animaciones, botones, control de elementos multimedia o cargado dinámico de contenido.

El cargado dinámico de contenido se refiere a incluir información nueva en el HTML original cuando sea necesario en vez de todo a la vez. Por ejemplo, en Twitter, al llegar al final de una página de Tweets, se amplía dicha página y aparecen más, eso es un cargado dinámico de contenido. Para este cargado dinámico, se hace una llamada a una API para cargar que envíe el contenido, se suele utilizar el formato JSON.

3.2.4. DOM, la representación en árbol

Se ha mencionado anteriormente que los documentos HTML son jerarquizados. Esto quiere decir que hay elementos dentro de otros con una relación padre-hijo. En el navegador se realiza paso de un archivo de texto en HTML a la representación en árbol DOM. En “Ilustración 4”, viene indicado un ejemplo de cómo se representa de forma gráfica este árbol para un código sencillo en HTML. Esta representación es clave para poder localizar elementos en un HTML y por ende clave para poder realizar *web scraping*. Una forma sencilla de acceder a esta representación es estar situado en una página web y hacer click derecho y en el menú pulsar la opción de “Inspeccionar Elemento” o “Inspect” como se ve en “Ilustración 3”.

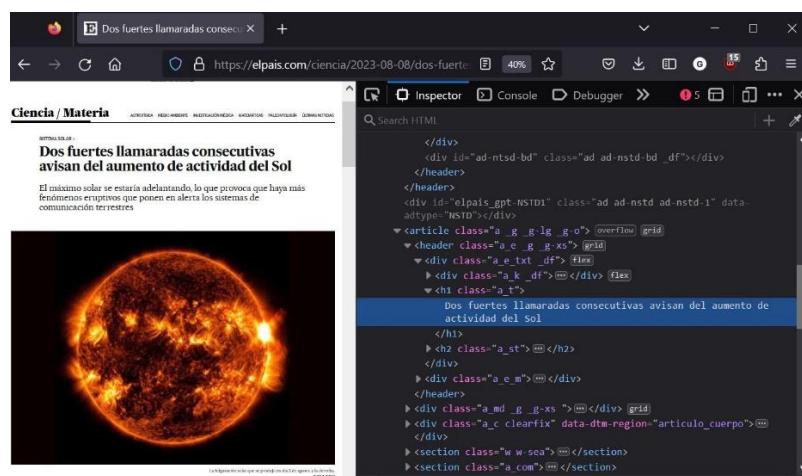


Ilustración 3 - Herramienta Inspeccionar

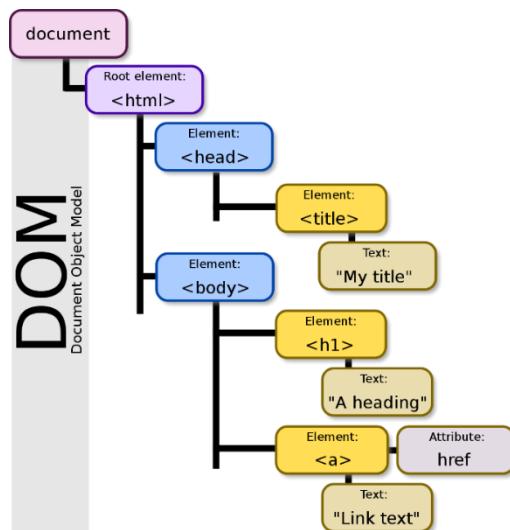


Ilustración 4 - Representación en DOM

3.3. Web Scraping

El *web scraping* se refiere a la extracción automática de datos de un sitio web. Se suelen obtener grandes cantidades de datos y, clásicamente, suelen estar en formatos no estructurados como HTML y resulta necesario, en ese caso, convertirlos a datos estructurados como una base de datos.

El *web scraping* se divide en dos partes fundamentales, el *parsing* y el *crawling*. El *parsing* es el proceso de obtención de los datos ya sea de un formato no estructurado como un HTML o de un formato estructurado como un archivo JSON. El *crawling* es el proceso de recorrer distintas webs o distintas partes de una web. Cuando se combinan dichos procesos obtenemos un programa que recorre la web (*crawling*) en busca de HTMLs de productos que los que podamos obtener sus datos (*parsing*).

Como se ha mencionado anteriormente, la forma más clásica de *web scraping* es la de tratar con formatos no estructurados como HTML que además coincide con el método más común de *scraping* del proyecto, por lo que nos centraremos en este aspecto a continuación.

Dada la siguiente captura de una noticia del periódico El País y su correspondiente código HTML, ¿cómo podemos hacer que un programa obtenga el título de la noticia?

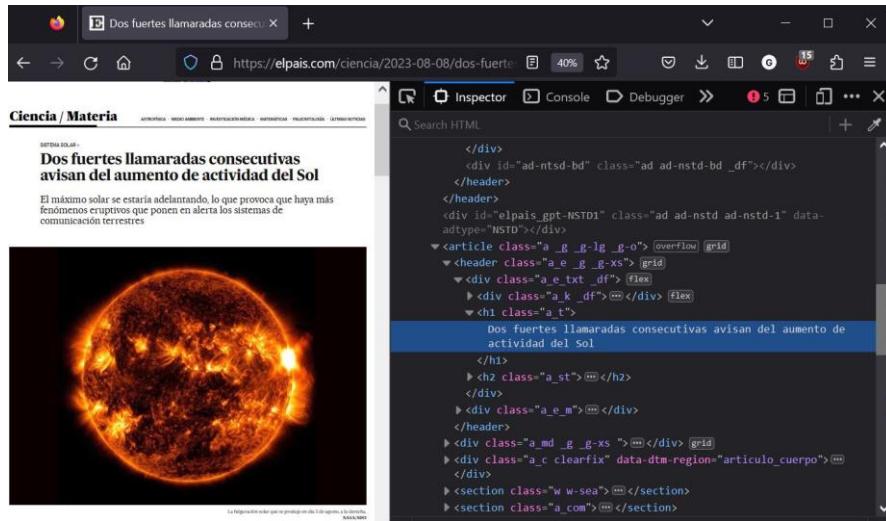


Ilustración 5 - Noticia ejemplo para XPath

Para ello se usará una herramienta llamada XPath¹⁹, y posteriormente, tras obtener dicha cadena y, si se hace necesario, se usarán expresiones regulares para realizar tareas de limpieza en dicha cadena para clasificarla o para modificarla.

3.3.1. Parsing

En este apartado nos centraremos en cómo se realiza el *parsing* o análisis sintáctico de las expresiones. Como se ha mencionado anteriormente, el *parsing* se refiere al proceso de obtención de datos de un archivo no estructurado como HTML y para poder seleccionar los elementos necesarios se usará un lenguaje llamado XPath. Además, muchas veces requiere de una limpieza de esos datos porque puede que todo el dato obtenido pueda no ser necesario y para ello se utilizarán expresiones regulares.

3.3.1.1. XPath

XPath²⁰ (*XML Path Language*) es un lenguaje diseñado para realizar búsquedas o modificaciones en documentos XML. Fue definido por la *World Wide Web Consortium*²¹ (W3C) en 1999 y también se usa para realizar las mismas funciones en lenguajes como HTML.

Dado que los archivos HTML se organizan de forma jerárquica, XPath juega con ese factor y utiliza una sintaxis de tipo ruta, como las rutas de archivos en un sistema de archivos, por ejemplo “/home/user/Desktop/file.txt” pero aplicado al documento, como “/html/body/div/p”, para identificar y navegar por los nodos de un documento XML/HTML.

A continuación, haremos una breve introducción teórica a los elementos más relevantes de XPath necesarios para comprender este proyecto.

¹⁹ *XML Path Language*, se trata de un lenguaje para recorrer documentos XML. Se tratará a continuación en el punto “3.3.1.1 XPath”.

²⁰ En español, Lenguaje de Dirección XML.

²¹ En español, Consorcio de la Web Global, es una organización líder en la creación de estándares para distintos aspectos de Internet.

Nodos

En XPath hay siete tipos de nodos, los elementos, los atributos, el texto, los espacios de nombres, las instrucciones de procesado, los comentarios y los nodos raíz. Nos centraremos especialmente en los elementos, atributos, texto y nodos raíz pues son los más comunes y los que se han utilizado en este proyecto.

Los documentos XML son tratados como árboles de nodos y todos tienen un elemento raíz del que parten el resto de los nodos.

En la “Ilustración 6” se muestra un código de ejemplo XML en el que se explicarán los distintos tipos de nodos.

```

<?xml version="1.0" encoding="UTF-8"?>

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>

```

Ilustración 6 - Ejemplo de código XML

- Nodo elemento: “<autor>J K. Rowling </autor>”. Un nodo elemento puede contener otros nodos elemento al igual que nodos atributo y texto.
- Nodo texto: “J K. Rowling” o “29.99”. Son cadenas de texto plano que suelen representar el valor que toma un nodo elemento.
- Nodo atributo: “Lang=”en””. Son variables con un valor asignado, siempre están dentro de un nodo elemento y proporcionan una característica que tiene este.
- Nodo raíz: “<bookstore>”. Es el nodo bajo el que se encuentran todos los nodos del documento XML. En HTML el nodo raíz siempre es “<html></html>”.

Los nodos tienen distintas relaciones entre ellos:

- Padre: todo nodo que contiene a otro. En “Ilustración 6”, “<book>” es padre de “<title>”. Todo nodo tiene padre menos el nodo raíz.
- Hijo: todo nodo contenido en otro. En “Ilustración 6”, “<title>” es hijo de “<book>” y “Lang=”en”” es hijo de “<title>”. Solo pueden tener hijos los nodos elementos, que pueden tener cero, uno o más hijos.
- Hermano: nodos que tienen el mismo padre. En “Ilustración 6”, los nodos “<title>”, “<author>”, “<year>”, “<price>” son nodos hermanos, pues todos comparten padre que es “<book>”

- Ancestro: el padre de un nodo, el padre del padre, etc. En “Ilustración 6”, los nodos ancestros del texto “Harry Potter” son “<bookstore>”, “<book>” y “<title>”.
- Descendiente: el hijo de un nodo, el hijo del hijo, etc. En “Ilustración 6”, los nodos descendientes de “<bookstore>” son “book”, “<title>”, “<author>”, “<year>”, “<price>”, “Harry Potter”, “J K. Rowling”, etc.

Sintaxis

Una vez entendido el concepto de nodo y sus tipos, en este apartado de XPath vamos a ver cómo es la sintaxis básica de las expresiones de este lenguaje.

Como se ha mencionado anteriormente, todos los elementos que hay en un documento XML o HTML son nodos, por lo tanto es importante saber cómo seleccionar nodos y predicados para especificar entre varios nodos por valores que contengan.

Expresiones para la selección de nodos:

- *Nombre_de_nodo*: selecciona todos los nodos con el nombre “*Nombre_de_nodo*”.
- `/`: selecciona un nodo desde el nodo raíz.
- `//`: selecciona todos los nodos desde el nodo actual que concuerden con el seleccionado independientemente de donde se encuentre.
- `.`: selecciona el nodo actual.
- `..`: selecciona el nodo padre del nodo actual.
- `@`: selecciona atributos.
- `*`: selecciona cualquier nodo.

En la “Ilustración 6”:

- `/bookstore/title`: selecciona al nodo “<title>” y a todos sus descendientes.
- `//book`: selecciona a todos los nodos “<book>” independientemente de dónde se encuentren en el archivo.

Los predicados siempre van entre corchetes, a continuación se mostrarán los más importantes:

- `/bookstore/book[1]`: se selecciona el primer nodo “<book>” hijo de “<bookstore>”. En XPath a diferencia de otros lenguajes de programación, se cuenta a partir del 1 no del 0, por lo que [1] indica el primer elemento.
- `//book[last()]`: se selecciona el último nodo “<book>” de todo el documento.
- `//title[@lang="en"]`: se selecciona todos los nodos “<title>” cuyo atributo “Lang” tenga el valor “en”.
- `//price[contains(text(), "9")]`: se selecciona todos los nodos que contengan el número 9 en su nodo texto.

Ejemplos

Se mostrarán ejemplos más realistas sobre la “Ilustración 6”:

- Tarea: seleccionar el primer nodo libro que contenga “Rowling” en su autor.
Solución: “`//book[contains(text(), “Rowling”)][1]`”
- Tarea: seleccionar el nombre de todos los nodo libro que cuyo idioma sea inglés (`leng=en`). Solución: “`//book/title[@lang=“en”]/text()`”

Volviendo ahora a la cuestión planteada para un caso real sobre la “Ilustración 5” para obtener el titular de una noticia de un periódico, el código XPath necesario sería `“//*[@class=“a_t”]/text()”`. Si nos fijamos, se trata de un texto hijo de un elemento `<h1>` cuya clase es “`a_t`”. A pesar de seleccionar el texto cualquier nodo cuya clase sea “`a_t`”, por lo que podrían salir más cadenas de nodos distintos, las clases suelen ser elementos únicos de ciertas porciones de texto especiales en un HTML, es decir, esa clase define un formato específico para un titular de noticia, por lo que es de esperar que si en el HTML que estamos analizando sólo hay una noticia y por ende, sólo un titular, esa expresión de XPath anterior sólo nos devolverá una única cadena que será el titular de la noticia.

3.3.1.2. Expresiones regulares

Muchas veces es necesario, tras obtener los datos mediante análisis sintáctico, clasificarlos o realizar tareas de limpieza como eliminación de distintos caracteres o modificación de otros y para ello es de un uso muy común el de las expresiones regulares.

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto.

En el caso de este proyecto, al ser realizado en gran parte en Python, se ha utilizado la biblioteca o módulo `re`, una biblioteca estándar de este lenguaje que se usa para manipular y crear expresiones regulares.

3.3.2. Crawling

Como se indicó a inicios del apartado “3.3 Web Scraping”, el *crawling* se refiere a la parte del *web scraping* que consiste en recorrer la web en busca de HTMLs a los que hacer *parsing*.

3.3.2.1. Archivo robots.txt

Hoy por hoy, muchos programas recorren todo Internet realizando distintas tareas como puede ser *web scraping* o simplemente haciendo *crawling* e indexando todas las páginas web que hay en Internet, como puede ser Google. Esto hace que un servidor de una página web se pueda saturar rápidamente, dado que estos programas son capaces de enviar una cantidad de solicitudes a páginas web por segundo mucho mayores que las de un humano además de ser un tráfico que no interesa, pues no interacciona con ellas ni realiza compras. Ante esto, se creó un estándar llamado *Robots Exclusion Protocol*²² por el cuál las páginas

²² En español, Protocolo de Exclusión de Robots, se trata de un estándar implementado por páginas web para indicar a programas que las recorran qué sitios tienen permiso para visitar.

web indicaban en un fichero llamado “robots.txt” un conjunto de normas a tener en cuenta por todos aquellos programas que decidieran recorrer su página web. Dichas normas son simplemente una sugerencia, pero es un buen hábito a seguir, pues así respetas a otros internautas, evitas ser bloqueado y ahorras tiempo evitando entrar en sitios de la página web que no suelen ser muy relevantes. Dicho archivo puede encontrarse en casi cualquier página web siguiendo este esquema de URL, <https://www.google.com/robots.txt>, siempre después de la dirección para los servidores DNS.

Para el siguiente fragmento de código analizaremos un archivo robots.txt de la página web www.twitch.tv, una plataforma especializada en vídeos en directo.

```
User-Agent: *
Allow: /
Allow: /directory
Allow: /directory/all
Allow: /directory/*
Allow: /videos/week
Allow: /.well-known/assetlinks.json
Disallow: /admin/*
Disallow: /email-unsubscribe/
Disallow: /login$
Disallow: /message/*
Disallow: /signup$
Disallow: /user/*
```

Sitemap: https://www.twitch.tv/sitemapv2_index.xml.gz

Cada directiva `User-Agent:` indica un bloque, lo que quiere decir que para el *User-Agent*²³ especificado se aplican unas normas hasta que se especifique otra directiva del mismo tipo o `Sitemap:`, el asterisco indica que vale para cualquier *User-Agent*. Las directivas `Allow:` y `Disallow:` indican a qué direcciones de la página se permite (*Allow*) y a cuáles no se permite (*Disallow*) entrar al *User-Agent*, especificado. En este caso se permite entrar a todos los sitios menos a los indicados por las directivas *Disallow*.

La directiva `Sitemap:` especifica un archivo *Sitemap* que se explicará a continuación en el apartado “3.3.2.2 Sitemaps”.

3.3.2.2. Sitemaps

Como se ha visto en el apartado “3.3.2.1 Archivo robots.txt”, hay una directiva que indica a un archivo llamado *sitemap*. Un *sitemap* es un archivo en formato XML donde se provee información de la dirección páginas, imágenes y otros videos de la página web en la que se encuentra y que ayuda a programas y buscadores a recorrer el sitio web de forma más eficiente. Está definido por un el protocolo *Sitemap protocol* que define los tipos de etiquetas de XML que puede contener el documento, sus posibles valores y su obligatoriedad.

Normalmente si el sitio web es muy grande o está disponible en varios idiomas en vez de haber un único *sitemap* hay varios en estructura jerárquica. En el archivo robots.txt se

²³ En español, Agente Usuario, se trata de una cadena de texto agregada en cada petición HTTP que identifica al navegador o sistema operativo entre otros, del navegador que realiza la petición.

apunta a un *sitemap* llamado índice o *index* en inglés. Dicho archivo contiene la dirección de *sitemap* en cada idioma disponible.

Este es el formato que tiene un *sitemap*:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.example.net/?id=who</loc>
    <lastmod>2009-09-22</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.8</priority>
  </url>
  <url>
    <loc>http://www.example.net/?id=what</loc>
    <lastmod>2009-09-22</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.5</priority>
  </url>
  <url>
    <loc>http://www.example.net/?id=how</loc>
    <lastmod>2009-09-22</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.5</priority>
  </url>
</urlset>
```

Las etiquetas de XML más relevantes en un *sitemap* son las siguientes:

- “<loc>”: indica la URL de un elemento. Es un campo obligatorio.
- “<changefreq>”: indica la frecuencia de cambio que tiene esa URL en el *sitemap*. Este campo puede tener valores de *always*, *hourly*, *daily*, *weekly*, *monthly*, *yearly* y *never*, que son respectivamente, siempre, a cada hora, diariamente, semanalmente, mensualmente, anualmente y nunca.
- “<lastmod>”: indica fecha del último cambio. No se ha usado en el proyecto.
- “<priority>”: indica a mayor valor entre 0 y 1, mayor prioridad. Se usa para los buscadores que indexan páginas de forma que las páginas con mayor prioridad son más importantes y aparecen más arriba al hacer una búsqueda. No se ha usado en el proyecto.

3.3.2.3. *Crawling*: tipos y problemas

Tipos de *crawling*

Hay principalmente tres tipos de *crawling*:

- Mediante *sitemap*: es un método sencillo, se recorre el *sitemap* URL a URL. La principal ventaja de este método, como ya se ha mencionado, es que es el más sencillo y la gran mayoría de enlaces que tienen cierta relevancia a la hora de hacer *web scraping* se encuentran en este archivo. La principal desventaja es que muchas veces los *sitemap* no están actualizados del todo y quedan secciones de las webs sin estar incluidas en el archivo, además de haber distintos tipos de enlaces, cada uno con una necesidad de tratamiento diferente.

- Mediante uso de API REST del sitio web: como se ha mencionado en el apartado “3.2.3 Carga dinámica de contenido”, hoy en día la gran mayoría de páginas webs cargan contenido de forma dinámica, así que es posible usar eso en favor del programador para hacer *web scraping* de forma eficiente, obteniendo datos en formatos estructurados como JSON y estando más actualizados. La principal desventaja es que muchas webs utilizan métodos de autenticación al usar APIs o están bien cubiertas, por lo que suele ser más complejo que el resto de los métodos.
- Siguiendo todos las URLs de la página web: es el método más sencillo y a la vez el que más fuerza bruta requiere. Se recorre prácticamente todos los enlaces de la página web por lo que consume mucho tiempo y gran parte de los enlaces a recorrer no son útiles. Además, ciertas páginas ponen URLs específicas para que programas que recorran la página web caigan y sean atrapados. No se usará en el proyecto.

Problemas del crawling

Dado que la velocidad a la que un programa puede recorrer una página web es muy superior a la que un humano tiene, puede llegar a dar problemas como una sobrecarga a un servidor, para evitar estos problemas, existen mecanismos de protección contra *web scraping*, que pueden denegar el acceso temporal o permanentemente a la página web. Cuando cualquier mecanismo de este tipo te prohíbe el acceso, en vez de devolverte el HTML de la URL solicitada, con un código HTTP de 200, que quiere decir que todo ha ido correcto, te devuelve un código HTTP de entre 400 y 500. Los más comunes son *403 Forbidden*²⁴ y en ciertos casos *410 Gone*²⁵. Otros motivos por los que existen estas capas defensivas son que un programa no va ni a ver publicidad que hay en la web ni a realizar compras, por lo que es una visita completamente vacía e incluso negativa en términos de ganancias económicas, pues se consumen recursos sin nada a cambio.

Es natural si se dispone de un programa que realice *web scraping* tratar de evitar estos mecanismos. Todas las soluciones tratan de hacer que el comportamiento del programa se asimile lo más posible al tráfico realizado por humanos y así camuflarse y no ser detectado. Algunas medidas que pueden tomarse para evitar estos mecanismos son las siguientes:

- Tener en cuenta robots.txt: es importante que al acceder a una página web sigamos las normas recomendadas, dado que, si no, nuestro comportamiento va a ser perjudicial y habrá mayores posibilidades de una prohibición de acceso.
- Modificar la cabecera HTTP: “Ilustración 7” muestra la cabecera de HTTP que tiene Scrapy por defecto, mientras que “Ilustración 8” muestra la cabecera HTTP que tiene Google Chrome desde un macOS. Vista una diferencia tan clara, es fácil determinar cuál de las dos ilustraciones se trata de tráfico humano y cuál no. Además, cobra especial importancia la clave *User-Agent*. Es importante poner los

²⁴ En español, 403 Prohibido, es un código HTTP que indica que se prohíbe el acceso al recurso solicitado. En el caso de *web scraping* suele ser porque se ha detectado que el emisor de la petición es un programa o que la ratio de peticiones es demasiado alta.

²⁵ En español, 410 Terminado/Acabado, es un código HTTP que indica que previamente el recurso existía pero que ya no se encuentra disponible. Suele ocurrir cuando se solicita una URL antigua.

distintos pares clave-valor de forma correcta en la cabecera, ya que el orden que siguen varía en los distintos navegadores.

```
Accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'
Accept-Language: 'en'
User-Agent: 'Scrapy/2.9 (+https://scrapy.org)'
```

Ilustración 7 - Cabecera HTTP por defecto de Scrapy

```
Connection: 'keep-alive'
Cache-Control: 'max-age=0'
sec-ch-ua: '" Not A;Brand";v="99", "Chromium";v="99", "Google Chrome";v="99"'
sec-ch-ua-mobile: '?0'
sec-ch-ua-platform: "macOS"
Upgrade-Insecure-Requests: 1
User-Agent: 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 ...
Accept: 'text/html,application/xhtml+xml,application/xml;q=0.9,
Sec-Fetch-Site: 'none'
Sec-Fetch-Mode: 'navigate'
Sec-Fetch-User: '?1'
Sec-Fetch-Dest: 'document'
Accept-Encoding: 'gzip, deflate, br'
Accept-Language: 'en-GB,en-US;q=0.9,en;q=0.8'
```

Ilustración 8 - Cabecera HTTP de Google Chrome desde un macOS

- Reducir la ratio de peticiones a la página web: muchas veces en el archivo robots.txt no pone cuál es la ratio recomendada para hacer *web scraping* por lo que entonces tener una velocidad de acceso razonable facilitará que no haya bloqueos además de camuflar el tráfico como si de un humano se tratase. Normalmente, una petición por segundo ya se considera una velocidad segura, pero depende del servidor.
- Aleatorizar el envío de peticiones a la página web: si alguien está mandando peticiones a un servidor de forma muy periódica es muy probable que se trate de un programa, ya que los humanos no seguimos un patrón a la hora de acceder a una página web. Aleatorizar el envío de peticiones hace que sea más difícil detectar si se trata de un humano o no.
- Contratar un proxy con opción de rotación de IPs: el gran problema de los puntos anteriores es que, a pesar de poder evitar una prohibición de acceso, algunos como reducir la ratio de peticiones suponen una bajada de rendimiento y se tarde una gran cantidad de tiempo en completar el proceso. La solución a esto es la contratación de un *proxy* con rotación de IPs.

Un *proxy* se trata de un servidor o servicio que permite realizar una conexión a Internet desde diferentes partes del mundo. Algunos servicios que ofrecen proxies dan también un servicio orientado a *web scraping* que consiste en ofrecer un conjunto de IPs por los que se pueden enviar peticiones. Esto aplicado al *web scraping* hace que, a pesar de enviar una gran cantidad de peticiones a un servidor, este ve que son realizadas desde diferentes IPs por lo que parece que son distintos usuarios los que solicitan páginas web en vez de uno, lo que hace que no haya una bajada de rendimiento. Como desventaja, un buen servicio de este tipo cuesta

dinero además de que ciertos sistemas de protección ya tienen controladas ciertas direcciones IPs que saben que pertenecen a *proxies* y en caso de usarse, prohíben el acceso a dicha IP.

3.4. Framework

Un *framework*²⁶ es un conjunto de reglas y convenciones que se usan para desarrollar software de forma más eficiente y rápida. En un lenguaje un poco más llano se trataría de una serie de bibliotecas unificadas que puede usar el programador.

En este proyecto se ha usado Scrapy como *framework* para *web scraping* y se va a explicar a continuación porque tiene un sistema de funcionamiento algo complejo que es clave entender para comprender cómo se han realizado las distintas etapas del *web scraping*.

3.4.1. Scrapy

Scrapy es un web *framework* gratuito de Python orientado a *web scraping* nacido en 2008. Es de código abierto, pero ha sido desarrollado principalmente por la compañía Zyte, anteriormente Scrapinghub. Está basado en otro *framework* de Python llamado Twisted, que está basado en eventos y que se centra en la creación de código asíncrono.

Se ha elegido este *framework* porque se encuentra a la vanguardia en las herramientas de *web scraping* por proveer un alto rendimiento y una gran cantidad de herramientas para personalizar el código al igual que una comunidad relativamente amplia.

3.4.1.1. Conceptos básicos

Spiders²⁷ o arañas

Las arañas son clases que definen cómo se va a hacer *web scraping* en un sitio web, es decir, cómo se va a recorrer el sitio web y cómo se van a obtener los datos de sus páginas.

El ciclo de funcionamiento de una araña es el siguiente:

1. Se genera una petición con inicial mediante un objeto *Request* (petición). La petición inicial se realiza con las direcciones dadas al atributo *start_urls*, en el que se indican las URLs que se quieren recorrer primeramente. El método *callback*²⁸ es un método llamado *parse()* que se activa cuando se recibe una respuesta por parte del servidor a cada petición enviada.
2. En el método de retrollamada, se analiza sintácticamente el HTML de la respuesta provista por la página web a la petición mandada como parámetro de entrada y devuelve un objeto Item. Es importante decir que el HTML que lleva consigo la respuesta recibida no tiene contenido cargado dinámicamente, pues no se ejecuta código JavaScript. Para obtener contenido cargado dinámicamente se tienen que hacer las llamadas de forma manual. Es en este método donde se usan selectores,

²⁶ En español, marco o entorno de trabajo

²⁷ En español, arañas

²⁸ En español, retrollamado, es una función que se activa cuando se da cierto evento en el sistema.

es decir, expresiones que usan XPath en este caso, para obtener los datos interesantes de la respuesta HTML.

3. Finalmente, los ítems devueltos por la araña pasarán por otras fases de procesado explicadas más tarde en este apartado.

Item

Las arañas devuelven los datos extraídos en forma de objetos Item, que es una clase con un formato de pares clave-valor y almacenan los datos extraídos.

Los atributos se declaran con una clase llamada Field() que es específica para guardar datos.

```
class Producto(scrapy.Item):  
    nombre = scrapy.Field()  
    precio = scrapy.Field()  
    stock = scrapy.Field()
```

Todas estas arañas e ítems declarados se enmarcan bajo un proyecto de Scrapy, de forma que si queremos hacer *scraping* de cuatro periódicos como son El País, El Mundo, ABC y Libertad Digital crearíamos un proyecto que podemos llamar “noticias” y una araña para cada periódico, AranaElPais, AranaElMundo, AranaABC y AranaLibertadDigital. Y una clase Item noticia como se ha mencionado anteriormente. En “5.2.1 Web Scraping” se detallará cómo se ha definido en el proyecto.

Configuración

La configuración de Scrapy permite personalizar el comportamiento de cualquier componente de Scrapy, desde las arañas hasta ciertos componentes de la arquitectura. Funciona gracias a un conjunto de pares clave-valor, de forma que a cierto parámetro de configuración se le asigna un valor.

La configuración se establece en varios niveles de prioridad, en este apartado veremos los dos más importantes, con orden decreciente de prioridad, que son la configuración de las arañas y la de proyecto:

- Configuración de araña: las arañas pueden definir su propia configuración a través del atributo *custom_setting* como se indica en “Ilustración 9”. Esto sobrecargará la configuración del proyecto, pues tiene mayor prioridad.

```
|class HmSpider(scrapy.Spider):  
|    name = "ache"  
  
|    custom_settings = {  
|        "IMAGES_STORE": 'imagenes/hm',  
|        "DOWNLOAD_DELAY": 2,  
|        "ROBOTSTXT_OBEY": True,  
|    }
```

Ilustración 9 - Configuración de arañas en Scrapy

- **Configuración de proyecto:** la configuración a nivel de proyecto es el método de más común de modificar la configuración para un proyecto de Scrapy, a este nivel se establecerán la gran mayoría de ajustes. Para logralo, hay que modificar el archivo *settings.py* como se indica en “Ilustración 10”.

```
settings.py*  ✘
1 # Scrapy settings for prendas project
2
3 BOT_NAME = "prendas"
4
5 SPIDER_MODULES = ["prendas.spiders"]
6 NEWSPIDER_MODULE = "prendas.spiders"
7
8 USER_AGENT = "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0"
9
10 ROBOTSTXT_OBEY = True
11
12 CONCURRENT_REQUESTS = 2
13
14 COOKIES_ENABLED = True
--
```

Ilustración 10 - Configuración de proyecto en Scrapy

3.4.1.2. Arquitectura

El flujo de datos en Scrapy es controlado por una clase llamada *Engine*²⁹ y se encuentra detallado en “Ilustración 11”. Los pasos son los siguientes:

1. La clase *Engine* obtiene los objetos *Request*³⁰ para recorrer la página web de una araña.
2. La clase *Engine* organiza los objetos *Request* iniciales y se las entrega a la clase *Scheduler*³¹ solicita el resto de objetos *Request* no iniciales.
3. La clase *Scheduler* devuelve el resto de peticiones a la clase *Engine*.
4. La clase *Engine* envía las peticiones a la clase *Downloader*³² pero pasando primero por las clases *Downloader Middlewares*³³.
5. Despues de que la clase *Downloader* obtiene una respuesta, que es el código HTML de la página solicitada, genera un objeto *Response*³⁴ con los datos de la respuesta pasándosela a la clase *Engine* pero siendo procesada antes de ello otra vez por las clases *Downloader Middleware*.

²⁹ En español, Mecanismo. En Scrapy la clase *Engine* se encarga de gestionar todo el flujo de datos en ejecución.

³⁰ En español, Petición, es una clase cuyos objetos representan peticiones HTTP.

³¹ En español, Planificador en español, es una clase que se encarga de poner los objetos *Request* en cola.

³² En español, Decargador, es una clase que se encarga de hacer las peticiones reales a las páginas web gracias a los datos de los objetos *Request*.

³³ En español, Agente Intermedio de Descarga, son unas clases que añaden funcionales previas al envío de peticiones de todas las arañas por parte del *Downloader*.

³⁴ En español, Respuesta, al igual que la clase *Response* representa la respuesta HTTP de un servidor ante una petición. Cada objeto *Response* tiene asociado un objeto *Request*.

6. La clase *Engine* recibe los objetos *Response* de la clase *Downloader* y los envía a las arañas para el procesado, pasando por las clases *Spider Middleware*³⁵, que son clases como las *Downloader Middleware* pero específicas para cada araña.
7. Las arañas procesan los objetos *Response* (aquí es cuando son procesados en la función de retrollamada *parse()* explicada anteriormente en este apartado) y devuelven los objetos *Item*, que son enviados a la clase *Engine* para ser procesados.
8. La clase *Engine* envía los ítems procesados a unas clases llamadas *Item Pipelines*³⁶. Estas clases son muy importantes pues sirven para procesar los ítems haciendo cosas como limpieza de parámetros, desechado de ítems con valores erróneos, descarga de imágenes en base a una URL, inserción de datos de ítems en una base de datos, etc.

Se especificará la estructura del proyecto actual y los distintos tipos de clases en “5.2.1 *Web Scraping*”.

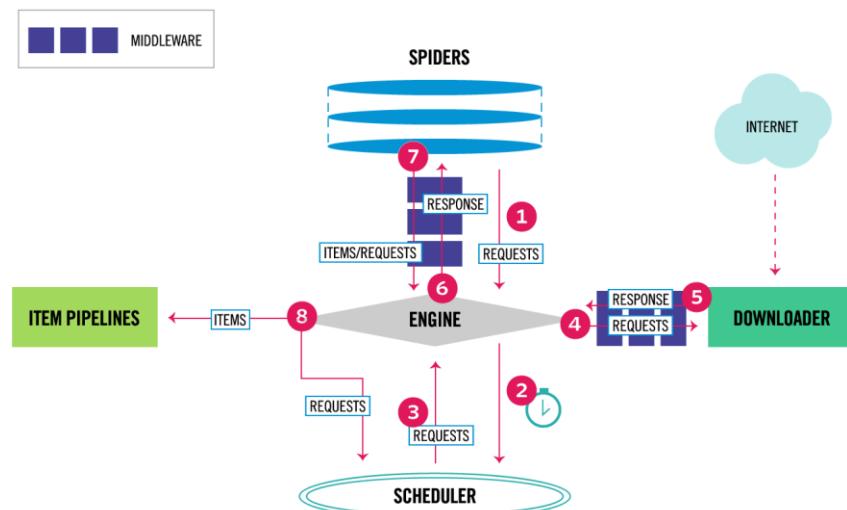


Ilustración 11 - Arquitectura de Scrapy

³⁵ En español, Agente Intermedio de la Araña, tiene una función parecida a la de *Downloader Middleware* pero específica a la araña.

³⁶ En español, Tuberías de Ítems.

4. TÉCNICAS Y HERRAMIENTAS

4.1. Herramientas y utilidades de desarrollo

4.1.1. Visual Studio Code

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Microsoft.
- Página web: <https://code.visualstudio.com/>
- Licencia de uso: licencia MIT.
- Coste: gratuito.

Visual Studio Code es un editor de código fuente de software libre y multiplataforma disponible para Windows, Linux y macOS. Algunas de sus características y funcionalidades son que puede integrar sistemas de control de versiones como Git, permite depuración de código y dispone de un gran número de extensiones para poder escribir código en una infinidad de lenguajes de programación distintos.

En el proyecto ha sido utilizado para escribir todo el código de tanto la parte de *web scraping* como del desarrollo de la web.

4.1.2. Scrapy

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Zyte.
- Página web: <https://scrapy.org>
- Licencia de uso: licencia BSD.
- Coste: gratuito.

Scrapy se trata de un *framework* escrito en Python especializado en *web scraping* alguna de las funcionales que tiene:

- Permite usar selectores CSS y expresiones XPath para extraer información de archivos como HTML/XML.
- Consola interactiva para interactuar con las respuestas de páginas webs dada una URL.
- Soporte de distintos formatos de salida de datos como JSON, CSV³⁷ (*Comma-Separated Values*) y XML.

³⁷ En español, Valores Separados por Coma, se trata de archivos de texto plano para almacenamiento de datos en los que, como su propio nombre dice, los valores se separan por comas.

En el proyecto ha sido utilizado para construir el algoritmo de búsqueda de prendas para las tres tiendas, Zara, H&M y Carhartt WIP, que incluye la búsqueda de las prendas en sí, la limpieza de los datos, la inserción de los datos en la base de datos, la descarga de imágenes de las prendas, la configuración de cómo se hace la búsqueda, que incluye el ratio de peticiones por página web, la personalización de la cabecera HTTP, la prioridad de la cola de peticiones y la definición de los *Item Pipelines* entre otros. Esta información se detallará más en “5.2.1 *Web Scraping*”.

4.1.3. MariaDB

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: MariaDB Foundation.
- Página web: <https://mariadb.com>
- Licencia de uso: GPLv2, LGPLv2.1.
- Coste: gratuito.

MariaDB es un sistema de gestión de bases de datos relacionales que nació al producirse un *fork* de MySQL con una licencia GPL menos restrictiva que la de este. Es muy compatible con MySQL porque comparten muchas órdenes, API y bibliotecas pudiendo cambiar de sistemas directamente.

En el proyecto, MariaDB es el sistema gestor de base de datos en el que se almacenan las propias bases de datos de prendas y de usuarios. Ha sido elegido por ser un sistema más robusto y rápido que MySQL y por tener una licencia menos restrictiva.

4.1.4. Flask

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Armin Ronacher.
- Página web: <https://palletsprojects.com/p/flask/>
- Licencia de uso: licencia BSD.
- Coste: gratuito.

Flask es un *framework* minimalista escrito en Python especializado en el desarrollo de aplicaciones web. Tiene un núcleo pequeño y fácil de extender. Usa jinja2 como gestor dinámico de plantillas, WSGI³⁸ (*Web Server Gateway Interface*) como especificación de conexión entre servidores web y aplicaciones web y Werkzeug como herramienta WSGI para implementar peticiones y respuestas.

En el proyecto ha sido utilizado en la parte de desarrollo de la aplicación, para desarrollar toda la web y la API de selección de imágenes. Se explicará en profundidad en “5.2.2 Aplicación web”.

³⁸ En español, Interfaz de Pasarela del Servidor Web.

4.1.5. Bootstrap

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Twitter.
- Página web: <https://getbootstrap.com/>
- Licencia de uso: licencia MIT.
- Coste: gratuito.

Bootstrap es una biblioteca multiplataforma de código abierto para diseño de aplicaciones web. Contiene todo tipo de elementos de HTML estilizados con código CSS y con código JavaScript al igual que plantillas de diseño. Está orientado únicamente a desarrollo de presentación o *front-end*³⁹.

En el proyecto ha sido utilizado como parte del apoyo al desarrollo de la parte de presentación de la página web.

4.1.6. Mozilla Firefox: herramienta para desarrolladores web

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Corporación Mozilla, Fundación Mozilla.
- Página web: <https://www.mozilla.org/es-ES/firefox/new/>
- Licencia de uso: MPL 2.0
- Coste: gratuito.

Mozilla Firefox es un navegador web multiplataforma, libre y de código abierto. Usa el motor Gecko para renderizar las páginas web. Este navegador posee una herramienta para desarrolladores web la cual está dividida en distintas pestañas, cada una con diferentes finalidades. Las más relevantes son, pestaña “*Inspector*”⁴⁰, permite analizar el DOM y su código HTML, la pestaña “*Console*”⁴¹ ejecutar comandos en una consola que tendrán efectos sobre la web objetivo como muestra la, la pestaña “*Network*”⁴², visualizar el intercambio de paquetes a nivel de aplicación entre el navegador y el servidor web objetivo como muestra, etc.

En el proyecto ha sido muy usada esta herramienta para desarrolladores web en el apartado de investigación de las webs y en el apartado de desarrollo de la aplicación en *web scraping*. Ha sido clave visualizar el DOM para formular posteriormente las expresiones XPath y probarlas en la consola y se ha dado uso de la pestaña de “*Network*”

³⁹ En español, frontal o interfaz de usuario, se denomina a la parte de diseño de una aplicación que se centra en la capa de presentación que está en contacto con el usuario.

⁴⁰ En español, Inspector, se refiere en este caso a la pestaña de Inspector de la herramienta para desarrolladores web de Mozilla Firefox

⁴¹ En español, Consola, se refiere en este caso a la pestaña de Consola de la herramienta para desarrolladores web de Mozilla Firefox

⁴² En español, Red, se refiere en este caso a la pestaña de Red de la herramienta para desarrolladores web de Mozilla Firefox.

Técnicas y Herramientas

para investigar si se ha dado carga dinámica de contenido interesante para después tratar de replicarlo.



Ilustración 12 - Pestaña Inspector de la herramienta para desarrolladores web de Mozilla Firefox



Ilustración 13 - Pestaña Console de la herramienta para desarrolladores web de Mozilla Firefox

Status	Method	Domain	File	Initiator	Type
200	GET	palletsprojects.com	pygments.css	stylesheet	css
200	GET	palletsprojects.com	main.js?h=96d2646b	script	js
200	GET	buttons.github.io	buttons.js	script	js
200	GET	fonts.googleapis.com	css?family=Raleway:400,400italic,500,500italic,700,700italic,300,300italic&subset=latin,latin-ext	stylesheet	css
200	GET	fonts.gstatic.com	1Ptug8zYS_SKggPNyC0ITw.woff2	font	woff2
200	GET	fonts.gstatic.com	1Ptug8zYS_SKggPNyC0ITw.woff2	font	woff2
200	GET	fonts.gstatic.com	1Ptug8zYS_SKggPNyC0ITw.woff2	font	woff2
200	GET	palletsprojects.com	fontawesome-webfont.woff2	font	woff2
200	GET	api.github.com	flask	button.js(xhr)	json
200	GET	palletsprojects.com	favicon.png	FaviconLoader.sys.mjs:176 (img)	png

Ilustración 14 - Pestaña Network de la herramienta para desarrolladores web de Mozilla Firefox

4.2. Lenguajes empleados

4.2.1. Python

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Python Software Foundation.
- Página web: <https://www.python.org>
- Licencia de uso: Python Software Foundation License
- Coste: gratuito.

Python es un lenguaje de programación de alto nivel, multiplataforma y orientado a objetos administrado por Python Software Foundation, una organización sin ánimo de lucro cuyo fin es el de fomentar su lenguaje de programación. Su sintaxis está enfocada en promover la legibilidad del código y por ende reducir costes de mantenimiento de los programas. Además, también se trata de un lenguaje interpretado y dinámico.

En el proyecto Python tiene un rol completamente protagonista. Es el lenguaje con el que se ha construido todo el proyecto. Dentro de la parte de desarrollo del proyecto, el algoritmo de búsqueda de prendas, es decir, todo el *web scraping* del proyecto se ha construido con Scrapy, un *framework* que utiliza Python y la parte de construcción de la página web con Flask un *framework* especializado en desarrollo web que también utiliza Python.

4.2.2. HTML y CSS

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: WHATWG (Web Hypertext Application Technology Working Group) para HTML y W3C (World Wide Web Consortium) para CSS.
- Página web: <https://html.spec.whatwg.org/>, <https://www.w3.org/TR/CSS/#css>
- Licencia de uso:
- Coste: gratuito.

HTML es un lenguaje de marcado diseñado para mostrar el contenido y la estructura de una página web en el navegador. Utiliza un conjunto de etiquetas para definir el texto y otros elementos como imágenes, audio, etc. Dichas etiquetas sirven para que el código sea estructurado y sólo el contenido se muestre al usuario.

CSS es un lenguaje cuyo fin es el de estilizar y presentar el contenido de una web de forma atractiva visualmente. Separa la estructura y contenido, que como se ha mencionado, corre a cargo de HTML y la presentación, que se define con CSS.

En el proyecto, se han utilizado ambos de forma muy básica para dar forma a las distintas páginas de la web.

4.2.3. SQL

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: ISO/IEC JTC 1 (Joint Technical Committee 1) / SC 32 (Subcommittee 32).
- Página web: <https://www.iso.org/standard/76583.html>
- Licencia de uso:
- Coste: gratuito.

SQL⁴³ (*Structured Query Language*) es un lenguaje diseñado para todas las tareas de gestión de bases de datos relacionales, como puede ser obtener datos, modificarlos, insertarlos o eliminarlos entre otros.

SQL ha sido utilizado en el proyecto en la parte de desarrollo de la aplicación y dentro de esta en la de *web scraping* para la realización de consultas e inserciones en la base de datos.

4.3. Otras herramientas

4.3.1. Microsoft Word

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Microsoft.
- Página web: <https://www.microsoft.com/en-us/microsoft-365/word>
- Licencia de uso: Trialware.
- Coste: gratuito en caso de ser estudiante adscrito al programa de Microsoft.

Microsoft Word es procesador de texto integrado en el paquete Microsoft Office cuyo uso es el de manipular documentos viendo mientras se escribe en ellos cómo sería el resultado final de los mismos. Se trata del procesador de textos más famoso y usado de la historia. Tiene funcionalidades extra como puede ser corrector ortográfico, gestor de citas, gestor de imágenes, tablas y más.

En el proyecto ha sido utilizado para escribir los puntos clave del avance del mismo en las reuniones que se han dado con el tutor del proyecto y para la redacción de la memoria principal y los anexos.

4.3.2. Visual Paradigm

Datos técnicos de la herramienta:

- Fabricante/Desarrollador: Visual Paradigm International Ltd.
- Página web: <https://www.visual-paradigm.com/>

⁴³ En español, Lenguaje de Consulta Estructurada.

- Licencia de uso: propietario pero con una edición gratuita para la comunidad.
- Coste: gratuito en caso de ser estudiante adscrito a la comunidad.

Visual Paradigm es una herramienta CASE⁴⁴ (*Computer Aided Software Engineering*) para modelar distintos diagramas de Ingeniería del Software así como la gestión del ciclo de vida del desarrollo del software.

En el proyecto ha sido utilizado para realizar toda los diagramas de la documentación de Ingeniería del Software presente en **Anexo II** y **Anexo III**.

⁴⁴ En español, Ingeniería del Software Asistida por Ordenador, son aplicaciones que permiten desarrollar y modelar diagramas de Ingeniería del Software.

5. ASPECTOS RELEVANTES DEL DESARROLLO

5.1. Investigación

En esta primera fase del proyecto, se ha centrado primero el foco en elegir las marcas de ropa cuyo catálogo estará posteriormente en la página web y posteriormente investigar las marcas elegidas, tanto en cómo obtener los campos necesarios para cada producto como los métodos de *crawling* para recorrer la web.

Los criterios de elección de las marcas, del número de estas y de los productos que se expondrán en la página web son los siguientes:

- Marcas orientadas a la ropa juvenil y que sean conocidas por el público general.
- La cantidad de marcas elegidas debe ser tres para facilitar las labores de estandarización de los datos, es decir, que cada dato tenga el mismo formato en todas las marcas y para ofrecer más funcionalidades en la página web.
- Que en la página de cada producto haya como mínimo todos los siguientes parámetros a la hora de hacer *parsing*: nombre del artículo, precio, URL de una imagen del producto, género/sexo de la prenda, descripción, color. Hay dos campos opcionales, composición y corte⁴⁵, que como mínimo deben estar presentes en dos de las tres marcas. Si no se cumple un campo de los considerados como obligatorios, no se considerará seguir explorando dicha marca.

	Zara	H&M	Carhartt WIP	Bershka	Pull&Bear
Nombre	√	√	√*	√	√
Precio	√	√	√	√	√
URL Imagen	√	√	√	√	~
Género	√*	√	√	√	√
Descripción	√	√	√	x	√
Color	√*	√	√*	√	√
Composición	~	√*	√*	~	~
Corte	x	√	√	x	x

Tabla 1 - Tabla de parámetros a obtener al hacer scraping

- Únicamente se seleccionará ropa de adultos para exponer en la página web, ni perfumes, ni ropa de baño, ni accesorios, ni zapatos, ni artículos de hogar, ni ropa de niños.
- Se premiará que el *crawling* sea lo más sencillo posible y se evite hacer *scraping* de ítems innecesarios como los mencionados en el punto anterior.

⁴⁵ Se refiere a cómo sienta una prenda a una persona. Por ejemplo, un corte *skinny* o ajustado en español será una prenda muy ceñida mientras que una prenda con un corte *relaxed* u holgado en español, será una prenda que dará mucha libertad de movimiento a la persona.

La simbología de la tabla “Tabla 1” es la siguiente:

- √: se puede obtener sin demasiados problemas. Como mucho requiere de una limpieza de la cadena obtenida con el XPath mediante expresiones regulares no muy complejas para obtener el valor deseado.
- √*: requiere más complejidad obtener el valor o estandarizarlo. Puede tener muchos valores de un tipo, estar en otro idioma o requerir de expresiones regulares complejas para obtener el valor deseado.
- ~: se cargan en la web original de forma dinámica, lo que requiere que se haga una llamada a la API de la web para obtener dicho valor.
- x: el valor no puede obtenerse porque no tiene un campo asociado en la página web. Es decir, si no hay un elemento HTML accesible mediante XPath que en todos o en la gran mayoría de los casos tenga un parámetro, como puede ser el corte de una prenda. En el caso de Zara hay camisetas en cuyo nombre se indica que tiene un corte *oversize*⁴⁶ sin embargo no hay un elemento HTML fijo que siempre contenga el tipo de corte de la prenda.

Se procederá a analizar cada marca de forma pormenorizada, donde se tratarán tres apartados. El primer apartado es “contexto”, en el cual se contextualizará y se darán datos sobre la marca de ropa. El segundo apartado es “análisis”, en el cual se realizará en detalle una explicación de los parámetros de la tabla más relevantes y se mostrarán las distintas opciones consideradas para recorrer dicha web de la forma más eficiente. Y el tercer y último apartado es el de “mecanismos de protección contra *web scraping*” que tiene dicha página web en el que se muestran cómo de agresivos son las herramientas anti *web scraping* que tiene la página web de la marca.

5.1.1. Zara

Contexto

Zara es una marca de ropa española nacida en Arteixo, A Coruña, España. Fue fundada por Amancio Ortega y Rosalía Mera en 1974. Es la marca insignia del grupo Inditex, una empresa multinacional de distribución textil también fundada por Amancio Ortega, bajo la cual se encuentran marcas como la ya mencionada Zara, Zara Home, Massimo Dutti, Pull&Bear, Bershka, Oysho, Lefties y Stradivarius.

En 2022 tuvo ventas por un valor de 32.569 millones de euros creciendo un 17,5% respecto al año anterior y tiene un total de 1939 tiendas físicas repartidas por todo el mundo.

El estilo de ropa que abarca va desde ropa más informal del día a día hasta ropa formal combinando ropa más juvenil e informal y de menor calidad con ropa con una mayor calidad que puede ser desde informal a más seria. Se suele inspirar en la moda de altas marcas pero vendiendo a precios más bajos. Tiene ropa para hombre, mujer y niños. Se

⁴⁶ En español, sobredimensionado o extragrande, se refiere al corte de prendas que son muy holgadas y grandes.

enmarca en la llamada *fast fashion*⁴⁷, pues genera una gran cantidad de colecciones y artículos nuevos cada año llegando a 18000 diseños nuevos al año.

Análisis

Análisis de los parámetros de la “Tabla 1” para *parsing* y de los posibles métodos de *crawling*:

- Parsing:
 - Color: fácil de obtener mediante XPath pero requiere de varias expresiones regulares para realizar la limpieza además de que hay un gran número de valores que puede tomar el campo.
 - Género: fácil de obtener mediante XPath pero además de tener los valores de “Hombre” o “Mujer” también puede tener “Niños” como se muestra en “Ilustración 15”. Esto hace que para determinar si un artículo es de niños, aunque en algunos casos pueda detectarse en la URL, en otros casos sólo va a ser posible haciendo *scraping* con ese método para ser rechazados.

ZARA / MUJER / KIMONO TERCIOPELO FAJÍN LIMITED EDITION

ZARA / NIÑOS / TOP PUNTO SEAMLESS

ZARA / HOMBRE / POLO PUNTO ALGODÓN - SEDA

Ilustración 15 - Valores que puede tomar el campo de género mediante XPath en Zara

- Composición: se obtiene a través de carga dinámica y por ende no se encuentra en el HTML de la respuesta. Se podría obtener mediante una llamada a la API, no obstante, el archivo *robots.txt* de Zara restringe la dirección de esa API como se muestra en “Ilustración 16”. Acceder a dicha dirección conllevaría no ser un programador responsable y exponerse a una prohibición de acceso a la página web en caso de ser pillado. No se obtendrá este campo.

⁴⁷ En español, moda rápida, se refiere a la captura de las nuevas tendencias de la moda por parte de las marcas de forma que se diseña, fabrica y saca a la venta ropa rápidamente atendiendo a dichas tendencias de modo que al final del año dichas marcas han sacado una gran cantidad de ropa diferente.

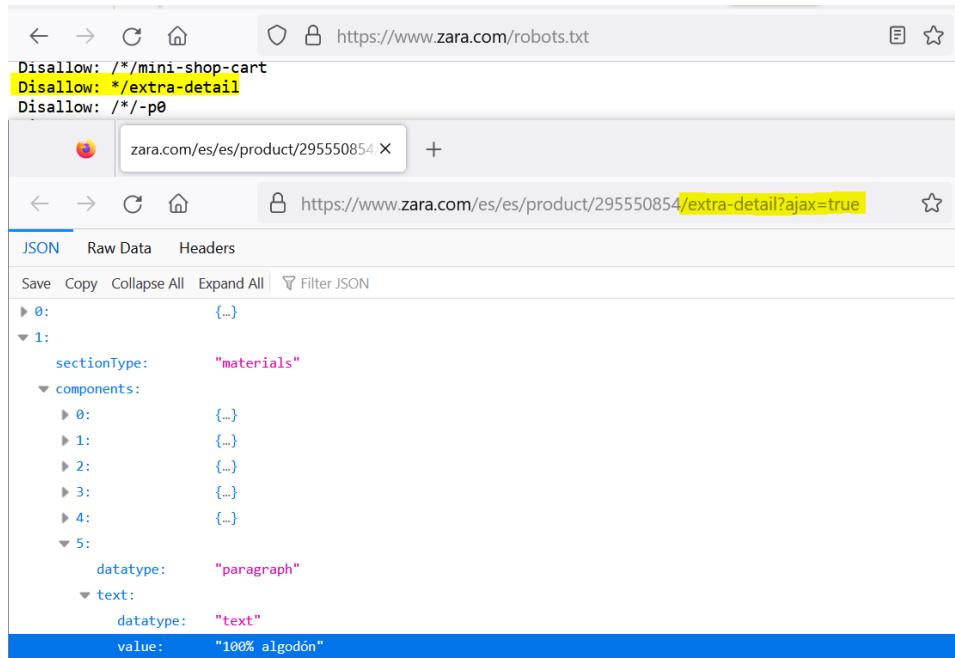


Ilustración 16 - Obtención de composición mediante la API de Zara

- Corte: no hay una sección en la página web de los productos donde ponga el tipo de corte que tiene. No se puede obtener este campo.
- Crawling:
 - API REST:
 - API de “Ver todos los productos”: a veces, la página web da la opción de “Mostrar todos los productos” y en ese caso mediante llamadas a la API puedes obtener un archivo JSON con todos los productos o por partes haciendo varias llamadas, como de un total de 1000 productos obtenerlos de 100 en 100. Zara no tiene de esta opción por lo que no se ha podido explorar la API.
No se elegirá este método de *crawling*.
 - API de “Buscar”: no es explotable. En ocasiones, se puede trucar la API de “Buscar productos” pasando como parámetro de búsqueda un espacio o “%20”, que es el carácter del espacio codificado, y puede llegar a mostrar todos los productos. Como se muestra en “Ilustración 17”, no devuelve ningún resultado.



Ilustración 17 - API de Buscar en Zara

No se elegirá este método de *crawling*.

- Sitemap: la estructura de *sitemap* es sencilla y fácil de recorrer. Está compuesta únicamente por un *sitemap index* y este apunta a su vez a los *sitemap* de productos en cada idioma disponible en la página web al igual que a *sitemap* de imágenes. En este caso nos interesan los productos en español, indicados por el archivo “sitemap-es-es.xml.gz” como muestra la “Ilustración 18”. “sitemap-es-es.xml.gz” cuenta con entre 15000 y 18000 URLs de productos y listas de productos.

Algunas desventajas de este método de *crawling* es que los *sitemap* no están del todo actualizados y no todos los productos disponibles en la web se encuentran en estos archivos. Además, habrá muchos productos, como perfumes, accesorios y en algunos casos, productos de niños, en estos archivos que no resultan interesantes y que habrá que rechazar. No obstante, el nombre del producto aparece incrustado en las URLs en Zara como la siguiente:

<https://www.zara.com/es/es/pantalon-cargo-relaxed-fit-p05520310.html>

Por lo que podría hacerse una lista de palabras que, en caso de formar parte de la URL, no se envíe petición a dicha URL.

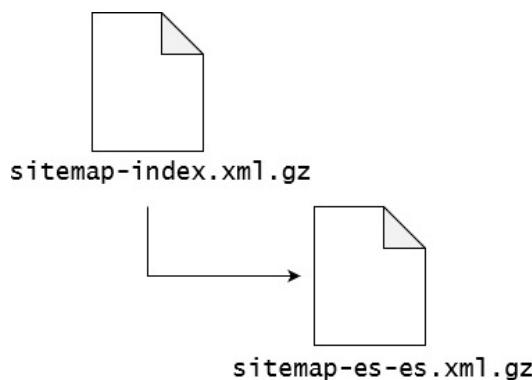


Ilustración 18 - Estructura de los sitemap de Zara

Este será el método escogido para el *crawling* de Zara

Se analizará el *sitemap* de productos “sitemap-es-es.xml.gz”. En “Ilustración 19” se muestra una parte del contenido de dicho archivo:

```
<url> <loc>https://www.zara.com/es/es/help/zara-qr-h67.html</loc> <changefreq>daily</changefreq> <priority>0.8</priority> </url>
<url> <loc>https://www.zara.com/es/es/help/zara-qr-information-general-h68.html</loc> <changefreq>daily</changefreq> <priority>0.8</priority> </url>
<url> <loc>https://www.zara.com/es/es/mujer-nuevo-11180.html</loc> <changefreq>daily</changefreq> <priority>0.8</priority> </url>
<url> <loc>https://www.zara.com/es/es/camisa-oversize-arrugada-limited-edition-p08216101.html</loc> <changefreq>weekly</changefreq> <priority>0.4</priority> </url>
<url> <loc>https://www.zara.com/es/es/vestido-macrame-bordados-limited-edition-p07521272.html</loc> <changefreq>weekly</changefreq> <priority>0.4</priority> </url>
<url> <loc>https://www.zara.com/es/es/kimono-terciopelo-fajin-limited-edition-p08110995.html</loc> <changefreq>weekly</changefreq> <priority>0.4</priority> </url>
```

Ilustración 19 - Contenido de sitemap-es-es.xml.gz de Zara

A continuación se expondrán las conclusiones que se han obtenido tras realizar una investigación del *sitemap*.

El contenido del *sitemap* cambia cada 8 días.

La etiqueta “*<changefreq>*” indica la frecuencia con la que se cambia el contenido de las URLs como se dijo en “3.3.2.2 Sitemaps”. En “sitemap-es-es.xml.gz” esta etiqueta sólo toma valores *daily*⁴⁸ y *weekly*⁴⁹.

La etiqueta “*<priority>*” no es relevante para el proyecto.

En el siguiente listado se clasificarán ahora los tipos de URL. El nombre del tipo se mostrará en color verde si son tipos de URLs que deban ser recorridas por tener cierto valor o en color rojo si carecen de valor. Se agruparán en dos grupos según el valor que toma la etiqueta “*<changefreq>*” y dentro de dichos grupos según una especie de prefijo que es “-x”:

- Para el valor *daily*:

- **Tipo /help o -h:** se refiere a páginas de ayuda de distintos temas como pueden ser modificaciones de pedidos o compra segura. Por ejemplo, <https://www.zara.com/es/es/help/modificaciones-de-pedido-h4.html>.

Carece de valor pues estas URLs no contienen productos ni otros otras URLs que lleven a productos.

- **Tipo -l:** son páginas de recopilaciones de productos. Por ejemplo, los ítems nuevos de mujer, vestidos, pantalones holgados, etc. Lo más relevante de estas URLs, es que contienen bastantes productos que no se encuentran en el *sitemap*. Además, a la gran mayoría de estas páginas se pueden llegar a través del menú principal de Zara o buscando a través de un buscador, por ejemplo a <https://www.zara.com/es/es/mujer-vestidos-11066.html> se puede llegar a través de Mujer → Vestidos en el menú de Zara, a <https://www.zara.com/es/es/mujer-vestidos-verdes-11741.html> no se puede llegar a través de la web pero sí si se busca “zara vestidos verdes” en el buscador del navegador.

Tiene valor pues contiene URLs de tipo -p, de productos, que no se encuentran en el *sitemap*.

- **Tipo -mkt:** páginas parecidas a las de tipo -l pero contienen fotoreportajes en los que recopilan productos o conjuntos. Por

⁴⁸ En español, diariamente.

⁴⁹ En español, semanalmente.

ejemplo, <https://www.zara.com/es/es/man-barbie-collection-mkt6401.html>. Contienen URLs tipo -pT o un tipo nuevo -pL, que a efectos generales es como un -pT o también puede contener URLs tipo -l ya existentes en el *sitemap*. Todas las prendas de este tipo de enlace se encuentran en el *sitemap*.

Carece de valor pues contiene URLs tipo -pT, cuyos productos están en el *sitemap* en su gran mayoría al igual que los tipo -l, que ya se encuentran en dicho archivo también.

- **Otros:** distintas URLs con formatos variados.
 - <https://www.zara.com/es/>: página principal de Zara en español.
 - <https://www.zara.com/es/es/z-tiendas-st1404.html>: página para buscar tiendas físicas de Zara.
 - <https://www.zara.com/es/es/z-trabajo-corp1398.html>: portal de trabajo de Zara.
 - <https://www.zara.com/es/es/--c1946312.html> y <https://www.zara.com/es/es/--c2297662.html>: páginas vacías. Puede cambiar el número después del -c.

Carecen de valor pues estas URLs no contienen ni productos ni otras URLs interesantes.

- Para el valor *weekly*:
 - **Tipo -p:** páginas de los productos de Zara normales. Por cada URL hay una página donde se muestra un único producto.
Hay subtipos dentro de estas URLs:
 - **-p0:** los verdaderamente importantes, llevan a páginas de prendas de ropa.
 - **-p1:** llevan a páginas de calzado o bolsos y mochila. No son interesantes para el proyecto.
 - **-p2:** llevan a páginas de maquillaje. No son interesantes para el proyecto.
 - **-p5:** llevan a páginas de tarjetas de regalo. No son interesantes para el proyecto.

Tienen valor pues son los productos cuya información se quiere almacenar.

- **Tipo -pT:** muestran conjuntos de ropa, es decir, varias prendas juntas por cada página de este tipo. Por ejemplo, <https://www.zara.com/es/es/conjunto-chaleco-y-pantalon-jacquard-estructura-pt0503963470.html>. En el momento del

análisis de los sitemap, en junio de 2023, existían un subtipo de estos enlaces que contenían /look, contenían conjuntos de ropa de una colección de Zara llamada Origins.

Carecen de valor pues la gran mayoría de los productos de las URLs tipo -p que contienen ya se encuentran en el sitemap.

- **Tipo -pG:** páginas no muy concluyentes, contienen resultados de búsquedas de perfumes mayoritariamente o perfumes.

Carecen de valor pues los perfumes son productos que no interesan según los criterios de este proyecto.

Para justificar las decisiones sobre si recorrer los distintos tipos de URLs se ha realizado por un lado una investigación manual para las URLs tipo -mkt y también se han creado dos *scripts* para investigar las de tipo -l y -pt, como se indica en “Tabla 2”. En el resto de los tipos de URLs no se ha realizado investigación pues no son enlaces que contengan productos o URLs de interés.

Nombre del <i>script</i>	Tarea que realiza
code.py	Explora las URLs tipo -l y -pT para ver si los productos que contienen están en el <i>sitemap</i> o no
tipos_sitemaps.py	Cuenta todos los tipos de URLs del sitemap

Tabla 2 - Scripts de investigación del sitemap de Zara

El resultado del script “tipos_sitemaps.py” se muestra a continuación en las siguientes ilustraciones “Ilustración 20 - Resultados de “tipos_sitemaps.py” del 09-06-20” y “Ilustración 21 - Resultados de “tipos_sitemaps.py” del 17-06-2023”, en las que además se muestra el cambio de contenido en el *sitemap*:

```
TIPO DAILY
- Tipo /help: 60
- Tipo -l[0-9]+: 1320
- Tipo -mkt[0-9]+: 21
- Otros: 4
Total: 1405

TIPO WEEKLY
- Tipo /look: 170
- Tipo -p[0-9]+: 14795
- Tipo -pT[0-9]+: 111
- Tipo -pG[0-9]+: 7
- Otros: 0
Total: 15083

Total WEEKLY y DAILY: 16488
```

Ilustración 20 - Resultados de "tipos_sitemaps.py" del 09-06-20

```
TIPO DAILY
- Tipo /help: 60
- Tipo -l[0-9]+: 1320
- Tipo -mkt[0-9]+: 21
- Otros: 4
Total: 1405

TIPO WEEKLY
- Tipo /look: 170
- Tipo -p[0-9]+: 15394
- Tipo -pT[0-9]+: 97
- Tipo -pG[0-9]+: 7
- Otros: 0
Total: 15668

Total WEEKLY y DAILY: 17073
```

Ilustración 21 - Resultados de "tipos_sitemaps.py" del 17-06-2023

Resultado del análisis de URLs tipo -l y tipo -pT del script “code.py”, almacenado en los archivos *codeResultados_l.txt* y *codeResultados_pT.txt* dentro de *prendas/resultados_scripts_investigacion*:

- Tipo -l: contienen listas de enlaces -p. La gran mayoría de ellos no se encuentran en el *sitemap* y se van añadiendo nuevos de forma diaria y desapareciendo otros. Esto es así porque los enlaces -p que van haciéndose antiguos se cargan de forma dinámica según se va bajando en la página web mientras que los nuevos, que son los que están previamente en el HTML están más arriba. No obstante, de un día para otro pueden repetirse

productos dentro de este tipo de enlace y en total hay varios miles de productos que no se encuentran en el *sitemap* en estos enlaces. Sería recomendable recorrerlos cada día para poder tener el catálogo entero de Zara actualizado, pero no repetir los ya existentes. Se hará *crawling* a los enlaces tipo -l.

- Tipo -pT: contienen conjuntos de ropa, es decir, de distintas URLs tipo -p. La gran mayoría de estos se encuentran en el propio *sitemap* y hay una minoría despreciable que no.

Como conclusión, no se hará *crawling* a los enlaces tipo -pT.

Como conclusiones sobre el *sitemap* de Zara decimos que:

- Únicamente se recorrerán las URLs tipo -l y tipo -p.
- La frecuencia para hacer *crawling* de las URLs tipo -l debe ser diaria mientras que recorrer los enlaces tipo -p basta con hacerse cada ocho días que es cuando cambia el contenido del archivo.

Mecanismos de protección contra web scraping

El mecanismo contra *web scraping* que tiene Zara es muy fino. Suele denegar el acceso si la ratio de peticiones por segundo es alta pero no si el volumen de peticiones es alto a lo largo del tiempo. Es decir, se pueden enviar 20000 peticiones a lo largo de 20 horas y que no haya prohibición de acceso, sin embargo, si la ratio es mayor que cuatro peticiones por segundo durante unos minutos, el servidor empieza a denegar acceso después de 500 peticiones aproximadamente. Es fino porque, aunque se haya denegado el acceso, si se cambia la ratio de peticiones a uno menor y se vuelve a lanzar el programa. Al poco rato, te permite el acceso. Esto no ocurre en otras páginas webs como se explicará en “5.2.1.3 Spider de H&M”.

Para evitar este mecanismo, como ya se ha comentado, hay que mantener una ratio baja de peticiones por segundo al servidor de Zara.

5.1.2. H&M

Contexto

H&M o Hennes & Mauritz es una cadena multinacional sueca de moda fundada en Västerås, Suecia, por Erling Persson. Es la marca insignia de la empresa sueca de moda H&M Group, que cuenta además con las marcas, Weekday, Monki, COS (*Collection of Style*⁵⁰), H&M Home, & Other Stories, Arket y Afound.

En 2021 obtuvo 1000 millones de euros de beneficio, nueve veces mayor que el año anterior y 4801 tiendas repartidas por todo el mundo en el mismo año.

El tipo de ropa que vende H&M es muy parecido al de Zara. Tiene desde ropa informal hasta más casual y formal alternando también baja con más alta calidad y diseños más

⁵⁰ En español, Colección de Estilo, es una marca de ropa británica perteneciente a la empresa H&M Group.

juveniles y atemporales. También se trata de una marca de *fast fashion*, pues saca al año una media de 16 colecciones. Tiene ropa para hombre, mujer, niños y un apartado de artículos de hogar.

Análisis

Análisis de los parámetros de la “Tabla 1” para *parsing* y de los posibles métodos de *crawling*:

- Parsing:

- Composición: requiere de un método complejo obtenerlo pues según el tipo de prenda puede tener distintos tipos de composición como únicamente composición principal o parte superior y parte inferior o exterior y forro de mangas como se muestra en “Ilustración 22”. El objetivo es el de obtener la composición principal o más relevante.

Materiales	Materiales & proveedores	Materiales
Composición	Composición	Composición
Revestimiento: Poliuretano 100%	Algodón 100%	Parte inferior: Algodón 100%
Forro: Poliéster 100%	Material: Punto	Parte superior: Algodón 100%
Tela exterior: Poliéster 100%		Material: Punto
Entretela: Poliéster 100%		

Ilustración 22 - Formato de composición en H&M

- Crawling:

- API REST:

- API de “Ver todos los productos”: en H&M hay opción de hacer compra por producto y se puede llegar a ver todos los productos de hombres y mujeres por separado. Esta característica es muy interesante pues en estas dos listas de productos estarían excluidos los productos de niños y de accesorios del hogar además de tener una lista actualizada de todos los productos para cada género, cosa que no ocurre con los *sitemap*.

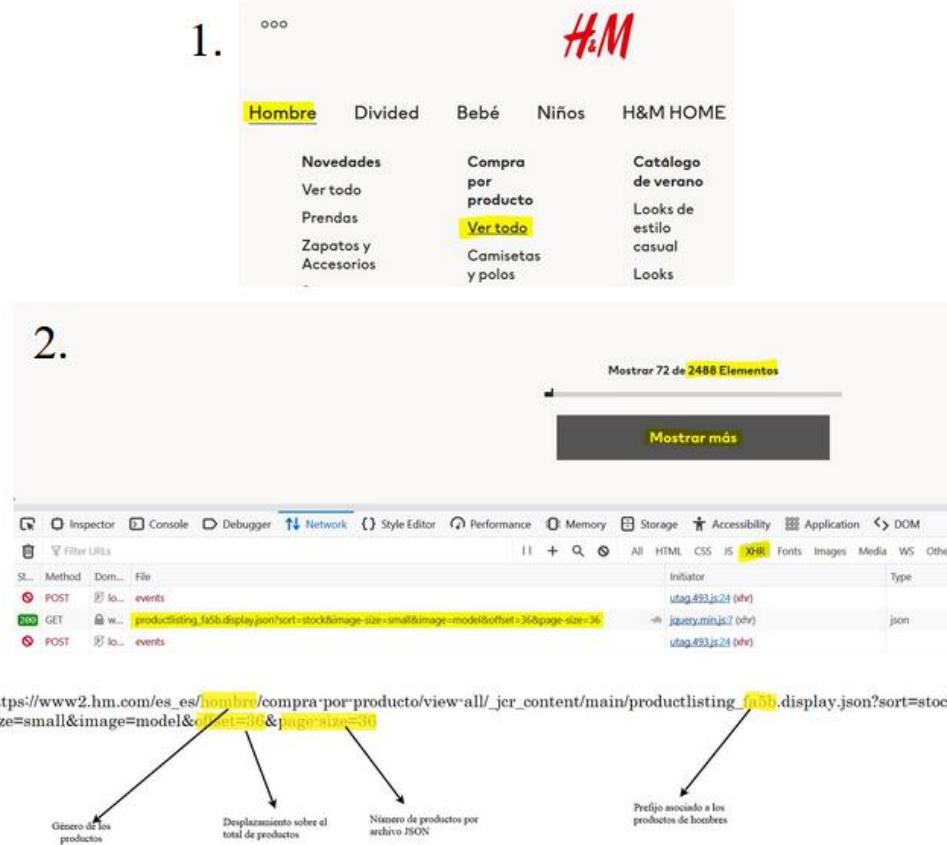


Ilustración 23 - Proceso de análisis de la API de carga dinámica de H&M

A continuación analizaremos los pasos del proceso de descubrimiento de la API de carga dinámica de todos los productos de hombre expuestos en “Ilustración 23”. Se presupone que el mismo método funcionará también para los productos de mujer:

1. Se selecciona el apartado “Ver todo” de “Compra por producto” en la sección de “Hombre” de la página principal de H&M. Se presupone que en la sección de “Mujer” la API funcionará igual cambiando ciertos parámetros de la dirección.
2. Se puede ver resaltado en color el número total de productos de hombre. Se abre la consola de desarrollador web de Mozilla Firefox y se abre la pestaña de Red o *Network* y se selecciona el filtro XHR, que muestra las peticiones que han devuelto archivos JSON. Se pulsa sobre el botón “Mostrar Más” y se observa en resaltado en color la URL de una API que devuelve un archivo JSON que contiene los datos de los productos cargados en la página web tras pulsar el botón.
3. Se analiza la URL de la API que se ha llamado. Como características concretas de los productos de hombre, se observa

que la API de hombre tiene “hombre” y “fa5b”, un prefijo que difiere del de la misma API de mujer que contiene “30ab”. Para el número de productos a cargar, se observa que la API tiene los parámetros *offset*, que indica el desplazamiento sobre el total de productos y *page-size*, que indica el número de productos que contendrá el archivo JSON. En este caso *offset=36* y *page-size=36*, por lo que contendrá un total de 36 productos, desde el producto 36 al 72 del total de 2488 prendas que hay para hombre.

Modificando ciertos parámetros de esta API se pueden llegar a obtener todos los productos de hombres y mujeres de forma sencilla y rápida y estando actualizados. Se estiman aproximadamente 2500 ítems para hombres y 10000 para mujeres, pues, aunque ahora mismo haya por ejemplo, 11000 ítems para mujeres, en otras ocasiones se han visto 9700 y lo mismo para los hombres, la cifra fluctúa.

Este será el método escogido para el *crawling* de H&M.

Los atributos de los objetos de los productos que contienen información interesante de los parámetros que se quiere recoger son los siguientes. También se muestran en “Ilustración 24”:

- Nombre: con el atributo “title”.
- Precio: con el atributo “price”.
- URL de la imagen: con el atributo “image”, que a su vez es una lista de otros objetos. Se elige el primero, índice 0, y el atributo “src”. Como se necesitará una imagen grande, manipulando la URL y cambiando la palabra “style” justo al final por “main” devuelve la imagen grande del producto en vez de la miniatura.

URL de la miniatura:
[https://lp2.hm.com/hmgoepprod?set=source\[/44/1f/441f78592a51dee886300adbc3aa19c8f7a2bb94.jpg\],origin\[dam\],category\[\],type\[LOOKBOOK\],res\[m\],hmver\[1\]&call=url\[file:/product/style1\]](https://lp2.hm.com/hmgoepprod?set=source[/44/1f/441f78592a51dee886300adbc3aa19c8f7a2bb94.jpg],origin[dam],category[],type[LOOKBOOK],res[m],hmver[1]&call=url[file:/product/style1])

URL de la imagen normal:
[https://lp2.hm.com/hmgoepprod?set=source\[/44/1f/441f78592a51dee886300adbc3aa19c8f7a2bb94.jpg\],origin\[dam\],category\[\],type\[LOOKBOOK\],res\[m\],hmver\[1\]&call=url\[file:/product/main\]](https://lp2.hm.com/hmgoepprod?set=source[/44/1f/441f78592a51dee886300adbc3aa19c8f7a2bb94.jpg],origin[dam],category[],type[LOOKBOOK],res[m],hmver[1]&call=url[file:/product/main])

- Género: como se explorarán las APIs de hombre y mujer por separado, cuando se explore la de hombre, se añade manualmente “Hombre” y viceversa cuando se explora la API de mujer.

- Descripción: no hay ningún atributo.
- Color: en el atributo “swatches”, que es una lista de objetos, se elige el primero, índice 0, y de este se elige el atributo “colorName”.
- Composición: no hay ningún atributo.
- URL del producto: con el atributo “link”, que muestra una URL parcial del producto, hay que añadirle la URL del dominio de H&M, “<https://www2.hm.com>”.

```
link:                      "/es_es/productpage.1169665001.html"
title:                     "Suéter sin mangas con cuello de pico Loose Fit"
▼ image:
  ▼ 0:
    ▶ src:                  "//lp2.hm.com/hmgoepprod?...rl[file:/product/style]"
    ▶ dataAltImage:          "//lp2.hm.com/hmgoepprod?...rl[file:/product/style]"
    ▶ alt:                  "Suéter sin mangas con cu...pico Loose Fit Auxiliar"
    dataAltText:             "Suéter sin mangas con cuello de pico Loose Fit"
price:                     "29,99 €"
▼ swatches:
  ▼ 0:
    colorCode:              "#ECE9E2"
    articleLink:             "/es_es/productpage.1169665001.html"
    colorName:               "Blanco/Naranja"
```

Ilustración 24 - Atributos del archivo JSON obtenido con la API de H&M

Se pueden obtener mediante los atributos de los objetos del archivo JSON todos los parámetros menos el de descripción y composición. Para obtener esos otros parámetros, habrá que hacer una petición a la URL del producto y después hacer *scraping* del HTML del mismo.

Dado que hay productos, como pueden ser trajes de baño o zapatillas o perfumes que no son necesarios para el proyecto, se puede hacer un filtrado de estos productos en base al atributo *category*⁵¹ como se indica en la ilustración siguiente, detectando categorías de productos no deseados:

```
category:                  "men_socks"
```

Ilustración 25 - Atributo category de los objetos del archivo JSON de la API de H&M

- API de “Buscar”: como se mencionó en el apartado “5.1.1 Zara”, si en la URL de la API de “Buscar” introducimos un carácter vacío “” o

⁵¹ En español, categoría. Se trata de un grupo de ropa similar, en el ejemplo se trata de *men_socks*, que en español quiere decir calcetines de hombre.

un espacio codificado “%20” como se ve en el subrayado a color de la URL en la “Ilustración 26”, a veces se pueden llegar a mostrar todos los productos como es este caso. La principal desventaja de este método respecto la API de “Mostrar todos los productos” es que hay una gran cantidad de prendas de ropa como pueden ser la de niños, bebés, de hogar (H&M Home) o de belleza (Beauty) que carecen de interés por parte del proyecto y habría que hacer un filtrado.

No se elegirá este método de *crawling*.



Ilustración 26 - Proceso de análisis de la API de Buscar de H&M

- *Sitemap*: la estructura de *sitemap* es algo más complicada que en el caso de Zara. El *sitemap* índice para el idioma español, “es_es.sitemap.xml”, contiene direcciones de *sitemap* para imágenes, productos, subdepartamentos, páginas de filtrado, tarjetas de regalo, etc. La que podría resultar interesante es la de por productos, como muestra la “Ilustración 27”. Cada *sitemap* de productos tiene aproximadamente 500 URLs.

Algunas desventajas de este método de *crawling* es que los *sitemap* no están del todo actualizados y no todos los productos disponibles en la web se encuentran en estos archivos. Además, a diferencia de Zara, el nombre del producto no viene incluido en la URL, por lo que sólo se podrá saber si un producto no es deseado una vez recibida la respuesta a dicha URL con la consecuencia de pérdida de rendimiento.

https://www2.hm.com/es_es/productpage.1091453001.html

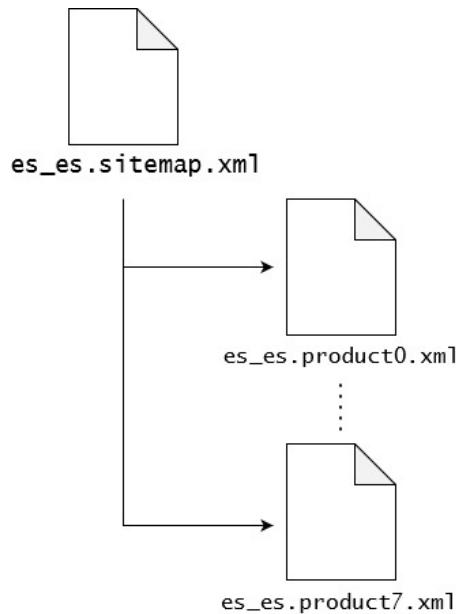


Ilustración 27 - Estructura de los sitemap de H&M

Mecanismos de protección contra web scraping

A diferencia de Zara, el sistema de defensa contra *web scraping* que tiene H&M no es tan eficaz ni tan fino, pero sí más punitivo. Se penaliza una ratio alta de peticiones y también si el volumen de peticiones supera las 7000 aproximadamente. El principal problema recae en que la prohibición se produce cuando acabas de hacer el *web scraping* o tras varios miles de peticiones si ha sido un proceso más largo, y dura entre tres y siete días. Este método penaliza mucho el poder hacer *web scraping* pues hay que hacerlo con una frecuencia de siete días por lo menos para, en caso de haber denegación de acceso, que esta haya acabado cuando se vuelva a ejecutar el algoritmo. También pueden no llegar a obtenerse todos los productos y no hay una ratio baja que evite la prohibición, pues se penaliza el volumen como se ha mencionado anteriormente.

5.1.3. Carhartt WIP

Contexto

Carhartt es una marca de ropa de exterior orientada al trabajo fundada en 1889 por Hamilton Carhartt en Dearborn, Michigan, Estados Unidos. Los productos que fabrica principalmente son chaquetas, abrigos, camisas, pantalones y ropa ignífuga y de caza, todo orientado a ser durable y resistente en ambientes de trabajo. Carhartt WIP, la marca con la que se trabajará en el proyecto, es una división de Carhartt fundada en 1994 por Edwin Faeh.

Carhartt WIP tuvo un valor en ventas de 2.1 millones de dólares en 2022 con 80 tiendas físicas por todo el mundo, sin embargo, su principal foco de ventas es el comercio electrónico.

Carhartt WIP produce prendas de ropa que son la ropa original de Carhartt adaptada a la vida y cultura urbana. Son prendas de muy alta calidad con un estilo informal y casual.

No se trata de una marca de *fast fashion* por lo que su catálogo es mucho más reducido y menos cambiante que marcas como H&M y Zara.

Análisis

Análisis de los parámetros de la “Tabla 1” para *parsing* y de los posibles métodos de *crawling*:

- Parsing:

- Nombre: está íntegramente en inglés como muestra “Ilustración 28”. Debe traducirse al español, al menos el tipo de prenda que es, para el público no bilingüe y para hacer más fácil tratarlo y hacer búsquedas.



W' S/S Pocket Heart T-Shirt
Squid
€ 35,00

W' S/S Bubbles T-Shirt
Grey Heather / Cassis
€ 45,00

Ilustración 28 - Nombre artículos de Carhartt

- Composición: hay un número ínfimo de productos que no tienen el apartado de composición y en otros aparece únicamente en inglés. Como el número de dichos productos es muy pequeño, se obtendrá este parámetro en cualquier caso.

- Crawling:

- Sitemap: diferencia de Zara y H&M, Carhartt WIP no es una marca de *fast fashion*, es decir, no tiene ni un catálogo tan grande, ni tan cambiante como las marcas anteriores. La estructura de los *sitemap* viene mostrada en “Ilustración 29”. El *sitemap* índice “sitemap.xml”, contiene las direcciones de los *sitemap* índice de cada idioma, en este caso el de español es “es/sitemap.xml” y contiene las direcciones de un *sitemap* de categorías de productos, otro de artículos de prensa y el de productos, que es el que nos interesa, “es/sitemap-products.xml”.

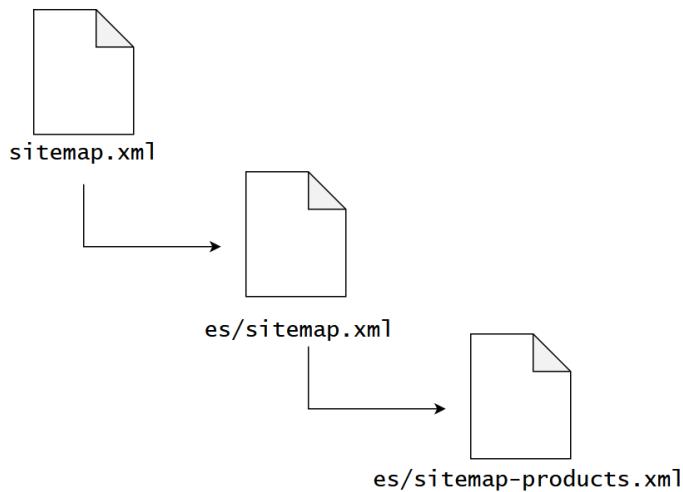


Ilustración 29 - Estructura de los sitemap de Carhartt

Como se ha mencionado anteriormente, al no tener un catálogo tan grande, aproximadamente “es/sitemap-products.xml” tiene un total de 750 productos entre hombres y mujeres, ni cambiar las colecciones con facilidad, por *sitemap* es un buen método para hacer *crawling*.

Este será el método escogido para el *crawling* de Carhartt WIP

Se analizará el *sitemap* de productos “es/sitemap-products.xml”. En “Ilustración 30” se muestra una parte del contenido de dicho archivo:

```

<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>https://www.carhartt-wip.com/es/mujer-pantalones/women-pants-w-jens-pant</loc>
    <lastmod>2023-06-10</lastmod>
  </url>
  <url>
    <loc>https://www.carhartt-wip.com/es/hombre-gadgets/accessories-gadgets-verse-rug</loc>
    <lastmod>2023-06-10</lastmod>
  </url>
  <url>
    <loc>https://www.carhartt-wip.com/es/hombre-bermudas-work/men-shorts-single-knee-short</loc>
    <lastmod>2023-06-10</lastmod>
  </url>
  <url>
    <loc>https://www.carhartt-wip.com/es/mujer-chaquetas/women-jackets-w-yanie-jacket</loc>
    <lastmod>2023-06-10</lastmod>
  </url>
  <url>
    <loc>https://www.carhartt-wip.com/es/hombre-gadgets/accessories-divers-trail-keyholder</loc>
    <lastmod>2023-06-10</lastmod>
  </url>
  <url>
    <loc>https://www.carhartt-wip.com/es/hombre-pantalones-work/men-pants-simple-pant</loc>
    <lastmod>2023-06-10</lastmod>
  </url>
  <url>
    <loc>https://www.carhartt-wip.com/es/hombre-camisetas-manga-corta/men-t-shirts-s-s-new-frontier-t-shirt</loc>
    <lastmod>2023-06-10</lastmod>
  </url>

```

Ilustración 30 - Contenido de es/sitemap-products.xml de Carhartt

- El campo de última modificación de las URLs de todos los productos sin excepción a 17 de agosto de 2023, “`<lastmod>`” queda fijado al 10 de junio, esto quiere decir que ningún producto se ha modificado después de esa fecha. Por lo tanto, la frecuencia del *web scraping* en Carhartt WIP debería de ser muy baja, como una vez por semana.
- El formato de todas las URLs es siempre el mismo como indica la siguiente “Ilustración 31”:

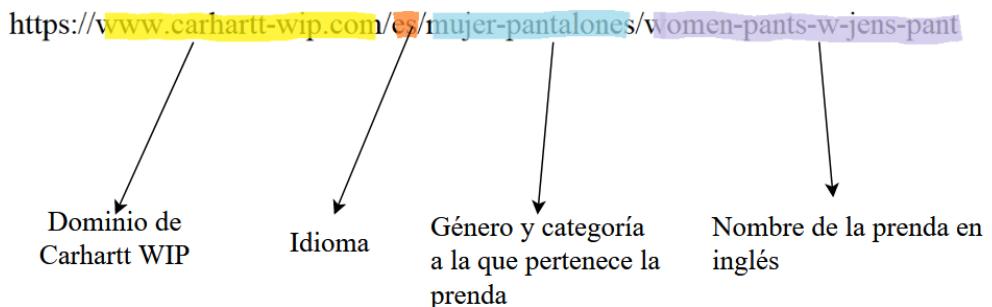
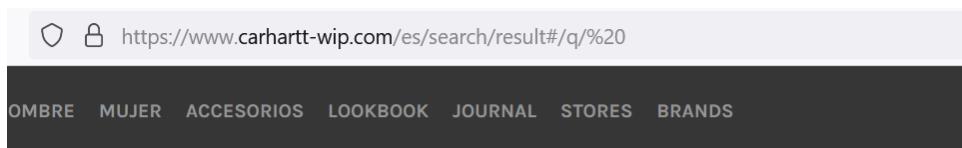


Ilustración 31 - Formato URLs de es/sitemap-products.xml de Carhartt

Se pueden usar las categorías de las URLs para ver si un producto es de hombre o de mujer, gracias al género, o para filtrar los ítems que no sean ropa de hombre o de mujer, como la categoría “hombre-gadgets”, o para añadir el nombre en español al nombre en inglés y así que se sepa qué tipo de prenda es para los clientes no bilingües. Por ejemplo, si en la categoría pone “pantalones” al nombre del artículo se le puede añadir “pantalón” y que quede el nombre final como “Women Pants W Jens Pant – Pantalón”.

- API REST:

- API de “Mostrar todos los productos”: Carhartt WIP carece de esta opción por lo que no tiene dicha API.
No se elegirá este método de *crawling*.
- API de “Buscar”: no es explotable, como se indica en la “Ilustración 32” no otorga ningún resultado.



O Resultados para ''

No hay resultados. Por favor, vuelva a intentarlo.

Ilustración 32 - API de Buscar de Carhartt

No se elegirá este método de *crawling*.

Mecanismos de protección contra web scraping

Dado que el catálogo de productos de Carhartt WIP es muy inferior al que tienen Zara y H&M, no se ha visto la necesidad de realizar un test sobre cómo de restrictivo es el mecanismo de protección contra *web scraping*, pues la recogida de información de esta

página acabará mucho antes que la de Zara o H&M, que son los cuellos de botella de todo el mecanismo búsqueda de prendas.

5.1.4. Bershka

Contexto

Bershka es una cadena de ropa juvenil perteneciente al grupo Inditex fundada en 1998 en Tordera, Barcelona, España.

Contaba con más de 1096 tiendas en 70 países en 2017 y generó en 2021 unos ingresos brutos de 321 millones de euros.

La ropa de esta marca está orientada a un público juvenil, tanto de hombres como de mujeres con una calidad media baja. Se trata de una marca de *fast fashion* por lo que genera muchos diseños nuevos al año y cuenta con un catálogo de prendas grande.

Análisis

Análisis de los parámetros de la “Tabla 1” para *parsing* y de los posibles métodos de *crawling*:

- Parsing:
 - Descripción: no contiene descripción. No se puede obtener este campo.
 - Composición: se obtiene mediante cargado dinámico. No se ha explorado.

Esta marca no será elegida para el proyecto pues no cumplen los campos mínimos exigidos en *parsing*, carece de descripción de productos.

Mecanismos de protección contra web scraping

Dado que la marca no será elegida para el proyecto, no se ha investigado cómo es el mecanismo anti *web scraping*. Sin embargo, al pertenecer al grupo Inditex, al igual que Zara, se presupone un sistema de defensa parecido, que no penaliza el volumen de peticiones sino una ratio alta que pueda saturar al servidor.

5.1.5. Pull&Bear

Contexto

Pull&Bear es una marca de ropa perteneciente al grupo español Inditex fundada en 1991.

Cuenta con un total de 970 tiendas repartidas por todo el mundo y en 2022 reportó unos ingresos de 1,876 billones de euros en todo el mundo.

La ropa que vende tiene una clara orientación al público juvenil recogiendo las últimas tendencias internacionales y con influencias claras de la cultura urbana. Se trata de una marca de *fast fashion* por lo que sus diseños son muy cambiantes a lo largo del año y cuenta con un gran catálogo de productos.

Análisis

Análisis de los parámetros de la “Tabla 1” para *parsing* y de los posibles métodos de *crawling*:

- Parsing:
 - URL Imagen: se hace un cargado dinámico muy complejo de las imágenes pues ni siquiera están las URLs de las imágenes en el HTML inicial. No se obtendrá este campo.
 - Composición: se hace cargado dinámico. No se ha explorado.

No será elegida pues no cumplen los campos mínimos exigidos en *parsing*, la obtención de los campos de URL Imagen es ciertamente complicada y otras marcas tienen una extracción más sencilla de datos.

Mecanismos de protección contra web scraping

Dado que la marca no será elegida para el proyecto, no se ha investigado cómo es el mecanismo anti *web scraping*. Sin embargo, al pertenecer al grupo Inditex, al igual que Zara, se cree que dispone de un sistema de defensa parecido, que penaliza una ratio alta que pueda saturar al servidor, pero no el volumen de peticiones.

Finalmente se elegirán las marcas Zara, H&M y Carhartt WIP. Son marcas muy conocidas entre el público juvenil en el caso de H&M y Zara y en menor medida Carhartt WIP. Todo el mundo, especialmente el público juvenil, está muy familiarizado con el tipo de ropa que hacen, por lo que la aplicación web a desarrollar sería de gran utilidad a la hora de realizar comparaciones entre dichas marcas. Las tres contienen todos los campos mínimos y se pueden obtener de forma más sencilla que en otras páginas web, criterio clave a la hora de descartar otras marcas e incluir estas.

5.2. Desarrollo

En esta segunda fase del proyecto, primero se ha desarrollado toda la parte de *web scraping* para crear el motor de búsqueda de los productos de Zara, H&M y Carhartt WIP y posteriormente se ha construido la página web que mostrará todos los resultados obtenidos al público.

5.2.1. Web Scraping

La parte de *web scraping* del desarrollo del proyecto tiene como protagonista al *framework* Scrapy, que ha sido el utilizado para la creación del motor de búsqueda.

En esta sección se expondrán los componentes usados más relevantes.

Los objetos de la clase *Item* que serán poblados será tratado en “5.2.1.1 Ítems”. Las clases *spider* creadas para recorrer Zara, H&M y Carhartt WIP en “5.2.1.2 Spider de Zara”, “5.2.1.3 Spider de H&M” y “5.2.1.4 Spider de Carhartt WIP” respectivamente. Cada sección de las arañas tendrá distintas partes, un sumario para recopilar las consideraciones principales que se tendrán en cuenta al diseñar la clase *spider*, cómo se ha realizado el

rechazo de ítems no deseados, cómo es el *parsing* y cómo se ha realizado el *crawling*. El procesado realizado gracias a las clases de *Item Pipelines* en “5.2.1.5 Item Pipelines”, el tratamiento a las respuestas del servidor gracias al *downloader middleware* “5.2.1.7 Middleware”, el cómo se ha realizado la ejecución de las arañas y cada cuánto en “5.2.1.8 Script de lanzamiento de las arañas” y por último, la configuración del *framework* utilizada en “5.2.1.6 Configuración”.

5.2.1.1. Ítems

En “Tabla 3” se muestran los atributos que tendrán los objetos de la clase *PrendasItem*, derivada de la clase *Item* de Scrapy con una pequeña descripción de qué contienen y del tipo de datos que son.

Campo/Parámetro	Descripción
ID	Identificador único para cada producto. Será de tipo <i>int</i> ⁵² de 10 dígitos
Nombre	Nombre del producto. Será tipo <i>string</i> ⁵³
Precio	Precio del producto. Será tipo <i>float</i> ⁵⁴
Género	Género, hombre o mujer, de la prenda. Será tipo <i>string</i> .
Color	Color de la prenda. Será tipo <i>string</i> .
Descripción	Pequeño texto descriptivo del producto. Será tipo <i>string</i> .
Composición	Materiales que conforman la prenda. Si la prenda está compuesta por varias partes, la más grande será la que marque este campo. Será tipo <i>string</i> .
Marca	La marca de la que es la prenda. Será tipo <i>string</i> .
Fecha de obtención	Fecha en la que se ha obtenido el producto mediante <i>web scraping</i> . Será tipo <i>datetime</i> ⁵⁵ .
URL del producto	URL original del producto. Será tipo <i>string</i> .
URL de la imagen	URL de la imagen principal del producto. Será tipo <i>string</i> .
Ruta de la imagen	Ruta en el sistema de archivos del servidor de la imagen principal. Será tipo <i>string</i> .
Ruta de la miniatura	Ruta en el sistema de archivos del servidor de la miniatura de la imagen principal. Será tipo <i>string</i> .

Tabla 3 - Campos que tiene cada ítem

⁵² Abreviatura de *integer*, que en español significa entero. Se refiere a un tipo de dato que almacena números enteros.

⁵³ En español, cadena de caracteres. Se refiere a un tipo de dato que almacena cadenas de caracteres.

⁵⁴ En español, números con coma flotante. Se refiere a un tipo de dato que almacena números decimales.

⁵⁵ En español, fecha. Se refiere a un tipo de datos que almacena fechas que tienen día, mes, año y horas, minutos y segundos.

En “Ilustración 33” se muestra una captura de la clase *PrendasItem*, cuyos objetos serán poblados con los datos reales de los productos.

```
class PrendasItem(Item):
    id = Field()
    nombre = Field()
    precio = Field()
    url_imagen = Field()
    ruta_imagen = Field()
    ruta_miniatura = Field()
    genero = Field()
    color = Field()
    descripcion = Field()
    marca = Field()
    composicion = Field()

    url = Field()
    fecha = Field()
```

Ilustración 33 - Clase derivada de Item en Scrapy utilizada

5.2.1.2. Spider de Zara

Sumario

Las principales consideraciones que se han tenido en cuenta para elaborar la araña de Zara son las siguientes.

- El único parámetro de la tabla “Tabla 3” que no se va a obtener es el de composición.
- El *crawling* se realizará a través del *sitemap* de productos en español “sitemap-es-es.xml.gz”.
- Del *sitemap* sólo se seguirán los enlaces tipo -l y tipo -p. Los enlaces tipo -l se seguirán de forma diaria, pues se actualizan día a día y los enlaces tipo -p una vez cada siete días.
- Aproximadamente se calcula que se obtendrán alrededor de 12000 productos pudiendo variar el número. No quiere decir que cada semana se añadan 12000 nuevos, sino que en total es el número, pues muchos se repetirán semana tras semana.

Rechazo de ítems no deseados

Dado que los productos elegidos para este proyecto son únicamente prendas de ropa normales de hombre y de mujer, en “Ilustración 34” hay una expresión regular que analiza las URLs del *sitemap* de forma que si aparece una de estas palabras la ignora. Las palabras de la dirección son en realidad el nombre del producto, que está incluido en la URL.

```
sitemap_rules = [
    """^((?!/help|-mkt[0-9]+|-pG[0-9]+|-pT[0-9]+|-p[1-9]|look-|kids|nino|girl|boy|baby|
    pendiente|abitorio|accessori|accesori|pack|boxer|mochila|calcetin|sombra|bolso|gafal|
    lip-|zapat|shoe|mini-|sujetador|refill|wallpaper|perfume|beachwear|
    beauty|lingerie|bag).)*""", 'parse')]
```

Ilustración 34 - Expresión regular de rechazo de URLs de productos que no son los elegidos

Como se ha mencionado también anteriormente, hay ciertos productos no deseados, como es el caso de productos de niños, que pueden no ser capturados a través de la URL, en dicho caso se rechazarán mediante el uso de *Item Pipelines* como se indica en “5.2.1.4 Item Pipelines”.

Parsing

Este apartado se subdivide en dos partes, el *parsing* de los productos y por ende, de los enlaces tipo -p, y el *parsing* de los enlaces tipo -l. Se mostrarán las expresiones XPath utilizadas para obtener la información relevante:

- Enlaces tipo -p: únicamente se extraen los datos para llenar los campos relevantes excepto el de composición, que se explicará cómo se rellena en “5.2.1.5 Item Pipelines”. Se hace *parsing* de estos enlaces en el método *parse_producto()*.

```
il.add_xpath('nombre', '//*[@class="product-detail-info_header-name"]/text()')
il.add_xpath('precio', '//*[@class="money-amount_main"]/text()', re='[0-9,]+')
il.add_xpath('url_imagen', '//meta[@property="og:image"]/@content')
il.add_xpath('genero', '(//*[@itemprop="name"])[2]/text()')
il.add_xpath('color', '//*[contains(@class, "product-color-extended-name")]/text()')
il.add_xpath('descripcion', '//*[@class="expandable-text_inner-content"]/p/text()', Join())
il.add_value('marca', 'Zara')

il.add_value('url', response.url)
il.add_value('fecha', datetime.now())
```

Ilustración 35 - Parsing de los enlaces tipo -p

- Enlaces tipo -l: de cada página de enlace tipo -l se extraen todos los enlaces tipo -p con diferentes expresiones XPath, se eliminan los duplicados y se envía una petición a cada enlace -p para que se haga después el *parsing* propio de este tipo de enlaces, como se ha visto arriba. Se hace *parsing* de estos enlaces en el método *parse_lista()*.

```
links_1 = response.xpath('//*[@class="product-link_item product-grid-product-info_name link"]/@href').getall()
links_2 = response.xpath('//*[@class="media-region link"]/@href').getall()
links_3 = response.xpath('//*[@class="product-link product-grid-product_link link"]/@href').getall()

links = links_1 + links_2 + links_3

links_unicos = list(set(links))

for link_p in links_unicos:
    if re.search('-p[0-9]+', link_p) and link_p not in self.productos_bd:
        yield Request(link_p, callback = self.parse_producto)
```

Ilustración 36 - Parsing de los enlaces tipo -l

Crawling

Se recorrerá el *sitemap* “sitemap-es-es.xml.gz” para ir obteniendo los enlaces tipo -l y -p. Para realizar esta tarea de forma más sencilla, se ha utilizado la clase derivada de *spider*, *SitemapSpider*, como clase base sobre la que crear la araña. Esta clase de Scrapy está especializada en recorrer y tratar con *sitemaps*.

```
class ZaraSpider(SitemapSpider):
    name = "zara"
```

Ilustración 37 - Clase SitemapSpider de Zara

Se ha puesto como *sitemap* a recorrer la dirección completa de “sitemap-es-es.xml.gz” en el atributo *sitemap_urls*.

```
sitemap_urls = ['https://www.zara.com/sitemaps/sitemap-es-es.xml.gz']
```

Ilustración 38 - Atributo *sitemap_urls* en Zara

Y para poder realizar el seguimiento de URLs tipo -l de forma diaria y los tipo -p de forma semanal, se ha utilizado el método *sitemap_filter()* cuya función es la de recorrer el *sitemap* filtrando los enlaces en base a las etiquetas. En este caso, en base al valor de la etiqueta de XML “changefreq”, que puede tomar el valor *daily* en enlaces tipo -l, se recorren de forma diaria, o *weekly* en enlaces tipo -p que se recorrerán los lunes. Antes de ejecutarse este método *sitemap_filter()*, a las URLs del *sitemap* se aplica la expresión regular de *sitemap_rules* en “Ilustración 34” y posteriormente las entradas restantes son las que se pasan como parámetro *entries*⁵⁶ al método *sitemap_filter()* de forma que como *entries* únicamente están los enlaces tipo -l y -p deseados.

```
def sitemap_filter(self, entries):
    for entry in entries:
        if entry.get('changefreq'):
            if entry['changefreq'].lower() == 'weekly' and (
                datetime.today().date().strftime("%A").lower() == 'monday'):
                yield entry
            elif entry['changefreq'].lower() == 'daily':
                yield entry
```

Ilustración 39 - Método *sitemap_filter()* de Zara

Después del método *sitemap_filter()* se crean objetos *Request* con las URLs de las entradas del mapa que sean devueltas por el método y la respuesta a dichas peticiones, como se ha comentado en “3.4.1.1 Conceptos básicos”, se gestiona en el método *parse()*. Una vez en este método, se hace un filtrado para que la respuesta a cada tipo de enlace sea tratada de forma distinta como se ha mencionado en el apartado de parsing.

⁵⁶ En español, entradas.

```
if re.search('-p[0-9]+', response.url):
    return self.parse_producto(response)

if re.search('-l[0-9]+', response.url):
    return self.parse_lista(response)
```

Ilustración 40 - Redirección a cada tipo de enlace con su método gestor

Por último, se ha observado que para una misma página de tipo -l, de un día para otro hay enlaces tipo -p repetidos y para evitar acceder a dichos enlaces repetidos de forma diaria y tener una pérdida clara de rendimiento, se ha obtenido una lista de las URLs de los productos de Zara almacenados en la página web y se pasan a la araña de Zara al iniciarse como se mostrará a continuación. Dicha lista de coteja cada vez que se obtiene un enlace tipo -p dentro de un -l como se muestra en *productos_bd* en “Ilustración 36”, que en caso de estar repetido, no se recorre.

```
def __init__(self, productos_bd=None, *args, **kwargs):
    super().__init__(*args, **kwargs)

    self.productos_bd = productos_bd
    if self.productos_bd:
        self.productos_bd = [tupla[0] for tupla in json.loads(productos_bd)]
    else:
        self.productos_bd = []
```

Ilustración 41 - Paso de lista de IDs de productos de Zara al iniciarse la araña de Zara

5.2.1.3. Spider de H&M

Sumario

Las principales consideraciones que se han tenido en cuenta para elaborar la araña de H&M son:

- Para el *crawling* se utilizará la API de H&M que devuelve un archivo JSON con todos los ítems de hombres y mujeres.
- En los atributos de los objetos del archivo JSON de la API se pueden obtener todos los campos especificados en “Tabla 3” excepto los de composición y descripción, para los que será necesario solicitar el HTML de cada producto para obtenerlos.
- Aproximadamente, se obtendrán en la primera ejecución de *web scraping* 13000 ítems. No se obtendrán tantos todas las semanas.

Rechazo de ítems no deseados

El rechazo de ítems en vez de realizarse a nivel de URL como en Zara y Carhartt, se ha hecho mediante los atributos *title* y *category* de los objetos JSON.

```
busqueda_categoria = re.search("""accessorie|kid|care|lingerie|underwear|shoe|
                                |swim|sock|slipper|beauty""", elemento['category'])
busqueda_nombre = re.search('pack', elemento['title'], re.IGNORECASE)
```

Ilustración 42 - Expresión regular de rechazo de productos no deseados por nombre y categoría H&M

Parsing y Crawling

En H&M el proceso de *web scraping* es el más complejo del proyecto y se mezclan los conceptos de *parsing* y *crawling*, por lo que se tratarán juntos como apartado.

Dado que no se recorre un *sitemap*, la araña de H&M se declara como una clase derivada de *spider*.

```
class HmSpider(scrapy.Spider):
    name = "ache"
```

Ilustración 43 - Clase Spider de H&M

Las URLs iniciales se declaran en el atributo *start_urls*, que son las páginas de “Ver Todo” de hombre por un lado y de mujer por otro.

```
start_urls = ["https://www2.hm.com/es_es/hombre/compra-por-producto/view-all.html",
              "https://www2.hm.com/es_es/mujer/compra-por-producto/view-all.html"]
```

Ilustración 44- Atributo start_urls en H&M

Se envía una petición a ambas páginas y la respuesta se procesa en el método *parse()*.

El procesamiento comienza con averiguar el número total de productos que hay para cada género con la variable *numero_articulos* y después se llama a una función para generar la petición a la API JSON. La variable *tamano_bloque* indica el número de ítems que debe tener el archivo JSON devuelto.

```
def parse(self, response):
    numero_articulos = re.search('[0-9]+', response.xpath('//*[@class="filter-pagination"]/text()').get()).group()
    tamano_bloque = 500
    if re.search('hombre', response.url):
        genero = 'hombre'
        prefijo_genero = 'fa5b'
        return self.crear_peticiones_json(genero, prefijo_genero, numero_articulos, tamano_bloque)

    elif re.search('mujer', response.url):
        genero = 'mujer'
        prefijo_genero = '30ab'
        return self.crear_peticiones_json(genero, prefijo_genero, numero_articulos, tamano_bloque)
```

Ilustración 45 - Preparación para llamar a la API JSON de H&M

Dado que no se piden el total de ítems de una vez sino en bloques de tamaño *tamano_bloque*, se hace un bucle que hace las llamadas necesarias a la API JSON por bloques hasta cubrir el total de ítems. Es necesario ir modificando la variable *offset* para que vaya cogiendo ítems diferentes.

Aspectos Relevantes del Desarrollo

```
numero_articulos = int(numero_articulos)
num_bloques = numero_articulos // tamano_bloque
resto = numero_articulos % tamano_bloque

offset = 0
requests = []
for i in range(num_bloques + 1):
    if i == num_bloques and resto != 0:
        url_json = f"https://www2.hm.com/es_es/{genero}/compra-por-producto/view-all/_jcr_content/main/
productlisting_{prefijo_genero}.display.json?sort=stock&image-size=small&image=model&offset={offset}&
page-size={resto}"
    else:
        url_json = f"https://www2.hm.com/es_es/{genero}/compra-por-producto/view-all/_jcr_content/main/
productlisting_{prefijo_genero}.display.json?sort=stock&image-size=small&image=model&offset={offset}&
page-size={tamano_bloque}"

    requests.append(Request(url=url_json, cb_kwargs={"genero": genero.title(),
                                                       callback=self.parse_item_but_description, errback=self.gestion_errores}))
    offset += tamano_bloque
return requests
```

Ilustración 46 - Llamada a la API JSON de H&M

En “Ilustración 47” se muestra cómo se gestiona el archivo JSON. Se va iterando ítem a ítem por el archivo JSON convertido en lista de objetos y se van poblando los ítems. No están terminados pues faltan los campos de descripción y composición, que hay que solicitar el HTML de los productos para obtenerlos.

```
lista_items = json.loads(response.body)
for elemento in lista_items.get('products'):
    if elemento.get('title') and elemento.get('category'):
        busqueda_categoria = re.search("""accessorie|kid|care|lingerie|underwear|shoe|
                                         swim|sock|slipper|beauty""", elemento['category'])
        busqueda_nombre = re.search('pack', elemento['title'], re.IGNORECASE)

    if busqueda_categoria is None and busqueda_nombre is None:
        il = ItemLoader(item=PrendasItem(), response=response)
        il.default_output_processor = TakeFirst()

    if elemento.get('link'):
        url_item = f"https://www2.hm.com{elemento['link']}"

    il.add_value('nombre', elemento.get('title'))

    precio = re.search('[0-9,]+', elemento.get('price'))
    if precio:
        il.add_value('precio', precio.group())

    if elemento.get('image') and len(elemento.get('image')) > 0 and elemento.get('image')[0].get('src'):
        il.add_value('url_imagen', re.sub('style', 'main', f"https:{elemento['image'][0]['src']}"))

    il.add_value('genero', genero)

    if "0" != elemento.get('swatchesTotal') and elemento.get('swatches') and (
            len(elemento.get('swatches')) > 0 and elemento.get('swatches')[0].get('colorName')):
        il.add_value('color', elemento.get('swatches')[0].get('colorName'))

    il.add_value('marca', 'H&M')
    il.add_value('url', url_item)
    il.add_value('fecha', datetime.now())

il.add_value('descripcion', respuesta)
```

Ilustración 47 - Parsing del archivo JSON de H&M

Se gestiona la respuesta al HTML de los productos y se obtiene mediante XPath la descripción. La composición es más complicada de obtener, pues como indica “5.1.2 H&M”, no hay un apartado obvio de composición. El criterio que se ha seguido para obtener la composición es extraer la composición principal si sólo hay una, si hay varias zonas de composición escoger la exterior, superior o delantera. Después de esta función, el ítem ya ha sido poblado y pasa a los *Item Pipelines*.

```

il.add_xpath('descripcion', '//*[@id="section-descriptionAccordion"]/descendant::p/text()')

ul = response.xpath('//*[@id="section-materialsAndSuppliersAccordion"]/ul[1]')

composicion = None
if len(ul.xpath('./*')) == 1:
    composicion = ul.xpath('./*/*/text()').get()
else:
    i = 0
    encontrado = False
    for grandchild in ul.xpath('./*/child::*'):
        contenido = grandchild.xpath('text()').get()
        if encontrado == True:
            composicion = contenido
            break
        if re.search('exterior|superior|delantera', contenido):
            encontrado = True
        if re.search('%', contenido) and i == 0:
            composicion = contenido
            break

    i += 1

il.add_value('composicion', composicion)

```

Ilustración 48 - Parsing del HTML de productos para añadir composición y descripción

5.2.1.4. Spider de Carhartt WIP

Sumario

Las principales consideraciones que se han tenido en cuenta para elaborar la araña de Carhartt son:

- El *crawling* se realizó recorriendo el *sitemap* “es/sitemap-products.xml”.
- El número aproximado de productos que se espera obtener es 800 la primera vez y cada semana ninguno hasta el cambio de temporada que se renovará el *sitemap*. Se extraerán todas las prendas mucho antes que Zara y H&M.
- La frecuencia de *web scraping* será de una vez por semana y aunque es poco probable que aparezcan nuevos ítems, puede que los precios varíen.
- Únicamente hay un tipo de enlaces en el *sitemap*, que es el de enlaces que llevan a productos.

Rechazo de ítems no deseados

Dado que los productos elegidos para este proyecto son únicamente prendas de ropa normales de hombre y de mujer, el filtrado de productos no deseados se hará detectando palabras clave de la categoría de la prenda en la URL.

```

sitemap_rules = [
    ('^((?!accesorio|gadget|ropa-interior|zapato).)*$', 'parse')]

```

Ilustración 49 - Expresión regular de rechazo de URLs de productos que no son los elegidos en Carhartt

Parsing

Sólo hay un único tipo de enlaces, el de los productos, por lo que el análisis sintáctico de los archivos HTML será igual para todos como se muestra en la siguiente imagen.

```
il.add_xpath('nombre', '//*[@class="product-title"]/descendant::*[@class="title"]/text()')
il.add_xpath('precio', '//*[contains(@class, "price country_eur")]/text()')

il.add_xpath('url_imagen', '//*[@data-mobile]/@data-mobile')

if re.search('hombre', response.url) is not None:
    genero = 'hombre'
    il.add_value('genero', genero.title())
elif re.search('mujer', response.url) is not None:
    genero = 'mujer'
    il.add_value('genero', genero.title())

campo_color = response.xpath('//[@property="og:description"]/@content').get()

if campo_color is not None:
    indice_ultima_barra = campo_color.rfind("/")
    if indice_ultima_barra > 0:
        campo_color = campo_color[indice_ultima_barra:]
        color = re.search('(?<=)/\s([a-zA-ZÁÉÍÓÚáéíóú]+)', campo_color)
        if color is not None:
            il.add_value('color', color.group(1))

il.add_xpath('descripcion', '//*[@class="inner_productDescription"]/p/text()')
il.add_value('marca', 'Carhartt')
il.add_xpath('composicion', '//li[contains(text(), "%")]/text()')

il.add_value('url', response.url)
il.add_value('fecha', datetime.now())
```

Ilustración 50 - Parsing de los productos en Carhartt

La obtención del color es compleja. Una cadena de ejemplo que contiene el color en Carhartt es la siguiente, de la que se corresponde el artículo de “Ilustración 51”. Hay que obtener el la subcadena “Verde”:

“Carhartt WIP Awake NY Teddy Jacket | Chaquetas ligeras - Awake NY Dark Green / Awake NY Wax / Verde | comprar online, entrega rápida ► envío gratuito”



Ilustración 51 - Artículo de Carhartt

Para obtener el color, en este caso “Verde”, primero se selecciona el último carácter “/”, a continuación. se selecciona la subcadena a partir del carácter “/”, es decir:

“Verde | comprar online, entrega rápida ► envío gratuito”

Y por último, se escoge la primera cadena de caracteres a partir del primer espacio, es decir, “Verde”.

Crawling

Se recorrerá la página web a través del *sitemap* en español de productos “es/sitemap-products.xml”. El *sitemap* de Carhartt WIP es pequeño y todas sus URLs tienen el mismo formato y llevan a productos.

Al igual que en Zara, se ha utilizado la clase derivada de *spider*, SitemapSpider.

```
class CarharttSpider(SitemapSpider):
    name = "carhartt"
```

Ilustración 52 - Clase SitemapSpider de Carhartt

Y se pone como dirección del *sitemap* a recorrer la dirección completa de “es/sitemap-products.xml” en el atributo *sitemap_urls*.

```
sitemap_urls = ['https://www.carhartt-wip.com/es/sitemap-products.xml']
```

Ilustración 53 - Atributo sitemap_urls en Carhartt

Como no hace falta hacer un tratamiento especial de las URLs del *sitemap* mediante las etiquetas XML no es necesario implementar *sitemap_filter()* como en Zara.

Por lo tanto, primero se aplican las reglas de “Ilustración 49”, se envía petición para dichas URLs y la respuesta se gestiona en el método *parse()* como se ha explicado en **Parsing**.

5.2.1.5. *Item Pipelines*

Las *Item Pipelines* o tuberías de ítems, son clases secuenciales que procesan los ítems una vez han sido poblados y devueltos por las arañas. Se utiliza para el procesado el método *process_item()*. Es en estas clases donde se pueden rechazar ítems, eso se realiza con la excepción *DropItem*, por lo que las siguientes clases secuenciales no recibirán dicho ítem eliminado.

Ahora se explicarán las diferentes clases usadas en la tubería de ítems en orden:

1. **Clase PipelineErroneos**: esta clase rechaza ítems que tengan ciertos campos obligatorios vacíos.

```
if item.get('nombre') is None:  
    raise DropItem(f"Descartado {item.get('url')}: articulo sin nombre")  
if item.get('precio') is None:  
    raise DropItem(f"Descartado {item.get('url')}: articulo sin precio")  
if item.get('url_imagen') is None:  
    raise DropItem(f"Descartado {item.get('url')}: articulo sin url_imagen")  
if item.get('genero') is None:  
    raise DropItem(f"Descartado {item.get('url')}: articulo sin genero")  
if item.get('descripcion') is None:  
    raise DropItem(f"Descartado {item.get('url')}: articulo sin descripcion")
```

Ilustración 54 - Clase PipelineErroneos para filtrar productos con campos vacíos

No cuentan campos como fecha, marca o URL, pues no se obtienen mediante *parsing* o XPath y se encuentran en todos los ítems o campos como ruta_minatura o ruta_imagen. Los campos color y composición vacíos se tratan en “5.2.1.5 Item Pipelines” además de que composición no es un campo obligatorio.

2. Clase PipelineNinos: su valor real es únicamente para los productos obtenidos en Zara ya que rechaza los ítems que tengan valor “Niños” en el XPath obtenido para el género.

```
if item['genero'].upper() == 'NIÑOS':  
    raise DropItem(f"Descartado {item['nombre']}: articulo de ninos")
```

Ilustración 55 - Clase PipelineNinos para filtrar productos de niños de Zara

3. Clase PipelineLimpieza: esta clase es la que hace toda la estandarización de los campos para que todos tengan valores o formatos parecidos. Esto hace que se puedan realizar operaciones sobre todos los ítems por igual y que sean homogéneos para el cliente.

- Nombre: se estiliza poniendo la primera letra de cada palabra en mayúsculas y en caso de ser el ítem de la marca Carhartt, se añade una traducción en el nombre basándose en la categoría de la URL a la que pertenece. Esto facilita el ver de qué prenda se trata para usuarios no bilingües y para búsquedas en la base de datos.

```

def limpieza_nombre(self, item):
    if item.get('marca'):
        if item['marca'] != 'Carhartt':
            return item['nombre'].title()

    traduccion = None

    if re.search('camiseta', item['url']):
        if re.search('larga', item['url']):
            traduccion = 'Camiseta Manga Larga'
        else:
            traduccion = 'Camiseta Manga Corta'
    if re.search('chaqueta', item['url']):
        traduccion = 'Chaqueta'
    if re.search('jersei', item['url']):
        traduccion = 'Jerséi'
    if re.search('sudadera', item['url']):
        if re.search('jogger', item['url']):
            traduccion = 'Pantalón Largo'
        else:
            traduccion = 'Sudadera'
    if re.search('camisa', item['url']):
        traduccion = 'Camisa'
    if re.search('pantalon', item['url']):
        traduccion = 'Pantalón Largo'
    if re.search('bermudas|shorts', item['url']):
        traduccion = 'Pantalón Corto'
    if re.search('mono', item['url']):
        traduccion = 'Mono'

    return f"{item['nombre'].title()} - {traduccion}"

```

Ilustración 56 - Tratamiento de nombres en la clase PipelineLimpieza

- Precio: se traduce de cadena de caracteres a *float* para poder realizar búsquedas por orden en la base de datos.
- ID: se crea un valor identificador para cada producto. Ha sido difícil encontrar el concepto adecuado, porque se buscaba un número que siempre sea igual para cada producto y que fuera un número exclusivamente entero, ni cadena de caracteres ni hexadecimal, de entre 9 y 12 dígitos para que las búsquedas y comparaciones se realizasen de la forma más rápida posible.

Todas estas condiciones dejaban fuera métodos populares como la opción de *auto increment* para crear los identificadores en la propia base de datos, que al introducir una tupla nueva el identificador se generaba incrementándose en uno sobre el mayor existente o números aleatorios en base a fechas, porque hacen que para un mismo ítem se generen identificadores diferentes cada vez.

La solución ha sido elegir un algoritmo de *hashing*, convertir el *hash* de un hexadecimal a entero y coger los 10 dígitos menos significativos. El algoritmo elegido ha sido MD5, que, aunque no sea muy fiable para *hashing*, para esta tarea no debería de haber problema y es el más rápido. Como parámetro de entrada al *hash* se ha elegido la URL, pues es el único atributo único de los ítems.

```

def crear_id(self, url):
    s = url.encode()
    return int(hashlib.md5(s).hexdigest(), 16) % 10000000000

```

Ilustración 57 - Creación del ID en la clase PipelineLimpieza

- Composición: al tratarse de un campo no obligatorio y que va a haber bastantes productos con este campo vacío, se ha decidido centralizar la gestión de este campo en la clase de *PipelineLimpieza*. Si está vacío se le asigna valor “sin_composicion” y en caso de ser de Carhartt, que la cadena tenía unos caracteres extraños de Unicode, se normaliza.

```
def limpieza_composicion(self, item):
    if item.get('composicion') is None:
        return 'sin_composicion'

    if item.get('marca'):
        if item['marca'] == 'Carhartt':
            composicion = re.sub('^[^0-9]+', '', item['composicion'])
            #composicion_normalizada = unicodedata.normalize('NFKD', composicion)
            composicion_normalizada = unicodedata.normalize('NFC', composicion)

    return composicion_normalizada
return item['composicion']
```

Ilustración 58 - Tratamiento de la composición en la clase PipelineLimpieza

- Color: primeramente, hay que hacer una limpieza de la cadena de color obtenida con el XPath pues hay información que no resulta interesante. Los distintos formatos de cadenas extraídas de Zara se muestran con los siguientes ejemplos. El tratamiento también sirve para H&M y Carhartt:

1. Marino | 5584/461
2. Beige claro | 4467/003
3. Crudo / Azul | 3057/051
4. 0120/510
5. azul-empolvado | 6045/366 o Marrón/Estampado

Ilustración 59 - Formatos de cadenas extraídas con XPath para el campo color

Se obtendrán, respectivamente de las cadenas de “Ilustración 59”, los colores marino, beige, azul, nada y azul o marrón mediante el uso de expresiones regulares. Si se da el caso 4, no tendrá color y se le asignará “sin_color”.

```
def limpieza_color(self, color_entrada):
    if color_entrada == None:
        return 'sin_color'

    # Primero se comprueba si tiene el formato más común, el 2 (aunque también vale para el 5)
    objeto_regex_color = re.match('^(a-zA-Záéíóú)+\s?[a-zA-Záéíóú]*', color_entrada.title())

    # Si la cadena de color es un número (caso 4)
    if objeto_regex_color is None:
        return 'sin_color'

    """
    Comprobamos si se trata de un formato de color normal (casos 1, 2 o 5) o formato "X / Y"
    - Normal: seleccionamos la primera parte del color, por ejemplo, de Verde Claro, Verde
    - X / Y: seleccionamos la parte Y de color, por ejemplo, de Crudo / Azul, Azul
    """
    objeto_regex_crudo = re.match('^(a-zA-ZÁÉÓÚáéíóú]+\s/\s([a-zA-ZÁÉÓÚáéíóú]+)', color_entrada.title())

    # Si no se da formato "X / Y"
    if objeto_regex_crudo is None:
        color_final = self.asignar_color(objeto_regex_color.group(1))

    # Si se da formato "X / Y"
    else:
        color_final = self.asignar_color(objeto_regex_crudo.group(1))

    return color_final
```

Ilustración 60 - Limpieza del campo color en clase PipelineLimpieza

Con la ilustración anterior simplemente se obtiene la cadena del color pero como se ha mencionado anteriormente, en Zara sobre todo, el campo color podía tomar una gran cantidad de valores. Concretamente han sido observados 112. En H&M y Carhartt muchos menos. Dado que ese gran número de colores hace que la gestión del campo color sea muy complicada en la base de datos, se ha hecho una tabla de asignaciones como la siguiente:

Negro	Negro, Antracita, <i>Black</i>
Gris	Gris, Cemento, Piedra, Grises, Acero, Hielo, Plomo, <i>Grey</i> , Grisáceo
Beige	Beige, Arena, Mantequilla, Camel, Cava, Crema, Crudo, Vigoré, Barquillo, Cuerda, Vainilla, Whisk
Marrón	Marrón, Chocolate, Tabaco, Tostado, Visón, Toffe, Ocre, Caramelo, Tierra, Terracota, Arcilla, Bronce, <i>Brown</i>
Azul	Azul, Índigo, Marino, Turquesa, Azules, Celeste, Azulina, Azulado, Azulón, Cobalto, <i>Blue</i>
Verde	Verde, Khaki, Kaki, Lima, Botella, Aceite, Esmeralda, Oliva, Verdosso, Menta, Pistacho, <i>Green</i>
Amarillo	Amarillo, Mostaza, Paja, <i>Yellow</i>
Naranja	Naranja, Naranjas, Calabaza, Teja, Mandarina, Melocotón, Salmón, <i>Orange</i> , Albaricoque
Rojo	Rojo, Rojos, Burdeos, Granate, Coral, Vino, Burgundy, Caldero, <i>Red</i>
Rosa	Rosa, Magenta, Fucsia, Rosado, Marsala, Rosas, Chicle, <i>Pink</i>
Morado	Morado, Lila, Púrpura, Malva, Berenjena, Violeta, Lavanda, <i>Purple</i>
Plateado	Plateado, Plata
Dorado	Dorado, Oro
Blanco	Blanco, Hueso, <i>White</i>
Multicolor	Multicolor, Estampado, Leopardo, Varios, Serpiente, Rayas

Tabla 4 - Tabla de asignación de colores en PipelineLimpieza

De forma que a cada color con nombre poco común se le asigne uno conocido por todos y sea posteriormente más fácil su gestión en la base de datos al hacer búsquedas y para el usuario seleccionarlo.

- Fecha: se hace un formateo específico para que siga el formato que tienen las fechas en la base de datos.
 - URL: se eliminan los parámetros que se le pasan después de “?” con la interrogación incluida.
4. Clase PipelineImagenes: en esta clase lo que se hace es descargar la imagen del producto cuya URL está almacenada en el campo url_imagen y se descarga a un directorio específico para cada marca. El nombre que se le da a cada imagen es el de su ID. Además, a partir de esa imagen se genera una miniatura que queda guardada en un directorio llamado *miniaturas*. La estructura de directorios queda reflejada en la siguiente imagen:

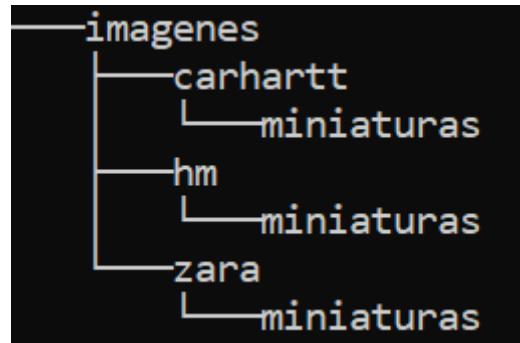


Ilustración 61 - Estructura de directorios donde se almacenan las imágenes y miniaturas de los productos

También se descartarán ítems cuya imagen no haya podido ser descartada y cuya marca sea desconocida.

```

def item_completed(self, results, item, info):
    print(results)
    nombres_imagenes = [x["path"] for ok, x in results if ok]
    if not nombres_imagenes:
        raise DropItem(f"Descartado {item.get('url')}: articulo sin imagen descargada")
    if not nombres_imagenes[0]:
        raise DropItem(f"Descartado {item.get('url')}: articulo sin imagen descargada")
    match(item['marca']):
        case 'Zara':
            ruta_relativa = "zara/"
        case 'H&M':
            ruta_relativa = "hm/"
        case 'Carhartt':
            ruta_relativa = "carhartt/"
        case _:
            raise DropItem(f"Descartado {item.get('url')}: articulo con marca desconocida")

    item['ruta_imagen'] = ruta_relativa + nombres_imagenes[0]
    item['ruta_minatura'] = ruta_relativa + 'miniaturas/' + nombres_imagenes[0]
    return item
  
```

Ilustración 62 - Asignación de rutas a las imágenes descargadas con la clase *PipelineImagenes*

5. Clase *PipelineBaseDatos*: la función de esta última clase en la tubería de ítems es la de insertar los ítems que hayan pasado todas las *Item Pipelines* a la base de datos.

Cada clase araña tendrá un conector de acceso a la base de datos que se activa cuando se inicializa la araña y se cierra cuando dicha araña acabe de hacer *web scraping* y se cierre.

```
def open_spider(self, spider):
    try:
        self.conn = mariadb.connect(
            user='scrapybot',
            password='aclassscraper',
            host='localhost',
            database='prendas'
        )

    except mariadb.Error as err:
        self.logger.error(f"Error al conectar a MariaDB: {err}")
    else:
        self.cursor = self.conn.cursor()

def close_spider(self, spider):
    self.cursor.close()
    self.conn.close()
```

Ilustración 63 - Inicialización y cierre del acceso a la base de datos en clase PipelineBaseDatos

La sentencia de inserción de ítems es bastante normal, cabe destacar que en caso de que se vaya a insertar un ítem ya existente se sobrescribe por si ha variado alguno de sus datos entre una extracción y otra.

```
def process_item(self, item, spider):
    sentencia = """
    INSERT INTO item (id, nombre, precio, url_imagen, genero, color, descripcion, marca, url, fecha)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    ON DUPLICATE KEY UPDATE
        nombre = VALUES(nombre),
        precio = VALUES(precio),
        url_imagen = VALUES(url_imagen),
        genero = VALUES(genero),
        color = VALUES(color),
        descripcion = VALUES(descripcion),
        marca = VALUES(marca),
        url = VALUES(url),
        fecha = VALUES(fecha),
        composicion = VALUES(composicion),
        ruta_imagen = VALUES(ruta_imagen),
        ruta_miniatura = VALUES(ruta_miniatura);
    """
    valores = (item['id'], item['nombre'], item['precio'], item['url_imagen'], item['genero'],
               item['color'], item['descripcion'], item['marca'], item['url'], item['fecha'],
               item['composicion'], item['ruta_imagen'], item['ruta_miniatura'])
```

Ilustración 64 - Inserción en la base de datos en clase PipelineBaseDatos

5.2.1.6. Configuración

La configuración se tratará a los niveles de araña, para el cuál los parámetros de configuración serán únicos de esta y para proyecto, que afectará a todos los componentes del proyecto.

Configuración a nivel de araña

- Zara: en la imagen siguiente se muestran las configuraciones de Zara. El parámetro *IMAGES_STORE* define la carpeta en la que serán almacenadas las imágenes descargadas de los productos de Zara.

```
custom_settings = {
    "IMAGES_STORE": 'imagenes/zara',
}
```

Ilustración 65 - Configuración de la araña de Zara

- H&M: en la imagen siguiente se muestra la configuración de H&M. El parámetro *IMAGES_STORE* indica el directorio donde se almacenarán los productos de H&M es diferente al de Zara.

```
custom_settings = {
    "IMAGES_STORE": 'imagenes/hm',
}
```

Ilustración 66 - Configuración de la araña de H&M

- Carhartt WIP: al igual que en Zara y H&M, la configuración de la araña de Carhartt se reduce a indicar el directorio de almacenamiento de las imágenes en *IMAGES_STORE*.

```
custom_settings = {
    "IMAGES_STORE": 'imagenes/carhartt',
}
```

Ilustración 67 - Configuración de la araña de Carhartt WIP

Configuración a nivel de proyecto

Se explicarán a continuación los parámetros más relevantes de la configuración a nivel de proyecto:

- Cabeceras HTTP y User-Agent: como se ha explicado en “3.3.2.3 Crawling: tipos y problemas”, la modificación de estos parámetros es clave para camuflar el tráfico del motor de búsqueda y evitar ser bloqueados.

```
USER_AGENT = "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0"

DEFAULT_REQUEST_HEADERS = {
    "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8",
    "Accept-Language": "en-US,en;q=0.5",
    "Accept-Encoding": "gzip, deflate, br",
    "Connection": "keep-alive",
    "Upgrade-Insecure-Requests": "1",
    "Sec-Fetch-Dest": "document",
    "Sec-Fetch-Mode": "navigate",
    "Sec-Fetch-Site": "none",
    "DNT": "1",
    "Cache-Control": "no-cache",
}
```

Ilustración 68 - Configuración de cabeceras y User-Agent

- Orden de envío de las peticiones: los parámetros de la siguiente imagen indican que el orden de envío de las peticiones es FIFO⁵⁷ (*First In First Out*) en vez de LIFO⁵⁸ (*Last In First Out*) que es el sistema por defecto.

```
DEPTH_PRIORITY = 1  
SCHEDULER_DISK_QUEUE = "scrapy.squeues.PickleFifoDiskQueue"  
SCHEDULER_MEMORY_QUEUE = "scrapy.squeues.FifoMemoryQueue"
```

Ilustración 69 - Configuración del orden de envío de las peticiones

- Seguimiento del archivo robots.txt: como se ha explicado en “3.3.2.3 Crawling: tipos y problemas” es importante seguir las indicaciones del archivo robots.txt para evitar ser bloqueados y para optimizar el tiempo no entrando en páginas irrelevantes para el propósito del proyecto.

```
ROBOTSTXT_OBEY = True
```

Ilustración 70 - Configuración de seguimiento de robots.txt

- Ratio de envío de peticiones: indica el número de segundos entre cada petición enviada a cada dominio web. Se pueden poner números decimales.

```
DOWNLOAD_DELAY = 1
```

Ilustración 71 - Configuración de ratio de envío de peticiones

- Cookies: las cookies se han habilitado pues ciertas páginas pueden llegar a bloquear el acceso en caso de navegar durante cierto tiempo por su web sin estar habilitadas.

```
COOKIES_ENABLED = True
```

Ilustración 72 - Configuración de cookies

- Orden de Item Pipelines: es en la configuración de proyecto donde se fija el orden que tendrán las clases de *Item Pipelines*. El número indica la prioridad siendo 0 la máxima prioridad y 1000 la mínima prioridad.

⁵⁷ En español, El Primero que Entra sale Primero, es una estructura de datos que simboliza una cola de las de toda la vida. La primera persona que entra en la cola es la primera que acaba, pero con objetos *Request* que serán peticiones HTTP en este caso.

⁵⁸ En español, El Último que Entra sale Primero, es una estructura de datos que simboliza una pila. Por ejemplo, en una pila de platos, el último que se pone es el primero que se quita, pero con objetos *Request* en que serán peticiones HTTP en este caso.

```
ITEM_PIPELINES = {
    "prendas.pipelines.erroneos.PipelineErroneos": 200,
    "prendas.pipelines.ninos.PipelineNinos": 300,
    "prendas.pipelines.limpieza.PipelineLimpieza": 400,
    "prendas.pipelines.imagenes.PipelineImagenes": 500,
    "prendas.pipelines.basedatos.PipelineBaseDatos": 600
}
```

Ilustración 73 - Configuración de orden Item Pipelines

- Orden de Downloader Middlewares: al igual que el orden de *Item Pipelines* se fija el orden de las clases de *Downloader Middlewares* en caso de haber varias. Al haber sólo una da igual el número de prioridad que se le asigne.

```
DOWNLOADER_MIDDLEWARES = {
    "prendas.middlewares.errorhttp.ErrorHttpDownloaderMiddleware": 500,
}
```

Ilustración 74 - Configuración de orden de Downloader Middlewares

- Reintentos: en caso de enviar una petición y no recibir respuesta por parte del servidor, el parámetro *RETRY_ENABLED* con valor *True*⁵⁹ habilita el repetir la petición. *RETRY_TIMES* indica el número de reintentos y *DOWNLOAD_TIMEOUT* cuántos segundos esperar a la respuesta antes de enviar reintentos.

```
RETRY_ENABLED = True
RETRY_TIMES = 3
DOWNLOAD_TIMEOUT = 20
```

Ilustración 75 - Configuración de reintentos

- Gestión de imágenes: con los siguientes parámetros se gestionan las imágenes. *IMAGES_URLS_FIELD* indica el campo de los productos en los que está almacenada la URL de la imagen a descargar. *IMAGES_RESULT_FIELD* indica el campo en el que se almacenará la ruta de la imagen descargada, *MEDIA_ALLOW_REDIRECTS* permite las redirecciones de URL en caso de que haya para descargar imágenes. *IMAGES_EXPIRES* indica el número de días que tiene validez las imágenes para que en caso de volverse a descargar la misma imagen no se haga hasta que expire. *IMAGES_THUMBS* indica la carpeta donde se almacenarán las miniaturas y el tamaño que tendrán para que Scrapy haga la reducción.

⁵⁹ En español, Verdadero o Activo.

```
IMAGES_URLS_FIELD = 'url_imagen'  
IMAGES_RESULT_FIELD = 'ruta_imagen'  
MEDIA_ALLOW_REDIRECTS = True  
IMAGES_EXPIRES = 90  
IMAGES_THUMBS = {  
    "miniaturas": (302, 453),  
}
```

Ilustración 76 - Configuración de almacenamiento de imágenes

- Log⁶⁰: con *LOG_LEVEL* se indica a partir de qué nivel de *log* de los cinco que hay, *debug*, *info*, *warning*, *error* y *critical* se escribe en el archivo de *log* designado por *LOG_FILE*. Se ha elegido el nivel de *warning* para abajo porque el código ya está depurado y únicamente interesa si se produce algún error inesperado.

```
LOG_LEVEL = 'WARNING'  
LOG_FILE = './logs/prendas.log'
```

Ilustración 77 - Configuración del log

5.2.1.7. Middlewares

Las clases *Middleware* en Scrapy son clases que permiten personalizar el tratado de los objetos *Request* y de los objetos *Response*, es decir, de las peticiones y respuestas a los servidores. Hay dos tipos principales, las clases *Spider Middlewares*, que, como su nombre dice, permiten tratar con las peticiones y respuestas que pertenezcan a la araña que las implemente o las clases *Downloader Middlewares*, que afectan a todas las peticiones y respuestas de todas las arañas.

En este proyecto sólo se ha desarrollado una clase *Downloader Middleware*. La función que cumple es que en caso de que una araña reciba cierto número de códigos de error HTTP 403 seguidos, el *middleware* le ordene acabar la ejecución. Esto es así porque cuando una página web deniega el acceso a una araña, en vez de devolver las páginas HTML solicitadas, devuelve ese código de error, por lo que si se ha denegado el acceso es mejor parar el *web scraping*.

5.2.1.8. Script de lanzamiento de las arañas

Para el lanzamiento de las arañas se ha elegido hacer un *script* en Python llamado *partystarter.py* de forma que se lancen todas, pues siguen unos plazos similares y es más fácil de gestionar.

A continuación se hará una pequeña recapitulación de los puntos más importantes de ejecución de las arañas:

- Zara: los enlaces tipo *-l* deben de ser explorados de forma diaria y los enlaces tipo *-p* cada semana. Además, es necesario antes de realizar la ejecución de la araña,

⁶⁰ En español, registro o historial, se refiere a un archivo en el que se suelen anotar automáticamente errores o incidencias que suelen ocurrir en un programa. En este caso, los errores que les ocurran a las arañas al hacer *web scraping*.

que se obtengan todas las URLs de productos para que sean contrastadas con las que hay en los enlaces tipo -l para no obtener día a día ítems redundantes.

- H&M: el sistema de protección anti *web scraping* puede llegar a penalizar varios días, por lo que la frecuencia de ejecución debe ser semanal.
- Carhartt WIP: los ítems del *sitemap* se renuevan cada varios meses, no obstante, como el catálogo es pequeño y para anticipar esos cambios cuando se den, se hará una frecuencia de ejecución semanal.

Antes de ejecutar las arañas, hay que acceder a la base de datos y obtener las URLs de los productos de Zara, para pasárselas a la araña de Zara y que sean contrastadas con las nuevas.

```
def get_zara_database_urls(cursor):
    try:
        sentencia = "SELECT url FROM item WHERE marca = 'Zara'"
        cursor.execute(sentencia)
        urls_sitemap = cursor.fetchall()

    except mariadb.Error as err:
        raise mariadb.Error(f"Error al obtener las urls de MariaDB por {_file_}: {err}")
    else:
        return urls_sitemap
```

Ilustración 78 - Obtención de las URLs de productos de Zara

Y posteriormente, se ejecutan las arañas. Se pasa por parámetros la lista de URLs a la araña de Zara y se fija como fecha de ejecución semanal los lunes. La ejecución semanal de los enlaces -p de Zara se regula a través de su método *sitemap_filter()* como se indica en “Ilustración 39”.

```
if productos_bd:
    productos_bd = json.dumps(productos_bd)

process.crawl(ZaraSpider, productos_bd=productos_bd)
if datetime.today().date().strftime("%A").lower() == 'monday':
    process.crawl(HmSpider)
    process.crawl(CarharttSpider)

process.start()
```

Ilustración 79 - Ejecución de las arañas en el script

5.2.2. Aplicación web

Para el desarrollo de la aplicación web se ha utilizado un *framework* de Python llamado Flask para el *back-end*⁶¹ y unas pinceladas del *framework* de *front-end* Bootstrap. Con la aplicación web se presentan todos los resultados del motor de búsqueda de forma atractiva e interactiva al usuario de forma que, en base a ciertos criterios, pueda elegir y buscar las prendas de ropa que prefiera.

⁶¹ En español, motor o capa de funcionamiento, se denomina a la parte de funcionamiento de una aplicación.

Para la realización de la aplicación web se ha seguido un tutorial de YouTube⁶² muy útil. La aplicación web desarrollada tiene ciertas similaridades con la que se desarrolla en el tutorial.

En este apartado se cubrirán distintos apartados como el cómo se ha hecho la API para cargar dinámico de imágenes, se mencionará el uso que se ha dado a Bootstrap, se hará un breve repaso de las distintas URLs que tiene la web y de los documentos HTML que presenta. En esta sección se hará hincapié en cómo se ha hecho el filtrado de las prendas, la parte más importante.

5.2.2.1. API Imágenes

El cargado dinámico de imágenes se produce cuando, al solicitar un cliente la URL de la página web donde hay imágenes y recibir el HTML correspondiente, no se almacena en el documento HTML la imagen incrustada, se almacena una URL que hace referencia a la API que contiene dicha imagen. Al recibir el cliente el HTML, automáticamente se hace una petición a la API para obtener dicha imagen e incrustarla posteriormente en su lugar. Por ende, hay que crear esta API.

```
id_imagen = request.args.get('id')
miniatura = request.args.get('miniatura')

if id_imagen and id_imagen.isdigit() and (miniatura == '0' or miniatura == '1'):
    item = Item.query.filter_by(id=int(id_imagen)).first()

    if not item:
        abort(404)
    if miniatura == '0':
        ruta_imagen = item.ruta_imagen

        return send_from_directory(app.config['UPLOAD_FOLDER'], ruta_imagen, mimetype='image/jpeg')

    ruta_imagen = item.ruta_miniatura
    return send_from_directory(app.config['UPLOAD_FOLDER'], ruta_imagen, mimetype='image/jpeg')

abort(400)
```

Ilustración 80 - API de cargado de imágenes de la aplicación web

Para solicitar una imagen, hay dos parámetros, el *id* que es el identificador de producto y también el nombre de la imagen y *miniatura* que es una variable booleana representada con un 0 si no se necesita miniatura y 1 si se necesita miniatura.

En caso de existir el ítem y que el valor de *miniatura* sea correcto se envía la imagen que se busca desde el directorio *UPLOAD_FOLDER* que apunta a “imágenes” donde se encuentran los directorios de las marcas, cada una con sus imágenes como se muestra en “Ilustración 61”. En caso de error se devuelve un código HTTP 400 *Bad Request*⁶³.

⁶² <https://www.youtube.com/watch?v=Qr4QMBUPxWo&t=4949s>

⁶³ En español, Pregunta Errónea, se refiere al código HTTP 400 que indica que hay un parámetro u otro factor erróneo en la petición.

5.2.2.2. Uso de Bootstrap

El uso del *framework* Bootstrap ha sido muy sencillo, se han obtenido ciertas hojas de estilo CSS y JavaScript y se han utilizado plantillas que se ofrecen gratuitamente en la página web de Bootstrap.

Un ejemplo es la sección de paginación, como se muestra a continuación. Sin embargo, han hecho ciertas modificaciones a las plantillas HTML cogidas.



Ilustración 81 - Visualización de plantilla de paginación de Bootstrap

Las hojas de estilo seleccionadas son las siguientes. Contienen clases de CSS y JavaScript.

```
<!-- Bootstrap CSS -->
<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js" >
```

Ilustración 82 - Código de CSS y JavaScript utilizado de Bootstrap

5.2.2.3. Distintas URLs y documentos HTML

Las URLs que se han habilitado en la página web son las siguientes:

```
@app.route('/')
@app.route('/galeria', methods=['GET', 'POST'])
def galeria_page(): ...

@app.route('/registro', methods=['GET', 'POST'])
def register_page(): ...

@app.route('/login', methods=['GET', 'POST'])
def login_page(): ...

@app.route('/logout')
def logout_page(): ...

@app.route('/<int:item_id>')
def item_page(item_id): ...

@app.route('/obtener_imagen')
def get_image(): ...
```

Ilustración 83 - Direcciones posibles de la página web

Cada dirección devuelve un documento HTML diferente.

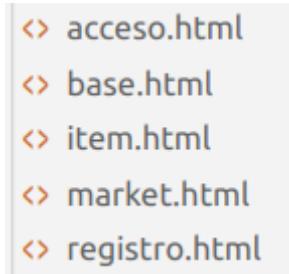


Ilustración 84 - Documentos HTML que puede devolver la página web

Todas las páginas heredan de *base.html* que fija la cabecera, donde se encuentran directivas necesarias como los *scripts* de Bootstrap de CSS y JavaScript.

- */market* o */*: página principal en la que se muestran los productos y donde se puede hacer el filtrado. Contiene el método GET para que el usuario pueda recibir el HTML y el método POST que se envía del cliente al servidor con la información de los filtros, de forma que este responda con un nuevo HTML de resultados filtrados. Devuelve el documento *market.html*.

La apariencia de la página web se puede ver en la siguiente imagen.

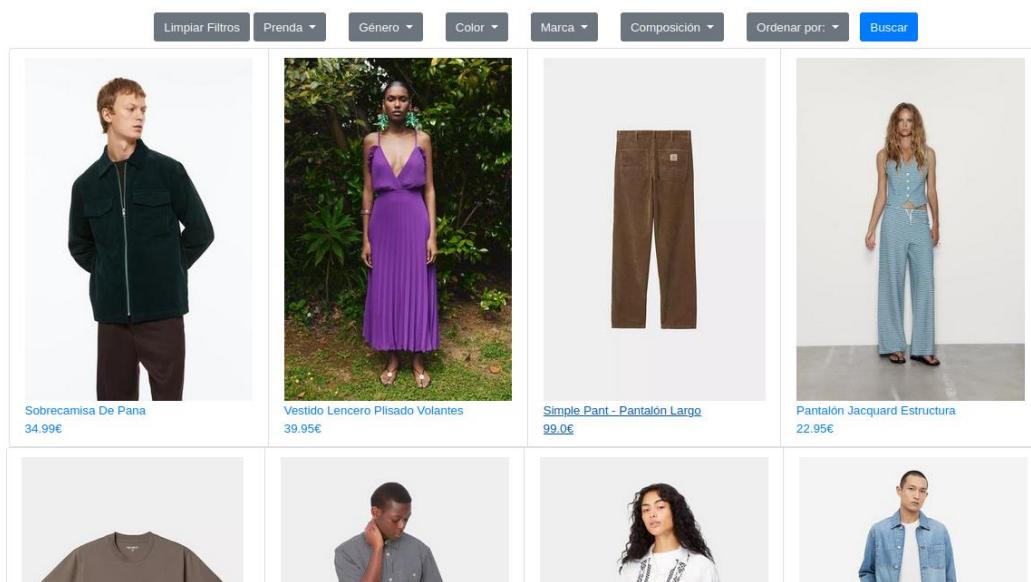


Ilustración 85 - Visualización de la página de *market.html*

El funcionamiento de la página de *market* para el filtrado de prendas, que funciona con el método POST, comienza con la obtención de las listas de filtros para cada parámetro, por ejemplo, si se marcan los colores 'Azul' y 'Blanco', la lista color tendrá dichos valores.

```
prenda = request.form.getlist('prenda')
genero = request.form.getlist('genero')
color = request.form.getlist('color')
marca = request.form.getlist('marca')
composicion = request.form.getlist('composicion')
orden_precio = request.form.get('ordenar')
```

Ilustración 86 - Obtención de los filtros seleccionados

Posteriormente se comprueban qué listas contienen valores y se genera una comprobación para cada parámetro seleccionado. Se pueden combinar distintos filtros, por ejemplo, color ‘Verde’ y marca ‘Zara’.

```

lista_query = []
if prenda:
    lista_query_prenda = []
    for p in prenda:
        lista_query_prenda.append(Item.nombre.like(f"%{p}%"))
        match p:
            case 'jerséi':
                lista_query_prenda.append(Item.nombre.like(f"%jersey%"))
            case 'pantalón':
                lista_query_prenda.append(Item.nombre.like(f"%jeans%"))
                lista_query_prenda.append(Item.nombre.like(f"%chinos%"))
                lista_query_prenda.append(Item.nombre.like(f"%bermuda%"))
                lista_query_prenda.append(Item.nombre.like(f"%shorts%"))
    lista_query.append(or_(*lista_query_prenda))

if genero:
    query_genero = Item.genero.in_(genero)
    lista_query.append(query_genero)
if color:
    query_color = Item.color.in_(color)
    lista_query.append(query_color)
if marca:
    query_marca = Item.marca.in_(marca)
    lista_query.append(query_marca)
if composicion:
    lista_query_composicion = []
    for c in composicion:
        lista_query_composicion.append(Item.composicion.like(f"%{c}%"))
    lista_query.append(or_(*lista_query_composicion))

```

Ilustración 87 - Obtención de los productos de la base de datos en base a los filtros seleccionados

También se trata la ordenación de precios en caso de haberse seleccionado, hay tres posibilidades de menor a mayor o viceversa y sin ordenación por precio.

```

if orden_precio and orden_precio == 'preciomenor':
    paginas = Item.query.filter(*lista_query).order_by(Item.precio.asc()).paginate(page=pagina, per_page=40)
elif orden_precio and orden_precio == 'preciomayor':
    paginas = Item.query.filter(*lista_query).order_by(Item.precio.desc()).paginate(page=pagina, per_page=40)
else:
    paginas = Item.query.filter(*lista_query).paginate(page=pagina, per_page=40)

```

Ilustración 88 - Ordenación de los productos en base a precio

- /registro: página para registrarse como usuario. Contiene el método GET para recibir el HTML y el método POST para enviar los datos de registro al servidor. Devuelve el documento *registro.html*.

Aspectos Relevantes del Desarrollo

Crear Nueva Cuenta

Nombre de Usuario

Correo Electrónico

Contraseña

Confirmar Contraseña

¿Ya tienes una cuenta?
[Accede a la página web](#)

Aceptar

Ilustración 89 - Visualización de la página de registro.html

- */login*: página para iniciar sesión. Utiliza el método POST para enviar los datos de inicio de sesión al servidor para que sean validados. Devuelve el documento *acceso.html*

Por favor, acceda

Nombre de Usuario

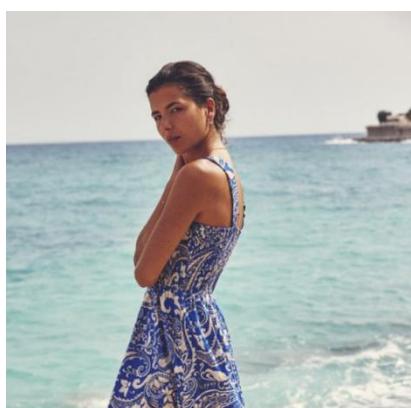
Contraseña

¿No estás registrado?
[Registrarse](#)

Acceder

Ilustración 90 - Visualización de la página de login.html

- */logout*: página para cerrar la sesión de usuario. Notifica al usuario que ha cerrado sesión con una alerta.
- */*: página individual de cada ítem que contiene toda su información. La dirección simplemente contiene el identificador del ítem. Devuelve la página *item.html*.



Vestido Midi Popelín Estampado

29.95 €

Vestido confeccionado con tejido 100% algodón. Escote recto y elástico con tirantes anchos.

Género: Mujer

Marca: Zara

Color: Azul

Composición principal: desconocida

[Ver en Sitio Original](#)

Ilustración 91 - Visualización de la página item.html

- */obtener_imagen*: API para obtener las imágenes de los productos. Se ha explicado el funcionamiento en el apartado “5.2.2.1 API Imágenes”.

6. TRABAJOS RELACIONADOS

Con la gran cantidad de negocios que han surgido gracias a la aparición del comercio electrónico, y por tanto, de una cantidad mayor de ofertas de todo tipo, se ha visto la necesidad de crear aplicaciones de software que analicen toda esa oferta y la presenten a los usuarios de forma que puedan filtrar todo aquello que no interese para poder realizar las compras de la forma más óptima posible. Así se ha visto que han triunfado páginas web como son SkyScanner o Rastreator.

En el apartado de la moda, las webs comparativas de marcas y prendas no han triunfado tanto como sus homónimos de comparación de seguros o vuelos. Algunas de las razones pueden ser que las páginas más potentes estén más centradas a un público de habla inglesa o que dado que la moda no implica un gasto demasiado elevado por prenda, no se haga necesaria una comparación. Otras razones pueden ser que gran parte de las marcas incluidas en estos comparadores sean marcas poco conocidas para los consumidores y que otros sectores tienen un mayor uso del comercio electrónico que el de la moda.

Algunas de las páginas web comparadoras de ropa más son las siguientes.

6.1. Lyst

Lyst es una plataforma de comparación de moda fundada en Londres en 2010 que incluye prendas de lujo. Tiene una selección de 17000 marcas de ropa y 8 millones de artículos almacenados en la web.

Tiene una gran cantidad de marcas y artículos y está centrada en gente con gran gusto por la moda y con dinero pues posee muchas marcas de nicho y no conocidas por el público general. También tiene marcas de ropa de lujo como pueden ser Armani, Saint Laurent o Prada. Tampoco tienen prendas de grandes marcas de ropa más alcanzables económicamente para el público medio como son Zara, H&M o Primark.

Tiene una interfaz minimalista, con el número mínimo de botones y con blanco como color principal. Esto hace que la navegación sea una experiencia sencilla, calmada y con una sensación de limpieza y pulcritud.

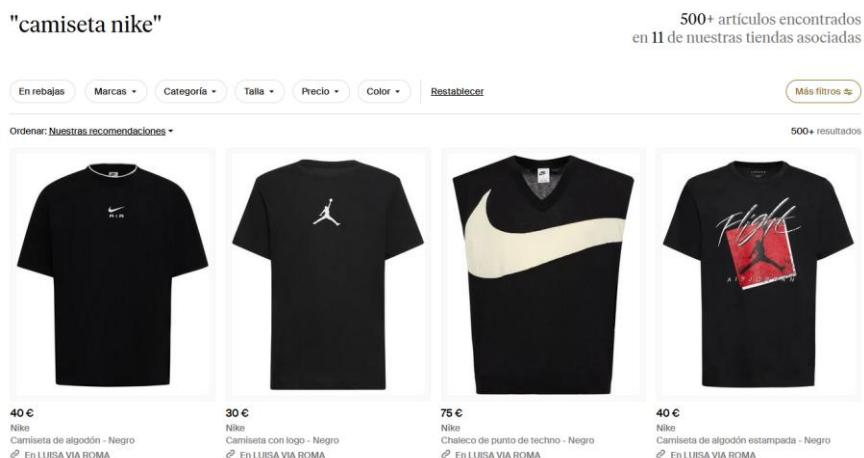


Ilustración 92 - Búsqueda de productos en Lyst

Trabajos Relacionados

Como características interesantes tiene selección para artículos en rebajas, para elegir entre rango de precios y tallas. Estas son opciones que no ofrece la aplicación del proyecto. Sin embargo, comparte con el proyecto selección por color, por marcas, por composición, por sexo y la opción de “Restablecer” para eliminar los filtros puestos. Los filtros se activan al seleccionarlos y no hace falta darle al botón de “Buscar”.

"camiseta nike"

500+ artículos encontrados
en 11 de nuestras tiendas asociadas

The screenshot shows the Lyst filtering interface. At the top, there are buttons for 'En rebajas' (Sales), 'Marcas' (Brands), 'Categoria' (Category), 'Talla' (Size), 'Precio' (Price), 'Color' (Color), and 'Restablecer' (Reset). A button for 'Más filtros' (More filters) is also present. Below these are sections for 'Sexo' (Gender), 'En rebajas' (Sales), 'Material' (Material), and 'Tienda' (Store). The 'Sexo' section has 'Mujer' and 'Hombre' options, with 'Hombre' selected. The 'En rebajas' section has 'Rebajado', 'A partir de -20 %', 'A partir de -50 %', and 'A partir de -70 %'. The 'Material' section lists 'Algodón', 'Caucho', and 'Lona'. The 'Tienda' section lists 'Amazon', 'ASOS', and 'Blue Tomato'. At the bottom, it says 'Filtros aplicados: Hombre' and 'Resultado: 500+ artículos'.

Ilustración 93 - Sistema de filtros en Lyst

Otro factor interesante que tiene para obtener los mejores precios es que analiza diferentes tiendas que tienen prendas de una misma marca, como es el caso de la siguiente imagen que tiene prendas de Nike de ASOS, Luisa Via Roma y JD Sports.

The screenshot shows a grid of Nike products from different stores. Top row: 1. Nike Camiseta manga corta - Blanco (35 €, Luisa Via Roma). 2. Nike Camiseta Hike con logo estampado - Negro (46 € 23 €, Luisa Via Roma). 3. Nike Camiseta Paris Saint Germain Futura - Negro (35 €, JD Sports). Middle row: 1. Nike Camiseta blanca extragrande con logo air - Blanco (49,99 €, ASOS). 2. Nike Camiseta de algodón - Negro (40 €, Luisa Via Roma). 3. Nike Camiseta Washed - Negro (45 € 30 €, JD Sports). Bottom row: 1. Nike Camiseta de algodón - Negro (40 €, Luisa Via Roma). 2. Nike Camiseta Washed - Negro (45 € 30 €, JD Sports).

Ilustración 94 - Productos en Lyst

Al hacer acceder a un producto, te redirige a la página original de la que se ha obtenido el producto para visualizarlo.

6.2. Fashiola

Fashiola es una plataforma de comparación de prendas holandesa nacida en 2012 con más de 5,5 millones de usuarios cada mes y con un gran rango de prendas y marcas de todo tipo.

Tiene un gran catálogo, pero a diferencia de Lyst, no está tan enfocada en ropa de nicho y de lujo y tiene más marcas y productos para el bolsillo medio como Zara y H&M.

Analizando la interfaz, es más recargada que la del proyecto y la de Lyst, tiene los filtros en la sección izquierda, que contrasta con la norma general que es tenerlo encima de los productos y además hace que ciertas secciones del filtro como estampado, tiendas y tiempo de entrega no se vean bien.

Tiene la opción de ver códigos de descuento de marcas.

Ilustración 95 - Búsqueda de productos en Fashiola

En la siguiente imagen se muestran las categorías que tiene el filtro. Añade como características interesantes el estampado, el rango de precios, las rebajas y el tiempo de entrega entre otras.

Ilustración 96 - Búsqueda de filtros en Fashiola

En la sección individual de cada producto no se muestra mucha información, únicamente el nombre, el precio, el color e información de la entrega, como los días y los gastos de envío. Sería un punto positivo incluir una breve descripción del producto y datos sobre la composición.

Ilustración 97 - Visualización de producto en Fashiola

7. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURO

7.1. Conclusiones personales

Hablando ya en primera persona, en este apartado me gustaría comentar todas aquellas resoluciones personales a las que he llegado tras tanto tiempo y esfuerzo puestos en este proyecto.

Un gran quebradero de cabeza ha sido la planificación temporal. Todas las estimaciones que he hecho respecto a los plazos han sido prácticamente en vano. El motor de búsqueda ha sido la parte que más tiempo me ha llevado desarrollar con diferencia, en muy gran parte porque el estudio e investigación de las páginas web ha sido largo y complejo. El desarrollo de la página web, sin embargo, ha sido un proceso mucho más liviano y sencillo. Todo ello ha causado cierta exasperación y poco control sobre el proyecto en algunos casos porque no me he sabido adaptar bien a los tiempos. La principal causa que veo es que nunca he realizado un proyecto tan grande y acostumbrado a trabajos más pequeños de la universidad ha hecho que no haya dado en el clavo y haya subestimado la dificultad del proyecto.

En cuanto a las conclusiones a nivel de resultados son relevantes el rendimiento del motor de búsqueda y la estandarización de los datos. El rendimiento del motor de búsqueda se ha visto muy lastrado por la posibilidad de bloqueo. Siendo la ratio una petición por cada segundo, se tarda aproximadamente 8 horas en obtener los ítems de Zara o H&M y eso es una gran cantidad de tiempo para la velocidad que se podría obtener sin limitaciones.

La parte de estandarización de los datos ha sido algo que no se pensó que fuera a ser tan importante como ha resultado siendo. La disparidad muchas veces de cómo se presentan los datos con sus valores en las páginas web es muy grande y hay que realizar grandes esfuerzos para que todos puedan ser tratados de la misma manera y que el usuario tenga la sensación de que todo venga de una misma fuente. El añadir una marca suponía tener en cuenta nuevas variables que no se han tenido antes en cuenta lo que también ha dificultado el trabajo y ha aumentado la complejidad de las funciones de estandarización.

El hecho de aprender de cero, únicamente con recursos gratuitos de Internet un lenguaje nuevo y dos *frameworks* y desarrollar una aplicación real entera supone para mí un orgullo muy grande. Hoy en día hay gente que es capaz de crear una carrera profesional entera gracias a haber empezado a programar con materiales gratuitos y para mí era un gran reto a nivel de disciplina y de aprender a aprender y lo he cumplido.

Me he dado cuenta de cómo de importante y necesario es realizar un Trabajo de Fin de Grado en la carrera de Ingeniería Informática. Es la piedra angular en la que para construirla pones en práctica mucho de lo aprendido en asignaturas de la carrera y creas algo muy grande. Porque la informática no son departamentos de conocimiento estancos, sino algo relacionado y que en el mundo real toma una dimensión totalmente diferente a cómo es en la universidad.

7.2. Conclusiones técnicas

En este apartado se analizarán el cumplimiento de los objetivos puestos en el punto “2 Objetivos del proyecto”.

- Investigación de las páginas web de Zara, H&M y Carhartt WIP para poder realizar un proceso de obtención de datos óptimo y correcto.

Se ha realizado un proceso de investigación muy exhaustivo de todas las páginas webs. Sobre todo, en Zara y H&M, que son páginas muy complejas, se ha logrado obtener un conocimiento útil y aplicable a la hora de realizar la implementación de los algoritmos.

- Obtención de los catálogos de ropa de hombre y mujer de Zara, H&M y Carhartt WIP.

A pesar de todas las dificultades, sobre todo la referente a la lentitud de extracción de datos por la existencia de los sistemas defensivos de la página web, se ha logrado obtener todos los catálogos de ropa de las tres marcas.

- Búsqueda y ordenación de los resultados mostrados por la aplicación web según distintos criterios.

Se ha logrado hacer un filtrado en base a muchos parámetros, en los que se puede combinar filtros y ordenar los resultados en base al precio. Esto da un gran poder y atractivo al proyecto.

- Gestión de usuarios.

Este ha sido el apartado que más ha flaqueado. No se le ha dado un uso real al tener perfil de usuario, no hay posibilidad ni de eliminar la cuenta ni de modificarla además de que se tampoco se envía un correo de confirmación para finalizar la creación, lo que puede ser un fallo de seguridad.

7.3. Líneas de trabajo futuras

Una vez finalizado, el proyecto tiene muchas posibilidades de expansión que en un futuro podría llevarse a cabo en caso de que quiera ponerse un escalón por encima de un proyecto simplemente académico:

- Añadir campo de ofertas: añadir a los campos obtenidos por *parsing* el de precio rebajado en caso de haberlo. Daría mucho valor a la página web saber qué ítems tienen rebajas.
- Añadir más marcas: se podrían flexibilizar los campos que se consideraron obligatorios de forma que elegir marcas fuera más fácil y se puedan añadir más, como pueden ser Bershka o Pull&Bear.
- Obtener un servicio de proxy con rotación de IPs: dado que el rendimiento del *web scraping* ha sido bajo debido a una baja ratio de peticiones por segundo, obtener

un servicio de *proxy* con capacidad de rotación de IPs haría que la obtención de los catálogos de las marcas fuera mucho más rápida y se reducirían las posibilidades de un bloqueo por parte de la página web.

- Eliminación de ítems que hayan desaparecido del catálogo: hacer una revisión de los productos expirados y retirarlos de la base de datos.
- Mejora de la interfaz: aprender más sobre Bootstrap y *front-end* y desarrollar una interfaz más atractiva al igual que un estudio de diseño centrado en el usuario. Esto haría que más usuarios accedieran a la página web si la interfaz fuera más profesional y fuera mejor la experiencia de compra.
- Añadir opción de ítems favoritos: que los usuarios puedan añadir un ítem como favorito a su perfil para que pueda hacer un seguimiento del mismo y tenerlo localizado, pues en la página se disponen de cientos de productos.
- Añadir existencias del producto: investigar cómo son las existencias de los productos en las páginas web y en caso de poder estandarizarlo, añadirlo como opción de los productos.

8. BIBLIOGRAFÍA

- Aguirre, J. M. (2019). *Generation of unique ids in the Database*. Obtenido de itnext.io: <https://itnext.io/generation-unique-ids-in-the-database-a9a7acd0e721>
- Ciberseguridad. (2021). *¿Qué es el web scraping y para qué se utiliza?* Obtenido de ciberseguridad.com: <https://ciberseguridad.com/guias/recursos/web-scraping/>
- Code, J. S. (2021). *Flask Course - Python Web Application Development*. Obtenido de youtube.com: <https://www.youtube.com/watch?v=Qr4QMBUPxWo&t=4949s>
- codeRECODE. (2022). *Run Scrapy Spiders from Python Script*. Obtenido de youtube.com: <https://www.youtube.com/watch?v=D22QqJ18rFg>
- codeRECODE. (2023). *Skip Pagination with Scrapy SitemapSpider: The Easiest Way To handle pages!* Obtenido de youtube.com: <https://www.youtube.com/watch?v=Z2yofe26K88>
- EDTeam. (2019). *¿Sabes qué es una API?* Obtenido de ed.team: <https://ed.team/comunidad/sabes-que-es-una-api>
- Hedgpeth, R. (2020). *How to connect Python programs to MariaDB*. Obtenido de mariadb.com: <https://mariadb.com/resources/blog/how-to-connect-python-programs-to-mariadb/>
- Jr, V. S. (2020). *Writing Scrapy Spiders in 2020*. Obtenido de stummjr.org: <https://stummjr.org/post/scrapy-in-2020/>
- Kouzis-Loukas, D. (2016). *Learning Scrapy*. Birmingham, United Kingdom: Packt Publishing Ltd.
- MariaDB. (2023). *Install MariaDB Connector/C*. Obtenido de mariadb.com: <https://mariadb.com/docs/skysql-previous-release/connect/programming-languages/c/install/>
- OpenAI. (2022). *ChatGPT*. Obtenido de <https://chat.openai.com/chat>
- Orús, A. (2023). *El comercio electrónico dentro de España - Datos estadísticos*. Obtenido de statista.com: <https://es.statista.com/temas/3167/el-comercio-electronico-dentro-de-espana/#topicOverview>
- ProgrammingKnowledge. (2020). *Python Flask Tutorial 10 - Pagination in Flask*. Obtenido de youtube.com: <https://www.youtube.com/watch?v=651dabuh7zE&t=185s>
- ReactNativeTutorial. (2020). *Finding hidden API of HM.com to web scrape all products*. Obtenido de youtube.com: <https://www.youtube.com/watch?v=6gtHzj4GMLo>
- Salas, J. (2023). *Dos fuertes llamaradas consecutivas avisan del aumento de actividad del Sol*. Obtenido de elpais.com: <https://elpais.com/ciencia/2023-08-08/dos-fuertes-llamaradas-consecutivas-avisan-del-aumento-de-actividad-del-sol.html>
- Schools, W. (s.f.). *XPath Nodes*. Obtenido de w3schools.com: https://www.w3schools.com/xml/xpath_nodes.asp

Bibliografía

- Scrapeops. (2022). *How To Solve A Scrapy 403 Unhandled or Forbidden Errors.* Obtenido de scrapeops.io: <https://scrapeops.io/python-scrapy-playbook/scrapy-403-unhandled-forbidden-error/>
- Scrapeops. (s.f.). *Web Scraping Guide: Headers & User-Agents Optimization Checklist.* Obtenido de scrapeops.io: <https://scrapeops.io/web-scraping-playbook/web-scraping-guide-header-user-agents/>
- Socratica. (2022). *Regular Expressions in Python || Python Tutorial || Learn Python Programming.* Obtenido de youtube.com: <https://www.youtube.com/watch?v=nxjwB8up2gI>
- StackOverflow. (2011). *How do I make XPath expressions more generic?* Obtenido de stackoverflow.com: <https://stackoverflow.com/questions/5545793/how-do-i-make-xpath-expressions-more-generic>
- StackOverflow. (2021). *Can't scrape an image url from Zara.* Obtenido de stackoverflow.com: <https://stackoverflow.com/questions/70511320/cant-scrape-an-image-url-from-zara>
- Wikipedia. (s.f.). *Document Object Model.* Obtenido de wikipedia.com: https://en.wikipedia.org/wiki/Document_Object_Model
- Wikipedia. (s.f.). *Expresión Regular.* Obtenido de wikipedia.org: https://es.wikipedia.org/wiki/Expresi%C3%B3n_regular
- Xirau, M. (2021). *Inditex gana (casi) tanto como Uniqlo, H&M, Gap y Primark en conjunto.* Obtenido de forbes.com: <https://forbes.es/empresas/132391/inditex-gana-casi-tanto-como-uniqlo-hm-gap-y-primark-en-conjunto/>
- Zyte. (2023). *Scrapy Documentation.* Obtenido de scrapy.org: <https://docs.scrapy.org/en/master/index.html>

