

# GALIBOOKS

Pablo Braga Paz  
Diego García López  
José Daniel García Ruiz

<b>Introducción</b>	<b>3</b>
<b>Páginas web utilizadas</b>	<b>3</b>
Buscalibre	4
Estado de la página	4
Crawling web	5
Libreria Paz	10
Estado de la página	10
Crawling web	11
Planeta del Libro	16
Estado de la página	16
Crawling web	19
Casa del libro	22
<b>Postprocesado de los datos</b>	<b>23</b>
Clusterización de las categorías	23
Obtención de categorías únicas	23
Creación del diccionario inicial	23
Agrupación basada en palabras clave	24
Agrupación basada en similaridad	25
Agrupación final	27
<b>Indexación</b>	<b>27</b>
Creación del índice	27
Inserción de documentos	29
<b>Interfaz Web</b>	<b>30</b>
Pasos iniciales	31
Modificaciones	31
Visualización	34

# Introducción

El mundo de la compra de libros por internet es inmenso y existen innumerables páginas para ello. En este proyecto intentaremos mostrar una pequeña aplicación que permita a los usuarios tener un punto centralizado para la búsqueda de libros en diversas páginas, de manera que pueda tener acceso a las diferentes ediciones, formatos y precios de un mismo libro, y también buscar por diferentes elementos como editorial, categoría o autor.

Para ello, necesitaremos recorrer diversas páginas web y obtener de cada uno de los libros que se oferten los siguientes campos:

- Título: nombre de la obra, elemento esencial para búsquedas.
- Autor: permite búsquedas específicas.
- Año de edición: ayuda a ordenar las obras por antigüedad, ofreciendo contenido más reciente antes.
- Editorial: permite búsquedas de ediciones específicas de un libro.
- Categorías: permite buscar por géneros específicos.
- Número de páginas: permite filtrar por tamaño del libro.
- Sinopsis: permite hacer búsquedas del contenido de la obra que no se incluya en el título.
- Portada: permite mostrar un elemento visual en relación a la obra que se muestra.
- Precio: permite filtrar y ordenar en base al coste del libro.
- ISBN: se trata de un número identificativo de una obra que permitiría buscar por un libro y edición específicos.

El proyecto se aloja en el siguiente repositorio: <https://github.com/pbpaz/RIWS.git>

## Páginas web utilizadas

Para la recopilación de contenidos nos hemos centrado en concreto en 3 páginas web. De esta manera, cubrimos un amplio abanico de libros, al mismo tiempo que trabajamos con diferentes tipos de páginas en cuanto al ámbito de la venta de libros online:

- Librería Paz: portal de compra de una librería específica.
- Planeta del Libro: portal de comparación de precios de libros.
- Buscalibre: plataforma de venta de libros exclusivamente online.

# Buscalibre

## Estado de la página

The screenshot shows the main landing page of Buscalibre. At the top, there's a banner advertising a 10% discount on imported books during the 'SEMANA DEL LIBRO IMPORTADO'. Below the banner, there's a sidebar for 'Libros en Inglés' featuring books like 'Grumpy Cat' and 'Human Acts'. The main content area includes a 'CATEGORÍA' sidebar with various book genres and a 'Novedades' section displaying several new releases with their covers and brief descriptions.

Como podemos ver, la página principal de Buscalibre tiene una interfaz muy clásica en el ámbito de los portales de venta online. En este caso, tenemos una barra de navegación, un bloque lateral que muestra categorías de libros y una sección principal que incluye algunas ofertas y los elementos más relevantes que están en venta en ese momento. Si navegamos a través de alguna de las categorías, lo que veremos será una pantalla mucho más útil para nuestro proyecto.

This screenshot shows a search results page for 'biografias, literatura y estudios literarios' on the Buscalibre website. The results are organized into a grid of book covers, each with its title, author, price, and a 'Envío GRATIS' badge. The sidebar on the left provides filters for categories like 'Biografías, Literatura Y Estudios Literarios' and 'Encuadernación'.

Al entrar en una categoría específica tendremos diferentes opciones a la izquierda, destacando el hecho de las subcategorías de la categoría específica. La parte central de la pantalla está ocupada por un listado de libros de dicha categoría paginados. Como último detalle, las URLs de estos listados por categorías siguen siempre el formato “[buscalibre.es/libros/categoría/subcategoría](http://buscalibre.es/libros/categoría/subcategoría)”. Si entramos dentro de un libro en específico, veremos lo siguiente:

The screenshot shows a product page for the book "Alas de sangre (Empíreo 1) Edición colecciónista enriquecida y limitada" by Rebecca Yarros. The page includes the book's cover image, title, author, publisher (Editorial Planeta), and format (Tapa Dura). It also displays the price (25,56 €), a discount (5% descuento from 26,90 €), and savings (Ahorras: 1,35 €). A "Libro Nuevo" button is present, along with quantity selection, purchase status (Estado: Nuevo), and a note about remaining units (Quedan 100+ unidades). Below the main price, there's a "Envío gratis en 1 día" badge. To the right, there's a "Compartir" button and a "Agregar a lista de deseos" link. The page also features a brief synopsis and a note about delivery times (Miércoles 13 de Noviembre y Jueves 14 de Noviembre).

Esta página nos muestra los campos del título, autor y editorial centrados, al lado de una ficha técnica que contiene información como el año de edición, número de páginas, categoría e ISBN, todo ello debajo de una imagen de la portada del libro. Finalmente y de nuevo en el centro, tenemos el precio y, más abajo, una sinopsis de dicho libro. En cuanto a la URL, el formato siempre seguirá la estructura “[buscalibre.es/libro-título...](http://buscalibre.es/libro-título...)”.

## Crawling web

Para ejecutar el crawling web de Buscalibre utilizamos un spider específico del cual podemos ver su configuración a través de las siguientes capturas:

```

class BuscalibreSpider(CrawlSpider):
    name="buscalibre_spider"

    allowed_domains = ['www.buscalibre.es']

    file = open('categories.txt', 'a', encoding='utf-8')
    cats = []
    #Start URL
    start_urls = [
        'https://www.buscalibre.es'
    ]

    custom_settings = {
        'ITEM_PIPELINES': {
            "riws.pipelines.ProcessBuscalibreSpiderPipeline": 1,
            "riws.pipelines.JsonWriterPipeline": 2,
        }
    }

    #Rules for the spider
    rules = (
        #Rule for the concrete book
        Rule(LinkExtractor(allow=(r'/libro-')), callback='parse', follow = False),
        #Rule for the sections
        Rule(LinkExtractor(allow=(r'/libros/')), follow = True),
    )

```

En esta primera captura vemos la configuración básica, que como vemos incluye el nombre del spider, que solo le permita crawlear en el dominio buscalibre.es y que inicie en dicha página de inicio. A mayores, vemos que tiene una configuración específica de este buscador, ya que ejecutará tanto el pipeline común que transforma los items a JSON como el pipeline específico de procesamiento de campos de este spider. Como podemos ver, las normas son muy sencillas, ya que lo único que hacemos desde la pestaña de inicio es seguir aquellos enlaces que posean la estructura “/libros/”, ya que se trata del listado de una categoría específica. De esta manera recorrerá todas las categorías y su paginación. Cuando entre a un libro específico, entrará por la primera regla, ya que todos los libros poseen la estructura “/libro-”. En este caso, no seguimos más enlaces y llamamos a la función parse.

```

#Parse method to extract the information of the concrete book
def parse(self, response):

    item = PazBookItem()
    item['url'] = response.url
    item['name'] = response.css('p.tituloProducto::text').get()
    item['author'] = response.css('p.font-weight-light a.font-color-bl::text').get()
    item['editorial'] = response.css('div#metadata-editorial a::text').get()
    item['edition_date'] = response.css('div#metadata-ano::text').get()
    item['category'] = response.css('div[id="metadata-categorías"] a::text').getall()

    for cat in item['category']:
        if cat.strip() not in self.cats:
            self.cats.append(cat.strip())
            self.file.write(str(cat.strip()))
            self.file.write('\n')

    item['isbn'] = response.css('div#metadata-isbn13::text').get()
    item['pages'] = response.css('div[id="metadata-número páginas"]::text').get()
    item['synopsis'] = response.css('span#texto-descripcion *::text').getall()
    item['cover'] = response.css('img#imgPortada::attr(data-src)').get()
    item['cost'] = response.css('p.precioAhora span::text').get()
    yield item

```

Como vemos esta función lo único que hace es llenar un item de libro con todos los datos necesarios. Aparte de la url que cogemos directamente, vemos que la mayoría de campos incluyen un id bastante esclarecedor del contenido de dicho elemento, como es el caso de tituloProducto, metadata-editorial, metadata-ano, metadata-isbn, metadata-número páginas o precioAhora. En concreto, destacamos como diferentes:

- Autor, que no posee un id claro y tenemos que utilizar su clase de css. Nos hemos asegurado de que este elemento es único en la página de manera que siempre devuelve correctamente el texto del autor.
- Categorías, ya que se trata de un div dentro del cual tenemos diversos enlaces a cada categoría a la que pertenece el libro. En este caso, queremos recuperarlos todos, por eso cogemos el texto de cada enlace.
- Descripción, que se trata de un campo en el que dentro incluye párrafos, saltos de línea, textos en negrita, etc. Por eso hemos tomado la decisión de tomar todos los elementos de texto dentro del span específico.
- Portada, donde tenemos que recuperar el atributo data-src en lugar del texto.

Una vez que recoge todos los campos, envía el item al pipeline específico de Buscalibre:

```

class ProcessBuscalibreSpiderPipeline:

    def process_item(self, item, spider):

        if spider.name != "buscalibre_spider":
            return item

        #Author
        if item['author'] is not None:
            authors = item['author'].strip().split(';')
            final_auth= []
            for author in authors:
                final_name=author
                if "," in author:
                    words = author.split(",")
                    final_name = words[1] + " " + words[0]
                final_auth.append(final_name.strip())
            item['author'] = final_auth

        if item['name'] is not None:
            item['name'] = item['name'].strip()
        #Editorial
        if item['editorial'] is not None:
            item['editorial'] = item['editorial'].strip()
        #Edition_date
        if item['edition_date'] is not None:
            item['edition_date'] = int(item['edition_date'].strip())

```

Como vemos, lo primero que hace este pipeline es asegurarse de que solo trabaja para el spider de Buscalibre. En caso de que no sea ese spider no realiza ninguna modificación. En caso de que sí, procesa cada uno de los campos en base a sus necesidades, que en este caso son las siguientes:

- El autor en Buscalibre puede ser un autor único o múltiple separado por puntos y comas. A mayores, el autor puede venir en el formato “Nombre Apellido” o en el formato “Apellido, Nombre”. Para unificarlo, este pipeline se encarga de separar los autores por puntos y comas, girar aquellos que vengan en el segundo formato para unificarlos todos como “Nombre Apellido” y almacenarlos en una lista.
- Los campos de título y editorial solo necesitan la ejecución de un strip para evitar elementos extras más allá del texto que incluía la página para elementos de estética(como tabulaciones, espacios y saltos de línea).

- El campo de año de edición necesita de nuevo el strip, pero en este caso hemos decidido almacenarlo como entero ya que nos será de más utilidad de esta manera para poder realizar ordenaciones o filtrados.
- El campo coste recibía el coste en el formato “XX,YY €”. Como solo nos interesa el valor numérico lo primero que hacemos es separar por el espacio y, a la primera cadena que contiene el número en un formato de texto no muy útil, lo transformamos a número decimal, haciendo primero un cambio de “,” a “.” para que la transformación funcione correctamente.
- De nuevo los campos de número de páginas e ISBN solo necesitan de un strip y una transformación a entero
- En cuanto a la sinopsis, tomamos la decisión de unir todos los fragmentos separados por un espacio asumiendo que todos ellos terminan correctamente una frase. A mayores, eliminamos algunos caracteres que nos podrían complicar la vida a la hora de disponer visualmente los elementos en la aplicación web, como los saltos de línea y las tabulaciones.
- Finalmente, a cada una de las categorías había, de nuevo, que aplicarle un strip antes de devolver el item final.

```

#Cost
if item['cost'] is not None:
    t1 = item['cost']
    t2 = t1.split()
    item['cost'] = float(t2[0].replace(',', '.'))

#Pages
if item['pages'] is not None:
    item['pages'] = int(item['pages'].strip())

#ISBN
if item['isbn'] is not None:
    item['isbn'] = int(item['isbn'].strip())

#Synopsis
if item['synopsis'] is not None:
    item['synopsis'] = " ".join(item['synopsis'])
    item['synopsis'] = item.get('synopsis').replace('\n', '').replace('\r', '').replace('\t', '')

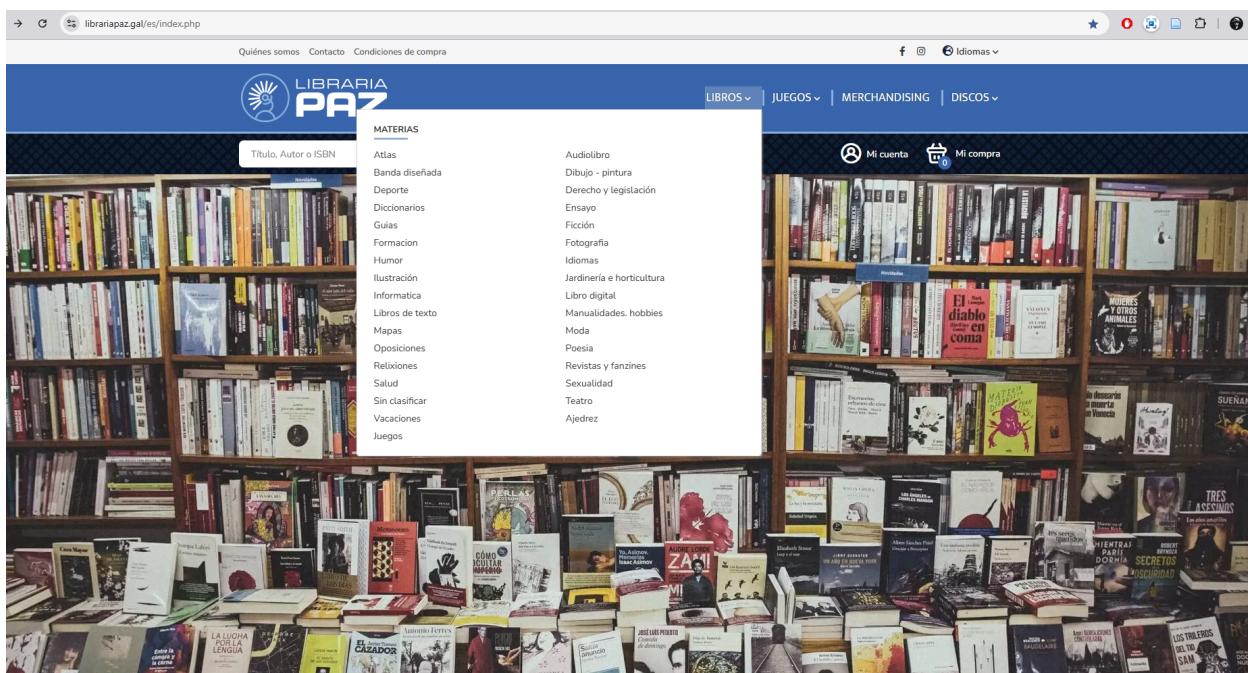
#Category
categories = item.get('category')
cat2 = []
for cat in categories:
    cat2.append(cat.strip())
item['category'] = cat2

return item

```

# Librería Paz

## Estado de la página



La web de Librería paz es bastante simple. Tiene una página principal en la que tenemos un buscador, una imagen de la librería y un listado con las novedades en cuanto a libros. En la parte superior presenta un header en el que podemos ver el logotipo de la empresa y los diferentes productos que tienen. En este header nos resultará especialmente útil el desplegable de “Libros”, ya que nos proporcionará una lista con todas las categorías disponibles en la librería y, por lo tanto, acceso a todos los libros existentes.

A screenshot of the "Libros de Ficción" section of the Librería Paz website. The header includes the "LIBRERÍA PAZ" logo and a breadcrumb navigation "INICIO / LIBRERÍA / FICCIÓN". The main content area is titled "Libros de Ficción" with the subtitle "104969 resultados". It features a grid of book covers for various fiction titles, each with its title, author, price, and a "Comprar" button. The books shown include "Funny Boy" by Shyam Selvadurai, "¿ESTÁS MALITO, SAM?" by Nest, Amy, "EL CÓDIGO SECRETO" by Ackerman, Sara, "HOTEL BU" by Rubio, María, "GATOTRASTADAS" by Martín, Enrique Carlos, and "CUANDO HARU ESTUVÓ AQUÍ" by Thao, Dustin. At the bottom, there are smaller book covers for "LOS PIRATAS CUATROPATAS" and "HASTA QUE TU MUERTE".

Si accedemos a una categoría específica, se presentarán de forma paginada todos los libros pertenecientes a dicha categoría. Podemos ordenarlos o elegir cuántos ejemplares ver por página, pero son detalles que no nos serán relevantes para el crawling. Por último, la URL de cada una de las categorías sigue el siguiente formato “librariapaz.gal/es/libros-de/categoría”.

The screenshot shows a web browser displaying the Librería Paz website at the URL [librariapaz.gal/es/libro/la-hija-de-la-novicia\\_722810](http://librariapaz.gal/es/libro/la-hija-de-la-novicia_722810). The page is for the book "LA HIJA DE LA NOVICIA" by ÁLVAREZ, ELENA. The book cover features a woman standing in front of a stone building with mountains in the background. The page includes the book's title, author, editorial (PLAZA & JANÉS EDITORES, S.A.), year of publication (2024), genre (Histórica), ISBN (978-84-01-03548-7), number of pages (288), and binding (Rústica). It also shows the price (22,90 € IVA incluido) and two buttons: "Añadir a mi cesta" and "Añadir a favoritos". Below the book details, there is a "Sinopsis" section with a detailed description of the plot, mentioning Carlomagno, Aquitania, and Elvira, a novicia who must flee from a convent in the mountains after her death. There are also social media sharing icons (Facebook, Twitter, LinkedIn, Pinterest, Google+, Email) and a green "En stock" button. At the bottom of the page, there is a "Sobre nosotros" link.

Si entramos dentro de un libro concreto, tenemos disponible toda la información relevante sobre él: imagen de portada, título, autor, editorial, año de edición, materia (categoría), ISBN, número de páginas, precio y sinopsis. Un detalle importante es que hay algunos libros en las que alguna información no está disponible; puede haber un libro sin portada o sin sinopsis, por lo que no será posible tener toda la información para todos los artículos. También hay en la parte inferior una serie de libros relacionados. En cuanto a la URL, sigue el formato “librariapaz.gal/es/libro/nombre-libro”.

## Crawling web

Para crawlear la página web, hemos creado un *CrawlSpider* simple que empiece en la página principal de la librería y vaya entrando a cada una de las categorías y libros. Como podemos ver en la primera imagen, tenemos 3 elementos definidos:

```

class PazSpider(CrawlSpider):
    name="paz_spider"
    allowed_domains = ['www.librariapaz.gal']

    start_urls = [
        'https://www.librariapaz.gal/es/index.php'
    ]

    custom_settings = {
        'ITEM_PIPELINES': {
            "riws.pipelines.ProcessPazSpiderPipeline": 1,
            "riws.pipelines.JsonWriterPipeline": 2,
        },
    }

    rules = (
        #Rule for digital books without information
        Rule(LinkExtractor(allow=(r'/es/libros-de/libro-dixital')), follow = False),

        #Rule for the concrete book
        Rule(LinkExtractor(allow=(r'/es/libro/')), callback='parse', follow = False),

        #Rule for the navigation on the sections
        Rule(LinkExtractor(allow=(r'/es/libros-de/.+pagSel=\d+')), follow=True),

        #Rule for the sections
        Rule(LinkExtractor(allow=(r'/es/libros-de/'))), follow = True),
    )

```

- *start\_urls*: Aquí indicamos en qué páginas (en nuestro caso es solamente una) se situará el *spider* para iniciar el proceso de crawling.
- *custom\_settings*: aquí definimos los *pipelines* por los que pasarán nuestros datos una vez hayan sido obtenidos. Primero, pasarán por un *pipeline* específico de la Librería Paz dedicado a formatear algunos campos para que sigan el formato deseado, y después pasarán al *JsonWriterPipeline* del que ya se ha hablado en el apartado anterior y es común a todas las librerías.
- *rules*: Aquí se definen las 4 reglas sencillas que nos permitirán desde la página principal llegar a cada uno de los libros concretos para extraer la información:
  - La primera se encarga de evitar la categoría de libros digitales, ya que son libros sin información y lo único que hace es no entrar dentro de esa categoría.
  - La segunda regla es la de los libros. Cuando encuentra un link de un libro concreto, se llama a la función encargada de procesar la información y los datos y no se sigue ningún link más.
  - La tercera regla se encarga de la navegación dentro de cada categoría. Cuando encuentra el link de la siguiente página de libros de esa categoría, lo sigue, para así obtenerlos todos.

- La cuarta y última regla se encarga de entrar a cada una de las categorías. Desde la página principal, entrará a cada una de las categorías y aplicando las reglas 2 y 3 conseguirá entrar a cada libro y obtener toda la información.

```
#Parse method to extract the information of the concrete book
def parse(self, response):

    item = PazBookItem()
    item['url'] = response.url
    item['name'] = response.css('h1#titulo::text').get()
    item['author'] = response.css('span.nomesigas::text').get()
    item['editorial'] = response.css('dd.editorial span.nomesigas::text').get()
    item['edition_date'] = response.css('dt:contains("Año de edición") + dd::text').get()
    item['category'] = response.css('dd a::text').get()
    item['isbn'] = response.css('dt:contains("ISBN") + dd::text').get()
    item['pages'] = response.css('dt:contains("Páginas") + dd::text').get()
    item['synopsis'] = response.css('div#tabsinopsis p.bodytext::text').getall()
    item['cover'] = response.css('div#detimg img#detportada::attr(src)').get()
    item['cost'] = response.css('div.infoprices span.despues::text').get()
    yield item
```

Una vez visto esto, podemos seguir con la función de extracción de los datos para cada uno de los campos de interés. Para todos ellos usamos el selector CSS. Veamos en detalle como extraemos cada uno de los campos:

- URL: para la URL simplemente obtenemos la URL del parámetro response.
- Name: En el HTML siempre viene dentro de un h1 con con ID “titulo”, así que obtenemos el texto asociado.
- Author: Para obtener el autor, está dentro de un span de clase “nomesigas”, que es el primero que aparece en la página. Por ello, podemos obtener el texto asociado y tendremos el autor del libro.
- Editorial: La editorial está dentro de un span “nomesigas”, pero a su vez dentro de un dd “editorial”. Especificando eso, podemos obtener la editorial.
- Edition\_date: La fecha de edición aparece en un dd que aparece justo después de un dt con el texto de “Año de edición:”. Para obtenerlo, buscamos dicho dt con contains() y obtenemos el texto del dd que aparece inmediatamente después.
- Category: En esta página solo hay una categoría por libro, y está en el texto de un elemento que a su vez está contenido en un dd. Obtenemos el texto asociado y tenemos la categoría.
- ISBN: Similar a la fecha de edición.
- Pages: Similar a la fecha de edición.
- Synopsis: La sinopsis viene dividida en varios párrafos, así que hacemos un getall() de todos ellos.
- Cover: Para obtener la imagen del libro, extraemos el atributo src de la imagen.

- Cost: El precio es el texto asociado a un span de class “despues” que está dentro de un div “infoprices”.

```

class ProcessPazSpiderPipeline:

    def process_item(self, item, spider):

        if spider.name != 'paz_spider':
            return item

        #Author
        if item['author'] is not None:
            t1 = item.get('author')
            t2 = t1.split(",")
            if len(t2) > 1:
                item['author'] = t2[1] + " " + t2[0]

        #Edition_date
        if item['edition_date'] is not None:
            item['edition_date'] = int(item.get('edition_date'))

        #Cost
        if item['cost'] is not None:
            t1 = item.get('cost')
            t2 = t1.split()
            item['cost'] = float(t2[0].replace(',', '.'))

        #Pages
        if item['pages'] is not None:
            item['pages'] = int(item.get('pages'))

```

Después de obtener todos los datos, se envía el item al pipeline correspondiente para procesar los datos y darles el formato correcto. Primero, nos aseguramos de que viene del spider que queremos, y comenzamos a procesar cada uno de los campos. Destacaremos algunos de los más relevantes:

- El campo author tiene la peculiaridad de que en nuestra página puede tener dos formatos: “Apellido, Nombre” y “Nombre Apellido”. Por ello, en los casos que aparezca con el primer formato tendremos que adaptarlo al segundo.
- El campo isbn en nuestra librería viene separado por ‘-’ y lo almacenamos como int, por lo que hay que eliminar dichos caracteres.
- En la sinopsis, tenemos que juntar todos los párrafos obtenidos para poder guardarlo como un string único. También limpiamos los saltos de línea o tabulaciones que pueda haber.
- Para las categorías, tenemos que guardarlas en forma de lista para seguir el formato establecido. En esta librería, aquellas sin categoría son asignadas como “Sin clasificar”,

por lo que tenemos que identificarlas y asignarlas a lista vacía, que es como las representaremos. Tras analizar la página, también vimos que algunas de ellas que presentaban más de una categoría venían separadas por ‘e’, ‘.’ y ‘-’, por lo que tuvimos que identificarlas y separarlas.

```
#ISBN
if item['isbn'] is not None:
    item['isbn'] = int(item.get('isbn').replace('-', ''))

#Synopsis
if item['synopsis'] is not None:
    item['synopsis'] = " ".join(item.get('synopsis'))
    item['synopsis'] = item.get('synopsis').replace('\n', '').replace('\r', '').replace('\t', '')

#Category
t1 = item.get('category')
if t1 == 'Sin clasificar':
    item['category'] = []
else:
    t1 = item.get('category')
    t2 = re.split(r' e |\. | - ', t1)
    if len(t2) > 1:
        item['category'] = t2
    else:
        item['category'] = [item.get('category')]

self.file = open('categories.txt', 'a', encoding='utf-8')
for element in item['category']:
    self.file.write(f"{element}\n")

return item
```

Una vez procesados todos los campos, se pasa el item al pipeline genérico que se encarga de guardarlos en formato JSON dentro de un archivo.

# Planeta del Libro

## Estado de la página

The screenshot shows the homepage of Planeta del Libro. At the top, there's a search bar with placeholder text "Escribe aquí el título, autor o ISBN" and a magnifying glass icon. Below the search bar is a navigation menu with links: Libros (selected), Más vendidos, Recomendados, Editoriales, Autores, Audiolibros, Revista, and Podcast. To the right are links for "MIS LISTAS" and "ENTRAR". The main title "Todos los libros de PlanetadeLibros" is displayed in bold, followed by the subtext "Entra y ponte cómodo, tienes mil maneras de buscar y encontrar tus libros favoritos.". A section titled "Temáticas de ficción" contains five categories with icons: "Novela literaria" (quill pen), "Novela negra" (detective hat and glasses), "Novela romántica" (hearts), "Novela histórica" (sword), and "Ciencia ficción" (eye with circuit board). Below this is a section titled "Temáticas de no ficción" with five categories: "Biografías" (person silhouette), "Historia" (temple icon), "Cocina" (kitchenware icon), "Relaciones" (speech bubble icon), and "Ciencia" (brain icon). At the bottom, there's a section titled "Libros destacados" with a grid of book covers for titles like "Cuando el cielo se vuelve amarillo" by Nerea Pascual, "La saga de los longevos 1. La Vieja Familia" by Eva García Sáenz de Urturi, "El Clan" by Carmen Mola, "Quedará el amor" by Alice Kellen, "Pecados 2. Rey de la soberbia" by Ana Huang, and "El amor que dejamos atrás" by Rebecca Yarros.

La página de inicio de **Planeta del Libro** está organizada en dos secciones principales. En la primera sección, se presentan las categorías principales en las que se clasifican los libros disponibles en la plataforma. En la segunda sección, se destacan libros seleccionados, con un énfasis especial en las novedades.

Al hacer clic en una de las categorías, la página despliega una nueva vista que incluye una serie de *tags* correspondientes a las subcategorías relacionadas, junto con los libros pertenecientes a la categoría seleccionada.

En esta sección, los libros se presentan de dos maneras. Primero, en listas simples divididas en tres grupos: una para los *best sellers*, otra para las novedades, y una última con recomendaciones destacadas. Segundo, en una lista más compleja y paginada que organiza los libros según los criterios definidos por el usuario mediante el filtro, proporcionando una experiencia de búsqueda más personalizada.

#### Últimas novedades en libros literarios

Descubre las novelas literarias que podrás ir encontrando en las librerías a lo largo del mes y déjate atrapar por la lectura.

The grid displays the following books:

- 1. Estuche Manolito Gafotas** by Elvira Lindo (Novela Literaria). Cover shows a boy in a suit.
- 2. La invención de la soledad** by Paul Auster (Novela Literaria). Cover shows a group of people in a room.
- 3. LA BUENA SOMBRA** by María de la Luz del Prado (Novela Contemporánea). Cover shows a woman sleeping on a log.
- 4. El país de las últimas cosas** by Paul Auster (Novela Literaria). Cover shows a house.
- 5. EL ÁRBOL DE LAS PALABRAS** by Andrés Pascual (Novela Literaria). Cover shows a large tree in a landscape.
- 6. La montaña del tesoro** by Martí Gironell (Novela Histórica). Cover shows a person walking through a tunnel towards a bright light.

#### Novelas literarias que son best sellers

Te presentamos las obras literarias más deseadas por miles de lectores en los últimos tiempos... ¿Ya las conoces?

The grid displays the following books:

- 1. Me piden que regrese** by Andrés Trapiello (Novela Literaria). Cover shows a group of people at a train station.
- 2. El mejor libro del mundo** by Manuel Vilas (Novela Literaria). Cover shows a person reading a book while hanging upside down from a tree.
- 3. La mala costumbre** by Alana S. Portero (Novela Literaria). Cover shows a woman with a bow in her hair.
- 4. Ropa de casa** by Ignacio Martínez de Pisón (Novela Literaria). Cover shows a couple in pajamas.
- 5. Un silencio lleno de murmullos** by Gioconda Belli (Novela Literaria). Cover shows a close-up of a young girl's face.
- 6. Conquistadores** by Éric Vuillard (Novela Literaria). Cover shows a group of people on a bridge.

**Subtemáticas Novela literaria**

- Novelas de no ficción
- Narrativa literaria clásicos
- Relatos

**Filtros**

**DESTACADOS**

- Novedades
- Más vendido
- Próximamente

**EDITORIAL**

- Austral Editorial (580)
- BackList (36)
- Booket (288)
- Click Ediciones (14)
- Crossbooks (4)
- MÁS EDITORIALES +

**FORMATOS**

- Audiolibro (513)
- Estuche (13)
- Libro (2079)
- Libro bolsillo (1007)
- Libro edición especial (1)
- MÁS FORMATOS +

[Limpiar filtros](#)

**Todas las novelas literarias**

Mostrando del 1 al 20 de 5857 libros

Ordenar por

**Me piden que regrese**  
Andrés Trapiello  
NOVELA LITERARIA

[Comprar](#)

**Tengo la impresión de que el cielo se prepara para la lluvia**  
Óscar Curielos  
NOVELA LITERARIA

[Comprar](#)

**Cuchara y memoria**  
Benito Talbo  
NOVELA LITERARIA

[Comprar](#)

**Mar de historias**  
Cristina Pacheco  
NOVELA LITERARIA

[Comprar](#)

**Los crímenes de la miel**  
Pep Coll  
NOVELA LITERARIA

**Los crímenes de la miel**  
Pep Coll  
NOVELA LITERARIA

Al seleccionar un libro, se accede a una página dedicada que muestra todos los detalles relacionados con el título. Esta incluye la portada del libro, una sinopsis que resume su contenido, el nombre del autor, el ISBN, y otros datos relevantes. Estos datos serán los que almacenaremos en el índice. Si seleccionamos una de las subcategorías nos mostrará su página homóloga.

**Planeta de Libros** Escribe aquí el título, autor o ISBN

Libros ▾ Más vendidos Recomendados Editoriales Autores Audiolibros Revista Podcast MIS LISTAS ENTRAR

Novela contemporánea / Drama / La buena sombra

**La buena sombra**

Maria de la Luz del Prado

Sé el primero en valorar este libro

Sinopsis de La buena sombra

La vida se abre paso en un cruce de caminos entre el fuego y la tormenta. Una novela reveladora sobre dos generaciones de mujeres que rompen con los convencionalismos del amor.

«Quisiera que fuese solo un juego, sin dolor, sin fuego —el que juega con fuego se quema—, pero en un ejercicio de franqueza consigo misma se reconoce cansada de pelear, quiere quemarse y citar a Cortázar: «Vos no elegís la lluvia que te va a calar hasta los huesos».

Esta historia no es lluvia, es un diluvio. Y Vera salta al vacío cuando una noche, saliendo del tablao Villa Rosa en la plaza de Santa Ana, su mirada verde chocó estrepitosamente con los ojos ámbar del Pantera: racial, sensual, la nueva...

[Leer más](#)

[Leer fragmento](#)

Opciones de compra

Elegir formato:

Libro	20.90 €
Casa de Libro	<a href="#">Comprar</a>
libros.com	<a href="#">Comprar</a>
amazon	<a href="#">Comprar</a>
agapea.com	<a href="#">Comprar</a>

Libro Electrónico (Epub 3) 8.99 €

[Guardar en mis listas](#) [Compartir](#)

FICHA TÉCNICA POR QUÉ LEER AUTORA OPINIONES CONTENIDO EXTRA SALA DE PRENSA OTROS LIBROS

**FICHA TÉCNICA**

Temáticas	Drama, Novela contemporánea, Novela Romántica contemporánea, Novela literaria, Novela romántica	Presentación	Rústica con solapas <th>Páginas</th> <td>400</td>	Páginas	400
Formato	15 x 23 cm	Editorial	Espasa	Código	0010349321
Publicación	30 oct 2024	ISBN	978-84-670-7446-8 <th>Tinta texto interior</th> <td>Blanco y negro</td>	Tinta texto interior	Blanco y negro
Sentido lectura	Occidental	Colección	ESPASA NARRATIVA		

## Crawling web

Para realizar el crawling de esta página hemos programado un Spider propio para ella. El primer paso para crearlo es establecer el dominio que vamos a crawlear, que reglas vamos a seguir para navegar por él y en qué url vamos a empezar.

```
class PlanetadelibrosSpider(CrawlSpider):
    name = 'planetadelibros'
    allowed_domains = ['planetadelibros.com']
    start_urls = ['https://www.planetadelibros.com/libros']

    custom_settings = {
        'ITEM_PIPELINES': {
            "riws.pipelines.ProcessPlanetadelibrosSpider": 1,
            "riws.pipelines.JsonWriterPipeline": 2,
        }
    }
```

Como se ve en la imagen el dominio que se ha seleccionado es el explicado anteriormente y la url inicial es la propia pantalla de inicio. A mayores se han definido ajustes propios para la araña para un correcto procesado de los datos.

```
rules = (
    #categorias
    Rule(
        LinkExtractor(allow=r'/libros/[a-zA-Z\-\-]+/\d+$'),
        follow=True
    ),
    Rule(
        LinkExtractor(allow=r'/libros/[a-zA-Z\-\-]+/\d+/p/\d+$'),
        follow=True
    ),
    Rule(
        LinkExtractor(allow=r'https://www.planetadelibros.com/[a-zA-Z\-\-]+/\d+$'),
        callback='parse_book',
        follow=False
    )
)
```

La primera regla que hemos definido permite al *crawler* seleccionar únicamente los enlaces que conducen a las páginas de cada categoría, evitando aquellos que no sean

relevantes. La segunda regla está diseñada para que el *crawler* pueda navegar a través de la lista completa de libros pertenecientes a una categoría específica. Finalmente, la última regla se encarga de dirigir al *crawler* a las páginas de detalles de cada libro.

Dado que no es necesario que el *crawler* continúe explorando más allá de esta última página, el parámetro **follow** de la última regla se ha configurado en **false**, limitando así la profundidad de la búsqueda.

Finalmente, en la creación de la araña, hemos definido la función **parse\_book**, encargada de construir el *ítem* correspondiente a cada libro. Esta función utiliza un selector que permite extraer la información requerida directamente desde el CSS de la página web. De esta manera, se asegura que los datos del libro, como el título, autor, ISBN, sinopsis, y otros detalles relevantes, sean obtenidos de manera precisa y organizada para su posterior procesamiento o almacenamiento.

La imagen no se encuentra de forma estática en el código fuente de la página, por lo que ha sido necesario analizar el JSON que almacena los datos utilizados para invocar los scripts de renderizado. Para ello, hemos cargado el JSON utilizando una librería específica de Python y navegado por su estructura con la función **get()** hasta localizar el atributo específico. Dado que el formato de estos JSON puede variar según el libro, hemos implementado una comprobación para determinar si ya hemos obtenido la URL deseada o si es necesario realizar pasos adicionales.

```
def parse_book(self, response):
    url = response.url
    book_title = response.css('h1.FichaLibro_fichaLibro_titulo__zoYiu::text').get()
    author = response.css('ul.LibroAutores_autoresList__ND_Mc li.LibroAutores_autoresListItem__i2Pk a::text').get()
    category = response.css('ol.Breadcrumbs_breadcrumb_list__Mwmc li span::text').getall()
    editorial = response.css('table.FichaTecnica_fichaTecnicaTabla__VKBCJ tr:contains("Editorial") td.FichaTecnica_fichaTecnicaValue__Tnr08 a::text').get()
    isbn = response.css('table.FichaTecnica_fichaTecnicaTabla__VKBCJ tr:contains("ISBN") td.FichaTecnica_fichaTecnicaValue__Tnr08::text').get()
    pages = response.css('table.FichaTecnica_fichaTecnicaTabla__VKBCJ tr:contains("Páginas") td.FichaTecnica_fichaTecnicaValue__Tnr08::text').get()

    json_data = response.css('script#\_\_NEXT_DATA__::text').get()

    cover = None
    if json_data:
        data = json.loads(json_data)
        cover = data.get('props', {}).get('pageProps', {}).get('page', {}).get('schema', {}).get('image', None)

        if isinstance(cover, dict) and 'path' in cover:
            cover = cover['path']

    synopsis = ''.join(response.css('div.mantine-Text-root *::text').getall())
    cost = response.css('button.OpcionesCompra_btnFormato__LQpT9 span.OpcionesCompra_btnFormato__precio__k3qx0::text').get()

    item = PazBookItem(
        url=url,
        name=book_title,
        author=author,
        editorial=editorial,
        isbn=isbn,
        cover=cover,
        pages=pages,
        cost=cost,
        synopsis=synopsis,
        category=category,
        edition_date=None
    )

    yield item
```

Para procesar cada dato extraído, tal como lo especificamos en el parámetro **custom\_settings**, se invocará el pipeline **ProcessPlanetadelibrosSpider**. Este pipeline se

encargará de recibir, validar y procesar la información recopilada por la araña, asegurando que los datos sean manipulados y almacenados adecuadamente según las necesidades del proyecto. Concretamente el pipeline se encarga de:

- Para el campo “cost”: Cambiar el carácter “,” por “.”
- Para el campo “pages”: Convertir el campo a tipo float.
- Para el campo “isbn”: Eliminar el carácter “-” y convertirlo en int.
- Para el campo “synopsis”: Eliminar caracteres como el “\n”, “\r” y “\t” que pueden conllevar problemas a la hora de visualizar el contenido.
- Para el campo “category”: Eliminar información innecesaria. La función `parse_book` extrae toda la información contenida en la etiqueta `<ol>` con la clase `Breadcrumbs_breadcrumb_link_OMirL` y la almacena en una lista. En esta lista, el primer elemento (categoría genérica "Libros") y el segundo (nombre del libro) son datos innecesarios y se eliminan para obtener la lista final con las categorías específicas de cada libro.

```

,
class ProcessPlanetadelibrosSpider:

    def process_item(self, item, spider):
        if spider.name != 'planetadelibros':
            return item

        #Cost
        if item['cost'] is not None:
            t1 = item.get('cost')
            t2 = t1.split()
            item['cost'] = float(t2[0].replace(',', '.'))

        #Pages
        if item['pages'] is not None:
            item['pages'] = int(item.get('pages'))

        #ISBN
        if item['isbn'] is not None:
            item['isbn'] = int(item.get('isbn').replace('-', ''))

        #Category
        if item['category'] is not None:
            if isinstance(item['category'], list):
                if len(item['category']) > 0:
                    item['category'].pop(0)
                if len(item['category']) > 0:
                    item['category'].pop(0)
                if len(item['category']) > 0:
                    item['category'].pop()

        #Synopsis
        if item['synopsis'] is not None:
            item['synopsis'] = item.get('synopsis').replace('\n', '').replace('\r', '').replace('\t', '')

        self.file = open('categories.txt', 'a', encoding='utf-8')
        for element in item['category']:
            self.file.write(f'{element}\n')

        return item

```

## Casa del libro

La página web de La Casa del Libro también fue estudiada para obtener elementos de ella y, en general, fue una de las candidatas para el scrapeado. A pesar de conseguir la mayoría de los elementos que nos interesaban para el proyecto decidimos renunciar al uso de esta página web debido a ciertos problemas a la hora de la obtención del precio, que se encontraba introducido como contenido dinámico, por lo que era menos accesible que las páginas que estudiamos previamente.

# Postprocesado de los datos

Una vez obtenidos los datos de cada una de las páginas web, el resultado es un JSON por cada una de ellas, que contiene los campos en los formatos y tipos establecidos. En general, con el procesado realizado en los pipelines específicos se consiguió obtener los datos esperados en todos los campos, pero en el caso de las categorías nos encontramos un problema: el número de categorías diferentes era demasiado alto, superando las 2000. Por ello, para poder realizar después filtrado y clustering por ellas, se optó por realizar un postprocesado para reducir ese número tan elevado.

## Clusterización de las categorías

Al ir analizando las categorías que íbamos encontrando durante el proceso de crawling nos dimos cuenta de que el número de categorías distintas era demasiado grande, sobre todo debido a la página de Buscalibre, cuyos temas eran demasiado específicos y, en nuestra opinión, poco útiles para poder filtrar o agrupar. Para ello decidimos hacer una agrupación de todas las categorías en unas preestablecidas que fuesen más amplias. El proceso para conseguirlo constó de 5 fases: obtención de las categorías únicas, creación de un diccionario inicial, agrupación basada en keywords, agrupación basada en similaridad y agrupación final.

### Obtención de categorías únicas

En esta primera fase, a la hora de crawlear cada una de las páginas, hemos ido obteniendo las diferentes categorías para cada una de ellas. Para cada una de las páginas se utilizó una aproximación algo diferente, pero todas son bastante triviales. Al final de este proceso, obtuvimos un fichero txt con las categorías únicas para cada una de las páginas:

- Librería Paz: 76 categorías.
- Planeta del Libro: 340 categorías.
- Buscalibre: 1978 categorías.

Para ilustrar un poco el problema, podemos ver como en Buscalibre las categorías son del tipo “Época de la exploración y la expansión en estados unidos | c. 1800-c. 1861” o “Época federalista estadounidense - c. 1783-c. 1800”. Son categorías demasiado específicas que encajarían dentro de una más general que fuese “Historia”.

### Creación del diccionario inicial

El siguiente paso es crear un diccionario inicial clave-valor que nos va a servir para definir las categorías que querramos tener en nuestro sistema. Para ello, establecimos 29 categorías diferentes y generales, las claves, y también una serie de valores para cada una de ellas. Así, en caso de que algún libro tuviese como categoría uno de esos valores, se le asignaría como categoría la clave del diccionario.

```
# Definir categorías generales
keywords = {
    "Poesía": ["poesía", "lírica"],
    "Hobbies": ["hobbie", "tiempo libre", "manualidades", "vacaciones"],
    "Salud": ["salud", "saude", "medicina"],
    "Educación": ["educacion", "ensino", "enseñanza"],
    "Thriller": ["thriller"],
    "Cómic": ["banda deseñada", "cómic", "banda diseñada"],
    "Romance": ["romance", "romántica", "amor"],
    "Ficción": ["Ficción", "Ciencia Ficción"],
    "Fantasía": ["Fantasía"],
    "Misterio": ["Misterio"],
    "Historia": ["Historia", "histórica", "siglo", "guerra"],
    "Biografía": ["biografía"],
    "Ensaya": ["ensayo"],
    "Psicología": ["psicología"],
    "Filosofía": ["filosfia"],
    "Ingeniería": ["ingenieria"],
    "Autoayuda": ["autoayuda"],
    "Religión": ["religión", "iglesia", "cristianismo", "judaísmo", "budismo"],
    "Biología": ["biología"],
    "Ciencia y Tecnología": ["Ciencia y Tecnología", "ciencias", "tecnología"],
    "Arte": ["arte", "escultura", "pintura", "musica", "cine", "película", "barroco", "gótico", "rococó", "renacimiento"],
    "Cocina": ["cocina", "recetas"],
    "Literatura Infantil": ["infantil"],
    "Literatura Juvenil": ["juvenil"],
    "Manga": ["manga", "shonen"],
    "Novela": ["novela"],
    "Mapas": ["mapas", "atlas"],
    "Deporte": ["deporte", "fútbol", "tenis", "baloncesto", "atletismo"],
    "Países": ["afganistán", "albania", "alemania", "andorra", "angola", "antigua y barbuda", "arabia saudita", ...]
}
```

## Agrupación basada en palabras clave

Con nuestro diccionario inicial ya definido y las categorías de las diferentes páginas en sus respectivos ficheros, empezamos con el primero de los métodos de agrupación: por palabras clave. Este es un proceso mucho más sencillo y rápido que una agrupación por similaridad, pero también menos efectivo en general.

```

# Keyword grouping function
def group_by_keywords(categories, dic):

    for category in categories:
        for general, key_list in keywords.items():
            if any(keyword in category.lower() for keyword in key_list):
                if category not in dic[general]:
                    dic[general].append(category)

    return dic

```

La agrupación basada en palabras clave consiste básicamente en recorrer nuestra lista con las categorías diferentes de cada página y ver si esa categoría coincide con alguno de los valores de nuestro diccionario:

- Si no coincide, simplemente se sigue y no se hace nada.
- Si coincide, significa que esa categoría puede ser agrupada con esa clave. Se comprueba si ese valor de categoría ya está en el diccionario, y si no está se añade. Por ejemplo, la categoría “Historia de Europa” caería dentro de la clave de “Historia”, y como esa categoría no es una de los valores el diccionario se actualizaría con ella. De esta forma, el diccionario se iría ampliando y mejorando poco a poco con las nuevas coincidencias para posteriormente poder realizar la agrupación.

Algunos de los problemas que puede haber en esta fase es que algunas categorías no se agrupen debido a que los valores establecidos al principio no son muy completos. Por ejemplo, “Época federalista estadounidense - c. 1783-c. 1800” no se agruparía dentro de “Historia” ya que no coincide con ninguno de los valores de dicha clave. Esto ocurre porque hay que buscar un balance adecuado, y este primer diccionario no debe ser ni muy genérico ni muy específico, ya que podría pasar lo contrario y que alguna categoría se clasificase en una clave que no le corresponde.

Al final de esta fase tendremos el diccionario de categorías actualizado con todos los valores nuevos añadidos.

### Agrupación basada en similaridad

La siguiente fase consiste en realizar una agrupación basada en similaridad de texto. Se ha considerado la utilización de modelos NLP como Word2Vec o Transformers, que nos permitirían tener en cuenta similaridad semántica, pero tras probarlos no acababan de funcionar del todo bien y eran mucho más pesados y difíciles de implementar. Por ello, se acabo utilizando una biblioteca más sencilla llamada *FuzzyWuzzy*.

Mediante el uso de esta librería se nos proporcionan métodos que nos permiten calcular la similaridad de texto usando la distancia Levenshtein. La distancia Levenshtein es el número mínimo de operaciones requeridas para convertir una cadena de texto en otra, entendiendo por operación una sustitución, eliminación o inserción de un carácter. Por ejemplo, la distancia Levenshtein entre “casa” y “cama” es de 1, ya que bastaría con una sustitución de la ‘s’ por la ‘m’.

Aplicando esto queremos poder añadir a nuestro diccionario aquellas categorías que son de la misma familia léxica, otras que están en gallego y cambian en alguna letra o terminación, y en especial los plurales y géneros de las palabras. Es necesario definir un umbral que mida a partir de qué nivel de distancia queremos incluir las categorías. Usando uno muy bajo, se incluirían algunas que no tuviesen nada que ver, y usando uno muy alto no se incluiría ninguna. En nuestro caso hemos optado por usar uno alto, de 0.85, ya que creemos que es mejor agrupar menos a agrupar mal.

```
# Levenshtein similarity grouping
def group_by_similarity(categories, dic):

    for category in categories:
        for general, key_list in keywords.items():
            for key in key_list:
                similarity = fuzz.ratio(category.lower(), key.lower())

                if similarity >= threshold:
                    if category not in dic[general]:
                        dic[general].append(category)
                    break

    return dic
```

El proceso es similar al anterior, así que no entraremos en detalle en él. Si se supera el umbral, se añade al diccionario, y si no lo supera no se hace nada. Una vez finalizada esta fase, tendríamos ya el diccionario final, con las 29 claves y los valores actualizados.

## Agrupación final

```
for element in paz_data:
    if "category" in element and isinstance(element["category"], list):

        if not element["category"]:
            element["category"] = ["Otros"]

    unique_categories = set()

    for i, single_category in enumerate(element["category"]):
        hit = False
        for key, value in categories_dict.items():
            if single_category in value:
                unique_categories.add(key)
                hit = True
        if not hit:
            unique_categories.add("Otros")

    element["category"] = list(unique_categories)
```

Con el diccionario ya definido, solo queda aplicarlo a los libros. Para ello, solo tenemos que recorrer cada uno de los libros de cada página y ver sus categorías; si coincide con algún valor, se le asigna la clave, y si no coincide o no tiene categoría se asigna a la categoría “Otros”.

El diccionario inicial constaba de las 29 claves con 237 valores en total. Al final de este proceso, tenemos para las mismas claves un total de 915 valores, por lo que se han agrupado más de 600. Esto nos permitirá tener un filtrado por categorías mucho más útil en nuestra aplicación final.

## Indexación

Para la indexación utilizamos, primeramente, la imagen de Docker de Elastic ejecutada en local, de manera que cada persona ejecuta su índice localmente. Para ello, utilizamos scripts de python que nos permiten tanto crear el índice como insertar los documentos en dicho índice.

## Creación del índice

El script de creación del índice es muy sencillo y se basa en la ejecución de un único comando:

```

es = Elasticsearch(
    hosts=["http://localhost:9200"]
)

index_name = "books"

mapping = {
    "mappings": {
        "properties": {
            "url": {"type": "text"},
            "name": {"type": "text", "fields" : { "suggest": {"type": "completion"}}},
            "author": {"type": "text",
                       "fields": {
                           "raw": {
                               "type": "keyword"
                           },
                           "suggest": {"type": "completion"}
                       }
                   },
            "editorial": {"type": "text",
                          "fields": {
                              "raw": {
                                  "type": "keyword"
                              }
                          }
                      },
            "edition_date": {"type": "integer"},
            "category": {"type": "text",
                         "fields": {
                             "raw": {
                                 "type": "keyword"
                             }
                         }
                     },
            "isbn": {"type": "text",
                      "fields": {
                          "raw": {
                              "type": "keyword"
                          }
                      }
                  },
            "pages": {"type": "integer"},
            "synopsis": {"type": "text"},
            "cover": {"type": "text"},
            "cost": {"type": "float"}
        }
    }
}

es.indices.create(index=index_name, body=mapping)
print(f"Index '{index_name}' successfully created.")

```

Como podemos ver, primero establecemos el objeto que conecta con nuestro puerto que está ejecutando el contenedor de Elastic y, tras ello, creamos un índice estableciendo el

nombre(en nuestro caso, “books”) y el mapeado de propiedades. Las decisiones en relación a las propiedades y sus tipos son las siguientes:

- El campo *url* se almacena como texto para poder acceder a él, aunque no se va a utilizar como elemento de búsqueda en ningún momento.
- El campo *name*, que hace referencia al título, se almacena como tipo texto, y también se establece el tipo *suggest* con el tipo *completion* con el objetivo de ofrecer al usuario autocompletado en base al título en funciones de búsqueda que veremos en la interfaz web.
- El campo *author* se almacena tanto como texto para poder hacer búsquedas sin conocer el nombre exacto de un autor como por keyword con el objetivo de acceder a todos los libros de un autor específico. De la misma forma que con el campo del título, habilitamos el tipo de completado para ofrecer la función de sugerencias en la interfaz web.
- El campo *editorial*, al igual que el autor, se ofrece como texto y keyword por los mismos motivos.
- La *edition\_date* se guarda como entero con el objetivo de usarla como método de ordenación y filtrado por fecha o época.
- *Category* se almacena como texto y keyword, con el objetivo de poder ofrecer a través del keyword funcionalidades de filtrado por categorías específicas.
- *Isbn* estaba pensando para almacenarse sólamente como keyword, ya que únicamente interesaría recuperar libros por un ISBN cuando este sea exactamente el mismo, y no por similitud a él, pero a la hora de realizar búsquedas en la UI por keyword daba fallo, así que se terminó guardando también como campo de texto para poder realizar búsquedas por él.
- *Pages* se almacena como entero para ofrecer de nuevo ordenación y filtrado por tamaño.
- *Synopsis* se almacena como campo de texto para poder hacer búsquedas en relación al resumen aparte del título y autor.
- *Cover* se almacena como texto y, al igual que la URL, es únicamente para poder tener acceso al campo posteriormente.
- *Cost* se almacena como float para, de nuevo, poder ordenar y filtrar en base al precio de los artículos.

## Inserción de documentos

El script de inserción de documentos es, de nuevo, un script muy sencillo.

```

es = Elasticsearch(hosts=["http://localhost:9200"])

index_name = "books"

with open ('../books_buscalibre_parsed.json', "r", encoding="utf-8") as file:
    buscalibre = json.load(file)
with open ('../books_paz_parsed.json', "r", encoding="utf-8") as file:
    paz = json.load(file)
with open ('../books_planeta_parsed.json', "r", encoding="utf-8") as file:
    planeta = json.load(file)

books = buscalibre + paz + planeta

actions = [
    {
        "op_type": "index",
        "_index": index_name,
        "_id": i,
        "_source":book
    }
    for i, book in enumerate(books)
]

try:
    helpers.bulk(es, actions, chunk_size=1000)
    print("All books inserted.")
except Exception as e:
    print("Error during insertion:", e)

```

Como vemos, primeramente se establece la conexión con el contenedor que está ejecutando Elastic. Una vez hecho eso, se abren los documentos que incluyen los datos de cada una de las páginas scrapeadas y se almacena la información en una variable que incluya toda la información. Finalmente, usando *helpers* de la librería de Elastic, creamos un objeto de acción que indexe cada libro en el índice “books”, con un id y los datos del libro. Esto se lo pasamos para que lo ejecute en bloques de 1000 libros.

De esta forma tan sencilla quedan indexados todos los libros scrapeados en el índice “books”.

## Interfaz Web

Para la creación de nuestra interfaz web hemos optado por utilizar la librería Search UI. Esta herramienta nos facilita tareas como la conexión con elasticsearch, la creación de componentes necesarios para buscar, filtrar y visualizar la información obtenida y otros procesos como realizar sugerencias al usuario sobre lo que está buscando.

## Pasos iniciales

Para comenzar a trabajar con la biblioteca, seguimos los pasos de instalación proporcionados en las diapositivas de la asignatura. Esto generó un proyecto con las dependencias necesarias ya configuradas, junto con una carpeta `src` que incluye la configuración inicial y un archivo `App.js`, donde se encuentra el prototipo base de la aplicación.

Dicho archivo contiene los componentes que nos aporta la biblioteca y utilizaremos para la práctica:

- SearchProvider: Este es un componente contenedor que actúa como la base para todo el sistema de búsqueda y da el contexto necesario a los otros componentes.
- ErrorBoundary: Componente de seguridad que captura errores en tiempo de ejecución dentro de otros componentes del árbol de la interfaz.
- SearchBox: Es el componente donde los usuario ingresan términos de búsqueda.
- Facet: Componente que se utiliza para implementar filtros interactivos basados en categorías o atributos específicos de los datos.
- WithSearch: Componente que proporciona acceso al estado y las acciones de búsqueda gestionadas por el contexto de **SearchProvider**.
- PagingInfo: Muestra información resumida sobre los resultados de búsqueda actuales, como el rango de resultados visibles y el número total de resultados encontrados.
- ResultsPerPage: Permite a los usuarios seleccionar cuántos resultados se deben mostrar por página.
- Paging: Gestiona la navegación entre las diferentes páginas de resultados.

## Modificaciones

Para la comodidad del usuario se ha programado tres componentes adicionales: clearFilters y un selector de tipos para la búsqueda.

- ClearFilters: Componente que se encarga de restablecer todos los filtros aplicados por el usuario en los componentes Facet.

```

import { withSearch } from "@elastic/react-search-ui";

function ClearFilters({ filters, clearFilters }) {
  return (
    <div style={{ paddingBottom: "20px" }}>
      <button onClick={() => clearFilters()}>
        style={{
          fontSize: "14px",
          padding: "10px 10px",
          justifyContent: "center",
          alignItems: "center",
          width: "270px",
          height: "35px",
        }}
      >
        Clear {filters.length} Filters
      </button>
    </div>
  );
}

export default withSearch(({ filters, clearFilters }) => ({
  filters,
  clearFilters
}))(ClearFilters);

```

- Selector: Este selector es el componente encargado de ofrecer al usuario la posibilidad de elegir el criterio de búsqueda de los libros de nuestra aplicación. Su funcionamiento se basa en la utilización de dos Hooks de React: useState y useEffect. **useState** se utilizará para gestionar el estado actual de nuestra configuración, permitiendo tanto acceder a su valor como actualizarlo según la variable seleccionada en el selector. Por otro lado, **useEffect** se encargará de volver a renderizar la página cada vez que el valor de la configuración cambie. Esto permitirá que la librería se comunique nuevamente con Elasticsearch, actualice los resultados mostrados y ajuste las sugerencias del SearchBox para que se alineen con el nuevo criterio de búsqueda. Además, como parte de la implementación, hemos modificado la función **buildSearchOptionsFromConfig** en el archivo **config-helper.js**. Esta función ahora acepta un parámetro adicional que representa el nuevo criterio de búsqueda, asegurando que las consultas se generen conforme a la configuración seleccionada por el usuario. A mayores, hemos modificado la función **BuildAutoCompleteQueryConfig** para ofrecer autocompletados en base al elemento seleccionado cuando tenga sentido (por autor en caso de buscar por autor y por título en cualquier otro caso). A pesar de numerosos intentos, no fuimos capaces de incluir un sistema de

autocompleteado por varios campos de manera simultánea, como podría ser por título y autor en una sola búsqueda, por lo que el sistema se limita a autocompletear por uno de ellos.

A su vez se han editado los componentes de la biblioteca de tal forma que se adapten a lo que queremos.

- Results: El componente results ofrece la posibilidad de cambiar como se visualiza los resultados a través de la propiedad **resultView**. Esta propiedad la hemos utilizado para mostrar el título del libro, el autor, el precio, la portada y los primeros 500 caracteres de la sinopsis.

```
<Results
  resultView={({ result }) =>
    return (
      <div className="grid-container" onClick={() => navigate("/viewDetails", { state: result })}>
        <img className="cover" src={result.cover.raw} alt="Cover" />
        <div className="info">
          <h1>{result.name.raw}</h1>
          <h2>{result.author.raw}</h2>
          <p className="basic">
            {result.synopsis.raw.slice(0, 500)}
            {result.synopsis.raw.length > 500 && '...'}
          </p>
          <p className="p-cost">{result.cost.raw} €</p>
        </div>
      </div>
    );
  }
/>
```

- Facets: En nuestra aplicación, hemos inicializado tres componentes **Facets** para implementar filtros basados en **categorías**, **precio** y **número de páginas**. La configuración de estos filtros se ha definido en el archivo **config-helper.js**, que nos permite estructurar y personalizar las opciones de forma centralizada. Los filtros son los siguientes:
  - Categorías: El filtro permite seleccionar las categorías sobre las que realizar la búsqueda, con un máximo de 100 categorías posibles.
  - Precio: Se han establecido **4 rangos** predefinidos para facilitar la selección:
    - Menos de 10€.
    - De 10€ a 19.99€.
    - De 20€ a 29.99€.
    - Más de 30€.
  - Número de páginas: Este filtro incluye **3 rangos** principales:
    - Menos de 100 páginas.
    - De 100 a 300 páginas.
    - Más de 300 páginas.

```

export function buildFacetConfigFromConfig() {
  const config = getConfig();

  const facets = (config.facets || []).reduce((acc, n) => {
    acc = acc || {};
    acc[n] = {
      type: "value",
      size: 100,
    };
    return acc;
  }, undefined);

  return {
    ...facets,
    cost: {
      type: "range",
      ranges: [
        { from: 0, to: 10, name: "Menos de 10€" },
        { from: 10, to: 19.99, name: "De 10€ a 19.99€" },
        { from: 20, to: 29.99, name: "De 20€ a 29.99€" },
        { from: 30, to: 100, name: "Más de 30€" },
      ],
    },
    pages: {
      type: "range",
      ranges: [
        { from: 0, to: 99, name: "Menos de 100 páginas" },
        { from: 100, to: 300, name: "De 100 a 300 páginas" },
        { from: 301, to: 5000, name: "Más de 300 páginas" },
      ],
    },
  };
}

```

## Visualización

La pantalla de inicio de nuestra página se divide en tres secciones principales. La primera sección es un encabezado que incluye un buscador, con el cual el usuario puede realizar sus búsquedas. Además, contiene un filtro que permite seleccionar el criterio de búsqueda deseado, así como dos botones: uno para ejecutar la búsqueda y otro para regresar al estado principal de la página.

La segunda sección se encuentra en el lateral izquierdo de la pantalla y presenta todos los filtros que se pueden aplicar a la búsqueda, junto con un botón para restablecerlos.

Finalmente, la tercera sección muestra parte de la información de los libros encontrados según el criterio de búsqueda actual. Además, incluye componentes que indican cuántos libros se han encontrado, cuántos se están mostrando actualmente y en qué página se encuentra el usuario. También se ofrece un selector para que el usuario especifique cuántos libros desea mostrar por página.

The screenshot shows a search interface with a header bar containing 'Inicio', 'Por defecto', a search input field 'Introduce el título, autor, ISBN...', and a 'Search' button. Below the header, there's a 'Clear 1 Filters' button and a 'Showing 1 - 20 out of 7602' message. On the left, there are three filter sections: 'CATEGORIAS' (with 'Literatura Infantil' checked), 'PRECIO' (with options like 'Menos de 10€'), and 'PÁGINAS' (with options like 'Menos de 100 páginas'). In the center, a book cover for 'Por que Tengo que Compartir (Mis Primeras Preguntas)' by Katie Daynes is displayed, along with its price of 10.4 €. At the bottom right, there's a 'Show 20' dropdown menu.

Una vez que se selecciona un libro, se abrirá la ventana de detalles del mismo. Esta página muestra toda la información recopilada sobre el libro, incluyendo su título, autores, categorías y la sinopsis completa. En la esquina inferior izquierda se indica el coste del libro y en qué página se encuentra la oferta correspondiente. Si el usuario pulsa el botón, la página lo redirigirá al dominio indicado, donde podrá proceder con la compra del libro.

Cabe destacar que, si un mismo libro está disponible en más de un dominio, este se mostrará varias veces en la lista de resultados. El usuario deberá seleccionar la oferta específica que desea, según el dominio y las condiciones correspondientes.

The screenshot shows a detailed view of a book titled 'Trono de Cristal' by Sarah J. Maas. It includes the author's name, publisher (Editorial Hidra), ISBN (9788410163621), and page count (492). A brief description in Spanish is provided: 'EDICIÓN ESPECIAL LIMITADA EN TAPA DURA CON LOS CANTOS PINTADOS A COLOR ¡SOLO HASTA FIN DE EXISTENCIAS! En las tenebrosas minas de sal de Endovier, una muchacha de dieciocho años cumple cadena perpetua. Es una asesina profesional, la mejor en lo suyo, pero ha cometido un error fatal. La han capturado. El joven capitán Westfall le ofrece un trato: la muerte a cambio de un enorme sacrificio. Celaena debe representar al príncipe en un torneo a muerte, en el que deberá luchar con los asesinos y ladrones más peligrosos del reino. Viva o muerta, Celaena será libre. Tanto si gana como si pierde, está a punto de descubrir su verdadero destino. Pero ¿qué pasará entretanto con su corazón de asesina?' At the bottom right, there's a note: 'BuscaLibre: 24.65 €'.