Facultade de Informática
UNIVERSIDADE DA CORUÑA

Scraping Fitness

Estudiante: Jaime Cabero Creus
Estudiante: David García Gondell

A Coruña, December de 2021.

Índice general

Capítulo 1
Introducción

Este primer capítulo de la memoria incluirá una explicación del sistema desarrollado y de los objetivos que buscamos conseguir con este proyecto.

1.1 Contextualización y descripción del sistema

Para este proyecto se han considerado multitud de dominios sobre los que extraer datos, para ello hemos tenido en cuenta que tuvieran una cantidad de información suficiente, que esta fuera accesible y por último que el propio dominio no contase con suficientes mecanismos para tratar esos datos. Finalmente, hemos decidido por extraer datos de una cadena de supermercados para proporcionar diferentes dietas y búsquedas de alimentos a nuestros usuarios.

Una vez elegido el dominio, se ha pensado en como hacer el proceso de extracción y tratamiento de los datos. El sistema ideado constará de un scraper en lenguaje Python que se encarga de recopilar información de la página web de supermercados Dia, de un servicio Elasticsearch para indexar la información y de una aplicación web SPA construida utilizando React que se encarga de mostrar las búsquedas en Elasticsearch.

Capítulo 2

Tecnologías empleadas

En este capítulo se expondrán las tecnologías empleadas en este proyecto. Se explicará el motivo de su elección y el uso dado a las mismas en el proceso de desarrollo.

2.1 Servicio

2.1.1 Python

Python es un lenguaje de programación de alto nivel. Se trata de un lenguaje multiplataforma y de código abierto, que no necesita de un compilador que lo traduzca a lenguaje máquina, sino que se sirve de un interpretador para ejecutarse. En este proyecto se ha empleado Python para la implementar el servicio ya que dispone del framework Scrapy para recolectar los datos que posteriormente vamos a tratar.

2.1.2 Scrapy

Scrapy es un framework de *scraping* y *crawling* de código abierto escrito en lenguaje Python; en nuestro proyecto lo utilizamos para realizar *web scraping*, que es una técnica de extracción de información de páginas web de manera automatizada. Scrapy permite declarar clases ”Spider” de forma sencilla que se encargan de obtener. la información de un tipo de dato disponible en un dominio o conjunto de dominios. Además, una vez obtenidos los datos se puede configurar un pipeline para tratar todos los documentos de uno en uno y enviarlos a otra plataforma, en nuestro caso Elasicsearch.

Decidimos emplear Scrapy frente a otras alternativas porque recibimos una charla sobre como utilizarlo y nos pareció que su funcionamiento era sencillo; además hay que destacar que la documentación existente de Scrapy es rica y está bien estructurada.

2.1.3 Elasticsearch

Eslasticsearch es un motor de búsquedas de código abierto basado en Lucene. Este motor cuenta con APIs que nos permiten crear índices de búsqueda desde el servicio y, una vez construidos, realizar consultas de todo tipo en formato JSON sobre estos índices. Decidimos emplear Elasticsearch debido a que nos pareció sencilla la API que exponía y cómodo su sistema de consultas. Además su API esta disponible tanto para python como para javascript, lo que facilita mucho su uso.

2.2 Aplicación web SPA

2.2.1 React

React es una biblioteca para la construcción de interfaces de usuario en el lenguaje de programación *JavaScript* . Está basado en componentes encapsulados que gestionan su propio estado y tiene un enfoque declarativo. Funciona actualizando y renderizando los componentes correctos de manera eficiente solo cuando los datos cambian. Decidimos emplear React en este proyecto debido a que es una tecnología familiar al equipo de desarrollo y nos permite crear interfaces de usuario de manera rápida y sencilla.

2.2.2 Redux

Redux es una herramienta creada para la gestión del estado en aplicaciones programadas en *JavaScript*, además funciona de manera óptima en conjunto con React, otra de las tecnologías empleadas en este proyecto. Se tomó la decisión de utilizar Redux en este proyecto debido a que al emplear React para el desarrollo de la aplicación web SPA, es muy recomendable realizar una gestión activa del estado de los componentes, para garantizar un comportamiento eficiente de la aplicación.

2.2.3 Material UI

Material UI es una librería de componentes de React con una estética común y una utilidad determinada, que están listos para ser integrados en cualquier proyecto que emplee React. Su diseño está basado en *Material Design*, una serie de especificaciones y guías publicadas por Google para la construcción de interfaces de usuario en aplicaciones Web y móviles. Decidimos utilizar Material UI para darle un aspecto visual atractivo y homogéneo a la aplicación. Además, el equipo de desarrollo está familiarizado con esta tecnología, con lo que se consigue una interfaz de usuario estética en tiempo récord.

Capítulo 3

Desarrollo

En este apartado de la memoria se describirán las funcionalidades y el proceso de desarrollo del proyecto. En él, se abarcará el desarrollo del servicio, el proceso de *scraping* e indexación de los resultados obtenidos y por último el tratamiento de los datos indexados y su representación en una interfaz de usuario con cierta funcionalidad disponible.

3.1 Funcionalidades

Las funcionalidades desarrolladas en esta práctica son las siguientes:

- Búsqueda de productos por palabras clave
- Búsqueda de productos por precio
- Búsqueda de productos por información nutricional
- Comprar producto (lleva a la página correspondiente en la web de Dia)
- Recomendación de productos de dieta
- Recomendación de productos para dieta de musculación

Estas funcionalidades se explican más en profundidad a continuación.

3.2 Servicio

Vamos a comenzar hablando del desarrollo del servicio. En primer lugar, se ha creado un proyecto de Python con el comando **”scrapy startproject riwsScraper”**.

Este comando creó un directorio ”riwsScraper” con el siguiente contenido:

```
riwsScraper/
    scrapy.cfg            # deploy configuration file

    riwsScraper/             # project's Python module, you'll
    import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your
    spiders
            __init__.py
```

Dentro de este directorio se pueden ver diversas clases en Python que ha creado Scrapy. Estas se reparten entre clases de configuración, la clase que sirve de pipeline, y un directorio que contendrá las clases *Spider* que son las encargadas de realizar el proceso de *crawling*, como ya se explicó anteriormente.

3.2.1 Fichero de configuración

El fichero *settings.py* es el encargado de albergar la configuración del servicio. En él viene una configuración inicial que se puede modificar según las necesidades del usuario.

En nuestro caso le pusimos un nombre a nuestro bot, le indicamos donde se ubicaba nuestro módulo de *Spiders* y configuramos nuestro pipeline. Además, limitamos el número de peticiones concurrentes a 8 y le introducimos un pequeño *delay* de un segundo. También, marcamos que nuestro bot siguiera las indicaciones de los ficheros *robot.txt* de las páginas web, los cuales indican a los bots qué URLs del sitio web son las que tienen permitido acceder.

Por último, nos identificamos con el *user-agent*: "$Mozilla/5.0(Macintosh; IntelMacOSX10.12.6)$ $AppleWebKit/537.36(KHTML, likeGecko)Chrome/61.0.3163.100Safari/537.36$"
Todo esto lo hemos hecho de manera preventiva para que no nos identificase como un bot de origen desconocido y nos bloqueasen el acceso.

3.2.2 Fichero *DiaSpider*

Es una clase que extiende la lógica de la clase *CrawlSpider*, y a la que se le da un nombre, un dominio y unas direcciones donde empezar a el proceso de *crawling*.

Este proceso sigue una reglas para discriminar lo que se considera un *item* de lo que no. Estas reglas están construidas como expresiones regulares que nos permiten describir la estructura modelo de una URL que se corresponde con el esquema de dirección de lo que se considera un *item*.

Cada uno de estos *items* los *parsea* utilizando una función $parse-item$, donde en nuestro caso asignamos a una serie de variables, el contenido de los elementos HTML de la pagina correspondiente a un *item*. Con estas variables construimos objetos, que devolvemos en formato JSON para poder indexarlos más adelante con la API de Elasticsearch.

3.2.3 Fichero *pipeline*

Este fichero es una clase que se inicializa al mientras se recopila la información de la web creando un indice de Elasticsearch mediante su API de Pyhton, en este caso llamado *products*. Luego, mediante esta API se indexan todos los documentos que encuentra scrapy durante el proceso de *crawling* utilizando un método que recibe cada *item* como parámetro de forma automática. En nuestro caso, cada *item* tiene un ID que corresponde a la URL del producto y como contenido los siguientes elementos:

- Nombre
- Subcategorías del producto
- Precio
- Ausencia de gluten
- KJ
- Kcal
- Grasas
- Grasas saturadas
- Carbohidratos
- Azúcares
- Proteínas
- Sal
- Ratio de gimnasio (Kcal/Proteínas)

3.3 Interfaz de usuario

Para el desarrollo de la interfaz de usuario se ha creado un proyecto de React mediante el comando **create-react-app**. Con este proyecto básico creado, hemos instalado las dependencias necesarias para poder emplear las tecnologías de Redux y Material UI.

Una vez estuvo lista esta configuración inicial, decidimos lo que íbamos a mostrar con los datos obtenidos. Para ello, primero comenzamos estableciendo dos componentes separados en la aplicación, uno dedicado a mostrar los productos obtenidos y el otro dedicado a realizar búsquedas y aplicar filtros sobre estos productos. Además, en los que cuenten con información nutricional, esta podrá consultarse en formato de tabla al presionar el botón del mismo nombre. El resultado puede apreciarse en la Figura 3.1.

Figura 3.1: Ejemplo de producto completo

En cuanto a los filtros que dispone la aplicación, esta se pensó para que contara con tres modos: el normal, el modo de dieta y el modo de musculación. Estos modos tienen en común un filtro por nombre, otro por precio y tres filtros que se consideran básicos, que nos permiten

obtener productos sin gluten para celíacos, productos sin sal (menos de 0.12g), o productos sin azúcar (menos de 0.5g).

El modo normal, es en el que se encuentra por defecto. Contendrá todos los productos obtenidos en el proceso de scraping y permitirá al usuario utilizar todos los filtros de los que dispone la aplicación. Esta consulta se hace con una query booleana en la que se filtra por nombre y subcategorías con la siguiente sentencia *should*:

```
should: [
    {
        match: {
            name: {
                query: nameSearch,
                fuzziness: "AUTO",
                zero_terms_query: "all"
            }
        }
    },
    {
        match_bool_prefix: {
            name: {
                query: nameSearch
            }
        }
    },
    {
        match: {
            subcategories: {
                query: nameSearch,
                fuzziness: "AUTO",
                zero_terms_query: "all"
            }
        }
    },
    {
        match_bool_prefix: {
            subcategories: {
                query: nameSearch
            }
        }
    },
]
```

Los filtros de información nutricional en cambio, se hacen gracias a la sentencia *filter* de

la siguiente forma:

```
filter: [
    {
        range: {
            glutenLess: {
                gte: noGlutenCheck ? true : false,
                lte: true
            }
        }
    },
    .
    .
    .
]
```

Además, la vista que se encarga de mostrar este modo se puede ver en la figura Figura 3.2.:

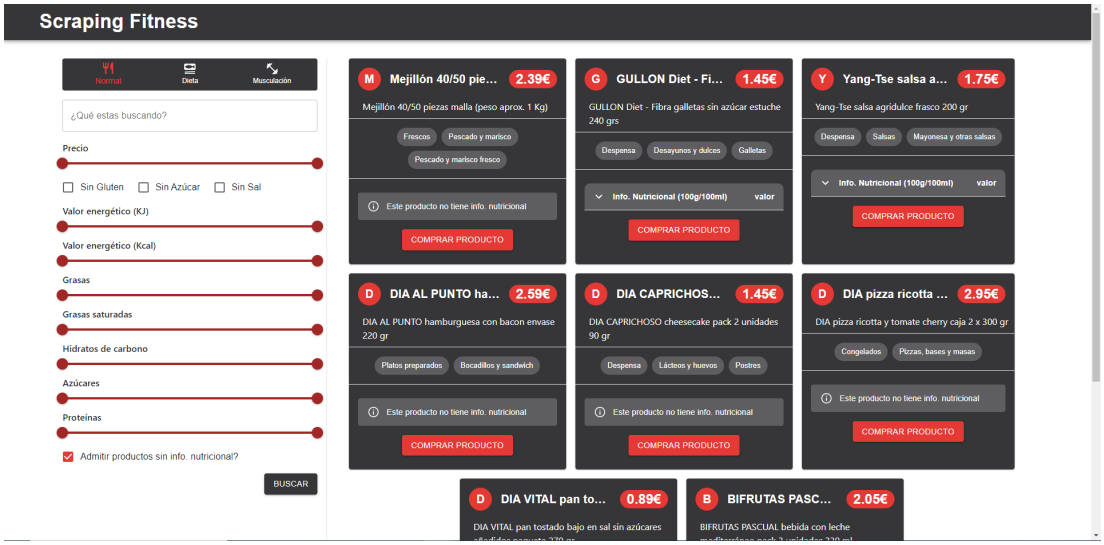

Figura 3.2: Aplicación en modo normal

El modo de dieta, muestra productos que cuenten con un número reducido de calorías, azúcares, grasas y grasas saturadas. En este modo también se podrá filtrar por precio y restringir los alimentos con gluten, sal o azúcar. Estos productos se recuperan de Elasticsearch empleando la siguiente consulta:

Para la consulta del modo dieta se emplea una query function score, que nos permite cambiar el score que le da Elasticsearch a los documentos poniendole pesos a subconjuntos de los mismos. Un ejemplo de la consulta sería el siguiente:

```
query: {
    function_score: {
        query: { ... },
        functions: [
            {
                filter: {
                    range: {
                        kcal: {
                            gte: 0,
                            lte: 100
                        }
                    }
                },
                weight: 30
            },
            {
                filter: {
                    range: {
                        kcal: {
                            gte: 100,
                            lte: 200
                        }
                    }
                },
                weight: 15
            }
            .
            .
            .
        ]
    }
```

En la figura Figura 3.3 se puede ver una consulta en el modo dieta donde se filtra además por productos sin gluten.

Por último, en el modo dedicado a la musculación se muestran principalmente productos que tienen una buena relación entre su aporte calórico y proteico, como se puede apreciar en la sección destacada en color azul en la Figura 3.4. Como sucedía en el modo de dieta, se puede filtrar también por precio, y eliminar de la búsqueda los alimentos con gluten, sal o azúcar. La consulta empleada para recuperar estos productos se hace con una query booleana en la que se filtra por el ”gimRatio”, que no es más que la relación entre las calorías del producto y sus proteinas, lo que da bastante buenos resultados:

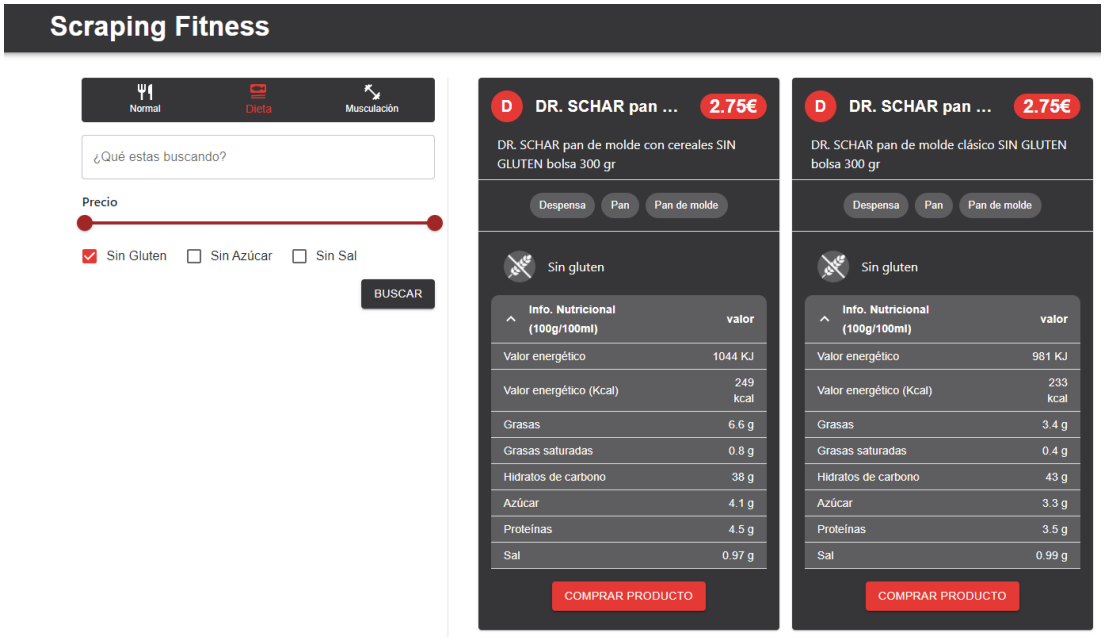

Figura 3.3: Modo de dieta

```
{
    range: {
        gymRatio: {
            gte: 0,
            lte: 15
        }
    }
}
```

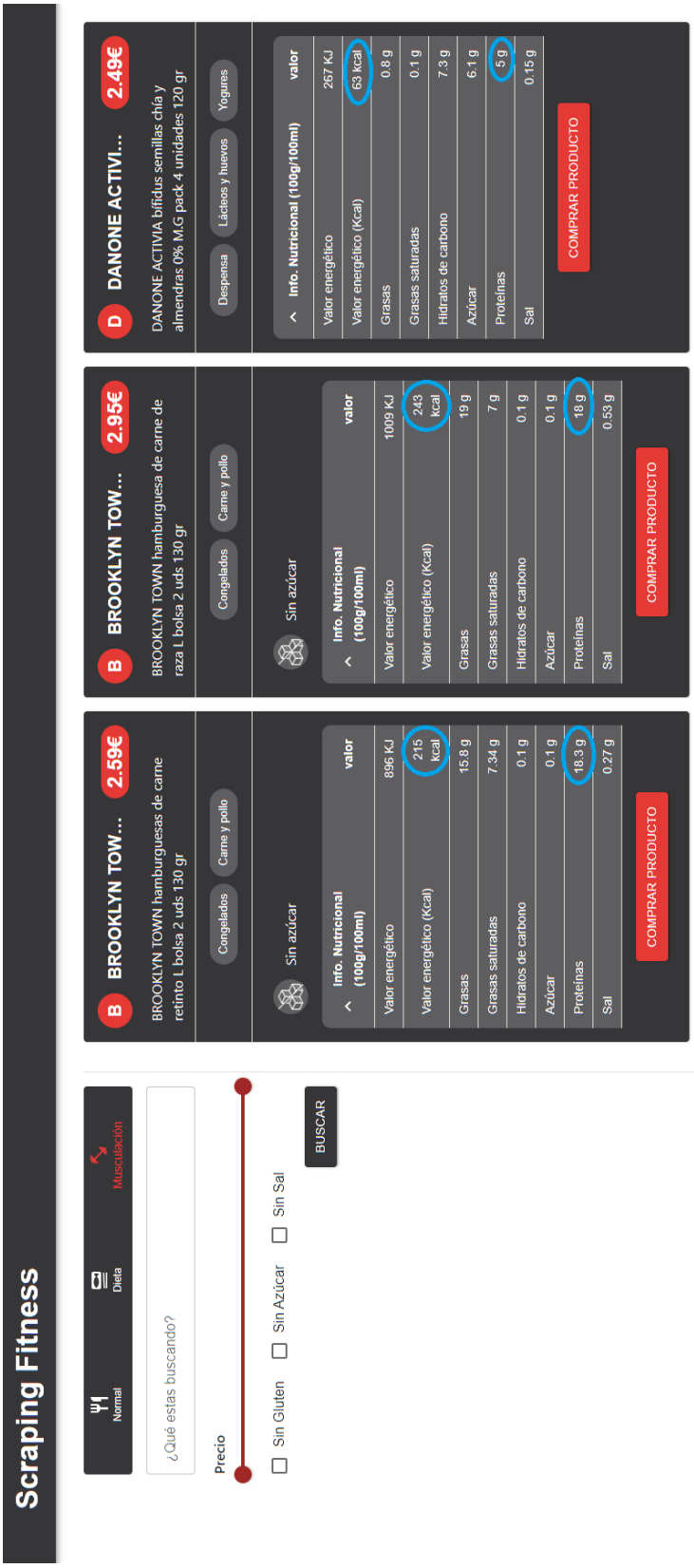

Figura 3.4: Modo de musculación