Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

Grado en Ingeniería Informática en Tecnologías de la Información

Área de Ingeniería Informática

TRABAJO FIN DE GRADO

Aplicación personalizable para la descarga de datos mediante web scraping

D. Picos Rodríguez, Adrián

TUTOR: D. Pérez Núñez, Pablo

COTUTOR: D. Diez Peláez, Jorge

FECHA: Julio de 2023

Índice de Contenido

Índice de Figuras

Índice de Tablas

Resumen

El web scraping es una técnica que permite recopilar información de sitios web de manera estructurada y automatizada. Este proyecto tiene como objetivo hacer uso de esta técnica para extraer datos de diversas páginas web predefinidas buscando facilitar la creación de conjuntos de datos que no están públicamente disponibles, no existen o no se poseen el formato conveniente para su análisis.

El proyecto se basa en el desarrollo de una aplicación de escritorio que permite a los usuarios seleccionar una página web específica y unos criterios de extracción de datos. A través de técnicas de web scraping, la aplicación accederá a la página web seleccionada, navegará por su contenido y extraerá los datos requeridos de acuerdo a los parámetros establecidos por el usuario. Además, se implementarán funcionalidades para exportar los datos en diferentes formatos facilitando su posterior uso y manipulación.

El objetivo final de este proyecto es proporcionar una herramienta fácil de usar que permita a los usuarios con pocos conocimientos web recopilar datos de distintas fuentes definidas de manera eficiente, ahorrando tiempo y esfuerzo en el proceso de extracción y formateo de datos. Además la aplicación está estructurada de tal forma que un usuario experto puede aumentar sus funcionalidades añadiendo nuevas webs.

1. Memoria

El objetivo de este documento es explicar los pasos que se han dado desde el principio del proyecto hasta el final y la explicación de las herramientas utilizadas.

1.1.- Introducción

En la actualidad la cantidad de información disponible es inimaginable y de alto valor para muchas aplicaciones como educación, investigación, análisis competitivo, seguimiento de noticias y un largo etcétera. A lo largo de la historia hemos oído de diferentes autores la popular frase de "*El conocimiento es poder*" [1] nos encontramos en un momento histórico con la mayor información disponible.

Tener tanta información disponible al alcance de todos, a priori parece una ventaja, pero nos plantea varios problemas. ¿Cómo la extraemos de forma eficiente?, ¿Cómo la almacenamos?, ¿Dónde la consultamos?

La respuesta sencilla a estos problemas sería buscar lo que necesito en las webs que haga falta, copiar los datos que me interesen y darles un formato. Esto hasta cierto volumen de datos podría ser un método viable. Pero existe una solución más sencilla, el web scraping. El **web scraping** la técnica que permite obtener un gran volumen de datos públicos de diferentes sitios web de forma automatizada y convertirlos en formatos más manejables como HTML, CSV, Excel, JSON, txt [2]. El proceso consta de 3 partes:

1. Analizar a través de un sitio web HTML.
2. Extraer los datos necesarios.
3. Almacenar los datos.

Ventajas que nos aporta:

- La extracción de datos es automatizada.
- Rapidez.
- Eficiencia.
- Orden y estructura en los datos.

Frente a esto, nos encontramos un inconveniente: se requiere de conocimientos de programación avanzados para poder utilizar las librerías y herramientas disponibles. Es cierto que existen opciones de alto nivel pero presentan varios problemas que veremos más adelante.

Con este proyecto planteamos una aplicación personalizable y fácilmente ampliable que permita la descarga de datos de diferentes webs y la exportación de los mismos .

Como comentamos al principio, la información es muy variada y dependiente de su uso, por lo que vamos a centrarnos en páginas webs de productos. Estas presentan varios conjuntos de información estructurada, distribuida en varias páginas y con varios niveles de profundidad. Debido a esto hace que sean el mejor ejemplo para implementar, ya que abordan los principales retos del web scraping.

1.2.- Objetivo y alcance del trabajo

Los objetivos de este proyecto son:

- Desarrollar una aplicación para poder consultar información de varias webs de productos sólo a través de interacción con la interfaz.
- Diseñar una aplicación escalable, que favorezca a futuras ampliaciones e implementaciones.
- Cubrir varias categorías dentro de la misma web.
- Poder especificar el numero de productos a exportar.
- Estructurar los datos para favorecer la exportación.
- Apariencia sencilla e intuitiva para el usuario.
- Ofrecer una forma sencilla de exportar la información en formatos conocidos.

1.3.- Resumen de las funcionalidades de la aplicación

Se van a detallar las opciones del usuario a la hora de utilizar la aplicación:

- El usuario podrá seleccionar la web y su respectiva categoría de una lista de disponibles.
- El usuario podrá seleccionar de una lista los atributos de cada producto que desea extraer.
- El usuario podrá definir cuantos productos desea exportar.
- Se exportará la información en el formato seleccionado (CSV, XML, etc).
- Ver un log durante el proceso.
- Interfaz sencilla e intuitiva.

1.4.- Descripción de la aplicación

La aplicación consta de dos componentes, la interfaz, con la cual el usuario interactúa y la funcionalidad, encargada de obtener, interpretar y presentar los datos solicitados por el usuario a través de la interfaz.

La aplicación ha sido desarrollada en Python, un lenguaje de alto nivel interpretado. Esto implica que el dispositivo en el que se vaya a ejecutar debe tener Python instalado [7].

Al ser un lenguaje interpretado, se requiere la presencia de un intérprete. El intérprete es el programa responsable de ejecutar las líneas de código que conforman la aplicación y realizar los cálculos necesarios. Además del intérprete, es necesario instalar una serie de librerías para asegurar el correcto funcionamiento del programa. Todo esto se explica en detalle en 6.

1.5.- Documentación que acompaña al proyecto

Esta es la lista de documentos que acompañan al proyecto:

- **Memoria**: Documento donde se va a incluir toda la información acerca del proyecto, incluyendo la descripción de la aplicación, información sobre el proceso de desarrollo, explicación detallada de las decisiones tomadas y el lenguaje de programación seleccionado, así como cualquier otra información relevante necesaria para comprender el proyecto en su totalidad.
- **Planificación y Presupuesto**: Este apartado incluye la planificación temporal de las actividades llevadas a cabo, junto con una estimación del presupuesto necesario para la realización del proyecto. La planificación incluirá los roles asignados a cada miembro del equipo, desglosados y definidos. El presupuesto tiene en cuenta los costes humanos y el coste hardware.
- **Requisitos del sistema**: Este apartado incluye descripción de los requisitos necesarios para el sistema en el cual se ejecutará la aplicación, así como los requisitos que el usuario debe cumplir para utilizarla correctamente. Esto abarca aspectos técnicos, como los recursos de hardware y software necesarios para el sistema, así como los conocimientos o habilidades que el usuario debe tener para interactuar con la aplicación de manera efectiva.
- **Estudio de Viabilidad del Sistema (EVS) y Análisis del Sistema de Información (ASI):** Este apartado incluye análisis exhaustivo de las necesidades del proyecto, teniendo en cuenta tanto las restricciones económicas como las técnicas.

Se consideran los recursos financieros disponibles y las limitaciones tecnológicas para determinar los requisitos del proyecto de manera realista. Además, se utiliza un Análisis de Sistemas de Información (ASI) para obtener una descripción detallada del sistema de información, basándose en el catálogo de requisitos donde se recopilan as funcionalidades y características necesarias del proyecto.

- **Diseño del Sistema de Información (DSI)**: Este apartado incluye la estructura del sistema, así como la interfaz con la cual el usuario final interactúa. Esto implica definir la arquitectura general del sistema, incluyendo los componentes, librerías y su organización. Además, se describe la interfaz de usuario, detallando los elementos visuales, la disposición de los mismos y la forma en que el usuario interactúa con el sistema.
- **Protocolo de pruebas**:Este apartado incluye un detallado registro de las pruebas a las que será sometida la aplicación, junto con las conclusiones obtenidas a partir de los resultados obtenidos. Se describen los diferentes tipos de pruebas así como la salida espera y la obtenida.
- **Manual de usuario**: Este apartado incluye instrucciones detalladas para que un nuevo usuario pueda instalar y configurar todos los elementos necesarios para que la aplicación funcione de forma correcta. Se incluye los pasos de forma clara para la instalación de los componentes del sistema, como el software principal, bibliotecas o dependencias adicionales, así como cualquier configuración requerida. Se incluye una guía de como implementar nuevas webs.
- **Conclusiones**: Este apartado incluye un resumen de las dificultades encontradas durante el proyecto. Se proporcionan sugerencias sobre posibles mejoras y ampliaciones que se podrían implementar en futuras iteraciones o versiones de la aplicación. Además, se incluye una reflexión general sobre el aprendizaje adquirido a lo largo del proyecto.
- **Futuras ampliaciones**: Este apartado incluye las mejoras y ampliaciones identificadas durante las fases de diseño y desarrollo de la aplicación, pero que no se implementaron debido a limitaciones de tiempo, conocimientos o porque se encontraban fuera del alcance del proyecto. Estas mejoras y ampliaciones podrían considerarse para futuras actualizaciones o versiones de la aplicación.

- **Referencias**: Este apartado incluye las páginas de donde se obtuvieron referencias e información para la realización del proyecto.
- **Anexos**: Se incluye una explicación de las partes principales del código que complementa al manual de usuario.

1.6.- Información adicional de interés

En este apartado se explicarán términos que se van a utilizar a lo largo del documento y que es interesante que el lector o usuario de la aplicación conozca a la hora de utilizar o ampliar el proyecto.

1.6.1.- Graphic User Interface (GUI)

La Interfaz Gráfica de Usuario, a partir de ahora nos referiremos a el como GUI por sus siglas en inglés, es un programa que consiste en un conjunto de elementos gráficos diseñados para que el usuario interactúe con un sistema operativo, dispositivo o aplicación. A diferencia de las interfaces de texto (como las interfaces de línea de comandos), que requieren conocimientos específicos por parte del usuario, las interfaces gráficas son más amigables y fáciles de utilizar.

La interacción con una GUI se realiza manipulando los objetos gráficos que la componen, lo que facilita su adaptación a diferentes dispositivos, como teléfonos móviles. Además de su facilidad de uso, las GUI permiten que el usuario interactúe con los elementos gráficos, como ampliar imágenes, desplazarse por ellas, etc.

Este tipo de interfaz se alinea con el modelo vista-controlador, el cual será explicado más adelante4.2.1. En este modelo, la funcionalidad se implementa de forma separada de la interfaz. Esto facilita la posibilidad de cambiar el aspecto del programa sin necesidad de modificar la funcionalidad subyacente de cada acción.

1.6.2.- Programación orientada a objetos (POO)

La programación orientada a objetos, a partir de ahora nos referiremos a ella como POO [17], es un paradigma de programación basado en la conceptualización y manipulación de objetos. Los objetos son estructuras que contienen información (atributos) y se definen mediante métodos que operan sobre esos datos. Los métodos permiten acceder a los campos de los objetos para obtener sus valores o modificarlos. La POO se utiliza como enfoque de programación para facilitar la escalabilidad y reutilización del código, evitando la repetición innecesaria de código. También es recomendable para el trabajo en equipo, ya que promueve una organización clara y modular del código.

- **Clases**: Las clases son los esqueletos o plantillas que definen los campos y métodos que tendrán los objetos pertenecientes a esa clase. En una clase se establecen los atributos (campos de datos) y los comportamientos (métodos) que serán utilizados por los objetos de esa clase.
- **Objeto**: Los objetos son las instancias concretas de una clase. Cada objeto tiene los campos definidos en la clase y posee valores específicos para esos campos. Además, los objetos pueden hacer uso de los métodos previamente definidos en la clase para llevar a cabo diferentes acciones y operaciones.

1.6.3.- HyperText Markup Language (HTML)

En este apartado se introduce brevemente a HTML [19], este es uno de los principales lenguajes utilizados en el desarrollo web. Junto a CSS, que veremos en el siguiente apartado, permiten crear y dar estilo a páginas web.

HTML es un lenguaje de marcado utilizado para estructurar y organizar el contenido de una página web. Proporciona una serie de elementos y etiquetas que definen la semántica y la estructura del contenido. Cada elemento HTML se coloca dentro de etiquetas corchetes angulares (<>) que indican su función y relación con otros elementos. Algunos elementos HTML comunes son:

- **Encabezados:** Representados por las etiquetas <h1>a <h6>, se utilizan para definir los títulos y subtítulos de una página.
- **Párrafos**: Representados por la etiqueta <p>, se utilizan para organizar el texto en bloques de párrafos.
- **Enlaces**: Representados por la etiqueta <a>, se utilizan para crear hipervínculos a otras páginas web o recursos.
- **Imágenes**: Representadas por la etiqueta <img>, se utilizan para insertar imágenes en una página web.
- **Listas**: Se pueden crear listas ordenadas (<ol>) o listas no ordenadas (<ul>) utilizando las etiquetas para cada elemento de la lista.
- **Tablas**: Representadas por la etiqueta <table>, se utilizan para organizar datos en filas y columnas.

1.6.4.- Cascading Style Sheets (CSS)

En este apartado se introduce brevemente a CSS [18], es uno de los principales lenguajes utilizados en el desarrollo web para darles estilo. **CSS**, es un lenguaje de estilo que se utiliza para controlar la presentación y la apariencia de los elementos HTML en una página web. Con CSS, puedes dar formato a los elementos HTML, lo que te permite personalizar el diseño de la página y mejorar la experiencia visual de los usuarios. Algunos conceptos y elementos CSS comunes son:

- **Selectores**: Permiten seleccionar los elementos HTML a los que se les aplicará un estilo. Pueden ser selectores de etiqueta, clase o identificador, entre otros.
- **Propiedades**: Son los atributos que se aplican a los elementos seleccionados. Por ejemplo, color, font-size, background-color, etc.
- **Clases y ID**: Se utilizan para asignar estilos específicos a elementos HTML. Las clases se definen utilizando el atributo class, mientras que los identificadores se definen con el atributo id.
- **Tipografía**: CSS ofrece propiedades como font-family, font-size y text-align para controlar la apariencia del texto, incluyendo la fuente, el tamaño y la alineación.

1.6.5.- Web scraping

Como vimos en la introducción 1 el web scraping es una técnica utilizada para extraer información y datos de sitios web de manera automatizada. Consiste en escribir programas o utilizar herramientas que navegan por las páginas web, analizan su contenido y extraen los datos de interés de manera estructurada. El proceso de **web scraping** generalmente sigue los siguientes pasos:

1. **Obtención de la URL**: El primer paso es identificar la página web de la cual se desea extraer los datos y obtener la URL correspondiente.
2. **Análisis de la estructura de la página**: Antes de realizar el scraping, es importante entender la estructura y el formato de la página web. Esto implica examinar el código HTML de la página y determinar qué elementos contienen los datos que se desean extraer.
3. **Extracción de datos**: Utilizando técnicas de programación, se desarrolla un programa o se utiliza una herramienta de scraping para acceder a la página web y extraer los datos deseados. Esto puede implicar buscar elementos específicos, seguir enlaces o interactuar con formularios, entre otras acciones.
4. **Transformación y almacenamiento de datos**: Una vez que se han extraído los datos, es común realizar transformaciones en ellos para limpiarlos y estructurarlos de manera coherente. Los datos pueden almacenarse en una base de datos, en un archivo CSV (valores separados por comas) u otros formatos según las necesidades.

2. Planificación y presupuesto

2.1.- Etapas de la planificación

El proyecto se compone de las siguiente etapas:

1. Estudio de viabilidad del sistema.
2. Análisis del sistema de información.
3. Diseño de la aplicación.
4. Desarrollo de la aplicación.
5. Protocolo de pruebas.
6. Documentación del proyecto.

2.1.1.- Estudio de la viabilidad del sistema

En esta parte se describe la etapa inicial del proyecto, la cual se enfoca en la descripción del mismo y la delimitación de su alcance. Se llevará a cabo un estudio de mercado con el objetivo de diseñar una aplicación que ofrezca servicios distintos a las existentes en el mercado. Durante esta etapa también se identifican y definen los usuarios involucrados en el proyecto, así como los requisitos que la aplicación deberá cumplir, tanto en términos funcionales como no funcionales.

Se estima que esta etapa requerirá aproximadamente **100 horas** de trabajo para completarse.

2.1.2.- Análisis del sistema de información

Esta es la segunda etapa del proyecto, la cual se enfoca en la definición del sistema y sus subsistemas. Se identifican los distintos componentes que conforman el proyecto y se establecen la interacción entre ellos. Además, se definen los actores que interactúan con la

aplicación, es decir, los usuarios finales. En esta etapa también se determina la estructura de los datos que serán utilizados por la aplicación. El último paso es la definición de las interfaces de la aplicación. Estas establecen cómo el usuario se comunica con la aplicación, qué acciones puede realizar y como se presentan los resultados.

Se estima que esta etapa requerirá aproximadamente **150 horas** de trabajo para completarse.

2.1.3.- Diseño de la aplicación

La siguiente etapa del proyecto es llevar a cabo dos tareas, la definición de las tecnologías que se utilizan para el desarrollo de la aplicación y el diseño de la arquitectura del sistema. Además, se realiza un primer esquema que describe como será la navegación del usuario dentro de la aplicación.

Se estima que esta etapa requerirá aproximadamente **150 horas** de trabajo de trabajo para completarse.

2.1.4.- Desarrollo de la aplicación

Después de finalizar la etapa de diseño, se da paso a la etapa de desarrollo, en la cual se lleva a cabo la implementación de toda la funcionalidad de la aplicación mediante la escritura del código. Siguiendo el diseño previamente realizado, se crea también la interfaz de usuario

Se estima que esta etapa requerirá aproximadamente **300 horas** de trabajo para completarse.

2.1.5.- Protocolo de pruebas

La finalidad de esta etapa es realizar validaciones y pruebas de la aplicación, con el objetivo de comprobar si satisfacen los requisitos de diseño y corregir posibles fallos de funcionamiento que se detecten.

Se estima que esta etapa requerirá aproximadamente **45 horas** de trabajo para completarse.

2.1.6.- Documentación del proyecto

A continuación, llevamos a cabo la etapa donde se redacta toda la documentación que acompaña al proyecto y los anexos.

Se estima que esta etapa requerirá aproximadamente **90 horas** de trabajo para completarse.

2.2.- Planificación temporal estimada

Al tratarse de un proyecto de fin de grado, está desarrollado por una sola persona, la cual asume distintos roles que se detallan en los siguientes apartados.

El total del tiempo empleado para completar el proyecto se estima en **1.193 horas**.

Etapa	**Duración (Horas)**
Estudio de viabilidad del sistema	100
Análisis del sistema de información	150
Diseño de la aplicación	150
Desarrollo de la aplicación	300
Protocolo de pruebas	45
Documentación del proyecto	448
Total	**1.193**

Tabla 2.1.- Planificación temporal estimada

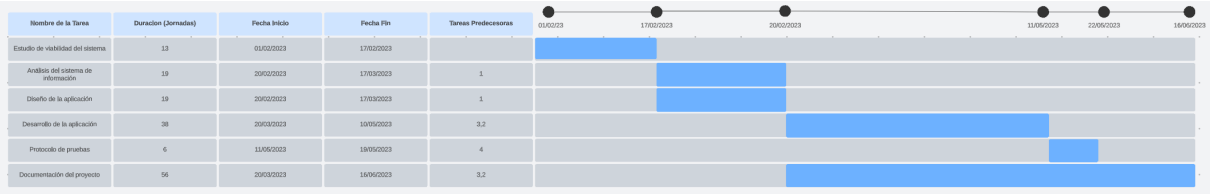

Figura 2.1.- Planificación temporal estimada

El proyecto se llevaría a cabo en **107 días**, considerando una jornada de 8 horas diarias que transcurre de Lunes a Viernes, y paralelizando el análisis del sistema de información y diseño.

2.3.- Reparto de roles

En este apartado se describe el equipo necesario, para realizar un proyecto de este tipo se debería contar al menos con los siguientes roles:

- Director de proyecto.
- Diseñador.
- Programador.
- Analista.
- Tester.

2.4.- Definición de roles

A continuación, se detallaran los roles junto con sus funciones y responsabilidades.

2.4.1.- Director proyecto

El máximo responsable del proyecto se conoce comúnmente como el **Project Manager** en inglés. Sus funciones principales incluyen la elaboración de la planificación del proyecto, establecer fechas límite para las distintas etapas y supervisar las tareas de los demás miembros del equipo.

2.4.2.- Analista

El analista tiene como responsabilidad elaborar el Estudio de Viabilidad del Sistema (EVS) y el Análisis del Sistema de Información (ASI).

2.4.3.- Diseñador

La labor del diseñador consiste en desarrollar interfaces de usuario que se adapten de forma óptima a las funciones específicas de la aplicación, considerando además el nivel de conocimiento que tendrán los usuarios finales.

2.4.4.- Programador

El programador asume la responsabilidad de elaborar el código que otorgará funcionalidad a la aplicación. Además, se encarga de obtener, analizar y estructurar los datos de manera adecuada para su correcta presentación. Es esencial que exista una coordinación fluida entre el programador y el diseñador para evitar incompatibilidades.

2.4.5.- Tester

La tarea del tester es asegurarse de que el trabajo realizado por el diseñador y el programador funcione correctamente. Esto se logra realizando pruebas exhaustivas sobre la aplicación y detectando cualquier posible fallo o error durante el proceso. El tester se encarga de identificar y señalar los problemas encontrados en sus pruebas, con el objetivo de mejorar la calidad y el rendimiento de la aplicación.

2.4.6.- Técnico

Es el responsable de la instalación de la aplicación y la elaboración de los manuales es el encargado de facilitar el proceso de instalación a los usuarios finales y de crear los manuales correspondientes.

2.5.- Asignación de roles

Una vez definidos los roles se va a asignar cada uno de los perfiles a las tareas descritas en el apartado 2.1.

- Estudio de viabilidad del sistema: Analista
- Análisis del sistema de información: Analista
- Diseño de la aplicación: Analista, diseñador
- Desarrollo de la aplicación: Diseñador, programador
- Protocolo de pruebas: Tester
- Documentación del proyecto: Técnico

El director del proyecto, además de en la planificación y el presupuesto va a supervisar el resto de las actividades del proyecto.

2.6.- Presupuesto estimado

En este aparatado se detallan los costes estimados. Empezando por el software utilizado para el desarrollo del proyecto y teniendo en cuenta que es completamente gratuito, lo que significa que el presupuesto se limita exclusivamente a los costes relacionados con el personal y el hardware.

Personal	**Horas totales**	**€/hora**	**Total (€)**
Director de proyecto	20	30,00	600,00
Analista	250	15,00	3.750,00
Diseñador	150	12,00	1.800,00
Programador	300	12,00	3.600,00
Tester	40	11,00	440,00
Técnico	80	11,00	880,00
		Total	**11.070,00**

Tabla 2.2.- Presupuesto estimado del personal

El equipo que se utiliza para llevar a cabo el proyecto es un ordenador portátil Dell XPS 15, con una vida útil aproximada de 5 años (60 meses) y un precio de mercado de 1.800€, lo que representa un coste mensual de 30,00€. Teniendo en cuenta que la duración del proyecto se estima en 5 meses, los cálculos se presentan en la tabla 2.3.

Nombre	**Cantidad**	**Duración**	**Precio**
Dell XPS 15	1	5 meses	30,00€/mes
		Total (€)	**150,00**

Tabla 2.3.- Presupuesto de hardware estimado

En la tabla 2.4 se presentan los costes humanos, de hardware y de software, que juntos conforman el costo de ejecución material. Estos tres componentes en conjunto representan un total de **11.220,00€.**

Recursos	**Coste (€)**
Humanos	11.070,00
Hardware	150,00
Software	0,00
Total	**11.220,00**

Tabla 2.4.- Presupuesto de recursos estimado

A los proyectos de ingeniería, por lo general, se les aplica un 13 % de coste adicional en concepto de gastos generales, y otro 6 % de beneficio industrial. Después de aplicar estos porcentajes, se añade un último 21 % de IVA, lo que resulta en el coste total, como se muestra en la tabla 2.5, de **16.157,68€.**

Concepto	**Coste (€)**
Ejecución material	11.220,00
Gastos generales	1.459,60
Beneficio industrial	673,20
Total antes de IVA	13.352,8
IVA	2.804,88
Total	**16.157,68**

Tabla 2.5.- Presupuesto total estimado

3. Especificación de requisitos

3.1.- Estudio de la situación actual

La primera aparición de lo que podríamos llamar web scraping sucede en 1993 cuando Matthew Gray desarrolla en Perl [4] el primer robot web conocido como **World Wide Web Wanderer**, la finalidad era recorrer sistemáticamente la web y recopilar sitios. Wanderer se convirtió en el primer agente web automatizado [3]. También en 1993 y posterior a Wanderer, aparece **JumpStation** de la mano de Jonathon Fletcher de Scarborough, se convierte en el primer motor de búsqueda basado en aranas (Crawlers). Este bot indexó millones de páginas web, convirtiendo Internet en una plataforma expansiva de código abierto antes los sitios webs ya que hasta entonces eran recopilados y editados a manos por los administradores [5]. En 2004 aparece **Beautiful Soup** [8], una biblioteca diseñada para Python y en 2006 aparece **visual de web scraping** [9], una plataforma que permite a los usuarios simplemente resaltar el contenido de una página web y estructurar esos datos en un Excel [4]. Tras esto, aparecen numerosos programas con la idea de hacer scraping la información de las webs con la finalidad de explotar posteriormente toda la información.

3.2.- Estudio de viabilidad del sistema (EVS)

3.2.1.- Ámbito y alcance del proyecto

A continuación se define el objetivo del proyecto, el cual es generar una aplicación sencilla, escalable y modular que permita a usuarios con bajos conocimientos de programación web descargar y personalizar conjuntos de datos de varias webs predefinidas, así como la posibilidad de ampliar e implementar nuevas.

La aplicación consta de varias webs seleccionables en las cuales el usuario podrá seleccionar una categoría de la cual se extraen los productos y exportan en el formato de

exportación elegido. Se puede elegir cuantos productos extraer y que deseamos exportar de los mismos.

3.2.2.- Lista de usuarios participantes

El proyecto cuenta con dos tipos de usuarios: el usuario administrador y el usuario del programa.

El usuario administrador, formado por personas como el director, diseñador, analista y programador, tendrá acceso a los archivos de la aplicación. Dependiendo de su rol, podrán realizar modificaciones en caso de problemas de funcionamiento, siempre bajo la supervisión del director.

Una vez desarrollada, solo habrá un tipo de participante: el usuario del programa. No será necesario realizar registros o identificaciones, por lo tanto, no se distinguirá entre diferentes tipos de usuarios de programa. Este tendrá acceso a todas las funcionalidades de la aplicación, pero no podrá realizar modificaciones en su funcionamiento.

Podría llegar a parecer un tercer rol, el usuario que desea ampliar con nuevas funcionalidades o implementaciones, este debe tener acceso al código fuente pero no estaría supervisado por el director del proyecto.

3.3.- Requisitos

A continuación, se definen los requisitos, los cuales se diferencian en dos tipos, requisitos funcionales y requisitos no funcionales.

3.3.1.- Requisitos funcionales

Podemos definir los requisitos funcionales como los encargados de definir los servicios proporcionados por el programa y como la aplicación debe comportarse cuando los usuarios u otros servicios interactúan con ella. A continuación, se definen los requisitos funcionales:

RF	Prioridad	Descripción
RF-1	Alta	El usuario debe poder seleccionar una página web en el combo box correspondiente.
RF-2	Media	Si se selecciona una web, se deben cargar las categorías correspondientes en el combo box de categorías.
RF-3	Media	Si se selecciona otra web, se deben recargar las categorías correspondientes en el combo box de categorías.
RF-4	Alta	El usuario debe poder ingresar el número de productos a buscar.
RF-5	Baja	El usuario debe poder seleccionar el formato de exportación de los productos.
RF-6	Media	El usuario debe poder marcar una casilla para mostrar el navegador durante el scraping.
RF-7	Alta	El usuario debe poder hacer clic en el botón *Iniciar* para iniciar el proceso de scraping.
RF-8	Baja	Al iniciar el scraping, se debe mostrar un mensaje al comenzar el scraping en el cuadro de texto de registro.
RF-9	Baja	Al finalizar el scraping, se debe mostrar un mensaje al finalizar el scraping en el cuadro de texto de registro.
RF-10	Baja	Durante el scraping, se debe mostrar un mensaje al encontrar un elemento en el cuadro de texto de registro.
RF-11	Baja	Durante el scraping, se debe mostrar un mensaje de los elementos restantes por añadir tras añadir uno nuevo en el cuadro de texto de registro.
RF-12	Baja	Durante el scraping, se debe mostrar un mensaje de los elementos añadidos tras añadir uno nuevo en el cuadro de texto de registro.
RF-13	Baja	Durante el scraping, se debe mostrar un mensaje cuando se pase de pagina en el cuadro de texto de registro.

RF-14	Baja	Durante el scraping, se debe mostrar un mensaje cuando se acepten las cookies.
RF-15	Alta	Durante la primera ejecución se deberán cargar las webs presentes en el fichero de configuración.
RF-16	Alta	Durante la primera ejecución se deberán cargar las categorías de la primera web añadida presentes en el fichero de configuración.
RF-17	Alta	Durante la primera ejecución se deberán cargar las categorías de la web actual presentes en el fichero de configuración.
RF-17	Alta	Durante la primera ejecución se deberán cargar los atributos de la web actual presentes en el fichero de configuración.
RF-18	Media	Durante el scraping solo se debe entrar en profundidad si se ha elegido al menos un atributo que lo requiera.
RF-19	Media	El usuario debe introducir uno o más en el campo de productos a buscar.
RF-20	Media	El usuario debe introducir al menos un atributo a exportar.
RF-21	Media	El fichero generado debe tener un nombre único.
RF-22	Media	El fichero de configuración debe estar en formato .json.
RF-23	Media	El fichero de configuración debe mantener la misma estructura que el resto de webs previamente implementadas.
RF-24	Baja	El botón *iniciar* debe des habilitarse hasta que finalice el scraping.

Tabla 3.1.- Requisitos funcionales junto con sus prioridades.

3.3.2.- Requisitos no funcionales

Podemos definir los requisitos no funcionales como aquellos que se refieren a las características y restricciones del sistema que no están directamente relacionadas con su funcionalidad. Estos requisitos se centran en aspectos como el rendimiento, la seguridad, la usabilidad y la mantenibilidad del sistema. A continuación, se definen los requisitos no funcionales:

RNF	Prioridad	Descripción
RNF-1	Alta	La interfaz de usuario debe ser intuitiva y fácil de usar.
RNF-2	Alta	El proceso de scraping debe ser eficiente y no debe haber retrasos significativos.
RNF-3	Media	La aplicación debe proteger los datos personales y la privacidad del usuario.
RNF-4	Alta	El código debe estar bien estructurado y documentado para facilitar el mantenimiento futuro.
RNF-5	Media	La aplicación debe ser compatible con diferentes sistemas operativos y entornos de ejecución.
RNF-6	Alta	La aplicación debe ser capaz de manejar grandes cantidades de datos y realizar scraping en diferentes sitios web.
RNF-7	Alta	La aplicación debe manejar errores y excepciones de manera adecuada, evitando bloqueos o cierres inesperados.
RNF-8	Media	El uso de recursos, como la memoria y el procesador, debe ser óptimo durante el proceso de scraping.
RNF-9	Media	La aplicación debe implementar hilos.
RNF-10	Media	El usuario debe tener conocimientos de HTML y CSS para nuevas implementaciones.
RNF-11	Media	El usuario debe tener conocimientos básicos de informática, debe ser capaz de instalar y ejecutar una aplicación.
RNF-12	Media	El usuario debe comprender Español para poder entender la interfaz.

RNF-13	Media	El equipo que ejecute la aplicación debe tener acceso a internet.
RNF-14	Media	El equipo debe tener espacio en disco suficiente para poder almacenar las exportaciones.
RNF-15	Media	El equipo debe tener las librerías necesarias para la ejecución del programa.
RNF-16	Media	El equipo debe tener Python instalado.
RNF-17	Media	El equipo debe tener Google Chrome instalado.
RNF-18	Media	El equipo debe tener un sistema operativo con GUI en x64 bits, se recomienda Windows 10 o posterior.
RNF-19	Media	La aplicación debe hacer las menores peticiones posibles a las webs.
RNF-20	Media	La aplicación debe espaciar las peticiones para evitar problemas de bloqueos.

Tabla 3.2.- Requisitos no funcionales junto con sus prioridades.

3.4.- Análisis de alternativas

En este apartado se van a explicar las decisiones tomadas a lo largo del proyecto.

3.4.1.- El mercado

Al comenzar el proyecto se buscó una aplicación similar que presentase gran parte de los requisitos que se buscaban. Existen numerosas aplicaciones dedicadas a extraer datos, tanto de pago como gratuitas y de código abierto. Podríamos categorizar las opciones según la forma de interactuar con ellas, se definen tres grupos.

APIs (Interfaces de programación de aplicaciones):
Las APIs son conjuntos de reglas y protocolos que permiten que diferentes aplicaciones o sistemas se comuniquen entre sí y compartan datos o funcionalidades de manera controlada. En el contexto del web scraping, las APIs pueden proporcionar mediante

peticiones datos de algunos sitios web específicos. Como alternativas para esta categoría tenemos:

- **Octoparse:** Es una plataforma basada en la nube que ofrece servicios de web scraping sin necesidad de escribir código. Proporciona una interfaz para configurar y ejecutar scraping en varios sitios web.

- **Import.io:** Es una API de extracción de datos web que permite extraer información de manera estructurada de sitios web complejos. Ofrece una amplia gama de funciones para personalizar las extracciones según tus necesidades.

Aplicaciones:

Las aplicaciones son programas o software completos que se ejecutan en una plataforma específica. En el contexto del web scraping, las aplicaciones proporcionan una interfaz visual o una interfaz de línea de comandos para configurar y ejecutar tareas de web scraping de manera más amigable para los usuarios que no tienen conocimientos de programación. Como alternativas para esta categoría tenemos:

- **ParseHub:** Es una aplicación de escritorio que permite extraer datos de sitios web sin necesidad de escribir código. Ofrece una interfaz gráfica intuitiva para seleccionar elementos y configurar la extracción.
- **WebHarvy**: Es una aplicación de Windows que facilita la extracción de datos de páginas web mediante técnicas de web scraping. Proporciona herramientas para extraer datos de tablas, listas, mapas y otros elementos.

Complementos para navegadores:

Los complementos o extensiones para navegadores son programas adicionales que se instalan en un navegador web para agregar funcionalidades específicas. En el contexto del web scraping, los complementos para navegadores pueden ofrecer herramientas visuales para seleccionar y extraer datos de manera más sencilla. Estos complementos a menudo permiten la interacción directa con elementos de la página y pueden proporcionar una interfaz intuitiva para configurar las reglas de extracción sin la necesidad de escribir código. Como alternativas para esta categoría tenemos:

- **Web Scraper (Chrome):** Es una extensión de Chrome que permite extraer datos de sitios web de manera visual. Proporciona una interfaz sencilla para seleccionar elementos y generar reglas de extracción.
- **Agenty (Chrome):** Similar a web scraper, es una extensión de Chrome que permite extraer datos de sitios web de manera visual. Proporciona una interfaz para seleccionar elementos en una página web y generar reglas de extracción sin necesidad de escribir código.

Aunque solo se citen un par de cada tipo existe gran variedad de alternativas, cada una de ellas tiene sus propias fortalezas y debilidades. Muchas de estas podrían cumplir gran parte de nuestros requisitos, pero no existe ninguna opción final que directamente sin ninguna preconfiguración nos permita extraer los datos de diferentes webs predefinidas, permita parametrizar que datos se desean y se puedan exportar con facilidad.
El web scraping puede llegar a ser un proceso complejo, ya que cada sitio web tiene su propia estructura y diseño particular, lo que requiere ciertas modificaciones y configuraciones específicas para extraer los datos deseados. La parametrización de los datos también puede variar según la estructura de cada sitio web y el tipo de información que se desea extraer. Debido a esto, es necesario llevar el proyecto a cabo.

3.4.2.- Lenguaje de programación

La primera decisión que hay que tomar, es el lenguaje de programación que se va a utilizar en todo el proyecto. Esta es la decisión más importante, ya que cada lenguaje tiene sus posibilidades, limitaciones y especialidades.

Entre las posibilidades se barajan dos, Python y C++. Ambos son lenguajes muy populares con amplia documentación y tienen como punto fuerte la programación orientada a objetos (POO), que es una parte importante de este proyecto.

Respecto a las diferencias entre uno y otro destacan:

- **C++** es un lenguaje compilado. Esto quiere decir que el código fuente se traduce a código de máquina y genera un programa ejecutable. Este archivo ejecutable

funciona sin necesidad de volver a ser compilado y no necesita la ayuda de otro programa.

- Por otro lado, **Python** es un lenguaje interpretado, que debe ser traducido y ejecutado por el intérprete cada vez que se ejecuta el programa y éste genera la salida del programa. El intérprete debe estar instalado en el equipo para que el programa pueda ser ejecutado.

En nuestro caso hemos seleccionado Python ya que ofrece ciertas ventajas como una sintaxis simple y legible, mayor facilidad de uso, una biblioteca estándar amplia, una comunidad activa y un ecosistema de bibliotecas amplio.

3.4.3.- Programa para interfaz gráfica (GUI)

Una vez elegido el lenguaje de programación, existen varias alternativas de frameworks de interfaz gráfica de usuario (GUI) para considerar. Las principales opciones son Tkinter, PyQt6 y wxPython.

- **PyQt6** es uno de los frameworks más ampliamente utilizados en el desarrollo con Python, lo que implica una gran base de usuarios y una amplia documentación disponible [14].
- **Tkinter** es otra opción popular que viene incluida con la biblioteca estándar de Python. Aunque tiene una apariencia más básica, es fácil de aprender y ofrece funcionalidad suficiente para aplicaciones GUI simples [15].
- **wxPython** es un binding de Python para wxWidgets, un framework de GUI multiplataforma escrito en C++. Proporciona una amplia gama de widgets personalizables y se enfoca en la natividad de la plataforma, brindando una experiencia de usuario coherente en diferentes sistemas operativos [16].

QtDesigner se basa en las llamadas *signals* para la comunicación entre objetos. Por ejemplo, cuando se presiona un botón, se emite una señal para ejecutar una función, o cuando se cambia el valor de una lista desplegable. Estas señales son muy útiles para crear una aplicación interactiva que responda al comportamiento del usuario, y se cree que

serán muy útiles para satisfacer los requisitos de diseño de la aplicación. Se escoge PyQt6 como framework de GUI para el proyecto debido a su potencia, flexibilidad, herramientas disponibles e integración con Python.

3.4.4.- Librería para exportar los datos

En cuanto a la elección de una alternativa para la exportación en diferentes formatos, podría considerarse la biblioteca interna de Python, que es una opción estándar para trabajar con archivos CSV. Sin embargo, se cree mejor opción utilizar Pandas para esta tarea por varias razones.

- En primer lugar, Pandas proporciona una funcionalidad más avanzada y flexible para la manipulación y análisis de datos en comparación con la biblioteca de Python. Pandas ofrece una estructura de datos llamada DataFrame, que facilita la manipulación de datos tabulares y ofrece métodos específicos para la exportación en diferentes formatos.
- Pandas tiene la capacidad de manejar datos de manera eficiente y optimizada, lo que es especialmente útil en escenarios con grandes volúmenes de datos. La biblioteca está diseñada para realizar operaciones eficientes en memoria y ofrece métodos de optimización para acelerar el procesamiento de datos.
- Pandas es su compatibilidad con una amplia gama de formatos de archivo, no solo CSV. Pandas permite exportar datos a formatos como Excel, JSON, SQL, HDF5, entre otros, lo que brinda flexibilidad al proyecto en términos de intercambio de datos con otros sistemas y herramientas.

3.4.5.- Librería para Scraping

Bibliotecas para lenguajes de programación:

Las bibliotecas son conjuntos de código predefinido que contienen funciones y métodos que los programadores pueden utilizar para facilitar la implementación de ciertas funcionalidades en sus programas. En el contexto del web scraping, las bibliotecas para lenguajes de programación proporcionan herramientas y funcionalidades específicas para

extraer información de las páginas web, como la extracción de datos de HTML, el análisis de XML, el manejo de solicitudes HTTP. Como alternativas par esta categoría tenemos:

- **BeautifulSoup (Python)**: Es una biblioteca de Python que facilita el análisis y la extracción de información de páginas web
- **Scrapy (Python):** Es un framework de web scraping en Python que permite la extracción estructurada de datos de sitios web.

En cuanto a la librería hemos considerado mejor opción Selenium [11], una alternativa hubiera sido Beautifulsoup pero debido a que esta es una librearía que solo parsea HTML y XML decidimos optar por selenium ya que emula completamente un navegador e interactúa como lo haría un humano.

Dentro de Selenium existen dos productos dentro del mismo suite. Selenium IDE y Selenium WebDriver son dos herramientas dentro del conjunto de herramientas Selenium que proporcionan soluciones para probar aplicaciones web y diseñar interfaces de usuario. Aunque tienen propósitos similares, existen diferencias significativas entre ambas que se adaptan a diferentes necesidades de prueba. Selenium WebDriver es una biblioteca de código que consiste en APIs esto permite a los testers y desarrolladores controlar los navegadores. Por otro lado, Selenium IDE es un complemento del navegador diseñado para grabar y reproducir pruebas. Actúa como un accesorio del entorno de pruebas y es adecuado para miembros del equipo de desarrollo [6]. Las principales diferencias son:
Selenium WebDriver:

- Biblioteca de API para controlar navegadores.
- Requiere habilidades de programación.
- Ofrece amplio soporte de lenguajes, compatibilidad con múltiples navegadores.
- Configuración más compleja con mayor participación de archivos y dependencias.
- Trata el navegador como un objeto y se puede utilizar en cualquier navegador.

Selenium IDE:

- Complemento del navegador para grabar y reproducir pruebas.
- No se requieren habilidades de programación.

- Proporciona accesibilidad mediante clics, ejecución de pruebas en paralelo, ejecución remota y grabación de comportamiento del usuario.
- Menos complicado que WebDriver, adecuado para testers sin experiencia avanzada en programación.
- Complemento disponible para los navegadores FireFox y Chrome.

Dentro de Selenium optamos por WebDriver ya que poder utilizar el navegador como objeto de Python facilita y hace más legible el código.

3.4.6.- Páginas web

Respecto a las webs elegidas existen multitud de páginas candidatas, implementar un comercio online era la mejor opción ya que disponemos de mucha información variada y previamente estructurada en categorías, tipos de producto y ordenada en páginas. Además presenta retos como entrar en varios niveles de profundidad, pasar páginas o aceptar las cookies. Esto hace que implementar páginas más simples, como por ejemplo una web de noticias, sea más sencillo ya que dispondrán de estas como guía y verán como se han abordado los retos.

El comercio online elegido en primer lugar es Amazon [12] , una web popular que resultará familiar y los datos extraídos de valor al usuario. Como segunda web, se opta por Aliexpress [13] , otro comercio online reconocido pero que por como está construido presenta nuevos retos a la hora de implementar.

3.4.7.- Alternativas seleccionadas

En este subapartado se presentarán las alternativas seleccionadas entre todas las opciones consideradas.

En cuanto al lenguaje de programación, la decisión final fue Python debido a varias razones. En primer lugar, Python ofrece compatibilidad con bibliotecas que no están disponibles en C++. Además, la sintaxis de Python es más sencilla y fácil de aprender para un usuario no avanzado.

Para el diseño de la interfaz gráfica se eligió PyQt, el framework más popular. PyQt cuenta con una documentación extensa y de calidad lo que facilita las nuevas implementaciones.

Para hacer scraping de los datos se elige Selenium Web Driver puesto que poder utilizar el navegador como objeto de Python nos aporta muchas facilidades y hace más comprensible el código.

Como webs a implementar se eligen Amazon y Aliexpress debido a que son dos comercios online populares y reconocidos.

Finalmente, en cuanto a la exportación de datos, se elige Pandas debido a su flexibilidad y variedad de formatos.

3.5.- Análisis del sistema de información (ASI)

En el Estudio de viabilidad 3.2 definimos el alcance del proyecto. El objetivo del Análisis del Sistema de Información (ASI) es crear un diseño basándonos en dicho análisis y cumpliendo los objetivos marcados en el Estudio de Viabilidad del Sistema 3.2

Se considera el siguiente subsistema, una única ventana, ventana principal, donde el usuario escogerá la web, la categoría, el número de productos que desea extraer, los atributos que desea exportar, así como el formato en el que lo quiere exportar. Dispondrá de un botón iniciar que comenzará a extraer la información. Durante la ejecución, el usuario verá unos mensajes de salida con el estado del proceso. Además podrá elegir si quiere o no ver el navegador que realiza la extracción de datos.

3.5.1.- Casos de uso

Los casos de uso [10] son una herramienta utilizada para definir la secuencia de acciones necesarias para lograr un resultado en un sistema. Se representan a través de diagramas que muestran la interacción entre el sistema y el usuario, detallando las relaciones y los flujos de información involucrados. Los casos de uso, proporcionan la estructura necesaria

para expresar tanto los requisitos funcionales como los no funcionales definidos en la sección 3.3.

Identificador	Campo identificador del caso de uso CU-XX.
Nombre	Nombre del caso de uso.
Actores	Partes involucradas.
Requisitos	Requisitos funcionales y no funcionales que se satisfacen.
Propósito	Descripción y objetivo.
Precondiciones	Situación en la que el sistema debe encontrarse antes de iniciar la ejecución del caso de uso de manera exitosa.
Flujo principal	Secuencia de pasos y acciones que los actores deben tomar.
Excepciones	Situaciones en las que el caso de uso no sigue el flujo principal y produce resultados inesperados.

Tabla 3.3.- Ejemplo de caso de uso.

3.5.1.1.- Casos de uso de la vista

Identificador	CU-1
Nombre	Seleccionar página web.
Actores	Usuario.
Requisitos	RF-1
Propósito	Permitir al usuario seleccionar una página web en el combo box correspondiente.
Precondiciones	El sistema muestra el combo box de páginas web.
Flujo principal	El usuario selecciona una página web en el combo box.
Excepciones	Ninguna.

Tabla 3.4.- Caso de uso: Seleccionar página web.

Identificador	CU-2
Nombre	Cargar Categorías .
Actores	Usuario.
Requisitos	RF-2
Propósito	Cargar las categorías correspondientes en el combo box de categorías al seleccionar una página web.
Precondiciones	Se ha seleccionado una página web en el combo box.
Flujo principal	El sistema detecta la selección de una página web en el combo box. El sistema carga las categorías correspondientes en el combo box de categorías.
Excepciones	Ninguna

Tabla 3.5.- Caso de uso: cargar categorías.

Identificador	CU-3
Nombre	Recargar categorías.
Actores	Usuario.
Requisitos	RF-3
Propósito	Recargar las categorías correspondientes en el combo box de categorías al seleccionar otra página web.
Precondiciones	Se ha seleccionado otra página web en el combo box.
Flujo principal	El sistema detecta la selección de otra página web en el combo box. El sistema recarga las categorías correspondientes en el combo box de categorías.
Excepciones	Ninguna.

Tabla 3.6.- Caso de uso: recargar categorías.

Identificador	CU-4
Nombre	Ingresar número de productos.
Actores	Usuario.
Requisitos	RF-4
Propósito	Permitir al usuario ingresar el número de productos a buscar.
Precondiciones	El sistema muestra el campo de entrada para el número de productos.
Flujo principal	El usuario ingresa el número de productos a buscar en el campo correspondiente.
Excepciones	Ninguna

Tabla 3.7.- Caso de uso: ingresar número de productos.

Identificador	CU-5
Nombre	Seleccionar formato de exportación .
Actores	Usuario.
Requisitos	RF-5
Propósito	Permitir al usuario seleccionar el formato de exportación de los productos.
Precondiciones	El sistema muestra las opciones de formato de exportación.
Flujo principal	El usuario selecciona el formato de exportación deseado.
Excepciones	Ninguna.

Tabla 3.8.- Caso de uso: seleccionar formato de exportación.

Identificador	CU-6
Nombre	Mostrar registro de mensajes.
Actores	Usuario.
Requisitos	RF-8
Propósito	Durante el proceso de scraping, mostrar un registro de mensajes en el cuadro de texto de registro.
Precondiciones	El sistema muestra el cuadro de texto de registro.
Flujo principal	Durante el proceso de scraping, el sistema registra mensajes y los muestra en el cuadro de texto de registro.
Excepciones	Ninguna.

Tabla 3.9.- Caso de uso: mostrar registro de mensajes.

Identificador	CU-7
Nombre	Iniciar proceso de scraping.
Actores	Usuario.
Requisitos	RF-7
Propósito	Permitir al usuario hacer clic en el botón *Iniciar* para iniciar el proceso de scraping.
Precondiciones	El sistema muestra el botón *Iniciar*.
Flujo principal	El usuario hace clic en el botón *Iniciar*. El sistema inicia el proceso de scraping.
Excepciones	Si no se especifica un numero entero positivo distinto de cero para el numero de productos y/o si no se selecciona ningún atributo.

Tabla 3.10.- Caso de uso: iniciar proceso de scraping.

Identificador	CU-8
Nombre	Marcar casilla de mostrar navegador.
Actores	Usuario.
Requisitos	RF-6
Propósito	Permitir al usuario marcar una casilla para mostrar el navegador durante el scraping.
Precondiciones	El sistema muestra la casilla de opción para mostrar el navegador.
Flujo principal	El usuario marca la casilla de opción para mostrar el navegador.
Excepciones	Ninguna.

Tabla 3.11.- Caso de uso: Marcar Casilla de Mostrar Navegador

3.6.- Descripción del sistema

3.6.1.- Esquema de ventanas

El esquema de la ventana consta de una interfaz única donde se encuentran todos los elementos descritos anteriormente.

En la parte superior de la ventana, se encuentra un combo box donde el usuario puede seleccionar la página web de su elección. Al seleccionar una página web, se activa el evento de carga de categorías correspondientes a dicha página web.

Justo debajo del combo box de la página web, se encuentra otro combo box donde se cargan las categorías correspondientes a la página web seleccionada. Si el usuario cambia la selección de la página web, las categorías se recargan automáticamente para reflejar las categorías asociadas a la nueva página web. A continuación, se encuentra un campo de texto donde el usuario puede ingresar el número de productos que desea buscar. Este campo de texto permite al usuario especificar la cantidad exacta de productos que se deben buscar durante el proceso de scraping.

Más abajo, se encuentra otro combo box que permite al usuario seleccionar el formato de exportación deseado para los productos obtenidos durante el scraping. El usuario puede elegir entre diferentes formatos, como CSV, HTML, entre otros.

También se incluye una casilla de verificación que permite al usuario marcarla para

mostrar el navegador durante el proceso de scraping. Si esta casilla está marcada, se mostrará el navegador durante el scraping para que el usuario pueda visualizar las acciones realizadas por el sistema.
En la parte inferior de la ventana, se encuentra un cuadro de texto donde se muestra el registro de mensajes relacionados con el proceso de scraping. Aquí se mostrarán mensajes informativos.

Finalmente, se encuentra un botón de ***Iniciar*** que el usuario puede hacer clic para iniciar el proceso de scraping. Una vez que se hace clic en este botón, se realiza el scraping de los productos según las configuraciones seleccionadas por el usuario.

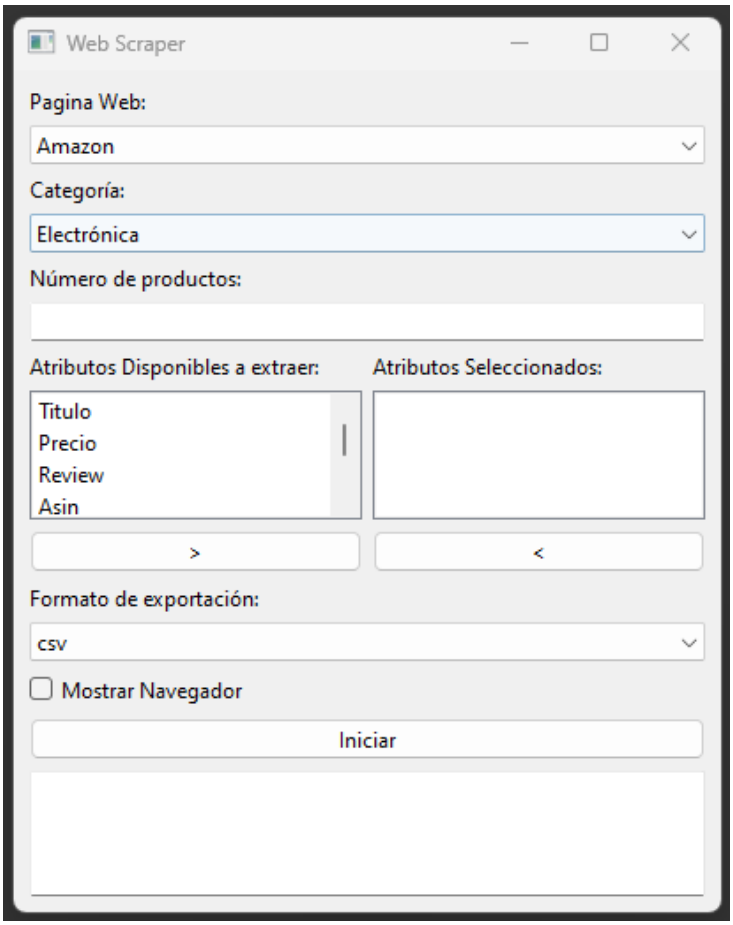

Figura 3.1.- Interfaz

4. Diseño del Sistema de Información (DSI)

Una vez concluido el análisis del sistema de información, se inicia la fase de Diseño del Sistema de Información (DSI). Durante esta etapa del proyecto, se establece la estructura del sistema, se definen las diversas interfaces con las que el usuario final interactuará y se determina el entorno tecnológico que se utilizará.

El Diseño del Sistema de Información se centra en la creación de una representación detallada y precisa del sistema, teniendo en cuenta los requisitos y las funcionalidades identificadas en la etapa de análisis. Se definen los componentes del sistema, sus relaciones y su interacción, para lograr una implementación eficiente y efectiva.

4.1.- Arquitectura del sistema

En la figura 4.1 se muestra el diagrama de arquitectura del sistema que refleja el flujo que siguen los datos hasta llegar a la aplicación. La aplicación, mediante Selenium obtiene los datos de la Web, existen dos webs que brindarán los datos Amazon o Aliexpress. Una vez obtenidos los datos, se deben almacenar y estructurar dependiendo de lo que haya seleccionado el usuario. Finalmente, se exportarán en el formato que eligió el usuario.

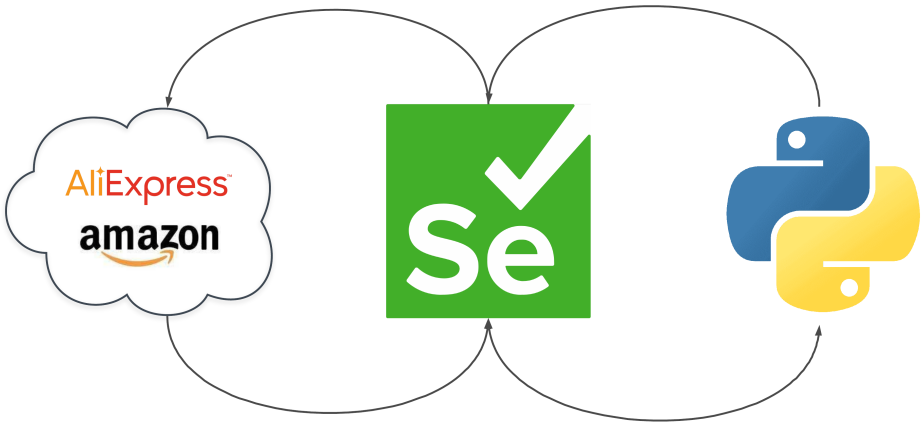

Figura 4.1.- Diagrama arquitectura del sistema

El objetivo es conseguir un diseño de aplicación sencilla para que el usuario en unos pocos clicks obtenga los datos extraídos.

Para los datos introducidos manualmente, se gestionan los posibles errores de entrada para asi evitar generar datos erróneos o incompletos.

4.2.- Patrones de diseño

Un patrón de diseño es una solución de diseño a un problema recurrente. El uso de patrones de diseño permite abordar problemas comunes de manera efectiva, promoviendo la reutilización de soluciones probadas y facilitando el desarrollo de software de calidad.

Un diseño se considera patrón cuando se ha comprobado que aplicándolo se consiguen los resultados esperados. Los elementos de un patrón de diseño son los siguientes:

- **Nombre**: Establece un vocabulario común que se va a utilizar a lo largo de todo el proceso de diseño.
- **Problema**: Describe la situación en la que se debe aplicar el patrón de diseño para que se obtengan los resultados esperados.
- **Solución**: Define la estructura que sigue el patrón, las relaciones entre sus componentes y una descripción.
- **Consecuencias**: Los resultados que se pretenden conseguir aplicando el patrón.

Existen diferentes tipos de patrones de diseño. En este proyecto se ha utilizado un patrón arquitectónico del tipo Modelo-Vista-Controlador (MVC).

4.2.1.- Modelo-Vista-Controlador

4.2.1.1.- Componentes del MVC

El patrón de diseño Modelo-Vista-Controlador (MVC) divide la organización de aplicaciones interactivas en tres componentes principales:

Modelo: Es el componente que contiene el núcleo de la funcionalidad y se encarga de la obtención y manipulación de datos. Representa la estructura y la lógica de negocio de la aplicación.

Vista: Es el componente encargado de mostrar la información al usuario y gestionar la interfaz gráfica. Se encarga de presentar los datos del modelo de una manera adecuada y proporciona la interacción visual con el usuario.

Controlador: Es el componente que gestiona la interacción del usuario con la aplicación. Se encarga de recibir las acciones y eventos del usuario, procesarlos y actuar en consecuencia. El controlador también se encarga de actualizar el modelo y la vista en función de las acciones realizadas por el usuario.

El patrón MVC es especialmente útil en la creación de aplicaciones basadas en interfaz gráfica de usuario, ya que proporciona una estructura clara y modularizada. Permite separar la lógica de negocio de la presentación, lo que facilita la reutilización de componentes y la mantenibilidad del código.

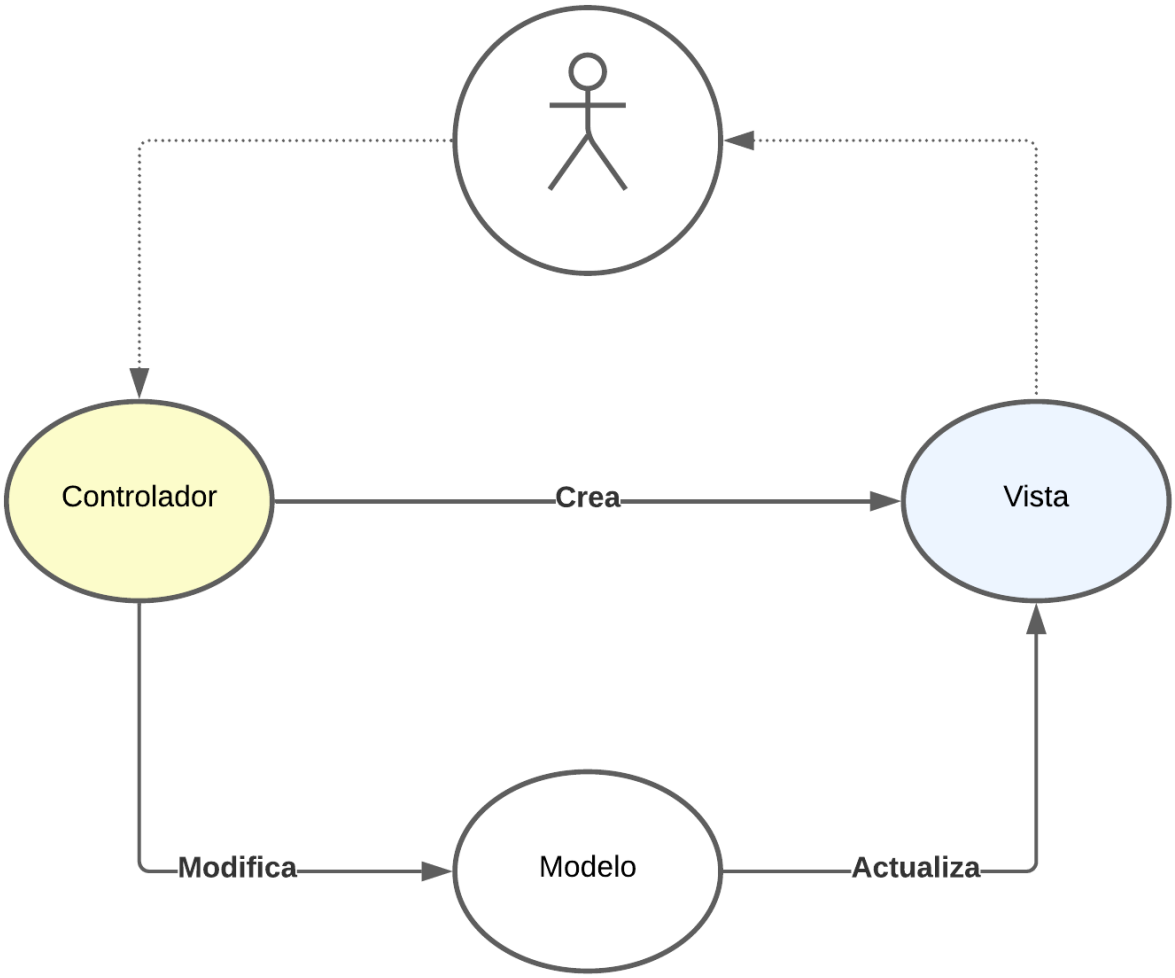

Figura 4.2.- Diagrama del patrón MVC

4.2.1.2.- Reglas del MVC

El patrón de diseño Modelo-Vista-Controlador (MVC) establece una clara separación de responsabilidades entre los componentes principales:

El **Modelo** es independiente de la Vista y el Controlador, y se enfoca en la funcionalidad de la aplicación. Su objetivo principal es la gestión de los datos y la lógica de negocio. El Modelo debe ser ajeno a la interacción directa con el usuario.

La **Vista** conoce el Modelo y se encarga de presentar los datos al usuario. La Vista se actualiza en función de las modificaciones que ocurran en el Modelo. Es importante destacar que la Vista no debe contener lógica de negocio ni realizar cálculos o acceso a datos, su única responsabilidad es la presentación.

El **Controlador** administra la interacción entre el usuario y la aplicación. Es responsable de recibir las acciones del usuario, modificar los parámetros del Modelo y crear las Vistas correspondientes. El Controlador no debe contener cálculos complejos ni acceso directo a datos, su función principal es coordinar la comunicación entre el usuario, el Modelo y la Vista.

Es esencial evitar la mezcla de contenido y presentación en el diseño MVC. La Vista y el Controlador deben mantenerse libres de lógica de negocio y cálculos, delegando estas tareas al Modelo. El Modelo se encarga de proporcionar los datos necesarios y realizar las operaciones correspondientes, mientras que la Vista se ocupa de presentarlos al usuario de manera adecuada.

4.3.- Entorno tecnológico de desarrollo

En este apartado se detallará el entorno tecnológico de desarrollo de la aplicación.

4.3.1.- Equipo hardware

- Procesador: Intel Core I9.
- Memoria: RAM 24GB RAM.
- Almacenamiento: SSD 1tb.
- Otros componentes relevantes: Windows 11 64 bits.

4.3.2.- Equipo software

- Microsoft Visual Studio Code 1.79.2.
- Python 3.10.4.
- Git 2.41.0.

4.3.3.- Tipos de archivos generados

Existen dos tipos de archivos generados en el proyecto:

- Archivos `.py`: Scripts de Python que pueden ser leídos y ejecutados por el intérprete.
- Archivos `.json`: Archivos de configuración utilizados para configurar la carga de la vista.
- Archivos .csv: Archivos resultado de la exportación de uno o varios productos.
- Archivos .html: Archivos resultado de la exportación de uno o varios productos.
- Archivos .feather: Archivos resultado de la exportación de uno o varios productos.
- Archivos .parquet: Archivos resultado de la exportación de uno o varios productos.
- Archivos .pickle: Archivos resultado de la exportación de uno o varios productos.

5. Pruebas

Además de las pruebas realizadas durante el desarrollo del programa para verificar su funcionamiento, una vez finalizado, se llevan a cabo pruebas unitarias y pruebas de integración. Estas pruebas tienen como objetivo asegurar que todos los componentes siguen cumpliendo su función correctamente y que las salidas obtenidas son las esperadas.

5.1.- Test unitarios

El objetivo de un test unitario es verificar que los fragmentos específicos de código funcionen correctamente. En un test unitario se aísla una parte del código del resto, sin tener en cuenta posibles conflictos con otros fragmentos. Para considerar un test válido, se debe obtener la salida esperada.

Test	Prueba carga de categorías.
Requisitos	RF-2
Acción	Seleccionar una página web.
Salida	Se cargan las categorías correspondientes en el combo box de categorías.
Resultado	El esperado .

Tabla 5.1.- Prueba carga de categorías.

Test	Prueba recarga de categorías .
Requisitos	RF-3
Acción	Seleccionar otra página web .
Salida	Se recargan las categorías correspondientes en el combo box de categorías .
Resultado	El esperado.

Tabla 5.2.- Prueba recarga de categorías.

Test	Prueba ingreso de número de productos.
Requisitos	RF-4
Acción	Ingresar un número válido en el campo de número de productos.
Salida	El número de productos se registra correctamente.
Resultado	El esperado.

Tabla 5.3.- Prueba ingreso de número de productos.

Test	Prueba selección de formato de exportación .
Requisitos	RF-5.
Acción	Seleccionar un formato de exportación en el combo box correspondiente.
Salida	El formato de exportación se registra correctamente.
Resultado	El esperado.

Tabla 5.4.- Prueba selección de formato de exportación.

Test	Prueba marcado de casilla de mostrar navegador.
Requisitos	RF-6
Acción	Marcar la casilla de mostrar navegador.
Salida	La casilla de mostrar navegador se marca correctamente.
Resultado	El esperado.

Tabla 5.5.- Prueba marcado de casilla de mostrar navegador.

Test	Prueba inicio del proceso de scraping.
Requisitos	RF-7
Acción	Hacer clic en el botón *Iniciar*.
Salida	El proceso de scraping se inicia correctamente.
Resultado	No genera mensajes de error y permite comenzar si no se introducen productos y/o seleccionan atributos, esto genera un cierre de la aplicación. Se deben contemplar y tratar dichos comportamientos.

Tabla 5.6.- Prueba inicio del proceso de scraping

Test	Prueba registro de mensajes durante el scraping.
Requisitos	RF-8
Acción	Realizar el proceso de scraping.
Salida	Se muestra un registro de mensajes en el cuadro de texto de registro.
Resultado	El esperado.

Tabla 5.7.- Prueba registro de mensajes durante el scraping

Test	Prueba ingreso de carácter de productos.
Requisitos	RF-9
Acción	Ingresar un carácter invalido en el campo de número de productos.
Salida	El número de productos no se registra correctamente generando un mensaje de error.
Resultado	No genera mensaje de error si se introduce un 0. Lo cual produce un cierre inesperado del código.

Tabla 5.8.- Prueba ingreso de carácter de productos.

Test	Prueba selección de atributos.
Requisitos	RF-10
Acción	Seleccionar uno o varios atributos.
Salida	Se agregan los productos a la vista.
Resultado	No genera mensaje de error si no se selecciona un atributo. Lo cual produce un cierre inesperado del código.

Tabla 5.9.- Prueba selección de atributos.

5.2.- Pruebas de integración

Test	Prueba de integración 1.
Requisitos	RF-1.1, RF-1.2
Acción desencadenante	Seleccionar la página web Amazon en el combo box correspondiente y verificar que se carguen las categorías correspondientes en el combo box de categorías.
Salida	Se verifica que se carguen correctamente las categorías correspondientes en el combo box de categorías.

Tabla 5.10.- Prueba de integración 1.

Test	Prueba de integración 2 .
Requisitos	RF-1.1, RF-1.2, RF-3.3
Acción desencadenante	Seleccionar la página web Aliexpress en el combo box correspondiente y verificar que se carguen las categorías correspondientes en el combo box de categorías. Seleccionar nuevamente la página web Amazon en el combo box correspondiente y verificar que se carguen las categorías correspondientes en el combo box de categorías.
Salida	Se verifica que se recarguen correctamente las categorías correspondientes en el combo box de categorías al seleccionar otra página web.

Tabla 5.11.- Prueba de integración 2

Test	Prueba de integración 3
Requisitos	RF-1.4, RF-5.2
Acción desencadenante	Ingresar el número de productos a buscar. Seleccionar el formato de exportación de los productos.
Salida	Se verifica que se exporte correctamente el número de productos seleccionado.

Tabla 5.12.- Prueba de integración 3.

6. Manual de usuario

6.1.- Introducción

En esta sección se explican los pasos a seguir para poder utilizar la aplicación, desde la instalación hasta como implementar nuevas webs.

6.2.- Requisitos del Sistema

Antes de utilizar la aplicación, asegúrese de que su sistema cumpla con los siguientes requisitos:

- Sistema operativo de 64 bits: Windows 10 o superior.
- Conexión a Internet.
- Google Chrome.
- Python 3.10 o superior instalado.

6.3.- Instalación

Si no lo tiene, deberá instalar Python en el equipo.El proyecto se ha realizado en Python 3.10, podrían existir incompatibilidades con otras versiones por lo que se recomienda utilizar un enviroment de Python:

1. Abrimos una terminal.
2. Navegamos al directorio raíz del proyecto.
3. Creamos el entorno virtual.

```
python -m venv <NOMBRE DEL ENTORNO VIRTUAL>
```

Código 6.1.- Crear el entorno

4. Ejecutamos la activación del entorno.

```
<NOMBRE DEL ENTORNO VIRTUAL>\Scripts\activate
```

Código 6.2.- Activar entorno

5. Configuramos el entorno en la primera linea de nuestro main.

```
#! <NOMBRE DEL ENTORNO VIRTUAL>\Scripts\python.exe
```

Código 6.3.- Configurar entorno en main.py

NOMBRE DEL ENTORNO VIRTUAL = El nombre que quieras poner a tu entorno.

A continuación se deben instalar las siguientes librerías:

Package	Version
attrs	23.1.0
certifi	2023.5.7
cffi	1.15.1
exceptiongroup	1.1.2
h11	0.14.0
idna	3.4
numpy	1.25.1
outcome	1.2.0
pandas	2.0.3
pip	23.2
pycparser	2.21
PyQt6	6.5.1
PyQt6-Qt6	6.5.1
PyQt6-sip	13.5.1
PySocks	1.7.1
python-dateutil	2.8.2
pytz	2023.3
selenium	4.10.0
setuptools	58.1.0
six	1.16.0
sniffio	1.3.0
sortedcontainers	2.4.0
trio	0.22.2
trio-websocket	0.10.3
tzdata	2023.3
urllib3	2.0.3
wsproto	1.2.0

Figura 6.1.- Librerías de Python

Para instalar las librerías ejecutad pip install *Nombre del paquete*, en este caso nos hacen falta solo los siguientes módulos:

```
pip install PyQt6
pip install selenium
pip install pandas
```

Código 6.4.- Instalar modulos

Una vez tenemos Python y las librerías debemos comprobar que la versión del Chrome Driver de Selenium y la versión de nuestro Chrome son la misma. Comprobar versión de Chrome:

1. Abrimos el Chrome.
2. En la esquina superior derecha le damos a los tres puntos y después a configuración.

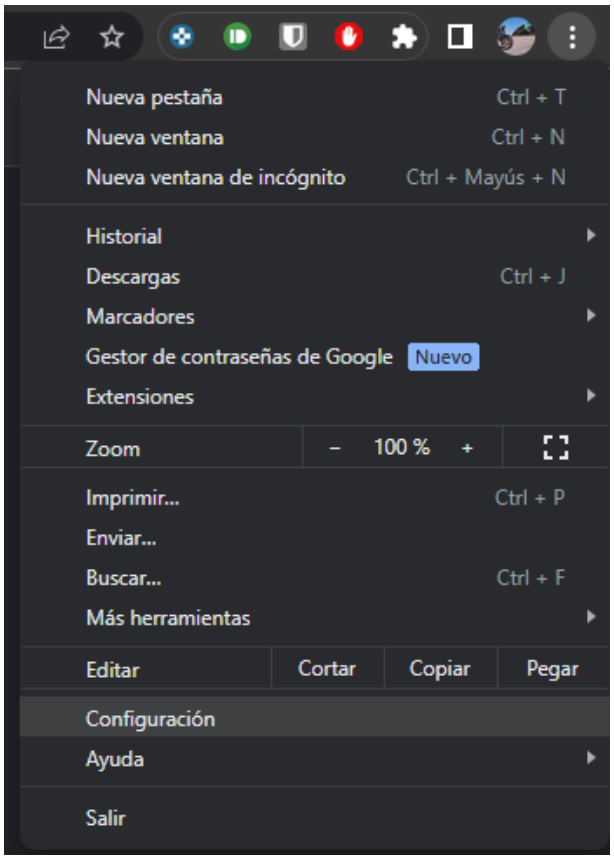

Figura 6.2.- Configuración Chrome

3. En el nuevo menú vamos a configuración de Chrome y vemos la versión. En este caso es la **114.0.5735.199**

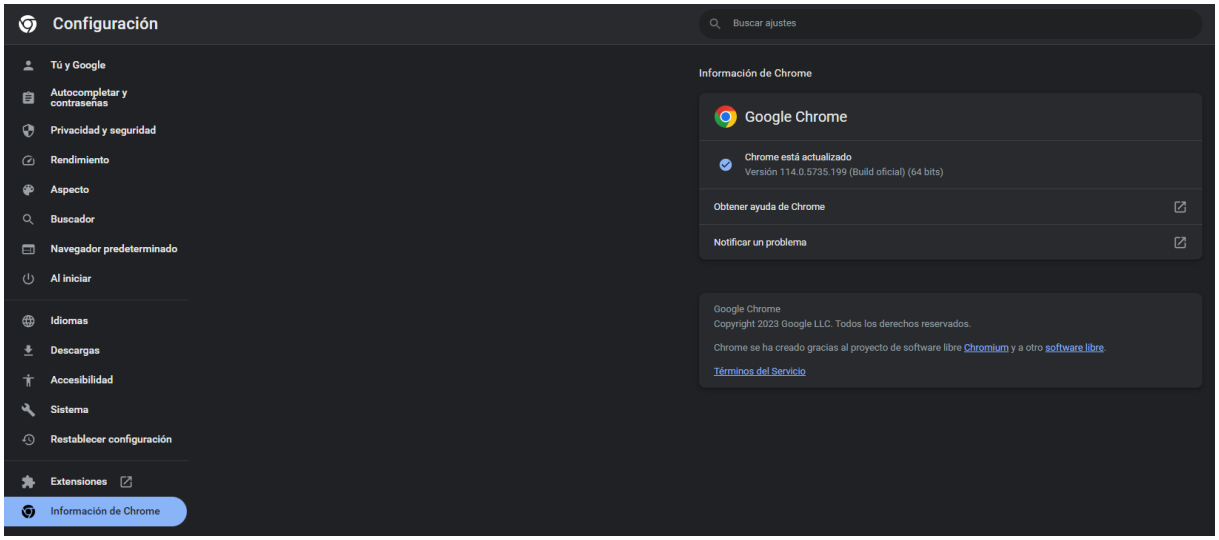

Figura 6.3.- Versión de Chrome

4. Descargamos la versión del Chrome Driver para nuestra versión en https://chromedriver.chromium.org/downloads
5. Almacenamos el Driver en la carpeta /Objects/Web/ bajo el nombre **chromedriver.exe**, si ya existe uno, lo sobrescribimos.

6.4.- Ejecución de la Aplicación

Podemos ejecutar desde consola o desde un editor de código o IDE

- **Ejecución desde consola:** Abriendo la consola y moviéndonos hasta la carpeta donde se encuentran los archivos que componen el programa, se escribe *python main.py,* esto ejecutará la ventana principal del programa.

```
(TFG) C:\Users\adrip\Desktop\VSC\Seleniumtfg2023>python main.py
```

Figura 6.4.- Ejecución consola main.py

- **Desde un editor de código o IDE:** En este caso se ha utilizado Visual Studio Code, a partir de ahora VSC, para iniciar la aplicación tanto en VSC como en

cualquier otro IDE, hay que hacer click en el botón *Run Python File* del menú superior teniendo importado el proyecto y abierto el main.py, esto ejecutará la ventana principal de la aplicación.

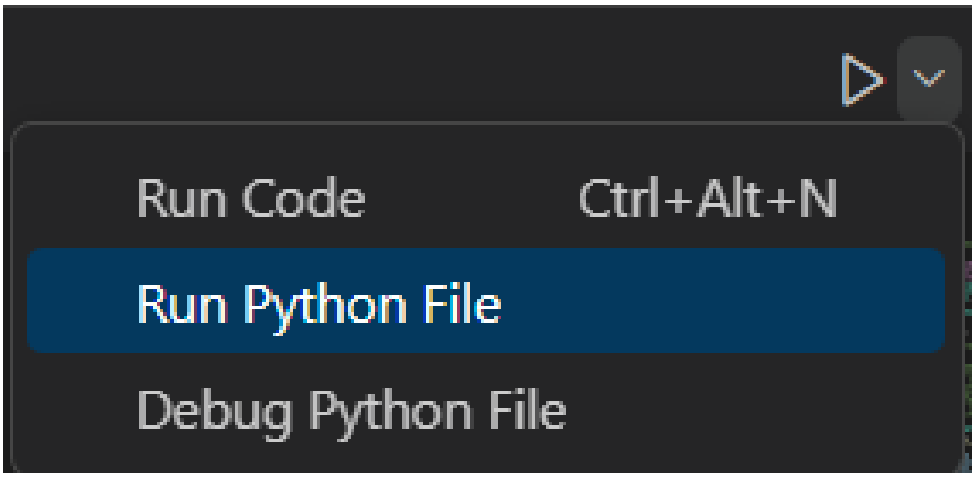

Figura 6.5.- Ejecutar main.py en VSC

6.5.- Uso de la aplicación

Una vez ejecutada en la ventana principal de la aplicación, encontrará un combo box donde puede seleccionar la página web de la cual desea realizar el scraping de productos. Las opciones disponibles incluyen Amazon y Aliexpress. Seleccione la página web deseada.

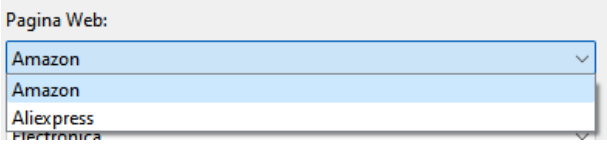

Figura 6.6.- Webs

Una vez seleccionada la página web, la aplicación cargará automáticamente las categorías disponibles en un combo box. Seleccione la categoría de productos donde desea hacer scraping.

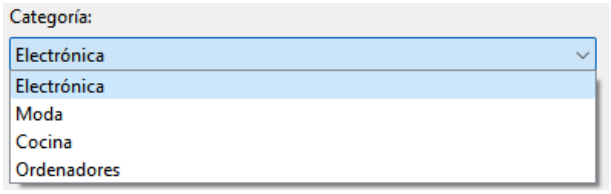

Figura 6.7.- Categorías

Ingrese el número de productos que desea buscar.

Figura 6.8.- Número de productos

Seleccione que atributos desea extraer. Ayúdese de los botones inferiores para mover los atributos entre listas.

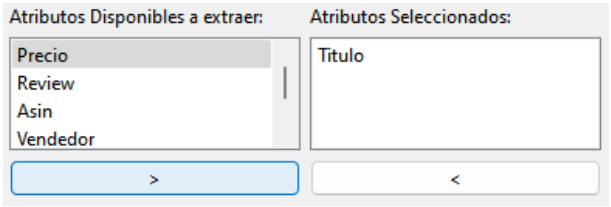

Figura 6.9.- Atributos

Seleccione el formato de exportación de los datos de productos.

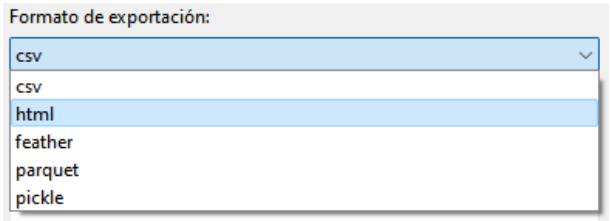

Figura 6.10.- Formato de exportación

Marque la casilla *Mostrar Navegador* si desea visualizar el proceso de scraping en tiempo real mediante la apertura de un navegador web

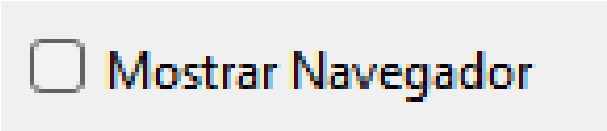

Figura 6.11.- Mostrar el navegador

Finalmente haga clic en el botón *Iniciar* para iniciar el proceso de scraping de productos.

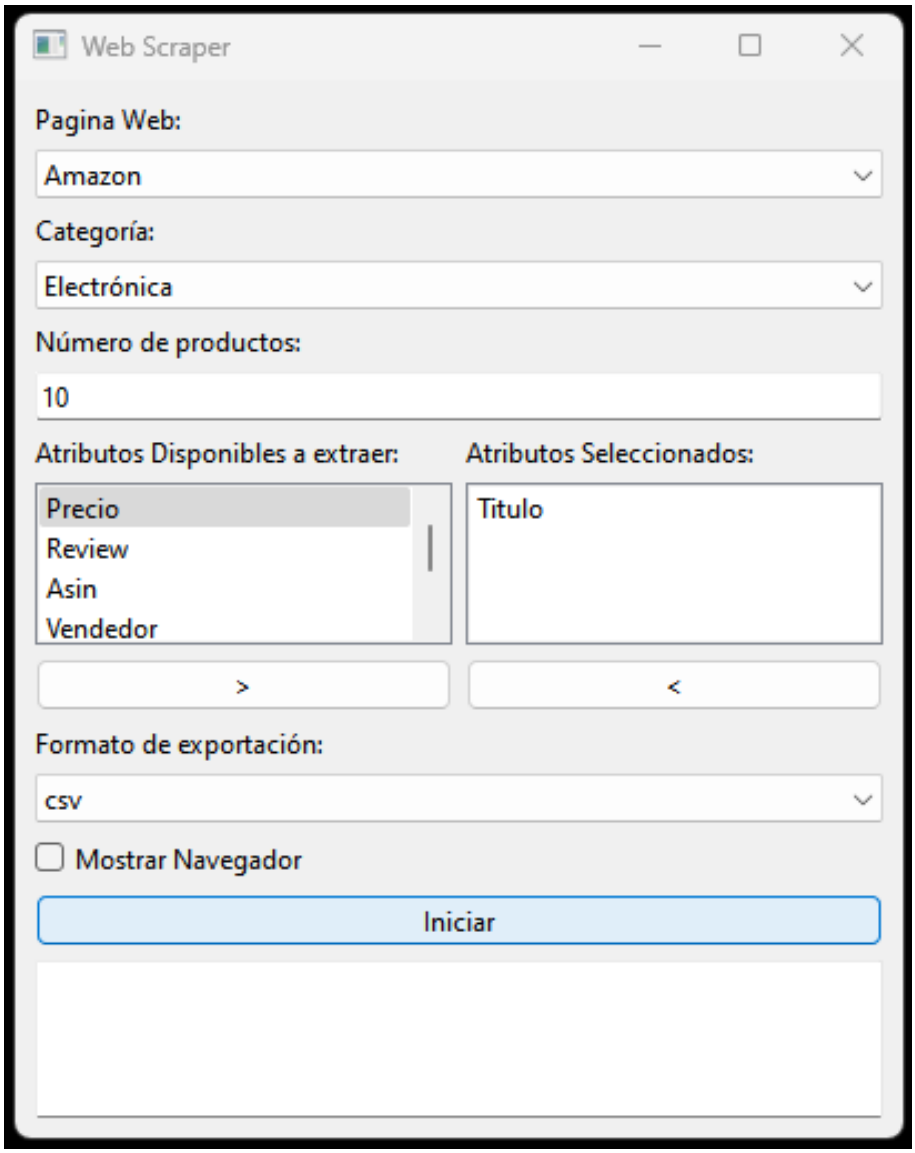

Figura 6.12.- Iniciar

Durante el proceso, se mostrará un registro de mensajes en la ventana de la aplicación.

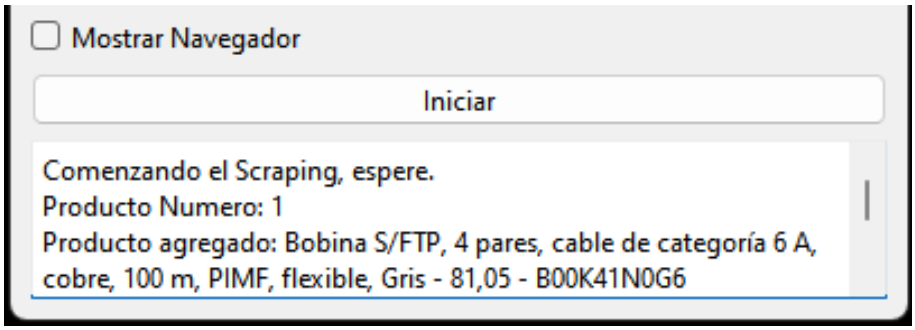

Figura 6.13.- Log

Una vez finalizado el proceso de scraping, veremos el mensaje de búsqueda de datos finalizada.

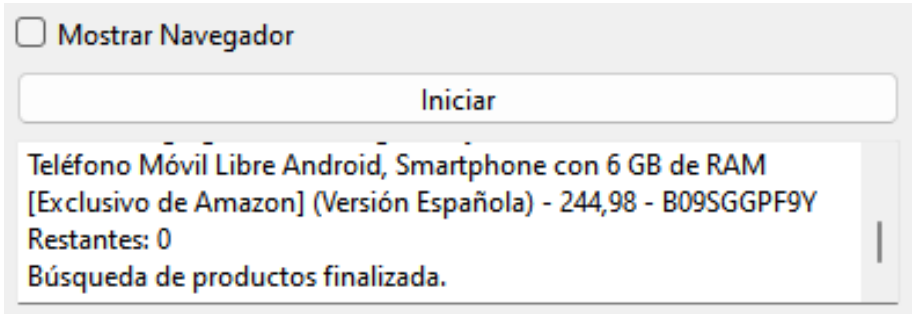

Figura 6.14.- Fin

Puede ver los resultados en el formato seleccionado. Navegue hasta la web y ruta de la categoría seleccionada.

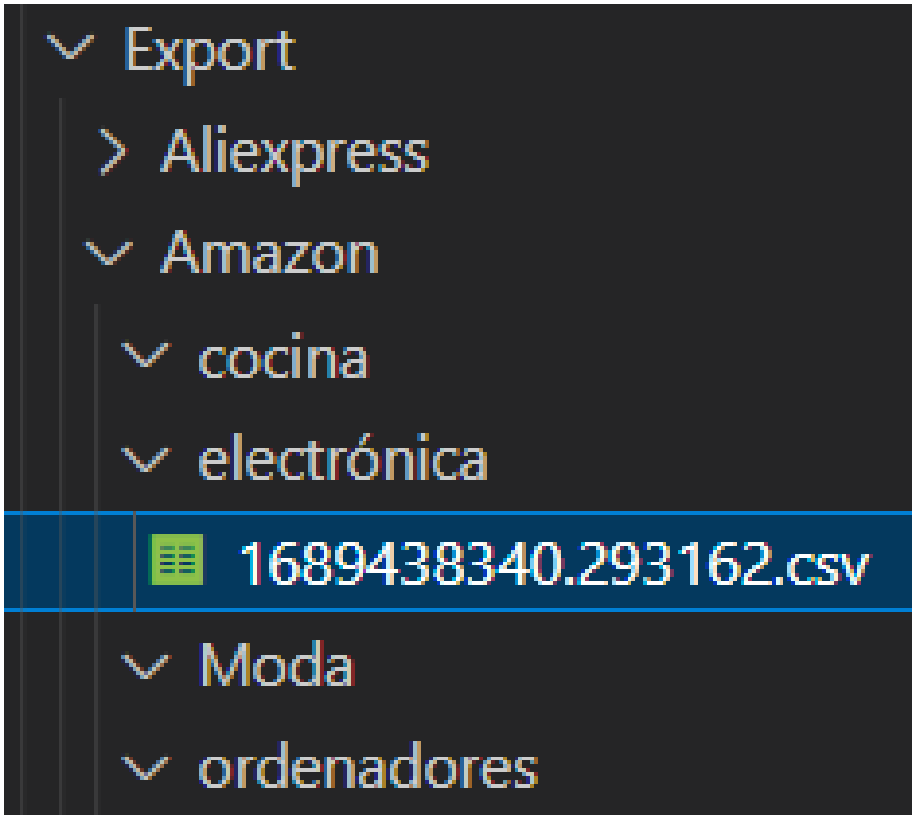

Figura 6.15.- Path

6.6.- Solución de problemas

Si experimenta algún problema durante el uso de la aplicación, siga los siguientes pasos para intentar solucionarlo:

1. Verifique que su conexión a Internet esté estable y funcione correctamente.
2. Asegúrese de tener instalada la misma versión de Python en su sistema.
3. Asegúrese de tener la versión de Chrome driver correcta para su versión de Chrome

6.7.- Implementar una nueva web

A continuación se va a explicar los pasos que un usuario avanzado debería de seguir para nuevas implementaciones y ampliaciones. Aquí se recoge como implementar una nueva página web así como algunos consejos y posibles problemas.A la hora de añadir una nueva debemos afrontar diferentes retos, los cuales, dependiendo de la web, serán diferentes.

Lo primero es entender como está estructurado el código. El código está estructurado en dos clases principales Web.py y producto.py estas clases presentan las funciones y atributos.El enfoque de tener una clase padre como *Web* y una clase *Producto* proporciona una estructura base sólida para implementar nuevas clases. La clase *Web* se encarga de la configuración del navegador y la extracción de datos básicos, mientras que la clase *Producto* define los atributos y métodos relacionados con un producto específico.

Estas clases servirán de estructura base a la hora de implementar nuevas clases además de que facilitan la organización y la reutilización del código, ya que puedes aprovechar la implementación existente en las clases base y solo enfocarte en las diferencias y características únicas de cada página web o producto. Los pasos a seguir son los siguientes:

6.7.1.- Agregar las clases hijas.

6.7.1.1.- Crear la clase hija de web.

Lo primero es crear una clase hija que herede de Web.

```
1 class MiWeb(Web):
2     def __init__(self, show_browser):
3         super().__init__(show_browser)
```

Código 6.5.- Ejemplo clase web

El siguiente paso es implementar los métodos específicos para nuestra web.

- **obtenerurl(categoría):** Este método se encarga de obtener la URL específica de tu página web para la categoría especificada en la vista. Recibe la categoría y debe devolver la URL de la misma.

- **extraeratributosproducto(elemento, atributosP):** Este método se encarga de extraer los atributos (Campos) de un producto dado. Recibe de forma obligatoria el elemento del cual extraer la información y un booleano que le indica si debe entrar en un segundo nivel a extraer algún elemento.
- **buscarproductos(categoría, numproductos, atributosenprofundidad, atributosaextraer, logcallback=None):** Este método se encarga de extraer los productos en la página actual, controlar si ya hemos extraído todos los indicados en la vista, si no es así iterar sobre las páginas hasta entonces y de llamar producto a producto a las funciones que extraen los atributos y los almacenan. Es el método que se encarga de orquestar el resto. En este método también se escribe gran parte del log que se manda a la vista. Este recibe, la categoría,el número de productos a extraer, si se extraen en profundidad o no, los atributos disponibles para extraer y el callback de la vista para el log. Esta función, devuelve la colección de productos con los campos extraídos.

6.7.1.2.- Crear la clase hija de producto.

A continuación, debemos implementar la clase hija herede de producto.

```
1 class MiProducto(Producto):
2     def __init__(self, Titulo, Precio, Atributo1, AtributoOpcional1=None,
      AtributoOpcional2=None):
3         super().__init__(Titulo, Precio)
4         self.Atributo1 = Atributo1
5         self. AtributoOpcional1= AtributoOpcional1
6         self. AtributoOpcional2= AtributoOpcional2
```

Código 6.6.- Ejemplo clase producto

El siguiente paso es añadir, si se requiere, atributos adicionales. Existen dos tipos de atributos dependiendo de la profundidad que se necesite para extraerlos. Los atributos simples son aquellos que adicionalmente están presentes en todos los productos para nuestra web y se encuentran en el primer nivel de profundidad, es decir en el propio HTML de la página sin necesidad de movernos entre diferentes URLS. Cuando necesitamos algún

atributo que esté dentro del propio producto o bien no esté presente en todos los productos disponemos de los atributos opcionales.

6.7.2.- Configurar la vista.

6.7.2.1.- Configurar el main

En el main, dentro de la función run() del Worker, debemos configurar el nombre de la web y la clase web a la que llamará.

```
def run(self):
        """
        Metodo que ejecuta el trabajo de web scraping.

        """
        web = None

        if self.web == "MiWeb":
            web = MiWeb(self.show_browser)
```

Código 6.7.- Ejemplo Main

Así como importar la clase web correspondiente.

```
from Objects.Web.Miweb_web import MiWeb
```

Código 6.8.- Crear el entorno

Esto podría hacerse importándolo desde el fichero de configuración pero es interesante definir con que clase de web se relaciona cada una de ellas ya que puede darse el caso de tener más de una clase web por página, dependiendo por ejemplo de la categoría. Esto también ayuda a simplificar la estructura del mismo.

6.7.2.2.- Agregar al fichero de configuración.

Por último debemos agregar al fichero de configuración las características de nuestra implementación. El fichero deberá seguir esta estructura.

```
{
 "MiWeb":
 {
 "categorias": ["Categoria1","Categoria2"],
 "atributos":{
   "Titulo":0,
   "Precio":0,
   "Atributo1":0,
   "AtributoOpcional1":1,
   "AtributoOpcional2":1,
       }
     },
 "OtraWeb":
 {
 ...Categorias y atributos de la otra web...
 }
 }
```

Código 6.9.- Fichero configuración

En el fichero definimos el nombre de la web, las categorías y los atributos. Los atributos simples deben llevar un cero y los atributos que deban entrar en profundidad un uno.

6.7.3.- Recomendaciones

Teniendo una idea de lo que se necesita a nivel de código, ahora vamos a ver que retos debemos afrontar con nuestra implementación y algunas recomendaciones de como afrontarlos. Lo primero de todo, hay que conocer y familiarizarse con la estructura HTML de la página web. Para ello lo mejor es acceder a la categoría de la web que queramos desde nuestro navegador de preferencia e inspeccionar elemento.

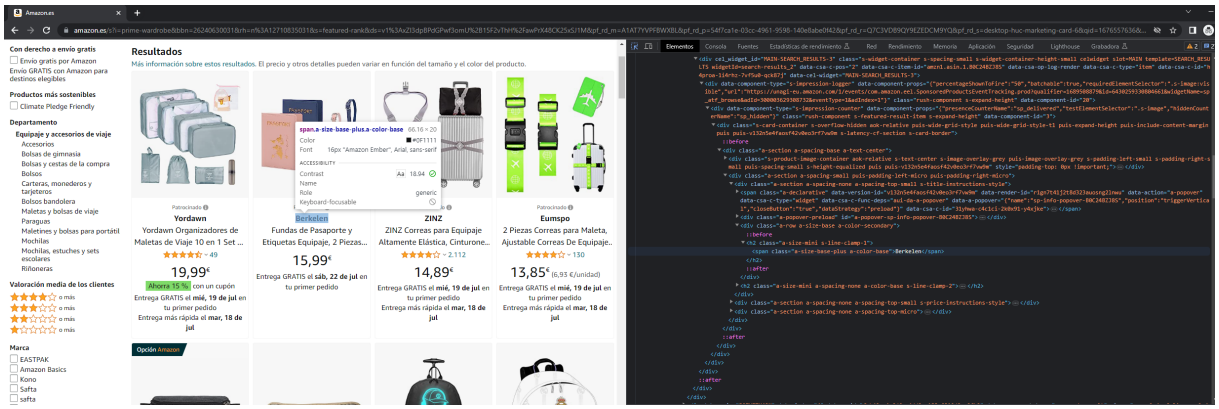

Figura 6.16.- Inspeccionar elemento

Dependiendo de la Web tendremos una estructura diferente, debemos fijarnos en las clases y los id de los elementos e intentar extraer el patrón que utilice la web Vamos a ver algunas recomendaciones para los principales retos que presenta la implementación.

6.7.3.1.- Listar los productos.

Deberemos ver que campo se repite en todos los productos y solo en los productos para poder iterar sobre ellos y extraer producto a producto. En ocasiones estarán anidados y nos servirán para extraer del mismo el resto de atributos, en otras ocasiones solo nos servirá para iterar sobre él y mantener el índice. Otro posible problema es que los elementos no aparezcan completos en la primera carga si no que se carguen a medida que recorremos la página. Para esto es buena idea subir y bajar en la página.

Podría hacerse de la siguiente manera:

```
sleep(1)
                    self.driver.find_element_by_tag_name('body').send_keys(Keys.END)
                    sleep(1)
                    self.driver.find_element_by_tag_name('body').send_keys(Keys.HOME)
                    sleep(1)
                    self.driver.find_element_by_tag_name('body').send_keys(Keys.END)
                    sleep(1)
                    wait.until(EC.presence_of_all_elements_located((By.XPATH,'//a[
    contains(@class,"earch-card-item")]')))
```

Código 6.10.- Subir y bajar en la página

Con *END* vamos al final de la página y con *HOME* al origen.

6.7.3.2.- Extraer los atributos.

Una vez listados los productos podremos ver que queremos extraer de cada uno. Normalmente siempre existe algo en elemento HTML que es común a todos. No es necesario seleccionar la *class* o el *id* completo, nos podemos aprovechar del comando *contains*.

```
wait = WebDriverWait(self.driver, 10)
            elementosList = wait.until(EC.presence_of_all_elements_located((By.XPATH,
    '//div[contains(@class, "s-result-item s-asin")]')))
```

Código 6.11.- Utilizar contains

Aquí podemos ver como seleccionamos todos los elementos de tipo *div* que contengan un *class* que dentro del texto tenga *s-result-item s-asin*

6.7.3.3.- Entrar en profundidad.

En algunas ocasiones es necesario entrar al producto para poder extraer cierta información interesante. Para ello hay que entrar y después volver atrás, esto es sencillo

ya que con Selenium podemos utilizar el *.get()* para acceder y el *.back()* para volver sin tener que almacenar la URL previa.

```
self.driver.get(url)
            try:
                atributos_extraidos["Review"] = self.driver.find_element(by=By.XPATH,
    value='.//span[@id="acrCustomerReviewText"]').text
            except Exception as e:
                atributos_extraidos["Review"] = "No encontrado"

            try:
                atributos_extraidos["Vendedor"] = self.driver.find_element(By.XPATH, '
    .//span[@class="a-size-small tabular-buybox-text-message"]').text
            except Exception as e:
                atributos_extraidos["Vendedor"] = "No encontrado"

            try:
                atributos_extraidos["Estrellas"] = self.driver.find_element(By.XPATH,
    './/span[@class="a-size-base a-color-base"]').text
            except Exception as e:
                atributos_extraidos["Estrellas"] = "No encontrado"

            self.driver.back()
```

Código 6.12.- Entrar en profundidad

6.7.3.4.- Evitar ser bloqueado.

Algunas webs presentan mecanismos contra autómatas o robots. Estos mecanismos detectan comportamientos repetitivos o demasiado rápidos. La solución para evadirlas es interactuar de una forma lo mas humana posible. Para ello espaciamos las peticiones sacrificando un poco de eficiencia. Esto lo podemos hacer con comandos como sleep(*Segundos*).

```
1 sleep(2) # Espaciamos las peticiones
2                 siguiente_pagina_url = self.driver.find_element_by_xpath('//a[contains
    (text(),"Siguiente")]').get_attribute('href')
3                 self.driver.get(siguiente_pagina_url)
```

Código 6.13.- Espaciar peticiones

6.7.3.5.- Aceptar las cookies.

Es una buena práctica aceptar las cookies, esto ayudará a tener una interacción más humana con la web. Normalmente, según entramos a la web, pasados un par de segundos nos aparece el botón para aceptarlas. Inspeccionamos y vemos que tipo de elemento HTML es.

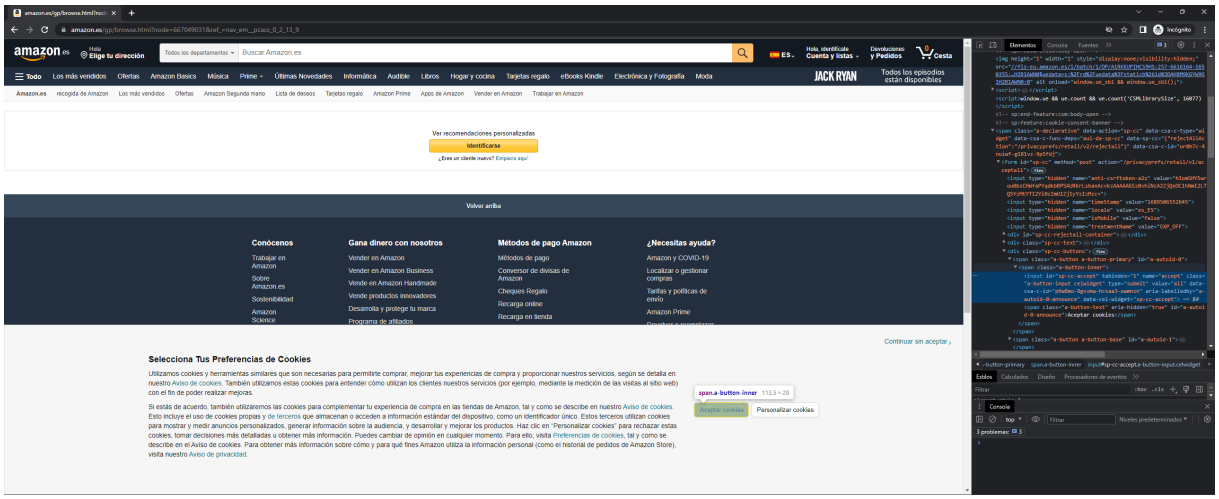

Figura 6.17.- Botón de cookies

En el código debemos seleccionar dicho elemento y mandar un evento del tipo .click.

```
# Aceptar las cookies
        sleep(3)
        try:
            accept_button = self.driver.find_element(By.ID, 'sp-cc-accept')
            accept_button.click()
        except Exception as e:
                print("No se encontro el boton de aceptar")
```

Código 6.14.- Crear el entorno

6.7.3.6.- Pasar de página.

Similar a aceptar las cookies, buscamos el elemento y hacemos click sobre el.

```
try:
                if log_callback is not None:
                    try:
                        sleep(1)
                        log_callback(f"---Pasando de pagina---")
                    except Exception as e:
                        print("Error al agregar el mensaje al registro:", str(e))
                sleep(2) # Espaciamos las peticiones
                siguiente_pagina_url = self.driver.find_element_by_xpath('//a[contains
    (text(),"Siguiente")]').get_attribute('href')
                self.driver.get(siguiente_pagina_url)
            except:
                if log_callback is not None:
                    try:
                        sleep(1)
                        log_callback(f"Fin, no hay mas productos a extraer")
                        sleep(1)
                        log_callback(f"Producto totales agregados: {Numero_Productos}"
    )
                    except Exception as e:
                        print("Error al agregar el mensaje al registro:", str(e))
                return productos
```

Código 6.15.- Pasar de página

6.7.3.7.- Controlar excepciones.

Debido a que las páginas pueden actualizarse de un día para otro es importante que cada llamada al driver este dentro de un *try* para evitar cierres abruptos del programa. En el *catch* si creemos que es de interés para el usuario podemos mostrarle un mensaje.

7. Conclusiones

De todos los proyectos que existían como opción este me llamó la atención ya que por mi cuenta había intentado realizar algunas extracciones de datos mediante la captura de peticiones GET. Veía en él una forma de indagar en el webscraping y la extracción de elementos, además de considerar realmente útil el concepto que perseguía la aplicación.

Durante el proyecto, me he encontrado con aún más webs que tenían información que quería pero que requerían de muchos clicks para consultarla y explotarla. Por poner un ejemplo, a nivel laboral, utilizo diariamente la web https://www.virustotal.com/gui/home/upload, una web que, pasado el hash de un archivo, te dice el nivel de confianza pasándolo por distintos motores de antivirus. Esta web es muy útil a la hora de analizar incidentes de seguridad y corroborar falsos positivos en alertas de antivirus, pero presenta un problema, si quieres automatizar la correlación de hash tiene una API limitada diariamente a nivel de peticiones y una modalidad de pago con ciertos niveles y limites. Si que es cierto, presenta muchas funcionalidades añadidas y proporciona mucha más información que el simple hecho de darte una puntuación de confianza sobre un fichero pero son funcionalidades que, en mi caso, no tendrían utilidad.

A donde quiero llegar con todo esto es que una implementación sencilla de este proyecto haría que pudiese extraer esa información en un formato como csv y tratarla de forma sencilla. Esto daría solución a una tarea muy mecánica y repetitiva.

Asimismo, que el proyecto se realizase en Python me ha gustado mucho, es un lenguaje que no se ve tan en profundidad en el grado y que resulta sencillo y natural. Cuando empiezas a trabajar con él, te das cuenta porque es un lenguaje tan popular hoy en día. Este proyecto me anima a conocer más acerca del lenguaje, librerías y disfrutar sobre todo de su gran comunidad, no hay problema que me haya encontrado que no hubiera experimentado otra persona antes y me ayudase a dibujar la posible solución a mi caso particular.

El trabajo ha supuesto varios retos, ente ellos el que más tiempo y esfuerzo me han llevado ha sido intentar no ser bloqueado por las webs. Estas incorporan técnicas para protegerse cuando detectan comportamientos extraños, por ejemplo en el caso de este proyecto, relanzar muchas veces la aplicación o extraer información de la forma más rápida posible hacía que se hiciese demasiadas peticiones a la misma URL. Esto generaba diferentes comportamientos, como redirecciones, verificar que no era un robot o directamente bloqueos temporales. Finalmente, se consiguió emular una interacción más humana para evitar tener problemas.

Para finalizar, considero alcanzado el objetivo del proyecto, puesto que la aplicación cumple los requisitos propuestos, además de permitir, debido a su modularidad, las implementaciones futuras, así como la sencillez de uso y extracción.

8. Ampliaciones

Según transcurre el proyecto se han visto posibles mejoras que no se han podido realizar por diferentes motivos. A continuación, se mencionan algunas para ayudar en las futuras ampliaciones.

- **Extracción de datos por consola:** De cara a integrar la aplicación con terceros o poder llamarla sin depender de la interfaz gráfica, sería una opción interesante poder llamarla desde una terminal pasándole todos los atributos como parámetros.
- **Aplicación web**: Publicar en un servicio web la aplicación la haría más utilizable, incluso abre la posibilidad de poder utilizarla desde dispositivos móviles y diferentes sistemas operativos.
- **Implementación de atributos específicos por categoría:** Por como está construido y tratado el fichero de configuración existen los mismos atributos para todas las categorías pero podría darse el caso de haber diferentes atributos dependiendo de la misma.
- **Optimización de la extracción:** Debido a intentar no ser bloqueado por las webs hay numerosos sleeps en el código, los cuales podrían ajustarse o eliminarse. Esto haría que el código se ejecutara de forma más rápida. Un modo que no muestre log al usuario también seria mucho más rápido, ya que mostrarlos los ralentiza la ejecución.

Bibliografía

[1] Francis Bacon, (1597), *Meditationes Sacrae* 30 de Mayo de 2023

[2] Qué es el Web Scraping y Por Qué es Importante https://www.octoparse.es/blog/por-que-web-scraping-es-importante 30 de Mayo de 2023.

[3] Servicios De Web Scraping: Cómo Comenzó y Qué Sucederá en El Futuro https://www.octoparse.es/blog/como-comenzo-y-sucedera-en-futuro 4 de Junio de 2023.

[4] World Wide Web Wanderer https://es.wikipedia.org/wiki/World_Wide_Web_Wanderer 4 de Junio de 2023.

[5] JumpStation https://en.wikipedia.org/wiki/JumpStation 5 de Junio de 2023.

[6] Web Driver VS IDE https://www.blazemeter.com/blog/selenium-ide-vs-webdriver 8 de Junio de 2023.

[7] Python https://www.python.org/ 18 de Mayo de 2023.

[8] Beautifulsoup https://pypi.org/project/beautifulsoup4/ 18 de Junio de 2023.

[9] Visual Web Scraper https://chrome.google.com/webstore/detail/visual-web-scraper/hnlnjfenbaiflgkdncofionfhnhmoono 18 de Junio de 2023.

[10] Definición casos de uso. https://es.wikipedia.org/wiki/Caso_de_uso 20 de Mayo de 2023.

[11] Selenium https://www.selenium.dev/ 5 de Junio de 2023.

[12] Amazon https://www.amazon.es/ 6 de Junio de 2023.

[13] Aliexpress https://es.aliexpress.com/ 5 de Junio de 2023

[14] Pyqt https://pypi.org/project/PyQt6/ 9 de Junio de 2023.

[15] Tkinter https://docs.python.org/es/3/library/tkinter.html#module-tkinter 9 de Junio de 2023.

[16] WxPython https://www.wxpython.org/ 9 de Junio de 2023.

[17] Programación orientada a objetos https://www.ibm.com/docs/es/spss-modeler/saas?topic=language-object-oriented-programming 6 de Mayo de 2023.

[18] CSS https://developer.mozilla.org/es/docs/Web/CSS 29 de Mayo de 2023.

[19] HTML https://desarrolloweb.com/home/html 19 de Mayo de 2023.

A. Anexo: Explicación del código

En el siguiente apartado se explica la organización del código, las clases padre web, producto así como sus implementaciones específicas de Amazon y Aliexpress. Anteriormente, se ha mostrado parte del código en el apartado 6.7, a continuación, volveremos a ver estas clases enfocándonos en las implementaciones especificas, y, por último, se comenta el main y la vista. El código se encuentra publicado en Github: https://github.com/adripi99/WebScraping.git.

A.1.- Estructura del proyecto y descripción

El proyecto está agrupado por carpetas, como vemos en la figura A.1, tenemos una carpeta *Objects* de la que cuelgan todas las clases de web y productos en sus respectivas carpetas, además tenemos la carpeta *Export* donde se almacenan los ficheros. En la raíz están las clases referentes a la vista y el main.

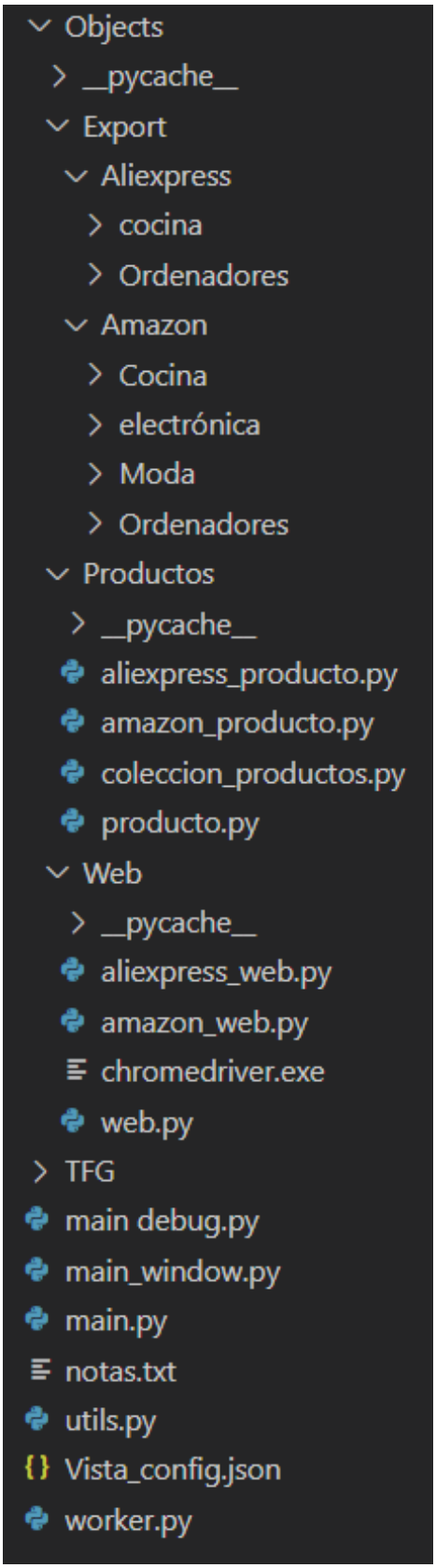

Figura A.1.- Estructura de carpetas

A continuación, vamos a comentar en orden descendente las carpetas y ver una breve descripción de su contenido. Dentro de *Objects* encontramos:

- **Export**:
- **Productos**:
  - *alixpress_productopy*: Clase hija de producto, contiene la implementación específica de un producto para la de la página web aliexpress.
  - *amazon_producto.py*: Clase hija de producto, contiene la implementación específica de un producto para la página web amazon.
  - *coleccion_productos.py:* Clase encargada de dar formato a la exportación de los productos con los atributos seleccionados.
  - *producto.py:* Clase padre, contiene los atributos básicos, sirve de base para generar nuevas implementaciones de productos.
- **Web**:
  - *alixpress_web.py:* Clase hija de web, contiene la implementación específica para la de la página web aliexpress.
  - *amazon_web.py:* Clase hija de web, contiene la implementación específica para la de la página web amazon.
  - *chromedriver.exe:* Driver de Chrome necesario para la clase padre web.
  - *web.py:* Clase padre, contiene las funciones necesarias para inicializar el webdriver y configurar el navegador. Contiene el esqueleto de las funciones a implementar en las clases hijas.

En la raíz están ubicados:

- **main_debug.py:** Un main diseñado para llamar directamente a las webs y facilitar la depuración del código de las clases hijas. No es parte del core de la aplicación pero puede resultar útil para futuras ampliaciones.
- **main_window.py:** Contiene la clase MainWindow, es la encargada de generar la vista y llamar al worker para extraer los datos.
- **main.py:** Contiene el main de la aplicación, se encarga de cargar el fichero de configuración y en llamar a la vista.

- **utils.py:** Contiene la función encargada de leer el .json de configuración. Se separa para facilitar las ampliaciones con diferentes ficheros de configuración y que podrían necesitar de diferentes funciones o utilidades.
- **Vista_config.json:** Contiene la configuración de la vista.
- **worker.py:** Contiene la clase worker, se encarga de crear y manejar el hilo que invoca la extracción de productos en las clases web.

A.1.1.- Clases web

Las clases web son las encargadas de invocar al driver y extraer los datos de las webs, son llamadas desde el Worker.py, se ayudan de las clases producto correspondientes para almacenar los datos y a la colección de productos para seleccionar y extraer los mismos.

```
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.chrome.service import Service
class Web:
    """
    Clase Web que representa una configuración de navegación web con Selenium.

    """
    def __init__(self,show_browser):
        """
        Inicializa una instancia de la clase Web.

        """
        self.driver_path = "Objects\Web\chromedriver.exe"
        self.show_browser=show_browser
    def configurar_navegador(self):
        """
        Configura el navegador web con las opciones especificadas.

        Args:
            show_browser (bool): Indica si se debe mostrar el navegador o ejecutar en
    modo headless.

```

```
        """
        service = Service(executable_path=self.driver_path)
        self.chrome_options = Options()
        if not self.show_browser:
            self.chrome_options.add_argument('--headless=new')  # Ocultar el navegador
        self.chrome_options.add_argument('--log-level=3')
        self.chrome_options.add_argument('--ignore-certificate-errors')
        self.chrome_options.add_argument('--ignore-ssl-errors')
        self.driver = webdriver.Chrome(service=service, options=self.chrome_options)

    def cerrar_navegador(self):
        """
        Cierra el navegador web.

        """
        self.driver.close()
        self.driver.quit()

    def obtener_url(self, categoria):
        """
        Obtiene la URL específica de la web para la categoría especificada.

        Args:
            categoria (str): Categoría para la cual se desea obtener la URL.

        Returns:
            str: URL específica de la web para la categoría especificada.

        """
        raise NotImplementedError("El método 'obtener_url()' debe ser implementado por cada clase hija.")

    def extraer_atributos_producto(self, elemento, atributosP):
        """
        Extrae los atributos específicos del producto en la web correspondiente.

        Args:
```

```
59              elemento: Elemento HTML del producto.
60              atributosP (bool): Indica si se deben extraer atributos en profundidad.
61
62          Returns:
63              dict: Diccionario de atributos extraídos del producto.
64
65          """
66          raise NotImplementedError("El método 'extraer_atributos_producto()' debe ser
    implementado por cada clase hija.")
67
68      def buscar_productos(self, categoria, num_productos, atributos_en_profundidad,
    atributos_a_extraer, log_callback=None):
69          """
70          Realiza la búsqueda y extracción de productos en la web correspondiente.
71
72          Args:
73              categoria (str): Categoría de productos a buscar.
74              num_productos (int): Número de productos a extraer.
75              atributos_en_profundidad (bool): Indica si se deben extraer atributos en
    profundidad.
76              atributos_a_extraer (list): Lista de atributos a extraer de los productos.
77              log_callback (func): Función de devolución de llamada para registrar
    mensajes de progreso (opcional).
78
79          Returns:
80              ColeccionProductos: Colección de productos extraídos.
81
82          """
83          raise NotImplementedError("El método 'buscar_productos()' debe ser
    implementado por cada clase hija.")
```

Código A.1.- Clase_Web.py

A continuación, vamos a explicar que realiza cada función. Cabe destacar que, en esta clase, las funciones obtener_url(), extraer_atributos_producto() y buscar_productos() están definidas con *NotImplementedError* ya que se implementan en las clases hijas que hereden de la clase Web. Vamos a ver las funciones en profundidad:

- **__init__(self, show_browser):** Este método es el constructor de la clase Web y se encarga de inicializar una instancia de la clase. Recibe un parámetro *show_browser* que indica si se debe mostrar el navegador o ejecutar en modo *headless*.
- **configurar_navegador(self):** Este método se encarga de configurar el navegador web con las opciones especificadas. Además, crea un objeto Service que específica la ubicación del controlador de Chrome, crea un objeto Options para configurar las opciones del navegador y, finalmente, crea una instancia de webdriver. Chrome pasando el controlador y las opciones.
- **cerrar_navegador(self)**: Este método se encarga de cerrar el navegador web.
- **obtener_url(self, categoria):** Este método debe ser implementado por las clases hijas. Recibe un parámetro categoría y debe devolver la URL específica de la web con la categoría especificada.
- **extraer_atributos_producto(self, elemento, atributosP):** Este método también debe ser implementado por las clases hijas. Recibe un parámetro elemento, que es el elemento HTML del producto, y atributosP, que indica si se deben extraer atributos en profundidad. Debe devolver un diccionario con los atributos extraídos del producto.
- **buscar_productos(self, categoria, num_productos, atributos_en_profundidad, atributos_a_extraer, log_callback=None):** Este método también debe ser implementado por las clases hijas. Recibe varios parámetros, incluyendo la categoría de productos a buscar, el número de productos a extraer, si se deben extraer atributos en profundidad, una lista de atributos a extraer de los productos y una función de devolución de llamada para registrar mensajes de progreso (este atributo es opcional). Este método realiza la búsqueda y extracción de productos en la web correspondiente.

A.1.1.1.- Clase hija Amazon_Web()

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from time import sleep
from Objects.Productos.amazon_producto import AmazonProducto
from Objects.Productos.coleccion_productos import ColeccionProductos
from Objects.Web.web import Web
from selenium.webdriver.chrome.options import Options

class AmazonWeb(Web):
    def __init__(self, show_browser):
        super().__init__(show_browser)

    def extraer_atributos_producto(self, elemento, atributosP):
        """
    Extrae los atributos de un elemento dado.

    Args:
        elemento (WebElement): Elemento del producto.
        atributosP (bool): Indica si se deben extraer atributos en profundidad.

    Returns:
        dict: Diccionario con los atributos extraídos del producto.
        """
        atributos_extraidos = {}
        try:
            titulo = elemento.find_element(By.XPATH, './/span[@class="a-size-base-plus
     a-color-base a-text-normal"]').text
            atributos_extraidos["Titulo"] = titulo
        except Exception as e:
            atributos_extraidos["Titulo"] = "No encontrado"

        try:
            precio = elemento.find_element(By.CSS_SELECTOR,'span.a-price-whole').text
            atributos_extraidos["Precio"] = precio
```

```
        except Exception as e:
            atributos_extraidos["Precio"] = "No encontrado"

        try:
            asin = elemento.get_attribute("data-asin")
            atributos_extraidos["Asin"] = asin
        except Exception as e:
            atributos_extraidos["Asin"] = "No encontrado"

        if atributosP:
            try:
                url_elemento = elemento.find_element(By.XPATH, './/a[@class="a-link-normal s-underline-text s-underline-link-text s-link-style a-text-normal"]')
                url = url_elemento.get_attribute('href')
                self.driver.get(url)
            except Exception as e:
                print("No se pudo entrar en profundidad")
            sleep(2)#esperamos a que cargue la página
            try:
                atributos_extraidos["Review"] = self.driver.find_element(by=By.XPATH, value='.//span[@id="acrCustomerReviewText"]').text
            except Exception as e:
                atributos_extraidos["Review"] = "No encontrado"

            try:
                atributos_extraidos["Vendedor"] = self.driver.find_element(By.XPATH, '//div[@tabular-attribute-name="Vendido por" and contains(@class, "tabular-buybox-text")]').text
            except Exception as e:
                atributos_extraidos["Vendedor"] = "No encontrado"

            try:
                atributos_extraidos["Estrellas"] = self.driver.find_element(By.XPATH, '//a[@role="button" and contains(@class, "a-popover-trigger")]/span[@class="a-size-base a-color-base"]').text
            except Exception as e:
                atributos_extraidos["Estrellas"] = "No encontrado"
```

```
66
67
68
69              try:
70                  self.driver.back()
71              except Exception as e:
72                  print("No se pudo volver para atrás")
73
74          return atributos_extraidos
75
76
77
78      def buscar_productos(self, categoria, num_productos,atributos_en_profundidad,
        atributos_a_extraer, log_callback=None):
79          """
80          Realiza la búsqueda de productos en la página de Amazon.
81
82          Args:
83              categoria (str): Categoría de productos a buscar.
84              num_productos (int): Número de productos a buscar.
85              atributos_en_profundidad (bool): Indica si se deben extraer atributos en
        profundidad.
86              atributos_a_extraer (list): Lista de atributos a extraer de los productos.
87              log_callback (func): Función de devolución de llamada para el registro de
        eventos.
88
89          Returns:
90              ColeccionProductos: Colección de productos encontrados.
91          """
92          try:
93              self.configurar_navegador()
94          except Exception as e:
95              print("Fallo al configurar el navegador")
96          try:
97              url = self.obtener_url_amazon(categoria)
98              self.driver.get(url)
99          except Exception as e:
```

```
            print("Fallo al obtener la url")
        productos = ColeccionProductos(atributos_a_extraer)
        Numero_Productos = 0
        # Aceptar las cookies
        sleep(2)
        try:
            accept_button = self.driver.find_element(By.ID, 'sp-cc-accept')
            accept_button.click()
        except Exception as e:
                print("No se encontró el botón de aceptar")
        while Numero_Productos != num_productos:
            try:
                wait = WebDriverWait(self.driver, 10)
                elementosList = wait.until(EC.presence_of_all_elements_located((By.
XPATH, '//div[contains(@class, "s-result-item s-asin")]')))
            except Exception as e:
                print("Fallo al obtener los elementos")
                self.cerrar_navegador()
                return productos
            for i in range(len(elementosList)):
                try:
                    elemento=self.driver.find_elements(By.XPATH, '//div[contains(
@class, "s-result-item s-asin")]')[i]
                    atributos_extraidos = self.extraer_atributos_producto(elemento,
atributos_en_profundidad)
                    producto = AmazonProducto(**atributos_extraidos)
                    productos.agregar_producto(producto)
                    Numero_Productos += 1
                except Exception as e:
                    print("Fallo al obtener el elemento")
                if log_callback is not None:
                    try:
                        sleep(1)
                        log_callback(f"Producto Numero: {Numero_Productos}")
                        sleep(1)
                        log_callback(f"Producto agregado: {producto.Titulo} - {
producto.Precio} - {producto.Asin}")
```

```
                        sleep(1)
                        log_callback(f"Restantes: {num_productos-Numero_Productos}")
                    except Exception as e:
                        print("Error al agregar el mensaje al registro:", str(e))

                if Numero_Productos == num_productos:
                    self.cerrar_navegador()
                    return productos

            try:
                if log_callback is not None:
                    try:
                        sleep(1)
                        log_callback(f"---Pasando de página---")
                    except Exception as e:
                        print("Error al agregar el mensaje al registro:", str(e))
                sleep(2) # Espaciamos las peticiones
                siguiente_pagina_url = self.driver.find_element(By.XPATH,'//a[contains(text(),"Siguiente")]').get_attribute('href')
                self.driver.get(siguiente_pagina_url)
            except:
                if log_callback is not None:
                    try:
                        sleep(1)
                        log_callback(f"Fin, no hay mas productos a extraer")
                        sleep(1)
                        log_callback(f"Producto totales agregados: {Numero_Productos}")
                    except Exception as e:
                        print("Error al agregar el mensaje al registro:", str(e))
                return productos

        return productos



    def obtener_url_amazon(self, categoria):
```

```
        """
        Obtiene la URL correspondiente a una categoría de productos en Amazon.

        Args:
            categoria (str): Categoría de productos.

        Returns:
            str: URL de la categoría en Amazon.
        """
        # Mapea cada categoría a una URL de Amazon
        categorias_urls = {
            "electrónica": "https://www.amazon.es/s?i=electronics&page=2",
            "moda": "https://www.amazon.es/s?i=fashion&page=2",
            "cocina": "https://www.amazon.es/s?i=kitchen&page=2",
            "ordenadores" :"https://www.amazon.es/s?i=computers&page=2"
            # -----------------Agrega más categorías Aquí-------------------------
        }
        return categorias_urls.get(categoria.lower())
```

Código A.2.- Clase hija Amazon_Web()

La clase AmzonWeb hereda de la clase web y contiene métodos específicos dedicados a la extracción de atributos de productos de la página web de Amazon. Vamos a ver las funciones en profundidad:

- **__init__(self, show_browser):** Este es el constructor de la clase AmazonWeb que llama al constructor de la clase padre Web y recibe el parámetro show_browser para indicar si se debe mostrar el navegador o ejecutar en modo headless.
- **extraer_atributos_producto(self, elemento, atributosP):** Este método se encarga de extraer los atributos de un elemento dado. Recibe el parámetro elemento, que es un objeto WebElement del producto, y atributosP, un booleano que indica si se deben extraer atributos en profundidad. Devuelve un diccionario con los atributos extraídos del producto.
- **buscar_productos(self, categoria, num_productos, atributos_en_profundidad, atributos_a_extraer, log_callback=None):** Este método se encarga de buscar los productos en la página de Amazon. Recibe varios parámetros, incluyendo la categoría de productos a buscar, el número de productos a buscar, si se deben extraer atributos en profundidad, una lista de atributos a extraer de los productos y el log callback para el registro de eventos. Devuelve una instancia de ColeccionProductos, que es la colección de productos encontrados.
- **obtener_url_amazon(self, categoria):**Este método se encarga de obtener la URL correspondiente a la categoría de productos en Amazon pasada como parámetro. Recibe el parámetro categoría y devuelve la URL de la categoría en Amazon.

A.1.1.2.- Clase hija Aliexpress_Web()

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from time import sleep
from Objects.Productos.coleccion_productos import ColeccionProductos
from Objects.Web.web import Web
from Objects.Productos.aliexpress_producto import AliexpressProducto
from selenium.webdriver.common.keys import Keys
class AliexpressWeb(Web):
    def __init__(self, show_browser):
        super().__init__(show_browser)

    def extraer_atributos_producto(self, elemento, atributosP,i):
        """
        Extrae los atributos del producto de un elemento dado.

        Args:
            elemento (WebElement): Elemento del producto.
            atributosP (bool): Indica si se deben extraer atributos en profundidad.
            i (int): Índice del elemento en la lista de elementos.

        Returns:
            dict: Diccionario con los atributos extraídos del producto.
        """
        atributos_extraidos = {}
        try:
            titulo = elemento.find_element(By.XPATH, './/h1[@class="manhattan--
    titleText--WccSjUS"]').text
            atributos_extraidos["Titulo"] = titulo
        except Exception as e:
            atributos_extraidos["Titulo"] = "No encontrado"

        try:
            precio_elemento = self.driver.find_elements(By.CLASS_NAME,'manhattan--
    price-sale--1CCSZfK')[i]
```

```
34            #precio_elemento = self.driver.find_element_by_class_name('manhattan--
    price-sale--1CCSZfK')
35            precio = precio_elemento.text.replace(',', '.')
36            atributos_extraidos["Precio"] = precio
37        except Exception as e:
38            atributos_extraidos["Precio"] = "No encontrado"
39
40        if atributosP:
41            try:
42                url_elemento = self.driver.find_elements(By.XPATH, '//a[contains(
    @class,"earch-card-item")]')[i]
43                url = url_elemento.get_attribute('href')
44                self.driver.get(url)
45            except Exception as e:
46                print("No se pudo entrar en profundidad")
47            sleep(2)#esperamos a que cargue la página
48            try:
49                atributos_extraidos["Estrellas"] = self.driver.find_element(By.XPATH,
    './/div[@class="ae-header-content-num"]').text
50
51            except Exception as e:
52                atributos_extraidos["Estrellas"] = "No tiene estrellas"
53
54            try:
55                atributos_extraidos["Vendedor"] = self.driver.find_element(By.XPATH, '
    .//a[@class="store-header--storeName--vINzvPw"]').text
56            except Exception as e:
57                atributos_extraidos["Vendedor"] = "No encontrado"
58            try:
59                self.driver.back()
60            except Exception as e:
61                print("No se pudo volver para atrás")
62        return atributos_extraidos
63
64    def buscar_productos(self, categoria, num_productos,atributos_en_profundidad,
    atributos_a_extraer, log_callback=None):
65        """
```

```
        Realiza la búsqueda de productos en la página de Aliexpress.

        Args:
            categoria (str): Categoría de productos a buscar.
            num_productos (int): Número de productos a buscar.
            atributos_en_profundidad (bool): Indica si se deben extraer atributos en profundidad.
            atributos_a_extraer (list): Lista de atributos a extraer de los productos.
            log_callback (func): Función de devolución de llamada para el registro de eventos.

        Returns:
            ColeccionProductos: Colección de productos encontrados.
        """
        try:
            self.configurar_navegador()
        except Exception as e:
            print("Fallo al configurar el navegador")
        try:
            url = self.obtener_url_aliexpress(categoria)
            self.driver.get("https://es.aliexpress.com")
            sleep(1)
            # Aceptar las cookies
            try:
                accept_button = self.driver.find_element(By.XPATH, './/button[@class="btn-accept"]')
                accept_button.click()
            except Exception as e:
                print("No se encontró el botón de aceptar")
            sleep(1)
            self.driver.get(url)
            self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.END)#Subimos y bajamos para que cargen todos los elementos de la pagina
            self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.END)
        except Exception as e:
            print("Fallo al obtener la url")
        productos = ColeccionProductos(atributos_a_extraer)
```

```
99          Numero_Productos = 0
100
101         while Numero_Productos != num_productos:
102             try:
103                 self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.END)
104                 self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.END)
105                 wait = WebDriverWait(self.driver, 20)
106                 elementosList = wait.until(EC.presence_of_all_elements_located((By.
    XPATH,'//a[contains(@class,"earch-card-item")]')))
107             except Exception as e:
108                 print("Fallo al obtener los elementos")
109                 self.cerrar_navegador()
110                 return productos
111             for i in range(len(elementosList)):
112                 try:
113                     elemento=self.driver.find_elements(By.XPATH, '//a[contains(@class
    ,"earch-card-item")]')[i]
114                 except:#Esto es necesario porque hay veces que dejan de aparecer todos
     los elemenos cuando estamos de la mitad para abajo
115                     self.driver.refresh()
116                     sleep(1)
117                     self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.END)
118                     sleep(1)
119                     self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.HOME)
120                     sleep(1)
121                     self.driver.find_element(By.TAG_NAME,'body').send_keys(Keys.END)
122                     sleep(1)
123                     wait.until(EC.presence_of_all_elements_located((By.XPATH,'//a[
    contains(@class,"earch-card-item")]')))
124                     try:
125                      elemento=self.driver.find_elements(By.XPATH, '//a[contains(@class
    ,"earch-card-item")]')[i]
126                     except:
127                         if log_callback is not None:
128                             try:
129                                 sleep(1)
```

```
                                log_callback(f"Error, no se puede extraer mas productos")
                                sleep(1)
                                log_callback(f"Producto totales agregados: {Numero_Productos}")
                                sleep(2)
                                return productos
                            except Exception as e:
                                print("Error al agregar el mensaje al registro:", str(e))
                                return productos
                        print("Error al encontrar el elemento:"+str(i))
                        return productos
                try:
                    atributos_extraidos = self.extraer_atributos_producto(elemento, atributos_en_profundidad,i)
                    producto = AliexpressProducto(**atributos_extraidos)
                    productos.agregar_producto(producto)
                    Numero_Productos += 1
                except Exception as e:
                    print("Fallo al guardar el elemento")
                if log_callback is not None:
                    try:
                        sleep(1)
                        log_callback(f"Producto Numero: {Numero_Productos}")
                        sleep(1)
                        log_callback(f"Producto agregado: {producto.Titulo} - {producto.Precio}")
                        sleep(1)
                        log_callback(f"Restantes: {num_productos-Numero_Productos}")
                    except Exception as e:
                        print("Error al agregar el mensaje al registro:", str(e))

                if Numero_Productos == num_productos:
                    self.cerrar_navegador()
                    return productos

```

```
162             try:
163                 if log_callback is not None:
164                     try:
165                         sleep(1)
166                         log_callback(f"---Pasando de página---")
167                     except Exception as e:
168                         print("Error al agregar el mensaje al registro:", str(e))
169                 sleep(3)  # Espaciamos las peticiones
170                 Burl=self.driver.find_element(By.XPATH, '//li[contains(text(),"
    Siguiente")]')
171                 Burl.click()
172                 sleep(2)
173             except:
174                 if log_callback is not None:
175                     try:
176                         sleep(1)
177                         log_callback(f"Fin, no hay mas productos a extraer")
178                         sleep(1)
179                         log_callback(f"Producto totales agregados: {Numero_Productos}"
    )
180                     except Exception as e:
181                         print("Error al agregar el mensaje al registro:", str(e))
182                 return productos
183 
184         return productos
185 
186     def obtener_url_aliexpress(self, categoria):
187         """
188         Obtiene la URL correspondiente a una categoría de productos en Aliexpress.
189 
190         Args:
191             categoria (str): Categoría de productos.
192 
193         Returns:
194             str: URL de la categoría en Aliexpress.
195         """
196         categorias_urls = {
```

```
197             "cocina": "https://es.aliexpress.com/category/204000021/consumer-
    electronics.html",
198             "ordenadores": "https://es.aliexpress.com/category/204000007/computer-
    office.html?category_redirect=1"
199             # Agrega más categorías y sus respectivas URL
200         }
201         return categorias_urls.get(categoria.lower())
```

Código A.3.- Clase hija Aliexpress_Web()

La clase AliexpressWeb hereda de la clase web y contiene métodos específicos dedicados a la extracción de atributos de productos de la página web de Aliexpress. Vamos a ver las funciones en profundidad:

- **__init__(self, show_browser)**: Este es el constructor de la clase AliexpressWeb que llama al constructor de la clase padre Web y recibe el parámetro show_browser para indicar si se debe mostrar el navegador o ejecutar en modo headless.
- **extraer_atributos_producto(self, elemento, atributosP, i):** Este método se encarga de extraer los atributos del producto de un elemento dado. Recibe como parámetro dicho elemento, que es un objeto WebElement del producto, atributosP, un booleano que indica si se deben extraer atributos en profundidad, e i, que es el índice del elemento en la lista de elementos. Devuelve un diccionario con los atributos extraídos del producto.
- **buscar_productos(self, categoria, num_productos, atributos_en_profundidad, atributos_a_extraer, log_callback=None):**Este método se encarga de realizar la búsqueda de productos en la página de Aliexpress. Recibe varios parámetros, incluyendo la categoría de productos a buscar, el número de productos a buscar, si se deben extraer atributos en profundidad, una lista de atributos a extraer de los productos y el log callback para el registro de eventos. Devuelve una instancia de ColeccionProductos, que es una colección de productos encontrados.
- **obtener_url_aliexpress(self, categoria):** Este método se encarga de obtener la URL correspondiente a una categoría de productos en Aliexpress. Recibe el parámetro categoría y devuelve la URL de esta.

A.1.2.- Clases Producto

Las clases producto son las encargadas de crear el objeto producto con los atributos pasados por la clase web, se encargan de componer el objeto para poder ser almacenado en la clase web.

A.1.2.1.- Clase producto.py

```python
class Producto:
    """
    Clase Producto que representa un producto con un título y un precio.

    Args:
        titulo (str): Título del producto.
        precio (float): Precio del producto.

    """
    def __init__(self, titulo, precio):
        """
        Inicializa una instancia de la clase Producto con el título y el precio especificados.

        Args:
            titulo (str): Título del producto.
            precio (float): Precio del producto.

        """
        self.Titulo = titulo
        self.Precio = precio

    def __str__(self):
        """
        Devuelve una representación en cadena del producto.

        Returns:
            str: Representación en cadena del producto en formato "Título - Precio".

        """
        return f"{self.Titulo} - {self.Precio}"
```

Código A.4.- Clase producto.py

La clase Producto representa un producto con un título y un precio, estos son los atributos imprescindibles para un producto. A continuación, vamos a ver las funciones en profundidad:

- **__init__(self, titulo, precio):** Este es el constructor de la clase Producto, el cual inicializa una instancia del producto con el título y el precio especificados. Recibe los parámetros titulo (str) y precio (float).
- **__str__(self):** Este método se encarga de devolver una representación en cadena del producto. Devuelve la cadena en el formato "Título - Precio", donde Título es el título del producto y Precio es el precio del producto.

A.1.2.2.- Clase amazon_producto.py

```
1  from Objects.Productos.producto import Producto
2
3  class AmazonProducto(Producto):
4      def __init__(self, Titulo, Precio, Asin, Review=None, Vendedor=None, Estrellas=
       None):
5          super().__init__(Titulo, Precio)
6          self.Asin = Asin
7          self.Review=Review
8          self.Vendedor=Vendedor
9          self.Estrellas=Estrellas
10     def __str__(self):
11         return f"{self.Titulo} - {self.Precio} - {self.Asin}\n"
```

Código A.5.- amazon_producto.py

La clase AmazonProducto hereda de la clase Producto y representa un producto específico de Amazon. A continuación, vamos a ver las funciones en profundidad:

- **__init__(self, Titulo, Precio, Asin, Review=None, Vendedor=None, Estrellas=None):** Este es el constructor de la clase AmazonProducto que llama al constructor de la clase padre Producto y recibe los parámetros *Titulo, Precio, Asin, Review, Vendedor* y *Estrellas*. Los primeros dos parámetros (Titulo y Precio) se pasan al constructor de la clase padre llamando al super. Los parámetros adicionales

(Asin, Review, Vendedor y Estrellas) se asignan a los atributos correspondientes de la clase AmazonProducto.

- **__str__(self):** Este método se encarga de devolver una representación en cadena del producto de Amazon. Esta cadena tiene el formato *Título - Precio - Asin,* donde Título es el título del producto, Precio es el precio y Asin es el identificador único del producto.

A.1.2.3.- Clase aliexpress_producto.py

```
from Objects.Productos.producto import Producto


class AliexpressProducto(Producto):
    def __init__(self, Titulo, Precio,Estrellas=None, Vendedor=None):
        super().__init__(Titulo, Precio)
        self.Estrellas=Estrellas
        self.Vendedor=Vendedor
    def __str__(self):
        return f"{self.Titulo} - {self.Precio}\n"
```

Código A.6.- aliexpress_producto.py

La clase AliexpressProducto hereda de la clase Producto y representa un producto específico de Aliexpress. A continuación, vamos a ver las funciones en profundidad:

- **__init__(self, Titulo, Precio, Estrellas=None, Vendedor=None)):** Este es el constructor de la clase AliexpressProducto que llama al constructor de la clase padre Producto y recibe los parámetros *Titulo, Precio, Vendedor* y *Estrellas*. Los primeros dos parámetros (Titulo y Precio) se pasan al constructor de la clase padre llamando al super. Los parámetros adicionales (Estrellas y Vendedor) se asignan a los atributos correspondientes de la clase AliexpressProducto.
- **__str__(self):** Este método se encarga de devolver una representación en cadena del producto de Aliexpress. Esta cadena tiene el formato *Título - Precio* , donde Título hace referencia al título del producto y Precio hace referencia al precio del producto.

A.1.2.4.- Clase Coleccion_productos.py

Esta clase es la encargada de recibir los productos desde la clase web, almacenarlos y posteriormente exportarlos en el formato solicitado.

```
import pandas as pd

class ColeccionProductos:
    """
    Clase ColeccionProductos que representa una colección de productos y proporciona funcionalidades como agregar productos, eliminar duplicados y exportar los datos en diferentes formatos.

    Args:
        lista_seleccion (list): Lista de columnas seleccionadas para la colección de productos.

    """
    def __init__(self,lista_seleccion):
        """
        Inicializa una instancia de la clase ColeccionProductos con la lista de selección especificada.

        Args:
            lista_seleccion (list): Lista de columnas seleccionadas para la colección de productos.

        """
        self.df = pd.DataFrame(columns=lista_seleccion)  #paso la lista de seleccion del user
        self.seleccion=lista_seleccion

    def agregar_producto(self, producto):
         """
        Agrega un producto a la colección.

        Args:
```

```
            producto (Producto): Producto a agregar a la colección.

        """
         nuevo_df = pd.DataFrame([producto.__dict__])
         self.df = pd.concat([self.df, nuevo_df], ignore_index=True)

    #Exportar al final de extraer los datos, recorrer en 2 niveles todos los produtos.
     Usar un dict en amazon web
    def eliminar_duplicados(self):
        """
        Elimina los productos duplicados de la colección.

        """
        self.df = self.df.drop_duplicates()

    def exportar(self, formato, path):
        """
        Exporta los datos de la colección en el formato especificado.

        Args:
            formato (str): Formato de exportación ("json", "csv", "html", "pickle", "
    parquet", "feather", "hdf").
            path (str): Ruta de archivo para guardar los datos exportados.

        Raises:
            ValueError: Si se proporciona un formato de exportación no válido.

        """
        df_seleccionado = self.df[self.seleccion]
        if formato == "json":
            json_data = df_seleccionado.to_json(orient="records", indent=4)
            with open(path, "w") as f:
                f.write(json_data)
        elif formato == "csv":
            df_seleccionado.to_csv(path, index=False)
        elif formato == "html":
            df_seleccionado.to_html(path, index=False)
```

```
62          elif formato == "pickle":
63              df_seleccionado.to_pickle(path)
64          elif formato == "parquet":
65              df_seleccionado.to_parquet(path)
66          elif formato == "feather":
67              df_seleccionado.to_feather(path)
68          elif formato == "hdf":
69              df_seleccionado.to_hdf(path, key="data", mode="w")
70          else:
71              raise ValueError("Formato de exportación no válido.")
```

Código A.7.- Clase Coleccion_productos.py

La clase ColeccionProductos representa la colección de productos y proporciona funcionalidades para agregar productos, eliminar duplicados y exportar los datos en diferentes formatos. A continuación, vamos a ver las funciones en profundidad:

- **__init__(self, lista_seleccion):** Este es el constructor de la clase *ColeccionProductos* que inicializa una instancia de la colección de productos. Recibe como parámetro *lista_seleccion*, que es una lista de columnas seleccionadas para la colección de productos. Crea un DataFrame vacío utilizando la librería Pandas, con las columnas especificadas en *lista_seleccion* y guarda esta lista en el atributo *seleccion*.
- **agregar_producto(self, producto)**: Este método se encarga de agregar un producto a la colección. Recibe el parámetro *producto*, que es un objeto de la clase *Producto*. Crea un nuevo DataFrame a partir del diccionario de atributos del producto y lo concatena con el DataFrame existente en la colección.
- **eliminar_duplicados(self)**: Este método se encarga de eliminar los productos duplicados de la colección. Utiliza el método *drop_duplicates()* de Pandas para eliminar las filas duplicadas en el DataFrame.
- **exportar(self, formato, path)**: Este método se encarga de exportar los datos de la colección en el formato especificado. Recibe los parámetros *formato*, que es una cadena que representa el formato de exportación, y *path*, que es la ruta del archivo para guardar los datos exportados. Dependiendo del formato especificado, utiliza los métodos correspondientes de Pandas para exportar el DataFrame seleccionado en

el formato deseado. Los formatos admitidos son ,*csv, html, pickle, parquet, feather* y *hdf.*

A.1.3.- Clases encargadas de la vista.

La clase que genera la vista es main_window.py, no obstante las demás clases en la raíz, colaboran para que la vista sea posible.
El main.py es el encargado de leer el fichero de configuración ayudandose de utils.py y llamar con su salida a la clase main_window.py.
La clase main_window.py se encarga de recibir la configuración de la vista, componer con esta la misma y una vez el usuario ha seleccionado la configuración deseada capturar el evento de inicio y llamar a la clase worker.py con lo que se desea extraer.

A.1.3.1.- La clase main_window.py

```
from PyQt6.QtWidgets import QMainWindow, QLabel, QGridLayout, QVBoxLayout, QHBoxLayout
    , QWidget, QLineEdit, QPushButton, QComboBox, QTextEdit, QCheckBox, QListWidget,
    QMessageBox
from PyQt6.QtCore import QRegularExpression
from PyQt6.QtGui import QRegularExpressionValidator
from worker import Worker
class MainWindow(QMainWindow):
    """
    Clase MainWindow que representa la interfaz de usuario principal.

    """
    def __init__(self,json):
        super().__init__()
        self.jsondata=json
        self.worker = None

        self.setWindowTitle("Web Scraper")
        self.setGeometry(100, 100, 400, 300)

        self.web_label = QLabel("Pagina Web:")
```

```
        self.web_combo = QComboBox()


        self.categoria_label = QLabel("Categoría:")
        self.categoria_combo = QComboBox()



        self.num_productos_label = QLabel("Número de productos:")
        self.num_productos_edit = QLineEdit()
        # validar para aceptar solo números enteros
        validator = QRegularExpressionValidator(QRegularExpression("[0-9]+"))
        self.num_productos_edit.setValidator(validator)


        self.available_products_list = QListWidget()  # Lista de atributos disponibles
        self.selected_products_list = QListWidget()  # Lista de atributos
    seleccionados

        self.move_to_selected_button = QPushButton(">")
        self.move_to_available_button = QPushButton("<")


        self.export_label = QLabel("Formato de exportación:")
        self.export_combo = QComboBox()
        formats = ["csv", "html", "feather", "parquet", "pickle"]
        for format in formats:
            self.export_combo.addItem(format)

        self.show_browser_checkbox = QCheckBox("Mostrar Navegador")

        self.start_button = QPushButton("Iniciar")

        self.log_text = QTextEdit()
        self.log_text.setReadOnly(True)
        left_layout = QVBoxLayout()  # Layout para la columna izquierda
        right_layout = QHBoxLayout()  # Layout para la columna derecha
```

```
55
56          left_layout.addWidget(self.web_label)
57          left_layout.addWidget(self.web_combo)
58          left_layout.addWidget(self.categoria_label)
59          left_layout.addWidget(self.categoria_combo)
60          left_layout.addWidget(self.num_productos_label)
61          left_layout.addWidget(self.num_productos_edit)
62
63          attribute_layout = QGridLayout()  # Layout en cuadrícula para la sección de
    atributos
64          attribute_layout.addWidget(QLabel("Atributos Disponibles a extraer:"), 0, 0)
    # Encima de la lista de atributos disponibles
65          attribute_layout.addWidget(QLabel("Atributos Seleccionados:"), 0, 1)  # Encima
     de la lista de atributos seleccionados
66          attribute_layout.addWidget(self.available_products_list, 1, 0)  # Lista de
    atributos disponibles
67          attribute_layout.addWidget(self.selected_products_list, 1, 1)  # Lista de
    atributos seleccionados
68          attribute_layout.addWidget(self.move_to_selected_button, 2, 0)  # Botón ">"
69          attribute_layout.addWidget(self.move_to_available_button, 2, 1)  # Botón "<"
70          left_layout.addLayout(attribute_layout)
71
72
73          left_layout.addWidget(self.export_label)
74          left_layout.addWidget(self.export_combo)
75          left_layout.addWidget(self.show_browser_checkbox)
76          left_layout.addWidget(self.start_button)
77          left_layout.addWidget(self.log_text)
78
79          main_layout = QVBoxLayout()  # Layout principal
80          main_layout.addLayout(left_layout)
81          main_layout.addLayout(right_layout)
82
83          widget = QWidget()
84          widget.setLayout(main_layout)
85          self.setCentralWidget(widget)
86          #Eventos
```

```
87
88          self.web_combo.currentTextChanged.connect(self.cambia_categoria)
89          self.start_button.clicked.connect(self.start_scraping)
90          self.move_to_selected_button.clicked.connect(self.move_to_selected)
91          self.move_to_available_button.clicked.connect(self.move_to_available)
92
93      def move_to_selected(self):
94          """
95          Mueve un atributo seleccionado de la lista de atributos disponibles a la lista
     de atributos seleccionados.
96
97          """
98          selected_item = self.available_products_list.currentItem()
99          if selected_item is not None:
100             self.available_products_list.takeItem(self.available_products_list.row(
    selected_item))
101             self.selected_products_list.addItem(selected_item.text())
102
103     def move_to_available(self):
104         """
105         Mueve un atributo seleccionado de la lista de atributos seleccionados a la
    lista de atributos disponibles.
106
107         """
108         selected_item = self.selected_products_list.currentItem()
109         if selected_item is not None:
110             self.selected_products_list.takeItem(self.selected_products_list.row(
    selected_item))
111             self.available_products_list.addItem(selected_item.text())
112     def Merror(self,error_text):
113         msg = QMessageBox()
114         msg.setIcon(QMessageBox.Icon.Critical)
115         msg.setText("Error")
116         msg.setInformativeText(error_text)
117         msg.setWindowTitle("Error")
118         msg.exec()
119     def start_scraping(self):
```

```
        """
        Inicia el proceso de web scraping con los parámetros especificados.

        """
        if self.worker is not None and self.worker.isRunning():
            self.worker.quit()
            self.worker.wait()
            self.worker.deleteLater()

        web = self.web_combo.currentText()
        categoria = self.categoria_combo.currentText()
        num_productos_text = self.num_productos_edit.text()
        export_format = self.export_combo.currentText().lower()
        show_browser = self.show_browser_checkbox.isChecked()
        atributos_a_extraer = [self.selected_products_list.item(i).text() for i in
    range(self.selected_products_list.count())]
        atributos_en_profundidad= False

        if any(atributo in atributos_a_extraer for atributo, valor in self.jsondata[
    web].get("atributos", {}).items() if valor == 1):
            atributos_en_profundidad = True

        try:
            num_productos = int(num_productos_text)
            if num_productos <= 0:
                self.Merror("El número de productos debe ser mayor a cero.")
                return
        except ValueError:
            self.Merror("Ingrese un número válido para el número de productos.")
            return

        if not atributos_a_extraer:
            self.Merror("Debe seleccionar al menos un atributo para extraer.")
            return

        self.start_button.setEnabled(False)  # Deshabilitar el botón de inicio
        self.log_callback("Comenzando el Scraping, espere.")
```

```
155
156         self.worker = Worker(web, categoria, num_productos, atributos_a_extraer,
    atributos_en_profundidad, show_browser,export_format, self.log_callback)
157
158         self.worker.finished.connect(self.scraping_finished)
159
160         self.worker.start()
161
162     def cambia_categoria(self):
163         """
164         Actualiza la lista de categorías disponibles según la página web seleccionada.
165
166         """
167         config = self.jsondata
168         if self.web_combo.currentText() == '':
169             webs = list(config.keys())
170             for webi in webs:
171                 self.web_combo.addItem(webi.capitalize())
172         web = self.web_combo.currentText()
173
174         self.categoria_combo.clear()  # Vaciar el contenido de categoria_combo\
175         self.available_products_list.clear()
176         self.selected_products_list.clear()
177         categoria_config = config[web].get("categorias", {})
178         for categoria in categoria_config:
179             self.categoria_combo.addItem(categoria.capitalize())
180
181         Atributos=config[web].get("atributos", {})
182         for atributo in Atributos:
183             self.available_products_list.addItem(atributo.capitalize())
184     def scraping_finished(self):
185         """
186         Se ejecuta cuando el proceso de web scraping ha finalizado.
187
188         """
189         self.log_callback("Búsqueda de productos finalizada.")
190         self.start_button.setEnabled(True)  # Habilitar el botón de inicio
```

```
191         self.worker = None
192 
193     def log_callback(self, message):
194         """
195         Agrega un mensaje al registro de eventos en la interfaz de usuario.
196 
197         Args:
198             message (str): Mensaje a agregar al registro.
199 
200         """
201         self.log_text.append(message)
202         self.log_text.verticalScrollBar().setValue(self.log_text.verticalScrollBar().
    maximum()) # Asegura que el texto agregado esté visible en la parte inferior del
    QTextEdit
```

Código A.8.- main_window.py

En el código se muestra la interfaz construida con la librería PyQt6. A continuación, vamos a ver las funcionalidades mas relevantes del código en vez de comentar función a función como vimos en los apartados anteriores. La clase MainWindow hereda de QMainWindow y representa la ventana principal de la interfaz de usuario. El constructor de la clase MainWindow recibe un parámetro json que contiene datos para la inicialización de la interfaz. Configura el título y la geometría de la ventana principal, y crea los diferentes widgets y layouts necesarios.

Los métodos move_to_selected y move_to_available se ejecutan cuando se hace clic en los botones. Estos métodos permiten mover atributos seleccionados entre dos listas: la lista de atributos disponibles y la lista de atributos seleccionados.

El método Merror muestra un cuadro de diálogo de error con un mensaje de error específico. Este se encarga de avisar al usuario si introduce un dato erroneo o si le falta.

El método start_scraping se ejecuta cuando se hace clic en el botón Ïniciar". Este método recopila los datos ingresados en la interfaz (como la página web, la categoría, el número de productos, etc.), realiza algunas validaciones y luego inicia el proceso de web scraping utilizando la clase Worker.

El método cambia_categoria se ejecuta cuando se cambia la selección de la página web en el combo box correspondiente. Actualiza la lista de categorías disponibles según la página web seleccionada.

El método scraping_finished se ejecuta cuando el proceso de web scraping ha finalizado. Realiza acciones como habilitar el botón de inicio y limpiar variables.

El método log_callback agrega un mensaje al registro de eventos en la interfaz de usuario, mostrándolo en un QTextEdit.

A.1.3.2.- La clase main.py

```
#!TFG\Scripts\python.exe
import sys
from PyQt6.QtWidgets import QApplication
from main_window import MainWindow
from utils import load_config

if __name__ == "__main__":
    app = QApplication(sys.argv)
    jsondata= load_config("Vista_config.json") #encoding utf
    window = MainWindow(jsondata)
    window.cambia_categoria()
    window.show()
    try:
        sys.exit(app.exec())
    except Exception as e:
        print("Se produjo un error:", str(e))
```

Código A.9.- main.py

Este código es el principal, se encarga de crear una instancia de la aplicación QApplication y de cargar la configuración desde un archivo JSON utilizando la función load_config. A continuación, se crea una instancia de la clase MainWindow, pasando los datos cargados desde el archivo JSON como parámetro. Se llama al método cambia_categoria para actualizar la interfaz según la selección inicial de la página web, esta primera llamada

sirve para hacer la primera carga de la vista. Finalmente, se muestra la ventana principal y se ejecuta la aplicación llamando a *app.exec()*.

A.1.3.3.- La clase worker.py

```
from PyQt6.QtCore import QThread,pyqtSignal
from Objects.Web.aliexpress_web import AliexpressWeb
from Objects.Web.amazon_web import AmazonWeb
import os
from datetime import datetime
class Worker(QThread):
    """
    Clase Worker que realiza el trabajo de web scraping en un hilo aparte.

    Args:
        web (str): Nombre de la página web a escrapear.
        categoria (str): Categoría de productos a buscar.
        num_productos (int): Número de productos a buscar.
        atributos_a_extraer (list): Lista de atributos a extraer de los productos.
        atributos_en_profundidad (bool): Indica si se deben extraer atributos en
    profundidad.
        show_browser (bool): Indica si se debe mostrar el navegador durante el
    scraping.
        export_format (str): Formato de exportación de los resultados.
        log_callback (func): Función de devolución de llamada para el registro de
    eventos.
    """
    finished = pyqtSignal()
    def __init__(self, web, categoria, num_productos, atributos_a_extraer,
    atributos_en_profundidad, show_browser,export_format, log_callback):
        super().__init__()
        self.web=web
        self.categoria = categoria
        self.num_productos = num_productos
        self.atributos_en_profundidad = atributos_en_profundidad
        self.show_browser = show_browser
        self.running = True
```

```
        self.log_callback = log_callback
        self.export_format=export_format
        self.atributos_a_extraer=atributos_a_extraer
    def run(self):
        """
        Método que ejecuta el trabajo de web scraping.

        """
        web = None

        if self.web == "Amazon":
            web = AmazonWeb(self.show_browser)
        elif self.web == "Aliexpress":
            web = AliexpressWeb(self.show_browser)
        else:
            self.log_callback("Web no reconocida.")
            self.finished.emit()
            return
        productos = web.buscar_productos(self.categoria, self.num_productos,self.
    atributos_en_profundidad,self.atributos_a_extraer, self.log_callback)
        # exportar los Objects\Export
        directorio="Objects/Export/"+self.web+"/"+self.categoria+"/"
        os.makedirs(directorio, exist_ok=True)
        productos.exportar(self.export_format,directorio+str(datetime.now().timestamp
    ())+"."+self.export_format)
        self.finished.emit()

    def stop(self):
        """
        Método que detiene la ejecución del trabajo.

        """
        self.running = False
```

Código A.10.- worker.py

A.1.3.4.- La clase utils.py

```python
import json
def load_config(file_path):
    """
    Lee la configuración desde un .json

    Args:
         file_path (str): Ruta del fichero de configuración.

        """
    with open(file_path, "r", encoding="utf-8") as f:
        config = json.load(f)
    return config
```

Código A.11.- utils.py

Este código hereda de la clase QThread se encarga de realizar el trabajo de web scraping en un hilo separado. Recibe varios parámetros en su constructor, incluyendo el nombre de la página web a scrapear, la categoría de productos a buscar, el número de productos a buscar, la lista de atributos a extraer de los productos, un indicador para extraer atributos en profundidad, un indicador para mostrar el navegador durante el scraping, el formato de exportación de los resultados y una función de devolución de llamada para el registro de eventos.

Tiene la clase Worker, la cual genera una señal *finished* que se emite cuando el trabajo de web scraping ha finalizado para poder ser capturada por Main_Window.py. Es importante también, el método run() es el punto de entrada del hilo y se encarga de ejecutar el trabajo de web scraping. Dependiendo del nombre de la página web especificada, se crea una instancia de la clase correspondiente (AmazonWeb o AliexpressWeb). A continuación, se llama al método buscar_productos() de la instancia web para buscar los productos en la categoría especificada. En penúltimo lugar, se exportan los productos en el formato especificado y se emite la señal *finished*. Por ultimo, el método stop() se utiliza para detener la ejecución del trabajo de web scraping estableciendo la variable *running* en False.

A.1.3.5.- Fichero de configuración.

```
{
  "Amazon":
  {
  "categorias": ["electrónica","moda","cocina","ordenadores"],
  "atributos":{
    "Titulo":0,
    "Precio":0,
    "Review":1,
    "Asin":0,
    "Vendedor":1,
    "Estrellas":1
        }
      },
  "Aliexpress":
  {
  "categorias": ["cocina","ordenadores"],
  "atributos":{
    "Titulo":0,
    "Precio":0,
    "Vendedor":1,
    "Estrellas":1
  }
  }
  }
```

Código A.12.- Vista_config.json

Este fichero contiene de forma estructurada las webs con sus categorías y atributos así como el nivel de profundidad de estos.