

# Trabajo Final de Máster.

**Máster de Programación Avanzada en Python para Big Data, Hacking y Machine Learning (IX Edición).**

Análisis Inmobiliario mediante Web Scraping y Machine Learning

Alumno 1: Jaime Muñoz Fisac (Web Scraping)  
Alumno 2: Vicente Sosa Alcolea (Base de Datos - Django)  
Alumno 3: Aliaksandr lakhvedau lakhvedau (Machine Learning)

NIF: 53406955N  
NIF: 48387900V  
NIF: 12817824Q

31 – enero - 2024

**Declaración Preliminar: Fines Didácticos y Limitaciones Legales**

Este trabajo se ha desarrollado única y exclusivamente con fines educativos y de investigación. Esta aclaración se realiza con la finalidad de eximirnos de cualquier responsabilidad derivada de la extracción de datos de la plataforma. En consecuencia, se recomienda encarecidamente utilizar estas prácticas de manera ética y respetando los términos y condiciones de los sitios web involucrados, garantizando así la integridad de la investigación y el cumplimiento de las normativas legales establecidas.

Adicionalmente, se establece expresamente la prohibición de compartir o publicar este trabajo con personas ajena a los evaluadores designados para la presente evaluación. La confidencialidad de la información y la restricción de su difusión se consideran aspectos fundamentales para el respeto de los términos y condiciones de Idealista y la integridad del proceso evaluativo.

**Repositorio GitHub**

Todo el trabajo y el código del programa, se encuentra en el siguiente repositorio de GitHub.

[https://github.com/JMFisac/TFM\\_analisis\\_inmobiliario\\_mediente\\_web\\_scraping\\_y\\_machine\\_learning](https://github.com/JMFisac/TFM_analisis_inmobiliario_mediente_web_scraping_y_machine_learning)

Para darle acceso, contacte con Jaime Muñoz Fisac a través de telegram o correo electrónico jaimemunozfisac@gmail.com

## ÍNDICE

Declaración Preliminar: Fines Didácticos y Limitaciones Legales.....	1
Repository GitHub.....	1
1. RESÚMEN .....	3
2. INTRODUCCIÓN.....	4
2.1. Planteamiento del problema. ....	4
2.1. Objetivos. ....	4
2.2. Justificación del Proyecto.....	5
2.3. Metodología Empleada.....	5
3. MARCO TEÓRICO. .....	6
3.1. Web Scraping.....	6
3.2. Bases de Datos.....	7
3.3. Machine Learning.....	7
4. METODOLOGÍA.....	9
4.1. Web Scraping.....	9
4.2. Base de Datos SQL.....	16
4.3. Machine Learning Aplicado. ....	17
4.4. Desarrollo de la Aplicación Web. ....	19
5. EVALUACIÓN DE LOS RESULTADOS.....	21
5.1. Web Sraping.....	21
5.2. Base de Datos.....	21
5.1. Machine Learning.....	21
6. POSIBLES MEJORAS Y TRABAJOS FUTUROS. ....	22
6.1. Web Sraping.....	22
6.2. Base de Datos.....	22
6.3. Machine Learning.....	23
7. CONCLUSIONES.....	23
8. REFERENCIAS .....	24
9. LISTA DE ANÉXOS.....	25

## 1. RESÚMEN

Ante la variación de los precios de venta y alquiler de inmuebles, y gracias a la existencia de portales inmobiliarios, podemos observar tendencias o aproximaciones de los precios de dichos inmuebles, que no llegan a ser más que conjeturas basadas en la observación, desde el punto de vista de un particular. En el caso de profesionales, parten de grandes bases de datos, y costosos estudios de mercado, que llevan en su mayoría un laborioso trabajo que implican grandes cantidades de dinero y tiempo.

Debido a la abundancia de información disponible en diversas plataformas sobre propiedades, es factible utilizar estos datos para crear modelos de predicción que permitan estimar el valor de una propiedad en el mercado en función de sus características específicas.

En este trabajo se quiere presentar una solución al mercado inmobiliario, que se podrá aplicar tanto a profesionales como a particulares. Para acotar el problema, se ha elegido la Isla de Gran Canaria para elaborar la Base de Datos.

Se pretende elaborar un modelo para la predicción del precio de mercado de un inmueble en Las Islas Canarias empleando técnicas de Machine Learning aprendidas durante el Master. Los datos para su realización se extraerán de la web de la plataforma idealista, mediante técnicas de *Web Scraping*.

Los datos se quieren incorporar en una Base de datos SQL para su posterior utilización en el algoritmo de aprendizaje, así como para explotar y analizar esos datos para realizar gráficas que ayuden a la hora de elegir una vivienda.

Por tanto, la aplicación no solo contará con el modelo de predicción, si no que podrá ser una herramienta de análisis inmobiliario, mostrando gráficas y datos de interés. Con opción a filtrado y elección de datos.

Al tratarse de un trabajo en grupo, utilizaremos una metodología Ágil para coordinar y repartir los trabajos, para que todos participemos de igual medida en la consecución de los objetivos marcados.

## 2. INTRODUCCIÓN.

### 2.1. Planteamiento del problema.

En el mercado inmobiliario, tanto particulares como profesionales se enfrentan a problemas al intentar estimar el valor de una propiedad. Los particulares dependen en gran medida de conjeturas basadas en la observación sobre todo de portales inmobiliarios, los profesionales invierten considerables recursos en estudios de mercado y análisis de grandes bases de datos. La existencia de portales inmobiliarios y la abundancia de información disponible ofrecen una oportunidad para aprovechar técnicas de Machine Learning y crear modelos de predicción que faciliten la estimación del precio de mercado de un inmueble.

### 2.1. Objetivos.

#### 2.1.1. Objetivo General.

El Objetivo General es desarrollar una solución integral basada en técnicas de Machine Learning para la predicción del precio de mercado de inmuebles la Isla de Gran Canaria.

#### 2.1.2. Objetivos Específicos.

- | Crear una Base de Datos SQL para almacenar y gestionar eficientemente los datos recopilados, permitiendo su utilización en el desarrollo del modelo de predicción.
- | Realizar un proceso de Web Scraping en la plataforma idealista para recopilar datos sobre propiedades en Gran Canaria.
- | Investigar y seleccionar las técnicas de Machine Learning más apropiadas para la creación de un modelo de predicción de precios de inmuebles.
- | Validar y evaluar la precisión del modelo de predicción, ajustando parámetros si es necesario para mejorar la eficacia del modelo.
- | Diseñar e implementar una aplicación que utilice el modelo de predicción desarrollado, ofreciendo funcionalidades de análisis inmobiliario, visualización de gráficas y opciones de filtrado y selección de datos.
- | Implementar una metodología ágil para coordinar las tareas del equipo.

## 2.2. Justificación del Proyecto.

La justificación de este trabajo radica en la necesidad de proporcionar a particulares y profesionales una herramienta integral y accesible que permita estimar de manera precisa y eficiente el valor de mercado de los inmuebles en Gran Canaria.

## 2.3. Metodología Empleada.

Para abordar este proyecto de manera eficiente y colaborativa, se ha empleado metodología ágil, en concreto, se ha adoptado la metodología Scrum, ampliamente utilizada en el desarrollo de software. Scrum proporcionará un marco de trabajo flexible que permite la adaptación continua a medida que avanza el proyecto.

Basándonos en la asignatura de “Desarrollo y Gestión Ágil de Proyectos”, hemos adaptado las técnicas aprendidas al equipo de trabajo, sin definir unos roles específicos para que de este modo todos aportáramos por igual al proyecto sin diferencia. Además, los pasos o puntos que hemos tenido en cuenta han sido los siguientes:

- | Definición del Producto: En una primera reunión definimos qué queríamos desarrollar, y con qué propósito.
- | Backlog del Producto: Creamos un backlog que incluía todas las tareas necesarias para el desarrollo del proyecto, priorizando según la importancia y la dependencia entre las tareas. Y de esta manera nos repartimos el trabajo.
- | Planificación de Sprint: Organizamos reuniones de planificación de sprint para seleccionar tareas del backlog y establecer los objetivos específicos para cada sprint.
- | Sprints: Forzados por la cantidad de tiempo disponible y sobre todo por lo difícil de cuadrar reuniones los hemos realizado cada 2-3 semanas.
- | Reuniones Diarias de Scrum: No hemos realizado como tal, nos actualizábamos el trabajo mediante telegram, para ir chequeando el estado del Sprint.
- | Revisión o retrospectiva de Sprint: Al final de cada sprint, hemos evaluado el trabajo completado y de esta manera obtendríamos retroalimentación, lo que permitía realizar ajustes en los futuros sprints.

Esta implementación de Scrum nos ha proporcionado un marco estructurado para la colaboración del equipo, la gestión de tareas y la adaptación ágil a medida que hemos enfrentado desafíos y se según se cumplían los objetivos del proyecto.

Para el reparto del trabajo, fundamentalmente lo dividimos en tres grandes bloques, Web Scraping, Base de Datos y Machine Learning. Encargádonos Jaime Muñoz Fisac del Web Scraping, Vicente Sosa Alcolea De la Base de Datos y Aliaksandr lakhvedau lakhvedau Del Machine Learning.

### 3. MARCO TEÓRICO.

#### 3.1. Web Scraping.

##### 3.1.1 Definición.

El Web Scraping es una técnica utilizada para extraer información de páginas web de manera automatizada. Permite acceder y recolectar datos relevantes de manera eficiente, siendo esencial en la extracción de información para este proyecto, donde se obtendrán datos inmobiliarios de la plataforma idealista y elaborar nuestra propia Base de Datos.

##### 3.1.2 Bibliotecas Python.

Para el proyecto se han utilizado las siguientes librerías auxiliares y específicas de web scraping:

- | HTTPx: Es una biblioteca que ofrece una implementación eficiente y asíncrona del protocolo Http. Su uso en conjunto con el Web Scraping permite realizar solicitudes de manera eficaz y manejar múltiples peticiones de forma concurrente, optimizando el proceso de extracción de datos.
- | Beautiful Soup (bs4): Es una biblioteca que facilita la extracción de datos desde documentos HTML y XML. En combinación con HTTPx, permite analizar la estructura del código fuente de las páginas web y extraer información específica de manera sencilla. Su flexibilidad y capacidad para manejar documentos malformados hacen que sea una herramienta poderosa para el Web Scraping en este contexto.

##### 3.1.3 Ética y Términos de Uso.

Es fundamental reconocer y respetar los términos de servicio de la plataforma objetivo y asegurarse de que la extracción de datos cumple con

las leyes y regulaciones aplicables. Se debe evitar el abuso del Web Scraping para preservar la integridad de la información y mantener la ética en la obtención de datos. En este sentido, es importante destacar que el objetivo de esta herramienta es puramente académico, sin buscar en ningún momento ningún tipo de lucro.

### **3.2. Bases de Datos.**

#### **3.2.1 Conceptos Básicos.**

Para el almacenamiento de los datos de las aplicaciones es fundamental incorporar mecanismos que puedan dar persistencia a los datos que los usuarios van registrando y las diferentes acciones que van sucediendo en el uso de la aplicación.

En la actualidad existen diferentes motores de bases de datos, siendo las más usadas del tipo relacionales como no relacionales.

Como ejemplo de estas tenemos para las relacionales SQL y para las no relaciones mongoDB.

Es una tarea muy importante el correcto mantenimiento y diseño de estas bases de datos para el correcto funcionamiento de las aplicaciones.

#### **3.2.2 Elección de la Base de Datos.**

Para la aplicación creada se ha elegido una base datos relacional.

En este caso SQL Lite. Esta base de datos viene integrada con Python y su elección ha sido ya que es sencillo su manejo, es de libre uso y cumple perfectamente para la aplicación.

### **3.3. Machine Learning.**

#### **3.3.1 Conceptos Básicos.**

Objetivo:

Se busca implementar un sistema de predicción de precios de viviendas mediante el uso de algoritmos de aprendizaje automático. Para ello, se lleva a cabo una serie de procesos fundamentales:

Carga de Datos: Se inicia con la carga de datos desde un archivo CSV ('completed\_houses.csv').

Limpieza y Preparación de Datos: Se realiza la identificación y tratamiento de valores nulos, así como la transformación y codificación de variables relevantes.

Entrenamiento del Modelo: Se exploran tres algoritmos de regresión (Regresión Lineal, Regresión Forestal Aleatoria y Regresión de Impulso Gradiente) para determinar el mejor modelo.

Evaluación del Modelo: Se evalúa el rendimiento del modelo seleccionado y se visualizan las predicciones frente a los valores reales.

Guardado del Modelo: El mejor modelo se guarda para su uso futuro.

### 3.3.2 Depuración y Elección de Datos.

En esta sección, se lleva a cabo la depuración de datos y la selección de características esenciales para el modelado.

### 3.3.3 Elección del Algoritmo.

En esta etapa, se detallan los procesos relacionados con la elección y entrenamiento de los algoritmos de regresión, así como la evaluación y almacenamiento del modelo final.

## 3.4. Buenas Prácticas.

### 3.4.1 Control de Errores.

El control de errores es esencial en el desarrollo de software para garantizar la robustez y la estabilidad del sistema. Es, por ello, que, basándonos en las asignaturas vistas durante el Master, tales como Buenas Prácticas de Programación, hemos implementado un manejo adecuado de excepciones y hemos generado archivos de log para el registro detallado de errores, para así facilitar la identificación y corrección de problemas durante el desarrollo y la ejecución de la aplicación.

### 3.4.2 Desarrollo Guiado por Pruebas.

El Desarrollo Guiado por Pruebas es una metodología que enfatiza la escritura de pruebas antes de la implementación del código. La adopción de la metodología contribuye a la detección temprana de errores y facilita el mantenimiento del código a medida que evoluciona el proyecto. La biblioteca elegida para este punto ha sido PYTEST. Es cierto que hay partes del código que no se han tratado mediante Pruebas, y hemos querido focalizar el esfuerzo de las pruebas en partes concretas del código. Es así que hemos

desarrollado pruebas robustas aplicadas a la parte de la Base de Datos y su archivo de gestión y a la parte de Web Scraping.

#### 3.4.3 Documentación.

La documentación adecuada es esencial para comprender, mantener y escalar el proyecto. Cada componente, desde el código fuente hasta la estructura de la base de datos, está debidamente documentado. Esto incluye explicaciones claras de funciones y métodos mediante el uso de comentarios. También hemos realizado un documento README con un resumen de la aplicación y sus funcionalidades. Pero consideramos que este documento en sí es la mejor documentación que hemos podido realizar.

#### 3.4.4 Control de Versiones.

La gestión efectiva del control de versiones es crucial para rastrear y coordinar los cambios en el código fuente y otros elementos del proyecto, además de facilitar el trabajo en equipo. La implementación de un sistema de control de versiones, hemos utilizado GitHub, permite mantener un historial claro de las modificaciones, facilita la colaboración en equipo, y proporciona la posibilidad de retroceder a versiones anteriores en caso de problemas. Un flujo de trabajo de ramificación y fusión bien definido contribuirá a una gestión eficiente del proyecto y a la prevención de conflictos, poniendo así en práctica lo aprendido durante el Master.

### 4. METODOLOGÍA

#### 4.1. Web Scraping.

##### 4.1.1 Detalles de la página idealista.com.

El Portal de Idealista, es un destacado portal inmobiliario donde los usuarios publican sus propiedades inmobiliarias, abarcando desde locales y pisos, hasta garajes y trasteros destinados a la venta, alquiler o compartición. En este trabajo nos enfocamos específicamente en las viviendas ofertadas en venta en la Isla de Gran Canaria. (Figura 1. Anexo I Detalles Idealista.com.).

Además, hay otros usuarios, que pueden acceder a dichas ofertas mediante filtrado para acceder a la compra, alquiler o compartición de dichos inmuebles.

En este contexto, es fundamental comprender la lógica detrás de proceso de Web Scraping, y para ello, iniciamos desglosando y explicando la estructura de la página de idealista.

La exploración comienza, en la página principal [www.idealista.com](http://www.idealista.com), seleccionando Gran Canaria, comprar, viviendas, accedemos a la primera parte de la navegación. En este punto, se presenta un mapa de Gran Canaria, marcando sus distintas zonas, y todas las viviendas ofertadas para la venta. (Figura 2. Anexo I Detalles Idealista.com.).

En la parte superior observamos una secuencia jerárquica:

| *Idealista > Las Palmas provincia > Gran Canaria*

Con un número debajo que indica el número de inmuebles de esa área en concreto ofertados.

Destacar que el formato de la URL de esta página es:

| <https://www.idealista.com/venta-viviendas/las-palmas/gran-canaria/mapa>

En cualquier momento, borrando mapa de esa URL, en lugar del mapa de la zona, nos muestra por pantalla el listado de las casas ofertadas de esa área seleccionada en ese momento.

Ese menú actúa como nuestra guía mientras exploramos todas las subzonas que conforman Gran Canaria. Al desplegar la lista de Gran Canaria, se revelan las diversas zonas, y al continuar explorando cada una, llegamos eventualmente a un punto en el que ya no es posible subdividir la región con mayor detalle. Este proceso nos permite realizar un análisis exhaustivo de las ofertas inmobiliarias disponibles en cada rincón de la isla. (Figura 3. Anexo I Detalles Idealista.com.).

Una vez, hemos alcanzado la última subzona, en lugar del mapa, la página nos muestra el listado de los inmuebles de esa última zona. (Figura 4. Anexo I Detalles Idealista.com.).

Destacar que el formato de la URL de esta página es:

| <https://www.idealista.com/venta-viviendas/las-palmas-de-gran-canaria/ciudad-alta/siete-palmas/>

Una vez llegado al listado, en la parte de abajo, podemos observar las páginas que forman el listado, con los botones correspondientes para navegar página

por página a través de todo el listado de casas. (Figura 5. Anexo I Detalles Idealista.com.).

En este listado, se puede leer información muy básica acerca del inmueble, como el título, la ubicación, el precio y unas pocas características del inmueble. Pero si hacemos click en uno de los inmuebles, accedemos a la información completa y detallada de todas las características. (Figura 6 y 7. Anexo I Detalles Idealista.com.).

Destacar que el formato de la URL de esta página es:

| <https://www.idealista.com/inmueble/103099271/>

Siendo el número final un identificador único para cada inmueble.

Conociendo bien la estructura del portal, y el funcionamiento de los desplegables y sus correspondientes URL nos permite realizar un Web Scraping efectivo para obtener información detallada sobre las propiedades inmobiliarias en Idealista.com.

#### 4.1.2 Extracción de datos relevantes.

Una vez entendida la estructura del portal inmobiliario, procedemos a extraer los datos relevantes para nuestra Base de Datos, siguiendo el flujo de trabajo detallado en el Anexo II Flujo de Trabajo de Web Scraping.

En primer lugar, se extraen todas las subzonas del área que se pida, en este caso, la isla de Gran Canaria. Basándonos en la estructura jerárquica de las zonas, e inspeccionando los elementos de dicha estructura, se puede realizar una función que vaya recorriendo elemento por elemento en busca de la última zona, e ir añadiéndolas a una lista. (Figura 1. Anexo III Extracción de Datos Relevantes.).

Una vez extraídas todas las subzonas de Gran Canaria, avanzamos al siguiente paso: recorrer página por página del listado de casas. Para ello inspeccionamos los botones responsables de cambiar de página, y además nos fijamos en la estructura de la URL:

| <https://www.idealista.com/venta-viviendas/las-palmas-de-gran-canaria/ciudad-alta/siete-palmas/>  
| <https://www.idealista.com/venta-viviendas/las-palmas-de-gran-canaria/ciudad-alta/siete-palmas/pagina-2.htm>  
| <https://www.idealista.com/venta-viviendas/las-palmas-de-gran-canaria/ciudad-alta/siete-palmas/pagina-3.htm>

Y así sucesivamente, con la particularidad de si nos pasamos de la última página, el portal automáticamente revierte a la página 1:

| <https://www.idealista.com/venta-viviendas/las-palmas-de-gran-canaria/ciudad-alta/siete-palmas/pagina-1.htm>

En este punto, exploramos la lista de inmuebles de una página específica, inspeccionando el contenedor principal que alberga el listado de casas. Extraemos los ID y precios de los inmuebles inicialmente, para luego profundizar en cada propiedad (Figura 2, Anexo III Extracción de Datos Relevantes).

Cada página de cada inmueble está formada por varios contenedores que contienen toda la información relevante para la Base de Datos.

Datos como, título, descripción, tipo de inmueble, precio, superficie, número de habitaciones, numero de baños, y si tiene características como jardín, terraza, parcela, trasteros...

Esta información se obtiene mediante una inspección meticolosa de cada elemento, identificando patrones repetitivos que indican las características buscadas.

#### 4.1.3 Código Fuente y Explicación.

El código fuente del back de Web Scraping se encuentra en el directorio scraping de la app scraping\_app de Django.

La estructura del código sigue un poco la misma estrategia detallada hasta el momento a la hora de recorrer las diferentes páginas del portal de Idealista.

Antes de comenzar con el flujo de trabajo en sí de recorrer el portal, explicaremos cómo hemos realizado las solicitudes para obtener el código HTML de las diferentes páginas.

Conexión con Idealista.com (req\_html.py):

1. Librería HTTPX. Como ya mencionamos en apartados anteriores, la librería que hemos utilizado para conectarnos a la web de Idealista ha sido HTTPX. Para ello creamos una clase `Httpx_client` en la que inicializamos un `httpx.client` y al que le añadimos unos header modificados para simular un uso humano, y de esta manera reducir las posibilidades de ser bloqueados.
2. `Get_html_from_url`. Ésta es la función principal que se encuentra dentro de la clase `Httpx_client` y es la encargada de solicitar el código html de la

url que le marquemos. Además ésta función tiene una serie de retardos aleatorios, que simulan igualmente un uso humano. Es la función que utilizaremos para cada parte del flujo de trabajo de web scrapping, para extraer los datos posteriormente con la librería BeautifulSoup BS4. (Figura 1. Anexo IV Código Fuente Web Scraping.).

Conocida ya como se establece la conexión con Idealista, podemos empezar con el flujo de trabajo:

**1.** Obtención de todas las zonas que componen Gran Canaria.

Get\_zones.py. Dentro de éste archivo, encontramos la función `get_zones_from_area`, que se encarga de obtener todas las subzonas que componen el área que se define por una URL, en éste caso la URL del mapa de Gran Canaria:

| <https://www.idealista.com/venta-viviendas/las-palmas/gran-canaria/mapa>

Se trata de una función recursiva, que se va adentrando en cada zona y subzona, buscando el desplegable de la zona que muestra todas las subzonas que la componen, y buscando en el código la lista de subzonas que componen la anterior zona, hasta que la función causa un error y es entonces como detecta que ha llegado a la última zona divisible, añadiéndola a una lista que posteriormente devuelve la función. Todo el código se encuentra detalladamente comentado para su mejor comprensión. (Figura 2. Anexo IV Código Fuente WebScraping.).

**2.** Recorrido de todas las páginas de cada zona.

Single\_zone\_all\_pages\_houses.py. Con la lista de zonas que componen el área de Gran Canaria, ya podemos optar por descargar todas las zonas o bien ir descargando zona por zona, en un caso u otro la función que usaremos será `get_and_save_houses_from_single_zone` que se encuentra dentro del archivo `single_zone_all_pages_houses.py`. La función hace uso de la url recurrente que explicamos anteriormente, siendo la primera parte de la Url común para todas las páginas de esa zona, y la última parte `"/pagina-{numero_pagina}.htm"`.

| <https://www.idealista.com/venta-viviendas/las-palmas-de-gran-canaria/ciudad-alta/siete-palmas/pagina-1.htm>

Teniendo esto presente, y sabiendo que cuando lleguemos a la ultima página, las siguientes nos rediregen a la página 1, utilizamos un bucle while para recorrer todas las páginas.

Ésta función, además, una vez ha descargado todas las páginas de la zona, realiza un doble chequeo de las casas que antes se encontraban en la base de datos y ahora no se han descargado, para ver si ha sido un error, o si bien ya no se encuentran ofertadas, en cuyo caso cambia el estado de la casa a inactivo. (Figura 3. Anexo IV Código Fuente Web Scraping.).

**3.** Obtención de la lista de casas de una sola página.

Single\_page.py .En este archivo podemos encontrar la función que nos descarga la lista, realmente un diccionario con {id:price} de todas las ids de las casas de esa página en concreto. Usando la función get\_housesId\_from\_page buscamos en la sopa creada por beautiful soup de la página, la sección de la clase “items-container” que contiene una lista de artículos, correspondiendo cada artículo a un anuncio de una casa. En cada artículo se encuentra el precio y el id de cada casa, que son los datos que agregamos a nuestro diccionario. (Figura 4. Anexo IV Código Fuente Web Scraping.).

**4.** Descarga de la lista de casas de una sola página.

Single\_page.py . Una vez que tenemos la lista de las id's de la página en concreto, las ordenamos aleatoriamente, para meter más filtros para simular control humano, y procedemos a descargar casa por casa usando la función get\_and\_save\_houses\_from\_page. Con ésta función, recorremos la lista de casas de la página desordenada, de siguiente manera: (Figura 5. Anexo IV Código Fuente Web Scraping.).

**4.1.**Primero buscamos el id y el precio en la base de datos

**4.2.**Si existe la casa con ese mismo precio, simplemente actualizamos la fecha del último check

**4.3.**Si la casa existe, entonces descargaremos los datos de la casa.

**5.** Descarga de las características de una casa.

Single\_house.py . La función que se encuentra en éste script, llamada parser\_house\_id es la encargada de extraer todos los datos característicos de una casa a través de la sopa extraída del html de la página mediante bs4. La función en sí ha sido creada estudiando meticulosamente el patrón del html usado en cada casa, que se repite, y haciendo uso del inspector de la página. Ésta función devuelve un diccionario con todos los datos de la casa, y en caso de error, un diccionario con el id de la casa, el precio que sí lo teníamos, la fecha y un estado de error. El diccionario seguidamente es pasado por la función

parser\_and\_save\_house\_id que se encarga de crear un objeto casa y guardarla en la base de datos usando la función que gestiona la Base de Datos. (Figura 6. Anexo IV Código Fuente Web Scraping.).

#### 4.1.4 Problemas encontrados y soluciones.

Para seguir la misma estructura del punto anterior, trataremos los problemas encontrados en cada punto del proceso:

| Conexión con Idealista.com (req\_html.py):

El principal problema con la conexión fue el bloqueo de nuestra IP cuando la página detectaba muchas solicitudes seguidas o un uso constante. Para ello en un principio, ideamos retardos aleatorios en las solicitudes.

Además, también utilizamos en un primer momento otras librerías que nos dieron más problemas, eligiendo finalmente HTTPX, y adicionalmente añadimos en las solicitudes unos headers personalizados para cada solicitud, en la que además añadíamos de qué página veníamos, dando muy buen resultado.

Pensamos en añadir proxies, pero la falta de un listado fiable de proxies, desechó esta opción.

| Obtención de todas las zonas que componen Gran Canaria:

La principal problemática en éste punto fue como detectar que habíamos llegado a la última zona explorable, ya que hay ciertas zonas aleatorias que no seguían un patrón fijo, con lo que se obtó por la recursividad y la detección del error a la hora de explorar las subzonas, momento en el que se paraba de explorar ese área.

| Recorrido de todas las páginas de cada zona:

No se encontraron grandes problemas

Obtención de la lista de casas de una sola página:

No se encontraron grandes problemas

| Descarga de la lista de casas de una sola página:

En este dentro de la lista de casas de la página había uno o dos anuncios metidos no de casas particulares, sino anuncios publicitarios incrustados, que dificultaban la búsqueda, se resolvió analizando el patrón de la lista, y viendo como únicamente los anuncios de casas tenían la propiedad del id.

| Descarga de las características de una casa:

La claridad y limpieza de ésta parte del programa es la pero sin duda, y tras muchas vueltas que le hemos dado, no hemos podido simplificar más la función, ya que es una búsqueda de texto dentro de un texto, y muy largo, además siendo cada característica diferente, en formato, de la anterior en la mayoría de los casos.

## 4.2. Base de Datos SQL.

### 4.2.1 Diseño de la estructura de la Base de Datos.

Para la realización de el diseño de la base de datos se ha tomado como entidades las viviendas y los precios de estas viviendas.

Se ha usado la metodología de creación de modelos de SQLAlchemy en los cuales se han incorporado los diferentes atributos de cada modelo como los diferentes métodos necesarios para las acciones y modificaciones de los objetos de dichos modelos.

Dichos modelos usan la base de programación mediante clases.

Ambas entidades se han relacionado de uno a muchos. Una vivienda puede tener muchos precios, debido a que en el caso real de la aplicación los precios de las viviendas pueden ir variando con el tiempo.

### 4.2.2 Proceso de inserción de datos de web scraping en la Base de Datos.

La inserción de las viviendas como los precios de las bases de datos se han realizado creando un módulo específico para ello. En el cual están las diferentes funciones relacionadas con la inserción, modificación, borrado, como cualquier cambio que la aplicación necesite realizar entre las entidades y las bases de datos.

El ORM de SQLAlchemy proporciona una forma de trabajar con la base de datos mediante el uso de objetos, no teniendo que realizar consultas en lenguaje propio de SQL facilitando la inserción como resto de acciones.

Actualmente existen diferentes formas con el ORM de crear las consultas.

Para la realización de esta aplicación se ha usado mayoritariamente el objeto de Query del motor de SQLAlchemy.

Con este objeto podemos realizar las consultas a la base de datos con el uso de métodos encadenados.

Durante la inserción de la base de datos la aplicación va realizando comprobaciones de su correcta inserción o en su caso actualización, marcando los errores que en el proceso pudieran surgir y capturando los errores con mecanismos como los try except para poder manejarlos.

<https://www.sqlalchemy.org/>

#### 4.2.3 Consideraciones de seguridad y escalabilidad.

La elección de uso de ORM frente al uso de consultas SQL nativas se ha decidido debido a que estos ORM ya proporcionan en sí una capa de seguridad, protegiendo de ataques por ejemplo de inyección a la aplicación.

También SQLAlchemy nos da herramientas para la validación de los datos antes de realizar la inserción en la base de datos.

Al usar este ORM tenemos una estructura fija y ordenada de cómo se deben tratar y crear cada entidad de la aplicación, proporcionando orden y fácil lectura del código.

#### 4.2.4 Problemas encontrados y soluciones.

Debido a que el ORM está realizando cambios en la forma de hacer las consultas hemos tenido problemas en determinar cuál es la mejor forma de realizar dichas consultas.

También al estar usando el Framework de Django el cual tiene su propio motor ORM, hemos tenido algunos problemas de configuración del fichero de settings y habían ciertas cosas que no se hacen de el mismo modo usando SQLAlchemy que el ORM de Django. Hay ciertas funciones que hubieran estado más integradas usando el propio ORM del framework, pero se ha preferido usar SQLAlchemy debido a que se puede usar en cualquier ambiente independientemente del framework.

En cualquier caso ambos ORM son muy robustos y hay bastante documentación en la que consultar

### 4.3. Machine Learning Aplicado.

#### 4.3.1 Preparación de los Datos

Objetivo:

Esta sección se enfoca en la preparación de los datos antes de la construcción del modelo de predicción de precios de viviendas.

### Pasos Relevantes:

- Carga de Datos:
  - Se utiliza la biblioteca Pandas para cargar los datos desde el archivo 'completed\_houses.csv'.
  - Se realiza una visualización inicial de los primeros registros y la información general del conjunto de datos.
- Limpieza de Datos:
  - Se identifican y tratan los valores nulos mediante visualizaciones y estrategias de imputación.
  - Columnas irrelevantes ('last\_active\_check\_date') se eliminan para facilitar el análisis.
  - Las fechas se convierten a un formato numérico para su uso en el modelo.
- Tratamiento de Columnas de Cadena:
  - Se eliminan caracteres especiales de las columnas de ubicación.
  - Las columnas categóricas se codifican utilizando one-hot encoding.

#### 4.3.2 Elección de Características

- Eliminación de Columnas:
  - Columnas no relevantes para la predicción de precios ('title', 'location\_4', 'zone\_url') se eliminan.
- One-Hot Encoding:
  - Se utiliza one-hot encoding para convertir las variables categóricas de ubicación en columnas numéricas.

#### 4.3.3 Modelo de Predicción

- Modelos Considerados:
  - **Regresión Lineal:** Implementación del modelo de regresión lineal con diferentes parámetros.
  - **Random Forest Regression:** Utilización de Random Forest Regression con ajuste de hiperparámetros.
  - **Gradient Boosting Regression:** Implementación de Gradient Boosting Regression con selección de parámetros.

#### 4.3.4 Evaluación del Modelo

- Selección del Mejor Modelo:
  - Utilización de GridSearchCV para evaluar y seleccionar el modelo con mejor rendimiento.

- Gráficos de Predicción:
  - Visualización de precios reales frente a precios predichos utilizando un gráfico de dispersión.
- Almacenamiento de Predicciones:
  - Se añade una columna con las predicciones al conjunto de datos original y se guarda el nuevo conjunto de datos.

#### 4.3.5 Problemas Encontrados y Soluciones

Desafíos Superados:

- Procesamiento de Fechas:
  - Dificultad en la conversión de fechas a datos numéricos.
  - Solución: Se utilizó la función `pd.to_numeric(pd.to_datetime(...).dt.strftime('%Y%m'))` para lograr la conversión adecuada.
- Selección de Características:
  - Identificación y elección de características relevantes.
  - Solución: Se empleó one-hot encoding y análisis visual para decidir qué columnas conservar y cuáles eliminar.
- Modelo con Mejor Rendimiento:
  - Elección del modelo con el mejor rendimiento.
  - Solución: Se aplicó GridSearchCV para evaluar varios modelos y seleccionar el más adecuado.

Conclusión de la Sección 4.3:

La sección de desarrollo y entrenamiento del modelo aborda la preparación y selección de datos, la elección de características, la implementación y evaluación de modelos, y la solución de problemas específicos encontrados durante el proceso. Este enfoque integral proporciona una base sólida para la construcción de un modelo preciso de predicción de precios de viviendas.

### 4.4. Desarrollo de la Aplicación Web.

#### 4.4.1 Selección del framework.

La elección de Django como framework para el desarrollo de la aplicación web se fundamenta en su robustez, eficiencia y facilidad de uso. Django, un framework de alto nivel para Python, ofrece una estructura organizativa sólida que agiliza el desarrollo y facilita el mantenimiento del código.

#### 4.4.2 Diseño del interfaz de usuario Web Scraping.

La interfaz gráfica diseñada para la parte de web scraping se caracteriza por su simplicidad y funcionalidad. En la primera pantalla, el usuario se enfrenta a un desplegable que le permite elegir entre dos opciones: obtener todas las zonas del área de Gran Canaria partiendo de los datos que ya disponemos en la base de datos o realizar el scraping de nuevo de todas las zonas.

En la segunda pantalla, se presenta una lista completa de todas las zonas disponibles, ya sean de la base de datos o de haber realizado el scraping de nuevo. Cada fila de la lista, representa una zona del área de Gran Canaria, así como el número de viviendas que hay en esa zona. Cuenta, además, con un botón para descargar individualmente esa zona en particular. Para descargar todas las zonas, el interfaz cuenta con un botón que cumple con esa función.

Esta estructura intuitiva y eficiente facilita al usuario una experiencia de navegación clara y directa, permitiendo la personalización del proceso de descarga según sus preferencias.

#### 4.4.3 Diseño del interfaz de usuario Base de Datos.

Para la inserción, como la visualización de los datos de la aplicación se ha usado el moto de plantillas que incorpora Django .

Dicho motor de plantillas que hemos usado a sido Twig

Este es un motor muy potente que facilita la generación dinámica de contenido de HTML.

El patrón de diseño que hemos usado a sido el modelo vista controlador.

En Django las vistas se generan desde los módulos de view de cada aplicación que renderizan los ficheros HTML.

A estos ficheros HTML es bastante sencillo el pasarles datos desde el fichero view usando sintaxis de {{}} para pasarle las variables

Estas plantillas nos han permitido también realizar usos de bucles y filtros para la darle el dinamismo a la aplicación.

{% for item in ítems % } {%end for %} es un ejemplo para poder realizar un bucle dentro de dichas plantillas.

También hemos usado archivos javascript asociados a dichos ficheros para la realización de carga de datos en las listas para evitar las recargas de las páginas y que sea más fluida la navegación.

#### 4.4.4 Diseño del interfaz de usuario Machine Learning.

Objetivo:

En esta sección, se describe el desarrollo de una aplicación web para predecir precios de viviendas mediante técnicas de aprendizaje automático. Además, se detalla el diseño del interfaz de usuario relacionado con el uso del modelo de machine learning.

Conclusión:

El desarrollo de la aplicación web incorpora un proceso completo, desde la conexión a la base de datos hasta la actualización de la misma con las predicciones del modelo. El diseño del interfaz de usuario facilita la interacción de los usuarios con el modelo (Random Forest Regression) de machine learning para obtener estimaciones de precios de viviendas de manera eficiente.

## 5. EVALUACIÓN DE LOS RESULTADOS

### 5.1. Web Scraping.

En el siguiente enlace se presenta una demo de la aplicación de web Scraping.

| <https://youtu.be/mD7hO3pCSUY>

### 5.2. Base de Datos.

En el siguiente enlace se presenta una demo de la aplicación de la Base de Datos.

| <https://youtu.be/iH5zVPeOrNA>

### 5.1. Machine Learning.

La evaluación de resultados se realiza mediante métricas clave como MSE, analizando la precisión del modelo de predicción de precios de viviendas. Se busca optimizar el rendimiento para garantizar una aplicación web confiable y precisa.

## 6. POSIBLES MEJORAS Y TRABAJOS FUTUROS.

### 6.1. Web Scraping.

Dados los buenos resultados, se deja para el futuro la implementación de las siguientes funcionalidades:

- | Incorporación de Proxies: Como mejora futura, se contempla la integración de proxies en el proceso de web scraping. Esto no solo agilizará la velocidad de descarga, sino que también contribuirá a la eficiencia y la evasión de posibles restricciones por parte de la plataforma objetivo.
- | Descarga de datos de otras plataformas: Se proyecta la expansión del alcance de la aplicación para incluir la descarga de datos desde otras plataformas relevantes, como Fotocasa. Esto permitirá enriquecer aún más la base de datos, proporcionando una visión más completa del mercado inmobiliario.
- | Descarga de datos de otro tipo de inmuebles: Como parte de las mejoras previstas, se considera la posibilidad de ampliar la variedad de inmuebles descargados. Además de viviendas, se contempla la inclusión de locales, garajes y trasteros, ofreciendo así una panorámica más integral y diversificada de las propiedades disponibles.
- | Descarga de los inmuebles en alquiler: Se plantea la expansión de la funcionalidad actual para abarcar la descarga de datos específicamente para inmuebles en alquiler. Esta ampliación permitirá capturar de manera exhaustiva tanto las propiedades en venta como aquellas disponibles para alquiler, brindando una cobertura completa de opciones inmobiliarias.

### 6.2. Base de Datos.

Como mejoras en respecto a la base de datos, se puede plantear la migración de esta a una base de datos de PostgreSQL ya que es mucho más potente que la base de datos de SQLite .

Así como el uso de contenedores DOCKER para la ubicación de dicha base de datos.

### 6.3. Machine Learning.

Se plantea la evaluación continua del modelo de Machine Learning, la incorporación de más características relevantes en la predicción, y la expansión de la aplicación a otras regiones.

Utilizando los datos de las viviendas (longitud y latitud) que se podría conectar a Google maps y evaluar la infraestructura de la zone donde se predice el precio.

## 7. CONCLUSIONES

Éste trabajo de fin de Master resuelve la problemática de estimar el valor de mercado de los inmuebles de la Isla de Gran Canaria.

Se ha destacado la importancia de respetar los términos de servicio de la plataforma Idealista.com y mantener prácticas éticas en el Web Scraping. La aplicación se ha desarrollado con fines académicos, evitando el abuso y preservando la integridad de la información.

La metodología ágil, específicamente la adaptación de Scrum, ha sido efectiva en la coordinación y ejecución de un trabajo en equipo. El reparto de tareas entre los miembros del equipo, centrando en Web Scraping, Base de Datos y Machine Learning, ha permitido una gestión eficiente y un progreso constante hacia los objetivos marcados.

Mediante la aplicación de técnicas de Web Scraping, se ha logrado recopilar información detallada de la plataforma Idealista, permitiendo la creación de una Base de Datos robusta y actualizada. Los desafíos encontrados durante el desarrollo, como el bloqueo de la IP y la complejidad en la extracción de características de las casas, fueron abordados con soluciones efectivas.

En la fase inicial, se consideró la implementación de una base de datos PostgreSQL, sin embargo, se optó por una solución más sencilla utilizando una base de datos SQL. La elección de la librería SQLAlchemy facilitó la interacción y gestión eficiente de la base de datos, proporcionando una solución práctica y efectiva.

La implementación del algoritmo de Machine Learning ha proporcionado un modelo de predicción que, si bien requiere una evaluación más detallada, representa un avance significativo en la estimación precisa del precio de los inmuebles en el mercado inmobiliario de Gran Canaria.

En cuanto al desarrollo de la aplicación, se ha optado por el framework Django, proporcionando una interfaz de usuario intuitiva y funcional. El módulo de Web Scraping permite la descarga y actualización de datos de manera personalizada, mientras que las secciones de Base de Datos y Machine Learning ofrecen herramientas visualizar los datos obtenidos, así como para el análisis y la predicción del valor de mercado de los inmuebles.

Como posibles mejoras y trabajos futuros, se plantea diferentes ampliaciones en la descarga de datos, ampliando la Base de Datos y dando rapidez en la descarga, además, la evaluación continua del modelo de Machine Learning, la incorporación de más características relevantes en la predicción, y la expansión de la aplicación a otras regiones.

En conclusión, este proyecto ha proporcionado una solución completa para abordar los desafíos en la estimación del valor de mercado de los inmuebles, integrando técnicas de Web Scraping, Machine Learning y una metodología ágil. Las lecciones aprendidas y las conclusiones obtenidas sirven como base para futuras mejoras y proyectos relacionados con el análisis y predicción en el mercado inmobiliario.

## 8. REFERENCIAS

<https://scrapfly.io/blog/how-to-scrape-idealista/>

<https://www.youtube.com/watch?v=1gQyRulpqUU>

<https://www.youtube.com/watch?v=JkwfzexrQS0>

<https://www.youtube.com/watch?v=2UyJv5oe570>

<https://www.sqlalchemy.org/>

<https://www.djangoproject.com/>

<https://docs.python.org/>

<https://www.kaggle.com/>

<https://stackoverflow.com/>

<https://docs.djangoproject.com/en/5.0/>

## 9. LISTA DE ANEXOS

ANEXO I. Detalles de Idealista.com.

ANEXO II. Flujo de trabajo de Web Scraping.

ANEXO III. Extracción de datos relevantes.

ANEXO IV. Código Fuente Web Scraping.

ANEXO V. Interfaz Gráfica de Web Scraping.

ANEXO VI. Interfaz Gráfica inserción y vista datos aplicación

## ANEXO I

Detalles Idealista.com

Figura 1

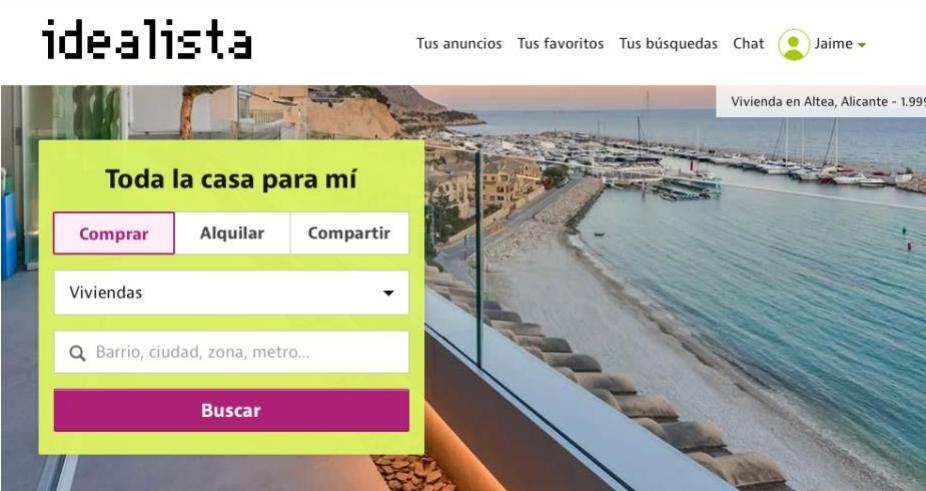


Figura 2



Figura 3

**idealista**

Tus anuncios Tus favoritos Tus búsquedas Chat Jaime ▾

idealista > Las Palmas provincia > Gran Canaria > Las Palmas de Gran Canaria > Carretera del Centro - Cono sur ▾

9.043 5.982 2.040 199

### Mapa de Carretera del Centro - Cono sur, Las Palmas de Gran Canaria: anuncios de viviendas en venta

[Ver las 199 viviendas](#)

[Dibujar tu propia zona](#)

**Zonas en Carretera del Centro - Cono sur**

- San Juan-San José 55
- Carretera del Centro 48
- San Juan de Dios-Salto del Negro-Playa de la Laja 48
- Marzagán-Los Hoyos-La Montañeta 35
- Vega de San José 13

Mapa que muestra la ubicación de las zonas de viviendas en venta en el Cono sur de Las Palmas de Gran Canaria. Se detallan las principales vías y localidades: GC-3, GC-4, Tafira, La Matula, P. San José, Vega de San José, San José Artesano, Ciudad San Juan de Dios, San Cristóbal, Hoya de la Plata, Salto del Negro, La Montañeta, La Data, Los Hoyos, Santa Margarita, Marzagán, Mercasipalmas, Playa de la Laja, entre otras.

[Ver las 199 viviendas](#)

Figura 4

**idealista**

Tus anuncios Tus favoritos Tus búsquedas Chat Jaime ▾

idealista > Las Palmas provincia > Gran Canaria > Las Palmas de Gran Canaria > Ciudad Alta > Siete Palmas ▾

9.043 5.982 2.040 430 127

### 127 casas y pisos en Siete Palmas, Las Palmas de Gran Canaria

Nuevos anuncios en tu email

[Guardar búsqueda](#)

Comprar Alquilar Obra nueva

[Listado](#) [Mapa](#)

Ordenar: Relevancia Baratos Recientes Más ▾

**Piso en calle Harald Flick, 2, Siete Palmas, Las...**

RE/MAX INSIGNIA

Top

239.000 € Garaje incluido  
2 hab. 103 m<sup>2</sup> Planta 1<sup>a</sup> exterior con ascensor  
¡No pierdas la oportunidad de vivir en este magnífico dúplex en la mejor zona de 7...

[Ver teléfono](#) [Contactar](#) [Mapa](#)

**Piso en calle Pedro Barber Jorro, 15, Siete...**

RE/MAX ATLANTIS

Top

148.000 € Garaje incluido  
3 hab. 67 m<sup>2</sup> Planta 2<sup>a</sup> exterior sin ascensor

Figura 5

Nuevos anuncios en tu email  
Guardar búsqueda

1/14 Ver teléfono Contactar

**Casa o chalet independiente en Amapola, 5, Siete Palma...**

**575.000 €**  
9 hab. 233 m<sup>2</sup>

AM Propiedades VENDE edificio de tres plantas en Las Palmas de Gran Canaria zona Las Torres a mu...

Ver teléfono Contactar

Precio medio 2.544 eur/m<sup>2</sup>

Cómo ordenamos los resultados

Ver más resultados: 1 2 3 4 5 Siguiente >

Σ

idealista

Tus anuncios Tus favoritos Tus búsquedas Chat Jaime ▾

« Viviendas en Siete Palmas 1 de 127 viviendas Siguiente >

**Piso en venta en calle Harald Flick, 2**  
Siete Palmas, Las Palmas de Gran Canaria Ver mapa

**239.000 €**

Calcular hipoteca Estudiar hipoteca

103 m<sup>2</sup> | 2 hab. | Planta 1º exterior con ascensor | Garaje incluido

Guardar favorito Descartar Compartir

Escribe una nota personal (sólo tú podrás verla)

**Pregunta al anunciante**

Hola, me interesa este piso y me gustaría hacer una visita. Un saludo

**Contactar por chat**

**Ver teléfono**  
Referencia del anuncio  
3478-03063

César Alvarez Sanz

**RE/MAX INSIGNIA**   
Las Palmas de Gran Canaria

Figura 7

**Piso en venta en calle Harald Flick, 2**

Siete Palmas, Las Palmas de Gran Canaria [Ver mapa](#)

**239.000 €**

[Calcular hipoteca](#) [Estudiar hipoteca](#)

103 m<sup>2</sup> | 2 hab. | Planta 1º exterior con ascensor | Garaje incluido

[Guardar favorito](#) [Descartar](#) [Compartir](#)

Escribe una nota personal (sólo tú podrás verla)

**Comentario del anunciante**

Disponible en: Español | English | Otros idiomas ▾

¡No pierdas la oportunidad de vivir en este magnífico dúplex en la mejor zona de 7 Palmas! Con una ubicación enviable frente al supermercado Mercadona y en la misma calle que el concesionario BMW, este inmueble te ofrece todas las comodidades que necesitas. Además, está cerca de RTVE y Correos, en una calle de un solo sentido con poco tráfico y con todos los servicios a tu alcance.

Este dúplex de 103 m<sup>2</sup> construidos y 86 m<sup>2</sup> útiles cuenta con dos habitaciones amplias en la planta alta, ambas con armarios empotrados y varios armarios para almacenamiento. En la planta baja encontrarás un amplio y luminoso salón comedor, una cocina completa con electrodomésticos y tres armarios despensas amplios, así como un baño reformado con plato de ducha de silestone y mamparas de lujo.

La vivienda es toda exterior, por lo que disfrutarás de una luminosidad excelente en todas las estancias. Además, el precio incluye una amplia plaza de garaje en segundo sótano con acceso directo desde el ascensor, para que puedas aparcar tu coche con comodidad y seguridad.

[Leer comentario completo](#)

Si tienes alguna duda recuerda que puedes hablar con **César** por chat.

**Características básicas**

- 103 m<sup>2</sup> construidos, 86 m<sup>2</sup> útiles
- 2 habitaciones
- 2 baños
- Plaza de garaje incluida en el precio
- Segunda mano/buen estado
- Armarios empotrados

**Edificio**

- Planta 1º exterior
- Con ascensor

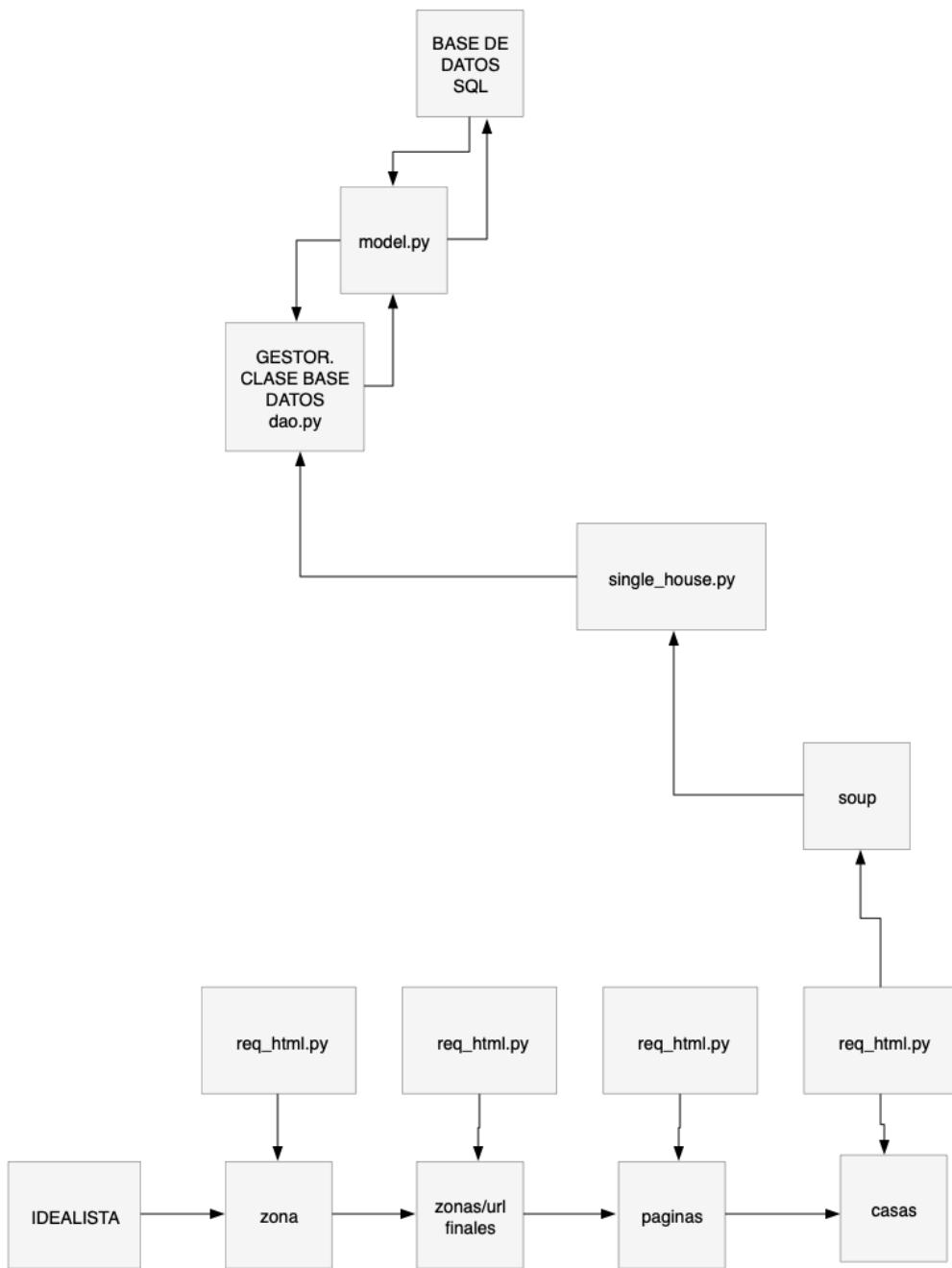
**Certificado energético**

- Consumo: G
- Emisiones: G

[Ver etiqueta de calificación energética ▾](#)

## ANEXO II

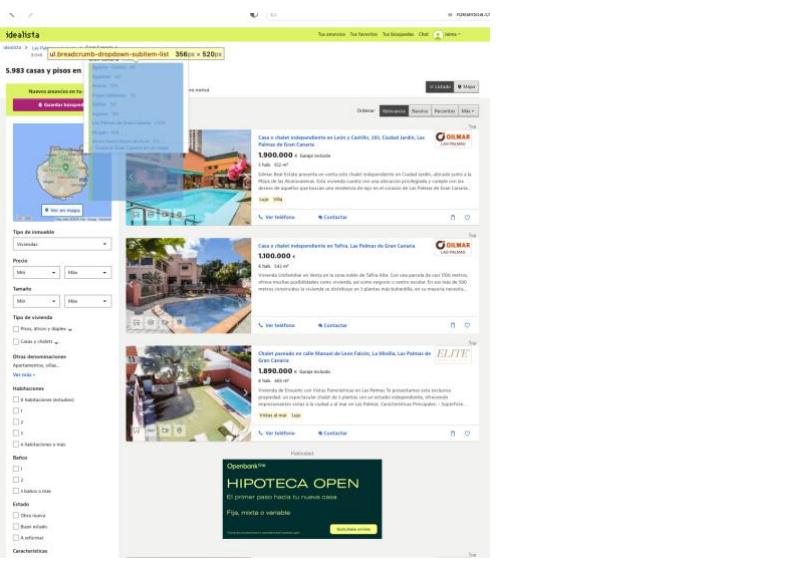
### Flujo de Trabajo de Web Scrapping



## ANEXO III

### Extracción de datos relevantes

Figura 1



The screenshot shows a real estate search results page for "5.983 casas y pisos en Gran Canaria". The results are displayed in a grid format, each listing a property's name, price, and a small image. A sidebar on the left contains various filters and a financing offer from Openbank.

```
<!DOCTYPE html>
<html lang="es" username="Jaime" data-userauth="true" class="open-breadcrumb-dropdown">
  <head>...
    <body>
      ...
      <div id="didomi-host" data-nosnippet="true" aria-hidden="true"></div>
      ...
      <div id="wrapper" class="open-breadcrumb-dropdown">
        ...
        <div class="content extended-grid">
          ...
          <div class="listing-top">
            ...
            <ul class="breadcrumb-navigation" itemscope itemtype="http://schema.org/BreadcrumbList">
              ...
              <li class="breadcrumb-navigation-element" itemscope itemprop="itemListElement" itemtype="http://schema.org/ListItem">
                ...
                <li class="breadcrumb-navigation-element" itemscope itemprop="itemListElement" itemtype="http://schema.org/ListItem" data-location-id="0-EU-ES-35-01-010">
                  ...
                  <a href="/venta-viviendas/las-palmas/gran-canaria/agarte-centro/">Agarte - Centro</a>
                  ...
                </li>
              </ul>
            ...
          </div>
        ...
      </div>
    </body>
  </html>
```

The code snippet shows the breadcrumb navigation structure, including the current location "5.983" and the path leading to it, such as "0-EU-ES-35-01-010" for Agarte - Centro.

Figura 2

The screenshot shows a search results page for "127 casas y pisos en Siete Palmas, Las Palmas de Gran Canaria". The interface includes a sidebar with filters for location, price, size, type, and features. The main area displays several property cards, each with a thumbnail, address, price, and a brief description. At the bottom, there is an advertisement for "CENTURY 21 GILMAR" featuring a building and the slogan "Disfruta, respira, vive!". Below the advertisement, there is a developer's contact information.

```

<html lang="es" env="es" username="Jaime" data-userauth="true" class="Desplazar">
    <head>...</head>
    <body class=" " id="didomi-host" data-nosnippet="true" aria-hidden="true">...</div>
    <div id="didomi-host" data-nosnippet="true" aria-hidden="true">...</div>
    <div id="main" data-islogged="true">...
        <div class="listing-top">...</div>
        <div class="header-bottom">...</div>
        <div id="fixed-toolbar" class="fixed-toolbar">...</div>
        <div id="aside-fixed-toolbar" class="aside-fixed-toolbar">...</div>
        <div id="listing-filter" class="segmented-btn-group list-map">...</div>
        <div id="order-by" class="order-by" data-param="ordenado-por">...</div>
        <main class="listing-items core-vitals-listing-map" id="main-content">...
            <section class="items-container items-list">...
                <article class="item item_contains_branding item_hightop extended-item item-multimedia-container" data-adid="103099271" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_hightop extended-item item-multimedia-container" data-adid="103552971" data-online-booking="false">...</article>
                ...
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="99476210" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="102097749" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="101651102" data-online-booking="false">...</article>
                ...
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="101662518" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="103015947" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="101662510" data-online-booking="false">...</article>
                ...
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="103137784" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="103182416" data-online-booking="false">...</article>
                <article class="item item_contains_branding item_preference-featured extended-item item-multimedia-container" data-adid="102613512" data-online-booking="false">...</article>
                ...
            </section>
        </main>
    </div>

```

Figura 3

The screenshot shows a property listing for a flat in Madrid. The main content includes:

- Title:** Piso en venta en calle Harald Flick, 2
- Price:** 239.000 €
- Features:** 103 m<sup>2</sup>, 2 habitaciones, 1 baño, Planta P exterior con ascensor, Garaje incluido.
- Description:** Este apartamento de 103 m<sup>2</sup> construidos y 86 m<sup>2</sup> útiles cuenta con dos habitaciones amplias en la planta alta, ambas con armarios empotrados y vestidor accesible para almacenamiento. En la planta baja encontramos un recibidor y un salón-comedor con chimenea que conecta directamente con la cocina abierta. La cocina dispone de electrodomésticos y tiene armarios desplazables amplios. Asimismo, el baño es muy luminoso y cuenta con plato de ducha de hidromasaje y manoplas de baño.
- Comments:** Includes sections for comments and a contact form.
- Characteristics:** Includes basic characteristics and energy certificate information.

The bottom part of the screenshot displays the detailed DOM structure of the page, showing the HTML code for the main content area, including sections like `<div>`, `<script>`, and various class names such as `main-info_title`, `main-info_title-block`, `detail-pagination`, and `details-box`.

## ANEXO IV

### Código Fuente Web Scrapping.

Figura 1

```

EXPLORADOR
EDTORES ABIERTOS
req_html.py ②
TFM
req_html.py ③
class Httpx_Client():
    """
    Clase para gestionar la sesión httpx para peticiones del html de la página web
    """

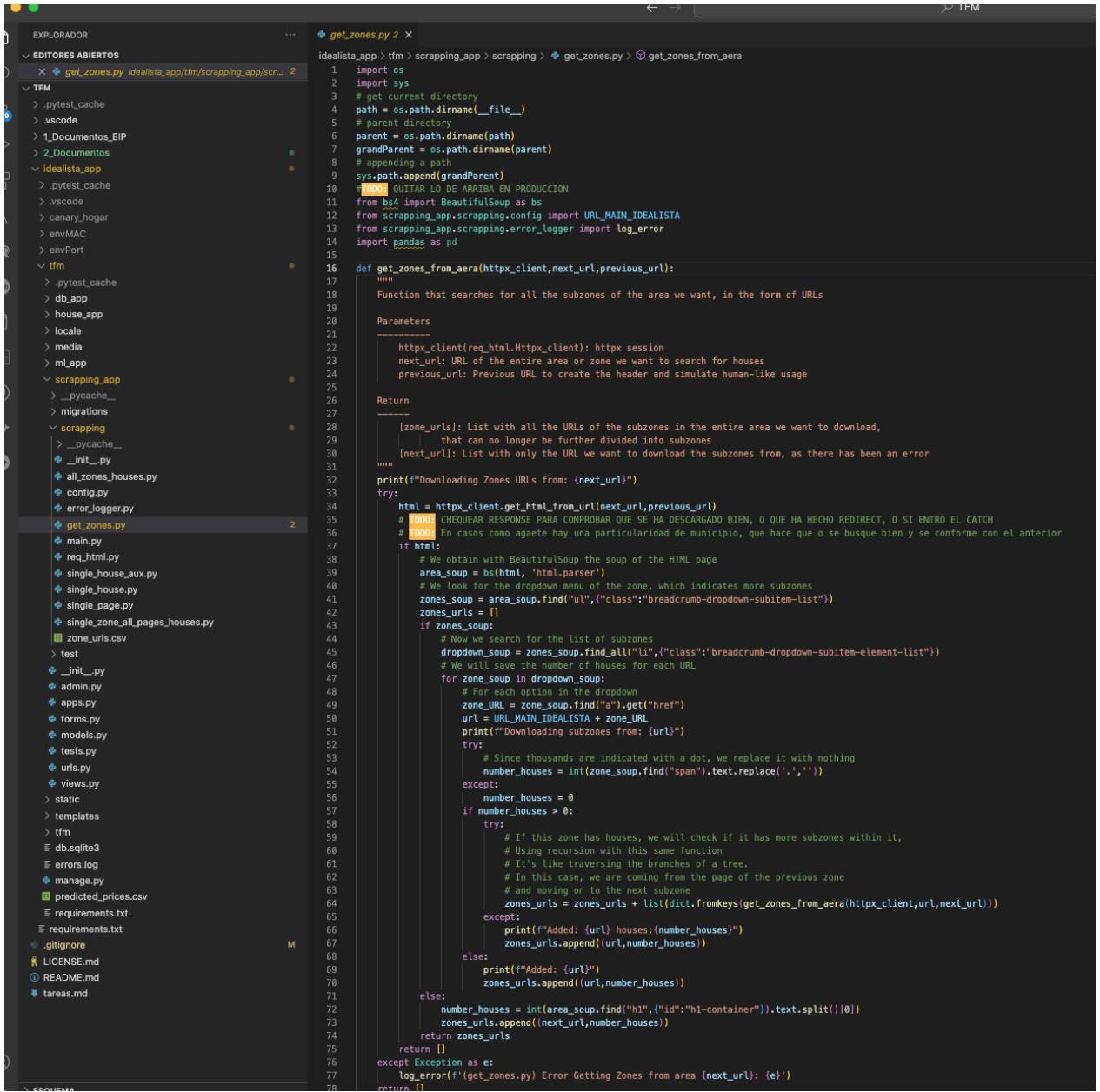
    def __init__(self,page):
        """
        Constructor que crea un Cliente/Sesión httpx
        """
        self.page = str(informativo de la página , que no servirá al crear los header de las peticiones
# comencemos viñetando de google
self.previous_url = URL_GOOGLE
try:
    self.session = httpx.Client(headers=self.generate_header(self.previous_url), follow_redirects=True)
except Exception as e:
    log_error(f'(req_html.py) Error init HTTPX Client: {e}')
    self.session = None

def generate_header(self,newreferer_url):
    """
    Función que nos genera unos headers para evitar problemas de catch de la página
    """
    newreferer_url: URL de la página que simulamos que venimos
    sec_fetch_site = 'same-origin' if self.page in newreferer_url else 'cross-site'
    header = {
        'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7',
        'accept-encoding': 'gzip, deflate, br',
        'accept-language': 'es-ES;es;q=0.9,en-US;q=0.8,en;q=0.7',
        'cache-control': 'no-cache',
        'pragma': 'no-cache',
        'referer': newreferer_url,
        'sec-ch-ua': "Chromium";v="110", "Not A(Brand";v="24", "Google Chrome";v="110",
        'sec-ch-ua-mobile': '?0',
        'sec-ch-ua-platform': "macOS",
        'sec-fetch-dest': 'document',
        'sec-fetch-mode': 'navigate',
        'sec-fetch-site': sec_fetch_site,
        'sec-fetch-user': '?1',
        'upgrade-insecure-requests': '1',
        'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/110.0.0.0 Safari/537.36'
    }
    return header

def get_html_from_url(self,url,previous_url):
    """
    Función que nos realiza la petición de la url
    """
    url: URL de la página que queremos solicitar
    previous_url: URL de la página que simulamos que venimos para simular
    uso humano
    Return
    html: Texto html de la petición de la URL
    """
    try:
        # [000] Meter un header diferente en cada request si metemos proxies
        header = self.generate_header(previous_url)
        r = round(random.uniform(1,7.0),2)
        sleep(r)
        # [000] Meter un proxie en cada request y quitar los tiempos de espera
        req = self.session.get(url,headers=header)
        # [000] CHEQUEAR RESPONSE PARA COMPROBAR QUE SE HA DESCARGADO BIEN, O QUE HA HECHO REDIRECT, O SI ENTRÓ EL CATCH
        html = req.text
    except Exception as e:
        log_error(f'(req_html.py) Error getting HTML from URL {url}: {e}')
        html = None
    return html

```

Figura 2



```

EXPLORADOR
EDItoRES ABIERTOS
  x get_zones.py idealista_app/tfm/scraping_app/src... 2
  ✓ TFM
    > .pytest_cache
    > vscode
    > 1.Documentos_EIP
    > 2.Documentos
    ✓ idealista_app
      > .pytest_cache
      > vscode
      > canary_hogar
      > envMAC
      > envPort
    ✓ tfm
      > .pytest_cache
      > db_app
      > house_app
      > locale
      > media
      > ml_app
    ✓ scraping_app
      > __pycache__
      > migrations
    ✓ scraping
      > __pycache__
      & __init__.py
      & all_zones_houses.py
      & config.py
      & error_logger.py
      & get_zones.py 2
      & main.py
      & req_html.py
      & single_house_aux.py
      & single_house.py
      & single_page.py
      & single_zone_all_pages_houses.py
      & zone_urls.csv
    > test
    & __init__.py
    & admin.py
    & apps.py
    & forms.py
    & models.py
    & tests.py
    & urls.py
    & views.py
  > static
  > templates
  > tfm
  & db.sqlite3
  & errors.log
  & manage.py
  & predicted_prices.csv
  & requirements.txt
  & requirements.txt
  & .gitignore
  & LICENSE.md
  & README.md
  & tareas.md
  ✓ ESTRUCTURA

  ⚡ get_zones.py 2 x
idealista_app > tfm > scraping_app > scraping > get_zones.py > get_zones_from_aera
1 import os
2 import sys
3 # get current directory
4 path = os.path.dirname(__file__)
5 # parent directory
6 parent = os.path.dirname(path)
7 grandParent = os.path.dirname(parent)
8 # appending a path
9 sys.path.append(grandParent)
10 # TODO: QUITAR LO DE ARRIBA EN PRODUCCION
11 from bs4 import BeautifulSoup as bs
12 from scraping_app.scraping.config import URL_MAIN_IDEALISTA
13 from scraping_app.scraping.error_logger import log_error
14 import pandas as pd
15
16 def get_zones_from_aera(httpx_client,next_url,previous_url):
17     """
18         Function that searches for all the subzones of the area we want, in the form of URLs
19
20     Parameters
21
22         httpx_client(req_html.Httpx_client): httpx session
23         next_url: URL of the entire area or zone we want to search for houses
24         previous_url: Previous URL to create the header and simulate human-like usage
25
26     Return
27
28         [zone_urls]: List with all the URLs of the subzones in the entire area we want to download,
29         | that can no longer be further divided into subzones
30
31         [next_url]: List with only the URL we want to download the subzones from, as there has been an error
32
33     print(f"Downloading Zones URLs from: {next_url}")
34     try:
35         html = httpx_client.get_html_from_url(next_url,previous_url)
36         # TODO: CHEQUEAR RESPONSE PARA COMPROBAR QUE SE HA DESCARGADO BIEN, O QUE HA HECHO REDIRECT, O SI ENTRÓ EL CATCH
37         # TODO: En casos como agaete hay una particularidad de municipio, que hace que o se busque bien y se conforme con el anterior
38         if html:
39             # We obtain with BeautifulSoup the soup of the HTML page
40             area_soup = bs(html, "html.parser")
41             # We look for the dropdown menu of the zone, which indicates more subzones
42             zones_soup = area_soup.find("ul", {"class": "breadcrumb-dropdown-subitem-list"})
43             zones_urls = []
44             if zones_soup:
45                 # Now we search for the list of subzones
46                 dropdown_soup = zones_soup.find_all("li", {"class": "breadcrumb-dropdown-subitem-element-list"})
47                 # We will save the number of houses for each URL
48                 for zone_soup in dropdown_soup:
49                     # For each option in the dropdown
50                     zone_URL = zone_soup.find("a").get("href")
51                     url = URL_MAIN_IDEALISTA + zone_URL
52                     print(f"Downloading subzones from: {url}")
53                     try:
54                         # Since thousands are indicated with a dot, we replace it with nothing
55                         number_houses = int(zone_soup.find("span").text.replace('.',''))
56                     except:
57                         number_houses = 0
58                     if number_houses > 0:
59                         try:
60                             # If this zone has houses, we will check if it has more subzones within it,
61                             # Using recursion with this same function
62                             # It's like traversing the branches of a tree.
63                             # In this case, we are coming from the page of the previous zone
64                             # and moving on to the next subzone
65                             zones_urls = zones_urls + list(dict.fromkeys(get_zones_from_aera(httpx_client,url,next_url)))
66                         except:
67                             print(f"Added: {url} houses:{(number_houses)}")
68                             zones_urls.append((url,number_houses))
69                         else:
70                             print(f"Added: {url}")
71                             zones_urls.append((url,number_houses))
72                         else:
73                             number_houses = int(area_soup.find("h1", {"id": "h1-container"}).text.split()[0])
74                             zones_urls.append((next_url,number_houses))
75                         return zones_urls
76                     return []
77                 except Exception as e:
78                     log_error(f'(get_zones.py) Error Getting Zones from area {next_url}: {e}')
79             return []

```

Figura 3

The screenshot shows a code editor with two panes. The left pane displays a file tree for a Python project named 'idealista\_app'. The right pane shows the content of the file 'all\_zones\_houses.py'.

```

idealista_app > tfm > scrapping_app > scrapping > all_zones_houses.py > ...
1 import os
2 import sys
3 # get current directory
4 path = os.path.dirname(__file__)
5 # parent directory
6 parent = os.path.dirname(path)
7 grandParent = os.path.dirname(parent)
8 # appending a path
9 sys.path.append(grandParent)
10 #TODO: quitar lo de arriba en produccion
11 from scrapping_app.scrapping.get_zones import get_zones_from_aera
12 from scrapping_app.scrapping.single_zone_all_pages_houses import get_and_save_houses_from_single_zone
13 from db_app.data_base.dao import Db_save_state
14 from scrapping_app.scrapping.error_logger import log_error
15
16 def get_and_save_results_from_zones(zones,httpx_client,previous_url,house_dao):
17     """
18         for loop for downloading all the houses from zones list
19
20     Parameters
21     -----
22     zones:[] url zones to be downloaded
23     httpx_client(req_html): httpx session
24     previous_url: Previous URL to create the header and simulate human-like usage
25     house_dao: Data Base Conection
26
27     Return
28     -----
29     results:[] list with all the houses downloaded and their state
30     """
31     # List to check that everything is going well and to save the results
32     results = []
33     errors = 0
34     if len(zones):
35         for zone in zones:
36             # From each subzone, we download the houses and their features
37             newResults = get_and_save_houses_from_single_zone(httpx_client,zone,previous_url,house_dao)
38             results = results + newResults[0]
39             if not newResults[1]:
40                 errors += 1
41             else:
42                 errors = 0
43             if errors == 2:
44                 log_error(f'connection loss')
45                 break
46             previous_url = zone
47     return results
48
49 def get_and_save_houses_from_zones(httpx_client,zones,previous_url,house_dao):
50     """
51         use get_and_save_results_from_zones and then print the results depending on the status
52
53     Parameters
54     -----
55     zones:[] url zones to be downloaded
56     httpx_client(req_html): httpx session
57     previous_url: Previous URL to create the header and simulate human-like usage
58     house_dao: Data Base Conection
59
60     Return
61     -----
62     results:[] list with all the houses downloaded and their state
63     """
64     # Once all the URLs of possible subzones for the entire area are downloaded
65     results = get_and_save_results_from_zones(zones,httpx_client,previous_url,house_dao)
66     for status in Db_save_state:
67         # Print OK results and KO results
68         print(Db_save_state(status).name)
69         r = [int(obj["id"]) for obj in results if obj["status"] == status]
70         print(r)
71         print(f'{len(r)} resultados con status {Db_save_state(status).name}')
72         #TODO: GUARDAR RESULTADOS EN txt
73     return results

```

Figura 4

```
# single_page.py 1 ●
dealista_app > tfm > scrapping_app > scrapping > ✘ single_page.py > ...
1 import os
2 import sys
3 # get current directory
4 path = os.path.dirname(__file__)
5 # parent directory
6 parent = os.path.dirname(path)
7 grandParent = os.path.dirname(parent)
8 # appending a path
9 sys.path.append(grandParent)
10 #1000: QUITAR LO DE ARRIBA EN PRODUCCION
11 from scrapping_app.scrapping.config import OperationType, URL_MAIN_HOUSE_IDEALISTA
12 import random
13 from scrapping_app.scrapping.req_html import Httpx_Client
14 from bs4 import BeautifulSoup as bs
15 from scrapping_app.scrapping.single_house import parser_and_save_house_id
16 from db_app.data_base.dao import Db_save_state
17 from scrapping_app.scrapping.error_logger import log_error
18
19 def get_housesId_from_page(houses_list_soup):
20     """
21         Returns all the IDs and prices of houses appearing on a single page.
22     """
23     Parameters
24     -----
25     houses_list_soup (BeautifulSoup): Soup of the page containing the list of houses.
26
27     Returns
28     -----
29     {id: price}: All the IDs and prices of the houses on the page, randomly unordered.
30     {}: If there are no houses or if there's an error.
31     """
32     try:
33         articles = houses_list_soup.find("section", {"class": "items-container"}).find_all("article")
34         ids_prices = {}
35         for article in articles:
36             id = article.get("data-adid")
37             if id:
38                 price = int(article.find("div",
39                               {"class": "item-info-container"}).find("div",
40                               {"class": "price-row"}).find("span",
41                               {"class": "item-price h2-simulated"}).text.replace(".", "").replace("€", ""))
42                 ids_prices[id] = price
43         return ids_prices
44     except Exception as e:
45         log_error(f'(single_page.py) Error getting houses ID from page {e}')
46     return {}
47
```

Figura 5

```
def get_and_save_houses_from_page(htttx_client:Httpx_Client,ids_price:dict,houses_list_url,zone_URL,dao):
    """
    Parameters
    -----
    htttx_client: Scraping session.

    ids_price (Dict): Dictionary of IDs using get_housesId_from_page from the page we are going to download.

    houses_list_url (str): The URL of the page where houses are located.

    url_zone (str): The URL of the zone.

    dao: House Dao DB Manager

    Returns
    -----
    [ {'id': house.id, 'price': price, 'date': date, 'status': Db_save_state} ]
    []: If no house has been added.
    None: If there are errors.
    """
    house_attempts = 0
    results = []
    ids = list(ids_price.keys())
    random.shuffle(ids)
    for id in ids:
        if id is None:
            continue
        price = dao.find_house_last_price_by_id(id)
        house = dao.find_houses_by_id(id)
        if house is not None and price.price == ids_price[id]:
            # The ID is in the database, and the price is the same.
            dao.update_ad_state(id,True)
            results.append({ 'id': id, 'price': price.price, 'date': price.date,'status':Db_save_state.house_and_price_duplicated})
            continue
        house_url = URL_MAIN_HOUSE_IDEALISTA + f"/{id}/"
        print("Downloading: " + house_url)
        html = htttx_client.get_html_from_url(house_url,houses_list_url)
        if html is None:
            print("Error downloading house")
            results.append({ 'id': id, 'price': None, 'date': None,'status':Db_save_state.error})
            house_attempts +=1
            if house_attempts == 3:
                print("Possible connection loss")
                return None
            continue
        single_house_soup = bs(html, 'html.parser')
        if "alquiler" in houses_list_url:
            operation = OperationType.rent
        else:
            operation = OperationType.sale
        result = parser_and_save_house_id(single_house_soup,id,operation,zone_URL,dao)
        print(f"Result: {result}")
        results.append(result)
        if result['status'] == Db_save_state.error:
            house_attempts +=1
        else:
            house_attempts = 0
    return results
```

Figura 6

```
single_page.py 1 ● single_house.py ✘
lista_app > tfm > scrapping_app > scrapping > single_house.py > parser_house_id

def parser_house_id(single_house_soup,id,operation,zone_url):
    """
    Get all single house data from idealista.

    Parameters
    -----
    single_house_soup: bs element from html single house
    id: id house from idealista
    operation: FROM CONFIG.PY rent or sale
    zone_url: House[5] Zone

    Return
    -----
    {'id':id, 'active' : True, 'last_active_check_date' : datetime.now(),
     'post_time':datetime.fromtimestamp(postTime), 'operation':operation,'title':title,
     'is_penthouse' : is_penthouse,'is_duplex' : is_duplex,'is_flat': is_flat,'is_studio' : is_studio,
     'is_apartment' : is_apartment,'is_loft' : is_loft,'is_ground_floor' : is_ground_floor,'is_semi_detached_house' : is_semi_detached_house,
     'is_townhouse' : is_townhouse,'is_bungalow' : is_bungalow,
     'is_country_house' : is_country_house,'is_large_country_house' : is_large_country_house,'is_villa' : is_villa,
     'is_terraced_house' : is_terraced_house, 'rooms_number': rooms_number,'baths_number': baths_number,
     'floors': floors, 'floor_number': floor_number,
     'built_area': built_area, 'usable_area': usable_area, 'constructed_area': constructed_area,
     'built_year': built_year, 'plot_area': plot_area,
     'has_lift': has_lift, 'has_parking':has_parking,'has_garden': has_garden,
     'has_swimming_pool': has_swimming_pool, 'has_terrace': has_terrace,
     'has_fitted_wardrobes': has_fitted_wardrobes, 'has_storage_room': has_storage_room,
     'has_balcony': has_balcony,
     'is_new_development': is_new_development, 'is_needs_renovation': is_needs_renovation,
     'is_good_condition': is_good_condition ,
     'latitude': latitude, 'longitude': longitude,
     'location_1': location_1, 'location_2':location_2,
     'location_3': location_3, 'location_4': location_4,
     'zone_url':zone_url,
     'price': price, 'date':datetime.now(),'status' : Db_save_state.pending}

    or

    {'id': id, 'price': None, 'date': None,'status':Db_save_state.error}: in case of error

    """
    errors = 0

    title,new_error = aux.extract_title(single_house_soup)
    errors += new_error

    description,new_error = aux.extract_description(single_house_soup)
    errors += new_error

    is_penthouse = aux.check_keywords_in_text(title + description, ["átilo", "última planta","atico"])
    is_duplex = aux.check_keywords_in_text(title + description, ["dúplex", "duplex"])
    is_flat = aux.check_keywords_in_text(title, ["piso"])
    is_studio = aux.check_keywords_in_text(title + description, ["estudio"])
    is_apartment = aux.check_keywords_in_text(title + description, ["apartamento"])
    is_loft = aux.check_keywords_in_text(title + description, ["loft"])
    is_ground_floor = aux.check_keywords_in_text(title , ("bajo"))
    is_semi_detached_house = aux.check_keywords_in_text(title + description, ["chalet adosado"])
    is_townhouse = aux.check_keywords_in_text(title + description, ["chalet pareado"])
    is_bungalow = aux.check_keywords_in_text(title + description, ["bungalow"])
    is_country_house = aux.check_keywords_in_text(title + description, ["finca rustica"])
    is_large_country_house = aux.check_keywords_in_text(title, ["caserón"])
    is_villa = aux.check_keywords_in_text(title + description, ["villa"])
    is_terraced_house = aux.check_keywords_in_text(title + description, ["casa terrera", "casa rural"])
    latitude,longitude,new_error = aux.extract_coordinates(single_house_soup)
    errors += new_error

    basic_c,new_error = aux.extract_basic_caracts(single_house_soup)

    has_fitted_wardrobes = 'armarios empotrados' in basic_c
    has_storage_room = "trastero" in basic_c
    is_ground_floor = is_ground_floor or ("bajo" in basic_c)
    plot_area = None
    has_terrace = has_balcony = 0
    built_year = None
    built_area = usable_area =  None
    floors = floor_number = 1

    for item in basic_c:
        try:
            if "construido en " in item:
```

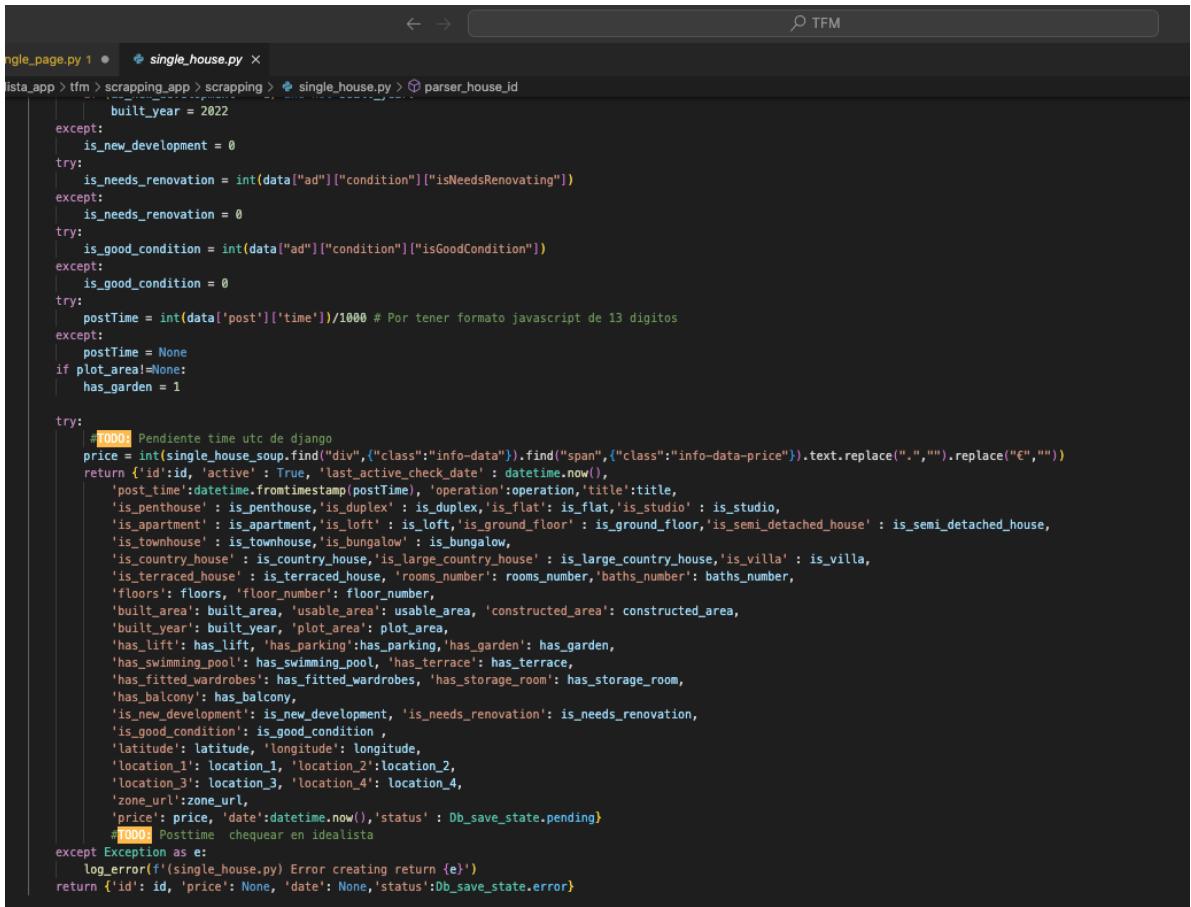
```
le_page.py 1 ● ⚡ single_house.py ✘
ta_app > tfm > scrapping_app > scrapping > ⚡ single_house.py > parser_house_id

for item in basic_c:
    try:
        if "construido en" in item:
            built_year = int(item.split("construido en ")[1])
        if "parcela de" in item:
            plot_area = float(item.split("parcela de ")[1].split()[0].replace(".", ""))
        if "terraza" in item:
            has_terrace = 1
        if "balcón" in item:
            has_balcony = 1
        if "construidos" in item and "útiles" in item:
            built_area = float(item.split()[0].replace(".", ""))
            usable_area = float(item.split()[3].replace(".", ""))
        elif "útiles" in item:
            usable_area = float(item.split()[0].replace(".", ""))
        elif "construidos" in item:
            built_area = float(item.split()[0].replace(".", ""))
        if "plantas" in item:
            floors = int(item.split()[0])
        if "planta" in item:
            try:
                string = item.split()[1]
                string = string[:-1]
                floor_number = int(string)
            except:
                floor_number = 0

    except Exception as e:
        log_error(f'(single_house.py) Error basic_c {e}')
        errors += 1

location_1, location_2, location_3, location_4, error = aux.extract_location(single_house_soup)
errors += error

try:
    scripts = single_house_soup.find_all("script")
    for script in scripts:
        if "var utag_data" in str(script):
            data = json.loads(str(script).split("var utag_data = ")(1).split(";")())
            break
except Exception as e:
    log_error(f'(single_house.py) Error var utag_data {e}')
    errors += 1
data = None
try:
    rooms_number = int(data["ad"]["characteristics"]["roomNumber"])
except:
    rooms_number = 0
try:
    baths_number = int(data["ad"]["characteristics"]["bathNumber"])
except:
    baths_number = 0
try:
    has_lift = int(data["ad"]["characteristics"]["hasLift"])
except:
    has_lift = 0
try:
    has_parking = int(data["ad"]["characteristics"]["hasParking"])
except:
    has_parking = 0
try:
    has_garden = int(data["ad"]["characteristics"]["hasGarden"])
except:
    has_garden = 0
try:
    has_swimming_pool = int(data["ad"]["characteristics"]["hasSwimmingPool"])
except:
    has_swimming_pool = 0
try:
    has_terrace = int(data["ad"]["characteristics"]["hasTerrace"]) or has_terrace or has_balcony
except:
    has_terrace = has_terrace or has_balcony
try:
    constructed_area = float(data["ad"]["characteristics"]["constructedArea"])
except:
    constructed_area = None
trv:
```



```

single_page.py 1  ⚡ single_house.py ✘
ista_app > tfm > scrapping_app > scrapping > ✘ single_house.py > ↗ parser_house_id
    built_year = 2022
except:
    is_new_development = 0
try:
    is_needs_renovation = int(data["ad"]["condition"]["isNeedsRenovating"])
except:
    is_needs_renovation = 0
try:
    is_good_condition = int(data["ad"]["condition"]["isGoodCondition"])
except:
    is_good_condition = 0
try:
    postTime = int(data["post"]['time'])/1000 # Por tener formato javascript de 13 digitos
except:
    postTime = None
if plot_area!=None:
    has_garden = 1
try:
    #TODO: Pendiente time utc de django
    price = int(single_house_soup.find("div", {"class": "info-data"}).find("span", {"class": "info-data-price"}).text.replace(".", "").replace("€", ""))
    return {'id':id, 'active' : True, 'last_active_check_date' : datetime.now(),
            'post_time':datetime.fromtimestamp(postTime), 'operation':operation,'title':title,
            'is_penthouse' : is_penthouse,'is_duplex' : is_duplex,'is_flat': is_flat,'is_studio' : is_studio,
            'is_apartment' : is_apartment,'is_loft' : is_loft,'is_ground_floor' : is_ground_floor,'is_semi_detached_house' : is_semi_detached_house,
            'is_townhouse' : is_townhouse,'is_bungalow' : is_bungalow,
            'is_country_house' : is_country_house,'is_large_country_house' : is_large_country_house,'is_villa' : is_villa,
            'is_terraced_house' : is_terraced_house, 'rooms_number': rooms_number,'baths_number': baths_number,
            'floors': floors, 'floor_number': floor_number,
            'built_area': built_area, 'usable_area': usable_area, 'constructed_area': constructed_area,
            'built_year': built_year, 'plot_area': plot_area,
            'has_lift': has_lift, 'has_parking':has_parking,'has_garden': has_garden,
            'has_swimming_pool': has_swimming_pool, 'has_terrace': has_terrace,
            'has_fitted_wardrobes': has_fitted_wardrobes, 'has_storage_room': has_storage_room,
            'has_balcony': has_balcony,
            'is_new_development': is_new_development, 'is_needs_renovation': is_needs_renovation,
            'is_good_condition': is_good_condition,
            'latitude': latitude, 'longitude': longitude,
            'location_1': location_1, 'location_2':location_2,
            'location_3': location_3, 'location_4': location_4,
            'zone_url':zone_url,
            'price': price, 'date':datetime.now(),'status' : Db_save_state.pending}
    #TODO: Posttime chequear en idealista
except Exception as e:
    log_error(f'(single_house.py) Error creating return {e}')
return {'id': id, 'price': None, 'date': None,'status':Db_save_state.error}

```

## ANEXO V

### Interfaz Gráfica Web Scrapping.

Figura 1

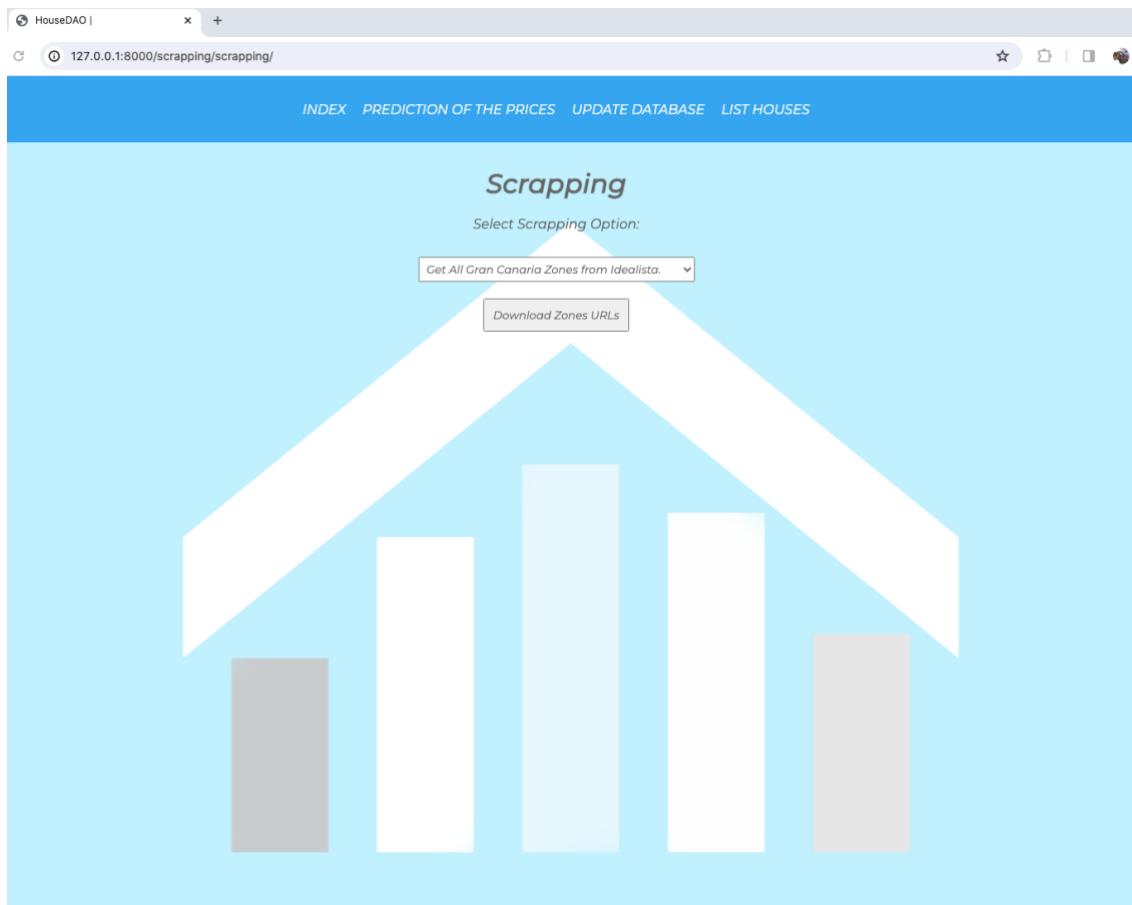


Figura 2



Figura 3

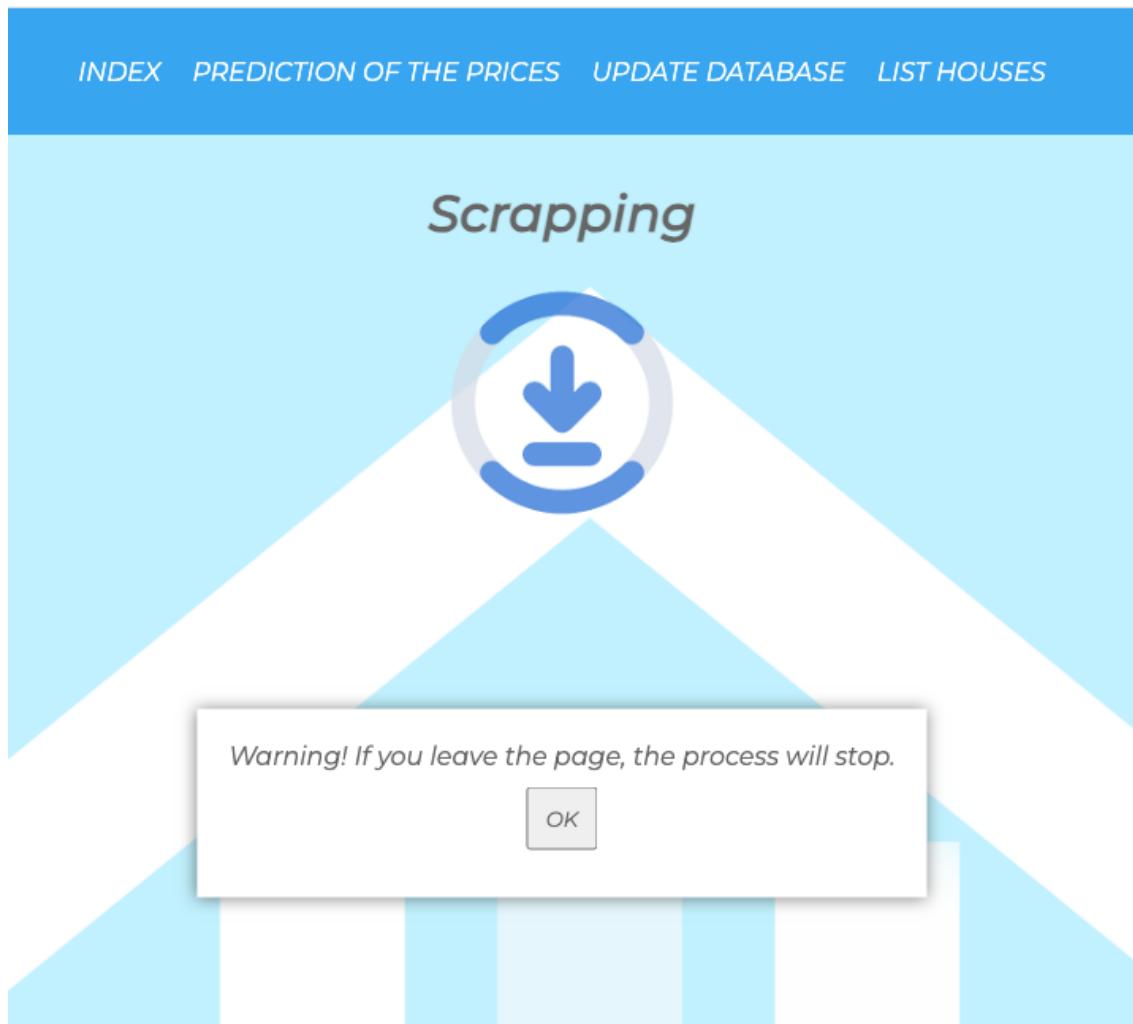


Figura 4

Index	URL	Houses	Status	
0	<a href="https://www.idealista.com/venta-viviendas/aguimes/aguimes">https://www.idealista.com/venta-viviendas/aguimes/aguimes</a>	23	0	<button>Download</button>
1	<a href="https://www.idealista.com/venta-viviendas/aguimes/cruce-de-arinaga">https://www.idealista.com/venta-viviendas/aguimes/cruce-de-arinaga</a>	49	0	<button>Download</button>
2	<a href="https://www.idealista.com/venta-viviendas/aguimes/montana-los-velez-vargas">https://www.idealista.com/venta-viviendas/aguimes/montana-los-velez-vargas</a>	16	0	<button>Download</button>
3	<a href="https://www.idealista.com/venta-viviendas/aguimes/playa-de-arinaga">https://www.idealista.com/venta-viviendas/aguimes/playa-de-arinaga</a>	62	0	<button>Download</button>
4	<a href="https://www.idealista.com/venta-viviendas/arucas/arucas-casco">https://www.idealista.com/venta-viviendas/arucas/arucas-casco</a>	80	0	<button>Download</button>
5	<a href="https://www.idealista.com/venta-viviendas/arucas/banaderos-el-puertillo-san-andres">https://www.idealista.com/venta-viviendas/arucas/banaderos-el-puertillo-san-andres</a>	23	0	<button>Download</button>
6	<a href="https://www.idealista.com/venta-viviendas/arucas/cardones-tinocas">https://www.idealista.com/venta-viviendas/arucas/cardones-tinocas</a>	21	0	<button>Download</button>
7	<a href="https://www.idealista.com/venta-viviendas/arucas/juan-xxiii">https://www.idealista.com/venta-viviendas/arucas/juan-xxiii</a>	19	0	<button>Download</button>
8	<a href="https://www.idealista.com/venta-viviendas/arucas/la-montaneta-la-goleta-el-cerrillo">https://www.idealista.com/venta-viviendas/arucas/la-montaneta-la-goleta-el-cerrillo</a>	14	0	<button>Download</button>

Figura 5

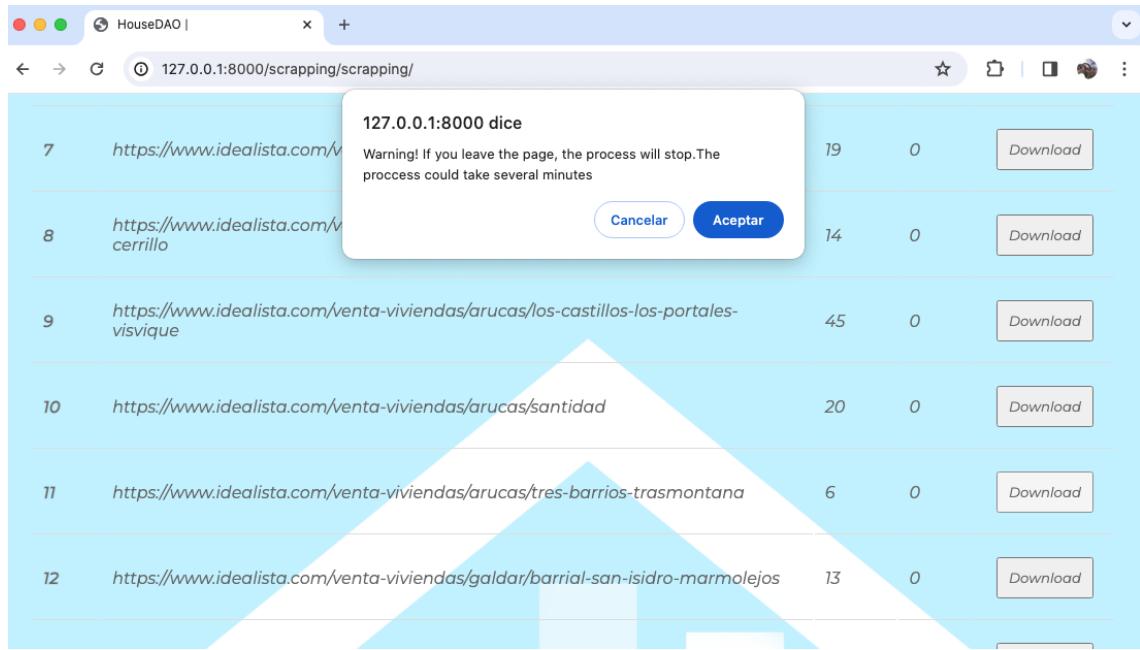


Figura 6



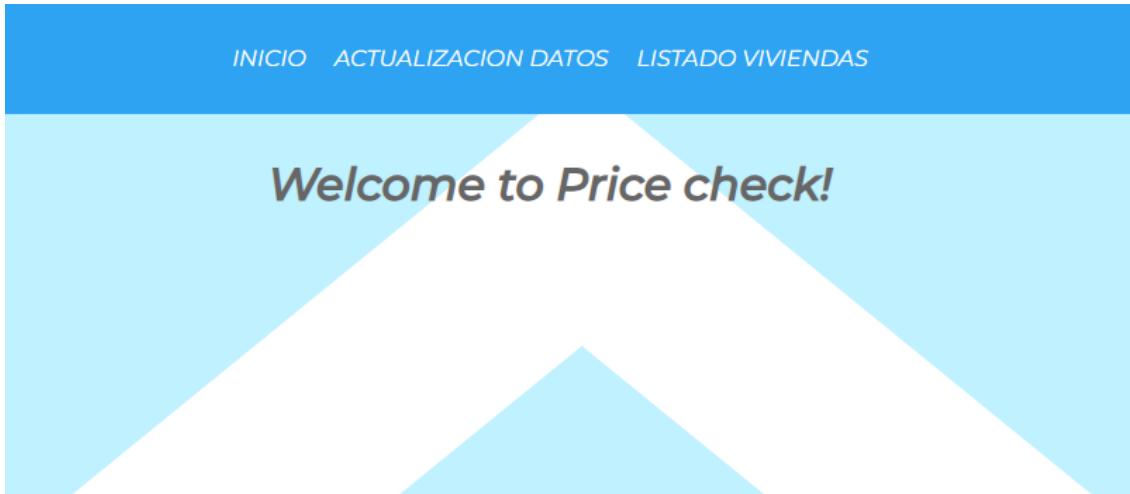
Figura 7



## ANEXO VI

### Interfaz Gráfica visualización Datos

Acceso a listado de casas menu



Proceso de carga de datos como realización de predicciones de precios de las viviendas.



Página general para ver los listados de las viviendas según los filtros seleccionados.

**Selecciona provincia:** Gran Canaria, Las Palmas

**Municipio:** Santa Brígida

**Buscar**

**Tipo Vivienda:**

- Atico
- Duplex
- Piso
- Estudio
- Apartamento
- Loft
- Planta baja
- Casa pareada
- Adosado
- Bungalow
- Casa de campo
- Casa de campo grande
- Villa
- Casa adosada

**Características:**

- Ascensor
- Estacionamiento
- Jardín
- Piscina
- Terraza
- Armarios\_empotrados
- trastero
- Balcón

**Estado:**

- Nueva.construcción
- Necesita reformas
- En buen estado

**Listado de Casas**

Icono de casa

« Primera Anterior Siguiente Última »

Grafico Análisis de Precios

Visualizar Gráfica

Una vez filtradas las características de las viviendas. Obtenemos las viviendas encontradas y el precio predicho para dicha vivienda. Indicando la diferencia de precios e icono indicativo de si la vivienda está por encima del precio calculado o si está por debajo

Esto realizado con paginación de las viviendas mediante peticiones AJAX

**Listado de Casas**

Icono de casa

Casa	Link	Precio Vivienda	Precio Calculado	Diferencia	Resultado	Exportar
Piso en venta en portada verde-lomo espino-guanche	<a href="#"></a>	122000	119600.7	2399.3		
Dúplex en venta en camino a la caldera, 28	<a href="#"></a>	138000	142373.19	-4373.19		
chalet adosado en venta en calle los pérez, 13	<a href="#"></a>	170000	271955.87	-101955.87		
Dúplex en venta en calle los pérez, 13	<a href="#"></a>	170000	175230.76	-5230.76		
Dúplex en venta en calle cura navarro, 28	<a href="#"></a>	195000	234705.4	-39705.4		

1 de 5 Siguiente Última »

Grafica de visualización de precios medios según las zonas donde se encuentran las viviendas. Como calculo de el precio medio predicho en las zonas.

Realizado con Matplotlib.

