UNIVERSIDADE DA CORUÑA

Práctica RIWS - Scraping & Crawling

Repositorio

Clara Valle Gómez
Antón Soto Ríos
Adrián Rego Criado

23 de noviembre de 2025

Índice

1. Motivación **2**

2. Desarrollo **2**
2.1. Scrapy . . . 2
2.2. Elasticsearch . . . 3
2.2.1. `create_index.py` . . . 3
2.2.2. `insert_docs.py` . . . 4
2.2.3. `search_docs.py` . . . 4
2.3. Elasticsearch UI . . . 4
2.3.1. `Engine.json` . . . 4
2.3.2. `App.js` . . . 5
2.3.3. `CustomResultView` . . . 6
2.3.4. `CustomFacetView` . . . 6
2.3.5. `DynamicValueFacetView` . . . 6
2.3.6. `Charts.js` . . . 7
2.3.7. App.css . . . 8

3. Instalación **8**
3.1. Scrapy . . . 8
3.2. Elasticsearch . . . 8
3.3. ElasticUI + React + Dependencias . . . 8

4. Ejecución **9**

1. Motivación

La situación del mercado de alquiler en la zona de A Coruña ha sido nuestra principal motivación para desarrollar esta aplicación de scraping y crawling, la cual permite monitorizar pisos disponibles en la ciudad a partir de los datos publicados en pisos.com. El mercado del alquiler en la ciudad ha sido declarado como tensionado (la demanda de viviendas supera con creces la oferta disponible), lo que aumenta los precios y dificulta que muchas personas jóvenes puedan independizarse. Ante esta realidad, disponer de una herramienta que recopile automáticamente todos los anuncios de alquiler de la zona, permite a los usuarios acceder a la información de forma más directa, acompañada de filtros específicos y gráficas que faciliten entender, de un solo vistazo, qué barrios son más económicos o qué opciones se ajustan mejor a un determinado presupuesto.

Centralizar los datos y la incorporación de filtros dinámicos hace posible comparar precios, zonas, número de habitaciones y otras características sin tener que navegar manualmente por la página una y otra vez, lo que agiliza la toma de decisiones, tanto para uso personal como para ver la situación del mercado.

Identificar una web que permitiera ser analizada de manera automática también fue un desafío, ya que muchas plataformas inmobiliarias implementan múltiples medidas para evitar el scraping. Por ello, pisos.com se presentó como la opción más viable para el desarrollo del proyecto.

Es importante recalcar que cualquier sistema de scraping debe realizarse respetando los términos de uso del sitio web y su archivo robots.txt, así como limitando la frecuencia de las consultas para no perjudicar el funcionamiento de la página.

En conclusión, una aplicación de este tipo ofrece una visión clara y accesible del mercado de alquiler en A Coruña, a la vez que ayuda a elegir de forma más informada un lugar adecuado para vivir en la ciudad.

2. Desarrollo

El desarrollo de este proyecto se divide en varias fases: la obtención de la información original mediante Scrapy, la indexación de estos datos en índices de Elasticsearch y el servidor encargado de suministrarlos y, por último, la interfaz web que se comunica con dicho servidor para obtener, agrupar y filtrar la información de forma conveniente.

2.1. Scrapy

El spider `pisos_live.py` está diseñado para extraer información de pisos en alquiler desde `pisos.com`, concretamente desde la página de alquiler de pisos en A Coruña. A partir de esta lista de pisos de alquiler, se accede a los enlaces que llevan a la ficha concreta de cada piso, donde se encuentra la información detallada.

Este programa es un **spider de Scrapy**, diseñado para recorrer automáticamente las páginas de listados de pisos y extraer de manera estructurada la información de cada ficha de inmueble. Hemos implementado las siguientes funcionalidades:

- **Inicio y búsqueda de pisos:**
  - Busca los enlaces que llevan a cada ficha de piso a partir de los enlaces de la página, atributos `data-href` y scripts.
  - Solo sigue URLs que cumplen ciertos patrones para evitar páginas irrelevantes.
- **Extracción de información:**
  - Primero intenta obtener datos estructurados en formato **JSON-LD** que la propia página proporciona. Esto incluye:
    - Título y descripción del piso.

    - Precio en euros.
    - Dirección completa: calle, barrio, municipio y provincia.
    - Imágenes principales del piso.
  - Si algún dato no está en JSON-LD, el spider lo busca directamente en la página:
    - Precio buscando el símbolo €.
    - Número de habitaciones, baños y superficie a partir de palabras clave.
    - Barrio desde el nombre de la URL.
    - Título del piso desde encabezados.

- **Normalización y filtrado:**
  - Convierte precios y superficies a números.
  - Descarta fichas sin título o sin precio.

- **Paginación:**
  - Tras procesar todos los pisos de una página, sigue el enlace a la página siguiente del listado.
  - Repite el proceso hasta no encontrar más páginas.

En resumen, este spider obtiene la información completa desde el listado de pisos hasta la ficha de cada uno, incluyendo título, descripción, precio, dirección e imágenes, entre otros. Esta información se organiza según la estructura definida en `items.py`, que especifica los campos que se extraen de cada piso, permitiendo exportar los datos de forma estructurada.

El archivo `settings.py` centraliza las configuraciones que afectan a los spiders. De esta forma, nos aseguramos que se respeten las normas de `robots.txt` (`ROBOTSTXT_OBEY = True`) y establecemos una velocidad para realizar las solicitudes (`DOWNLOAD_DELAY = 0.5`) de forma que no se perjudique el correcto funcionamiento de la página. Además, define un `USER_AGENT` y encabezados HTTP por defecto (`DEFAULT_REQUEST_HEADERS`) para que el bot se identifique y envíe las cabeceras indicadas en cada solicitud.

2.2. Elasticsearch

2.2.1. `create_index.py`

Este archivo se encarga de crear y configurar el índice de Elasticsearch `pisos_index`. Sus funcionalidades principales son:

- Inicializa un cliente de `Elasticsearch` apuntando a `http://localhost:9200`.
- Define un **mapping** que incluye únicamente los campos implementados en el código:
  - Campos de texto: `title`.
  - Campos categóricos: `url`, `listing_id`, `neighborhood`, `images`.
  - Campos numéricos: `price_eur`, `rooms`, `bathrooms`, `surface_m2`.
- Implementa la función `create_index()`, que borra el índice si ya existe y lo crea nuevamente con el mapping definido.

2.2.2. `insert_docs.py`

Este archivo inserta los documentos contenidos en `pisos_a_coruna.jsonl` en el índice `pisos_index` de Elasticsearch.

- Establece una conexión con `Elasticsearch` apuntando a `http://localhost:9200`.
- Implementa la función `load_jsonl(file_path)`, que:
  - Lee un archivo en formato `JSONL` línea por línea.
  - Convierte cada línea en un diccionario de Python mediante `json.loads`.
  - Devuelve una lista de documentos.
- Implementa la función `insert_docs(file_path)`, que:
  - Carga los documentos usando `load_jsonl`.
  - Inserta todos los documentos en Elasticsearch mediante `helpers.bulk`.
  - Imprime el número total de documentos insertados.

De esta forma, se insertan de manera eficiente todos los pisos en el índice `pisos_index`, permitiendo su consulta y análisis posterior mediante el servidor de Elasticsearch.

2.2.3. `search_docs.py`

No se usa realmente. La lógica de la búsqueda y filtrado está implementada en el frontend, pero sirve para comprobar que se pueden filtrar los pisos insertados de forma correcta.

2.3. Elasticsearch UI

Para el desarrollo de la interfaz hemos desarrollado diferentes archivos clave para interactuar con el índice creado anteriormente y el desarrollo de diferentes componentes para la correcta visualización de estos.

2.3.1. `Engine.json`

El archivo `engine.json` define la configuración de un motor de búsqueda para el índice de pisos creado anteriormente `pisos_index`, indicando cómo se deben buscar, mostrar y filtrar los datos. Sus principales componentes son:

- **Campos de búsqueda:**
  - `searchFields`: usa `title` para la búsqueda por texto.
- **Campos de resultados:**
  - `resultFields`: campos que se mostrarán en los resultados, como título, barrio, precio, habitaciones, baños, superficie, imágenes y URL.
  - `titleField`, `urlField`, `thumbnailField`: campos específicos para mostrar el título, el enlace y la imagen de miniatura en la interfaz.
- **Ordenamiento y filtrado:**
  - `sortFields`: campos disponibles para ordenar los resultados.
  - `facets`: campos utilizados para filtros o facetas en la búsqueda, como barrio, número de habitaciones, baños, precio o superficie.

De esta forma se proporciona la información necesaria para que interfaz pueda consultar el motor de búsqueda y permitir filtrado y ordenamiento de los pisos.

2.3.2. `App.js`

El archivo `App.js` implementa la interfaz web de búsqueda y visualización de pisos utilizando `React 18` junto con `@elastic/react-search-ui` y un conector a Elasticsearch. Esta interfaz permite a los usuarios buscar, filtrar, ordenar y explorar los resultados del índice `pisos_index` de forma sencilla.

- **Conexión a Elasticsearch:** Se establece mediante `ElasticsearchAPIConnector` en el puerto 9200 que contiene el índice `pisos_index`.
- **Configuración de búsqueda:**
  - `search_fields`: define los campos sobre los que se realiza la búsqueda de texto, en este caso `title`.
  - `result_fields`: indica qué campos del documento se muestran en los resultados, como `listing_id`, `title`, `price_eur`, `rooms`, `bathrooms`, `surface_m2`, `neighborhood`, `images` y `url`.
  - `facets` y `disjunctiveFacets`: permiten filtrar los resultados mediante facetas para barrio, número de habitaciones, baños, precio y superficie.
- **La interfaz está constituida por los siguientes elementos:**
  - `SearchBox`: barra de búsqueda para introducir consultas de texto.
  - `Facet`: componentes para filtrar por habitaciones, baños, superficie, precio y barrio usando componentes personalizados, integrados en `CustomFacetView`.
  - `Results`: muestra los resultados de la lista usando el componente `CustomResultView`
  - `Charts`: visualiza datos con gráficos usado el componente `Charts`.
  - `PagingInfo`, `ResultsPerPage` y `Paging`: permiten la navegación y paginación de resultados.

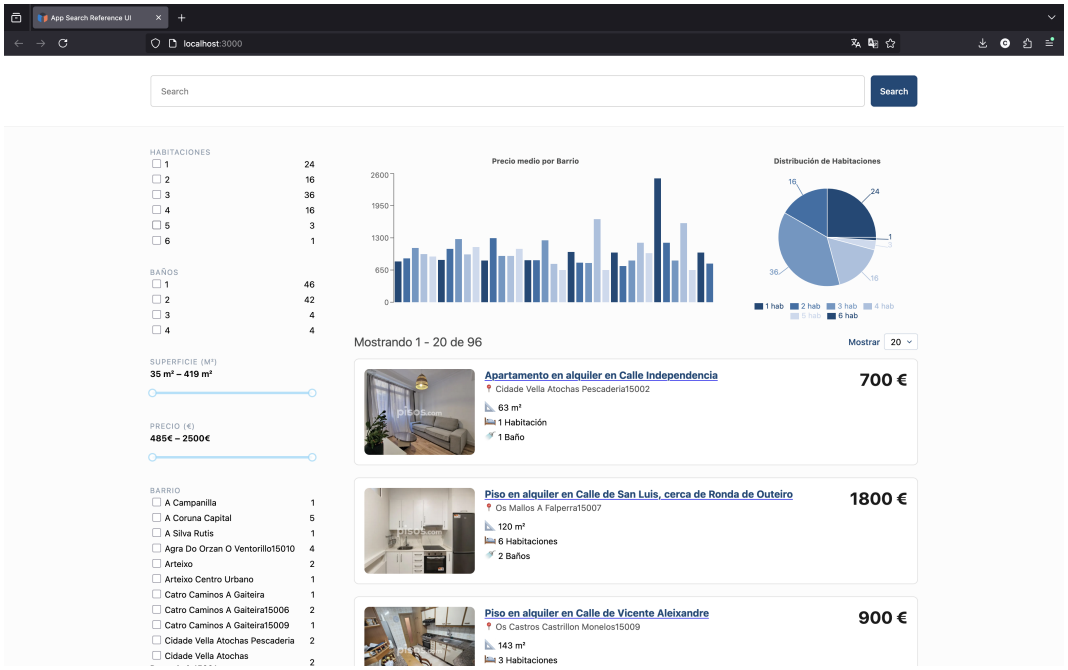

Figura 1: Interfaz sin aplicar filtros de la aplicación web

2.3.3. `CustomResultView`

El componente `CustomResultView` es responsable de renderizar cada uno de los resultados de búsqueda obtenidos desde Elasticsearch, presentándolos en forma de lista como "tarjetas". Para ello, extrae del objeto `result` los campos relevantes como el título, el precio, las habitaciones, los baños, la superficie, el barrio y la URL del anuncio. También procesa el campo `images` para seleccionar una imagen principal válida, asegurando que cada tarjeta muestre una fotografía representativa del piso.

El diseño del componente organiza la tarjeta en dos columnas: una sección izquierda destinada a la imagen y otra derecha en la que muestra los datos concretos de la propiedad como el título, barrio, precio o número de habitaciones y baños, utilizando iconos para mejorar la legibilidad.

2.3.4. `CustomFacetView`

El componente `CustomFacetView` proporciona una vista personalizada para mostrar filtros en la interfaz.

- **Filtros con slider**
  - Si el filtro es un rango, obtiene el valor mínimo y máximo presentes en el índice.
  - Usamos la librería `rc-slider` para una sencilla implementación
  - Cada vez que el usuario mueve el slider, se actualiza el filtro y se hace la búsqueda.
- **Filtros con checkbox**
  - Muestra una lista de opciones con casillas de selección.
  - Cada opción incluye el valor y el número de resultados asociados.
  - Las opciones que tienen un contador igual a cero aparecen desactivadas.

Este componente es la base para aplicar filtros, que pueden implementar otros componentes como `DynamicValueFacetView`.

2.3.5. `DynamicValueFacetView`

El componente encargado de la lógica de los filtros es `DynamicValueFacetView`, que implementa la función `createDynamicValueFacetView()`, la cual devuelve un componente que cambia los filtros (*facets*) dinámicamente, incluso cuando los resultados de búsqueda cambian. Esto permite evitar la desaparición o reordenación (que a la vista produce un cambio muy brusco) de opciones al aplicar o eliminar filtros. Esta función permite:

- **Guardado de filtros:** Guarda todos los valores que existen sin filtrar para que al hacer una búsqueda o aplicar filtros estos no desaparezcan de la lista.
- **Fusión de opciones nuevas con las originales:** A partir de los valores iniciales, se genera una lista que combina:
  - las opciones actuales devueltas por la búsqueda o filtrado
  - los valores que no corresponden con la búsqueda o filtrado se muestran a 0 y con un color gris
- **Ordenación:** Las opciones se ordenan numérica o alfabéticamente.

Finalmente, este componente se integra con `CustomFacetView` para proporcionar la visualización final de cada filtro. Todos los filtros de la aplicación utilizan este mismo mecanismo, ya que implementan la función `createDynamicValueFacetView` para tener valores dinámicos. De esta forma, los componentes `BathroomsFacetView`, `NeighborhoodFacetView`, `PriceFacetView`, `RoomsFacetView` o `SurfaceFacetView` no contienen lógica propia, sino que simplemente instancian la vista generada por esta función.

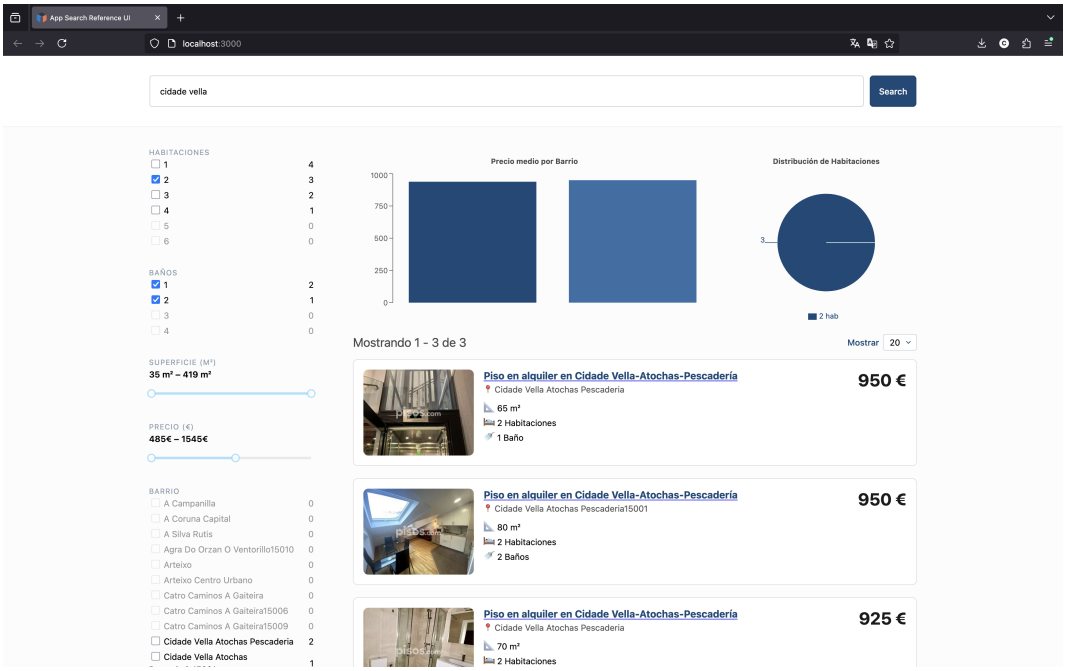

Figura 2: Interfaz con búsqueda y filtros aplicados

2.3.6. `Charts.js`

Por último, el componente `Charts.js` se encarga de generar las gráficas estadísticas a partir de los datos almacenados en el índice de Elasticsearch, usando la librería `recharts`. El componente se actualiza con el estado de la búsqueda a través de `WithSearch`, pero en lugar de depender del texto que el usuario va escribiendo, utiliza el texto final de búsqueda (`resultSearchTerm`). Esto garantiza que las gráficas se actualicen únicamente cuando se lanza una búsqueda completa, y no en cada pulsación de tecla.

A partir del texto final de búsqueda y de los filtros activos, el componente construye una consulta a Elasticsearch que solo solicita agregaciones (sin recuperar documentos individuales). Se obtienen, para la gráfica de barras los precios medios por barrio y para la gráfica tipo "pie" la distribución del número de habitaciones. De este modo, el usuario puede visualizar rápidamente la situación total de precios y distribución de habitaciones en el mercado, o una visión general de la búsqueda o filtrado que ha realizado.

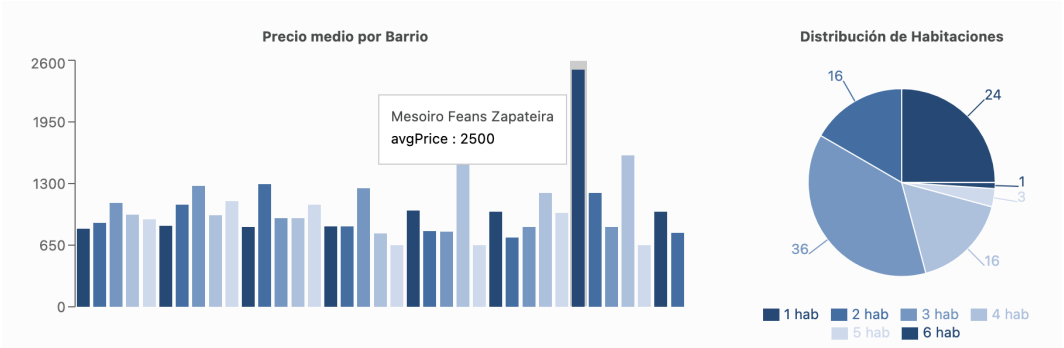

Figura 3: Gráficas de precio medio por barrio y de distribución de habitaciones

2.3.7. App.css

El archivo `App.css` define los estilos globales de la aplicación como tipografías, tamaños, márgenes y colores para unificar la apariencia visual. Además, personaliza elementos de la librería *Search UI*, como el cuadro de búsqueda, los filtros y la paginación. También proporciona estilos específicos para las tarjetas de resultados, dando como resultado una presentación limpia y consistente en toda la interfaz.

3. Instalación

3.1. Scrapy

1. Instalar scrapy e iniciar proyecto

```
$ pip install scrapy
$ scrapy startproject buscoocasa
```

3.2. Elasticsearch

1. Descarga e instalación

```
      $ cd elasticsearch-9.2.1
      $ bin/elasticsearch
(mac) $ xattr -d com.apple.quarantine jdk.app
```

2. Desactivar autenticación

```
$ nano config/elasticsearch.yml
  + xpack.security.enabled: false
  + xpack.security.enrollment.enabled: false
```

3. Uso con python

```
$ pip install elasticsearch
```

3.3. ElasticUI + React + Dependencias

1. Importante tener instalado React18 para que no de errores de dependencias

```
$ npm install react@18 react-dom@18
```

2. Para el uso de gráficas es necesario instalar `recharts`

```
$ npm install recharts
```

3. Para el uso de sliders es necesario instalar `recharts`

```
$ npm install rc-slider
```

4. Descargar la aplicación de Search UI

```
$ curl https://codeload.github.com/elastic/
  app-search-reference-ui-react/tar.gz/master | tar -xz

$ cd app-search-reference-ui-react-master
```

```
(mac) $ brew install yarn
(mac) $ brew install npm
```

5. Configuración `elasticsearch.yaml`

```
http.cors.enabled: true
http.cors.allow-origin: "*"
http.cors.allow-headers: X-Requested-With, X-Auth-Token,
Content-Type, Content-Length, Authorization,
Access-Control-Allow-Headers, Accept, x-elastic-client-meta
http.cors.allow-methods: "OPTIONS, HEAD, GET, POST, PUT, DELETE"
```

6. Instalar el conector para Elasticsearch

```
$ yarn add @elastic/search-ui-elasticsearch-connector
```

7. Añadir a `src/App.js` para importar el nuevo conector de Elasticsearch

```
+ import ElasticsearchAPIConnector from
  "@elastic/search-ui-elasticsearch-connector";
+ const connector = new ElasticsearchAPIConnector({
  host: "http://localhost:9200",
  index: "pisos_index"
  });
```

8. Renombrar `src/config/engine.json.example` a `src/config/engine.json`

4. Ejecución

1. Iniciar el proyecto y generar el json

```
$ cd buscoocasa
$ scrapy crawl pisos -O data/pisos_a_coruna.jsonl
  -s FEED_EXPORT_ENCODING=utf-8
```

2. Iniciar el servidor Elasticsearch (donde lo hayas descargado)

```
$ cd elasticsearch-9.2.1
$ bin/elasticsearch
```

3. Generar índices

```
$ cd buscoo.casa/elasticsearch
$ python3 create_index.py
$ python3 insert_docs.py
```

4. Iniciar Servidor

```
$ cd buscoo.casa/app-search-reference-ui-react-master
$ npm start
```