GetACar: Recuperación y búsqueda de automóviles usando APIs de terceros

Entrega Final

Informe Obligatorio 2022

Grupo 02

Renzo Beux - 50769058

Joaquín Menes - 54938495

Facundo Fleitas - 50721349

Docente

Libertad Tansini

1. Presentación de problema a abordar

Hoy en día tenemos toda la información a dos clics y no tenemos que visitar concesionarias presencialmente a la hora de comprar un vehículo, generalmente los usuarios utilizan distintas páginas web o aplicaciones móviles ya que ahí es donde podemos encontrar la mayor parte de la información. Sin embargo, esto también trae problemas para el usuario, son muchas las variables que tenemos que considerar a la hora de comprar un nuevo vehículo.

La magnitud de la información y la descentralización logran que el usuario se sienta agobiado y provocan que pueda perder distintas oportunidades por el tiempo que consume visitar todos los sitios.

Con este proyecto, buscamos mejorar estos puntos, además brindar una mayor accesibilidad y centralización en la búsqueda de autos en la web.

2. Enfoque de la solución

Para poder solucionar el problema planteado, necesitamos recopilar las publicaciones de autos en venta de las páginas más populares (podrían implementarse más con el tiempo), como por ejemplo: Mercado Libre y CarOne y llevarlo a una base común.

Una vez construido el corpus, necesitamos construir índices para facilitar la búsqueda del usuario sobre este conjunto de datos.

Observamos también que la importancia de un usuario sobre un auto puede ser muy relativa, y puede variar la relevancia. Considerando esto, creemos necesario desarrollar un sistema de ranking para presentarle al usuario lo que consideramos las mejores opciones y así mejorar la experiencia de usuario y solucionar uno de los problemas del exceso de información.

Esta función de ranking utilizará la información provista por las distintas publicaciones, utilizando weights en los distintos atributos para generar un ranking del auto. Este ranking será utilizado por los motores de búsqueda para poder ordenar los resultados y mostrarle al usuario la mejor opción primero.

Por otro lado, además buscamos centralizar la información de distintos sitios para que el usuario no pierda tiempo buscando en cada uno de ellos.

Se incorpora además un sistema de filtros para aumentar la usabilidad y permitir al usuario buscar por rango de precios, kilómetros, y años.

3. Diseño del sistema e implementación

a. Diseño

Para poder lograr el sistema propuesto, se diseña una arquitectura que seguirá un modelo básico de tres capas, separando las capas de cliente, lógica y capa de base de datos:

Capa de Frontend

También llamado capa de cliente, es la encargada de mostrar las pantallas de búsqueda y los resultados de las búsquedas a los usuarios. El usuario interactúa con esta capa mediante distintos botones y campos, que desencadenan acciones con el resto del sistema.

Esta capa se comunicará directamente con el servidor de backend, quién le otorgará los distintos datos para que se muestren en pantalla.

Capa de Backend

También llamada capa lógica. Tiene distintas funcionalidades:

En primer lugar, será el nexo de nuestro sistema, funcionando como intermediario entre la capa de Frontend y la capa de datos. Expondrá una API

para que la capa de cliente pueda consumir e interactuar con el sistema, por ejemplo, una búsqueda.

Además, será encargado de la etapa de obtención, puntuación y limpieza de los datos de las distintas publicaciones. Se activará un proceso periódicamente en el cuál se traerán los datos de las marcas que forman parte de la base de datos en los distintos sitios web, convirtiendo los datos que llegan a un modelo propio.

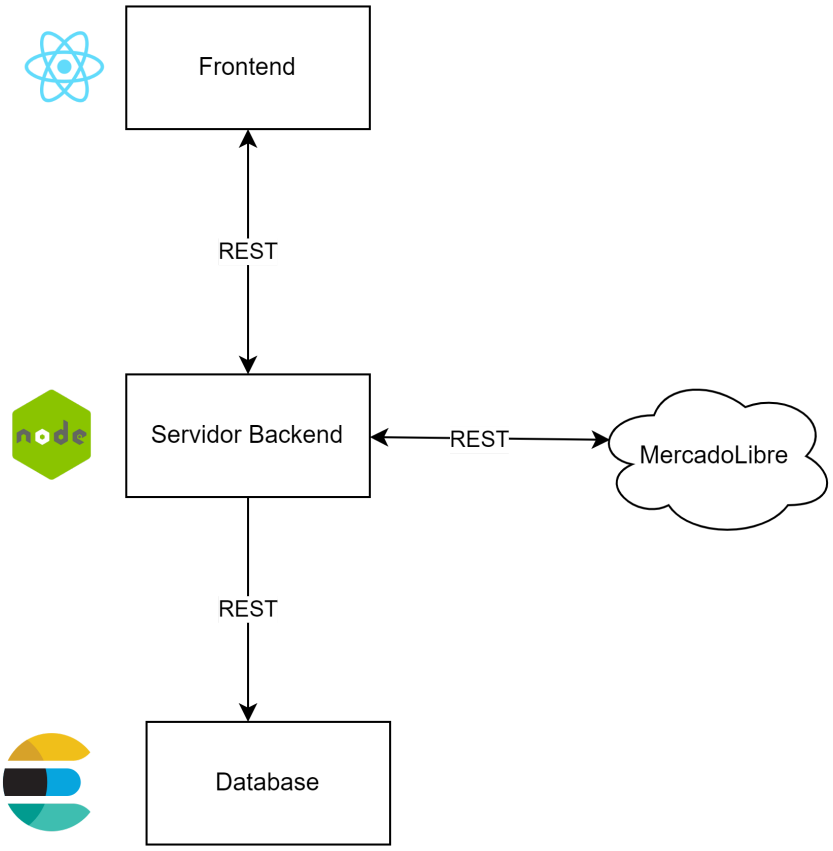
Capa de Datos

Esta capa será la responsable de almacenar todos los datos de manera normalizada, se almacenarán las distintas marcas con sus respectivos modelos y además las publicaciones que se obtengan de los distintos sitios.

Interoperabilidad

Se definió además que la comunicación entre los distintos componentes utilizará el estilo de arquitectura API REST.

Teniendo en cuenta todo lo anterior, la arquitectura resultante es la siguiente:



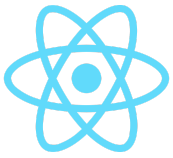
Arquitectura de componentes e interoperabilidad en alto nivel

b. Implementación

Capa de Frontend

La capa de frontend se implementa con **React**, utilizando Client Side Rendering.

React

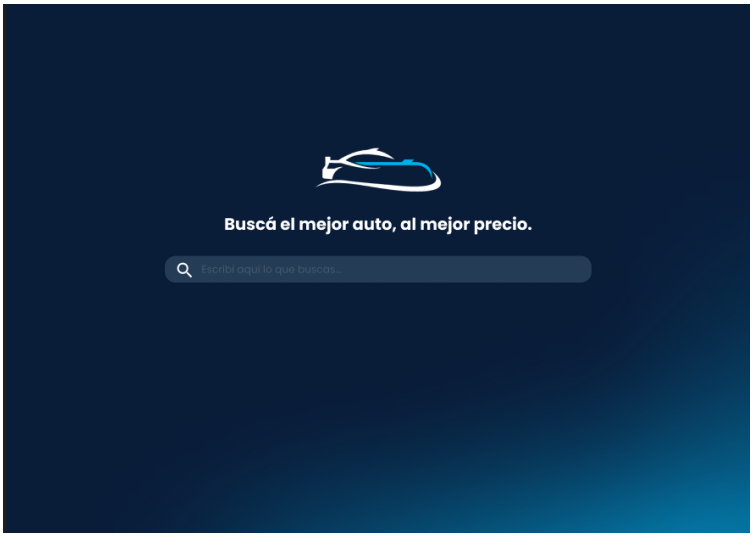


React es una biblioteca de Javascript creada para construir interfaces de usuario de forma declarativa y basada en componentes. Esto permite la reutilización de código y permite mejorar la legibilidad del mismo. Es muy sencillo y rápido de usar.

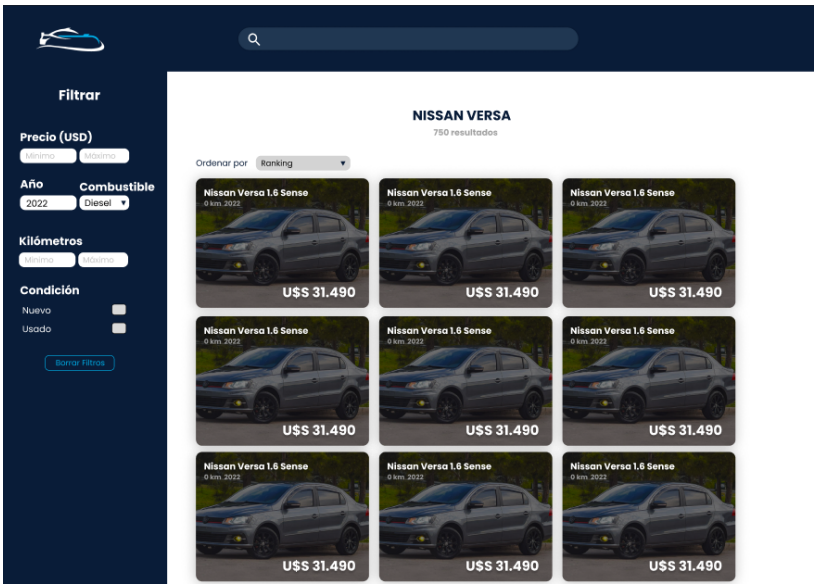
En esta capa se realizan distintas páginas para que el usuario pueda interactuar y buscar el auto más apropiado, agrupados por marca-modelo.

UI/UX

Se realizaron distintos mockups en una herramienta de diseño colaborativo en tiempo real (FIGMA), buscando la mejor experiencia para el usuario.



Página principal (búsqueda)

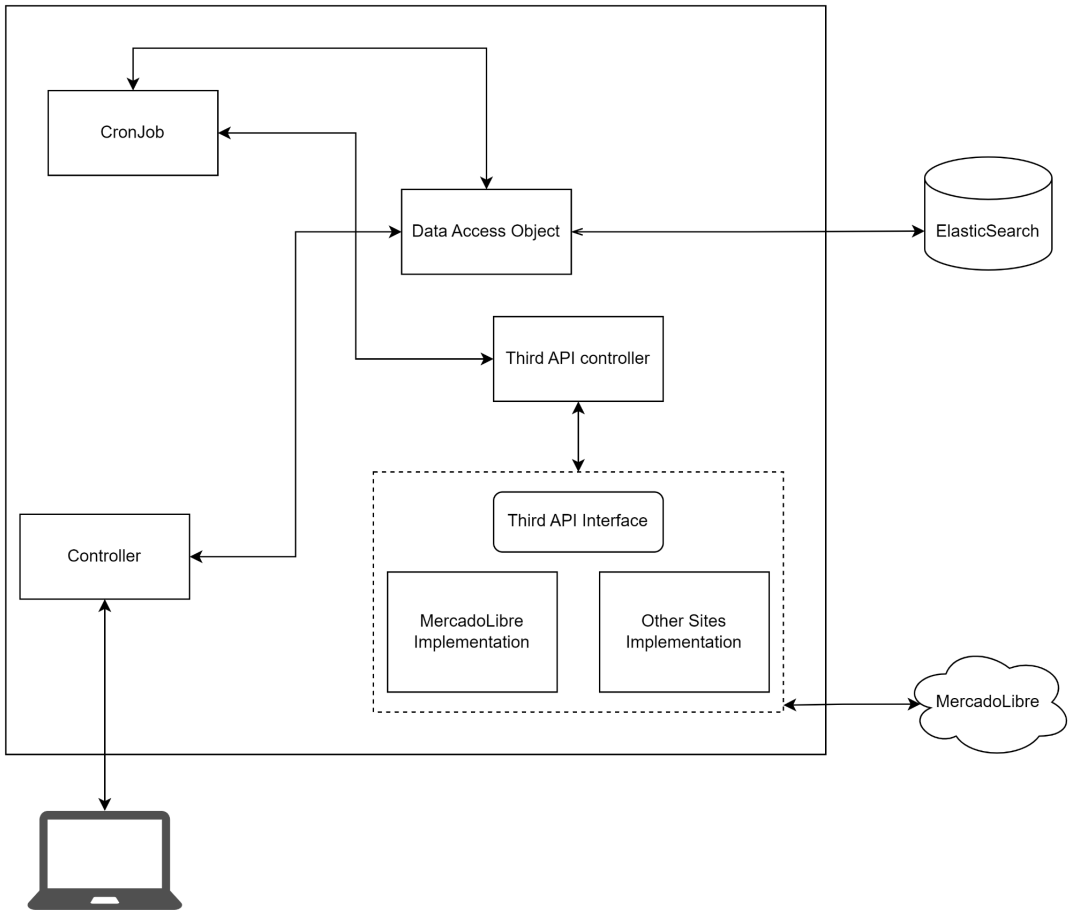


Página de resultados obtenidos.

Capa de Backend

La capa de Backend se implementará utilizando **NodeJS**, con el lenguaje de programación TypeScript.

Se busca aplicar las mejores prácticas de ingeniería, y tratar de tener un sistema escalable, en el que integrar nuevos sitios sea relativamente sencillo, por el momento solo se empezará con Mercado Libre y CarOne.



Arquitectura de componentes de Servidor Backend

El componente **CronJob** se encargará de obtener periódicamente los datos de Mercado Libre u otros sitios, a través del controlador Third API.

El componente **Controller** se encarga de exponer los servicios (API) con una arquitectura REST para que el cliente pueda consumir. Uno de los servicios CORE de este controlador es el servicio:

/search?q=<*búsquedaUsuario*>

Este servicio interpretará la búsqueda del usuario, enviando querys a la base de datos para determinar si el usuario está buscando un modelo en específico, una marca, o simplemente quiere ver todas las marcas que el sistema ofrece.

Para el acceso a los datos, se implementa el patrón de diseño **DAO**, estos DAO son los que realmente pueden interactuar con la base de datos, a través de REST API que expone ElasticSearch, separando el modelo de la capa de persistencia.

Una vez obtenidos los datos de los sitios, los convertiremos en un modelo que sea capaz de guardarse en nuestra base: **Advertisements**, de esta forma, normalizamos los datos a un tipo de modelo propio de nuestro sistema.

Para esta versión del proyecto se integra la API de MercadoLibre, durante el desarrollo se intentó además integrar la API de CarOne, pero por un cambio a último momento tuvimos que realizar un script para scrapear de su página web.

API MercadoLibre

Esta API que publica MercadoLibre es posible consumirla de manera gratuita y nos provee los datos de las publicaciones para poder poblar nuestra base e indexar Marcas, Modelos y además Publicaciones. Además existe una documentación bastante detallada, la cual nos sirvió de guía para poder utilizarla.

En primer lugar obtenemos las marcas y sus respectivos modelos de MercadoLibre gracias a esta API, y luego consultamos por las primeras 1000 publicaciones (el límite de la API) para cada una de las marcas.

Luego de obtener los resultados, los indexamos en ElasticSearch.

Función de Ranking

Además, en esta capa se valorarán los distintos vehículos de acuerdo a una fórmula de ranking que utilizaremos para ordenar los resultados, y almacenarlos en la base ya valorizados en base a los atributos de la publicación original.

Para ordenar las publicaciones utilizamos una función de ranking diseñada en base a datos extraídos de distintas investigaciones (enlaces en Referencias). Utilizamos los parámetros de año, kilómetros y precio. Los datos obtenidos afirman que cada 20.000 millas el vehículo degrada aproximadamente su valor un 20% y que por cada año que pasa se degrada aproximadamente un 20%. Les aplicamos la función a los vehículos y los presentamos de manera de que, el que tenga mayor puntaje de ranking es el más recomendado.

Capa de persistencia

Para la capa de persistencia decidimos implementar todo con ElasticSearch.

ElasticSearch

ElasticSearch es un motor de búsqueda basado en la API de código abierto para recuperación de información Lucene. El mismo provee un motor de búsqueda y analítica distribuido, gratuito y abierto para varios tipos de datos como pueden ser: textuales, numéricos, geoespaciales, estructurados y no estructurados.

Este motor de búsqueda nos permite buscar dentro de toda la base los vehículos relevantes para el usuario muy rápidamente, aplicando técnicas que vemos en el curso, como por ejemplo la construcción de índices.

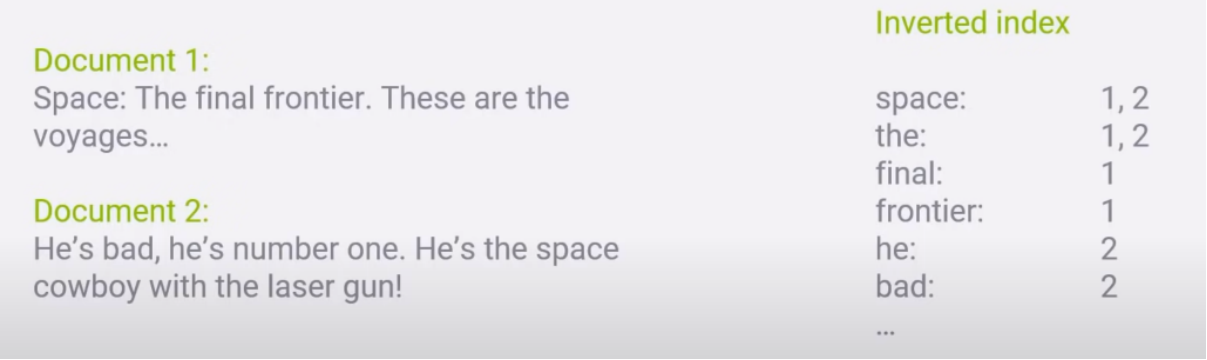
Cómo funciona ElasticSearch

Para poder interactuar con la capa de persistencia, ElasticSearch expone una interfaz REST para poder crear, modificar o eliminar datos. Estos datos se almacenarán como documentos en archivos JSON.

En nuestro caso, tendremos un índice llamado **advertisements** que tendrá el siguiente esquema:

"id": "<id>",
"title": "<título publicación>",
"price": **<precio>**,
"currency": "<usd o uyu>",
"condition": "<new o used>",
"url":"<url publicación original>",
"photoUrl": "<url foto>",
"year": **<año>**,
"brand": "<marca>",
"model": "<modelo>",
"kilometers": **<kms>**,
"ranking": **<ranking>**

ElasticSearch construye **índices invertidos** que permiten hacer búsquedas en todas las advertisements de manera más eficiente.



Construcción de índice invertido

También se almacenan datos de la frecuencia de los términos (Term Frequency) o qué tanto aparece un término en todos los documentos (Document Frequency).

Si hacemos un GET a la siguiente ruta podemos observar estos datos para un documento en particular:

/advertisements/_termvectors/<id>?
term_statistics=true&field_statistics=true&fields=brand


```
"brand": {
    "field_statistics": {
        "sum_doc_freq": 16612,
        "doc_count": 15641,
        "sum_ttf": 16613
    },
    "terms": {
        "nissan": {
            "doc_freq": 1045,
            "ttf": 1045,
            "term_freq": 1,
            "tokens": [
                {
                    "position": 0,
                    "start_offset": 0,
                    "end_offset": 6
                }
```


Estadísticas del campo Brand

Fuzzy Query

La capa de backend construirá las querys que llegarán a la capa de ElasticSearch, estas querys se harán sobre los atributos del índice advertisements, permitiendo también un márgen de error. Este motor de búsqueda permite realizar un fuzziness sobre los términos de la query para matchear con lo que más se parezca.

Por ejemplo, si el usuario ingresa ***Fidt***, la query se interpreta y se ordenarán los resultados por un puntaje, este puntaje es más alto mientras más se parezca. Probablemente lo que nos llegue como resultado serán todos los modelos de la marca Fiat.

Autocompletador sugiriendo posibles búsquedas

Suggester

También se utilizarán las querys de ElasticSearch para implementar un autocompletador en la búsqueda del usuario, esto se llama *Suggester.*

Por ejemplo, si el usuario ingresa **"Fi"**, se realiza una query sobre la base y se traen sugerencias más relevantes como Fiat, Fiat Uno, entre otros, tratando de predecir al usuario y así mejorar la experiencia.

Suggester en funcionamiento

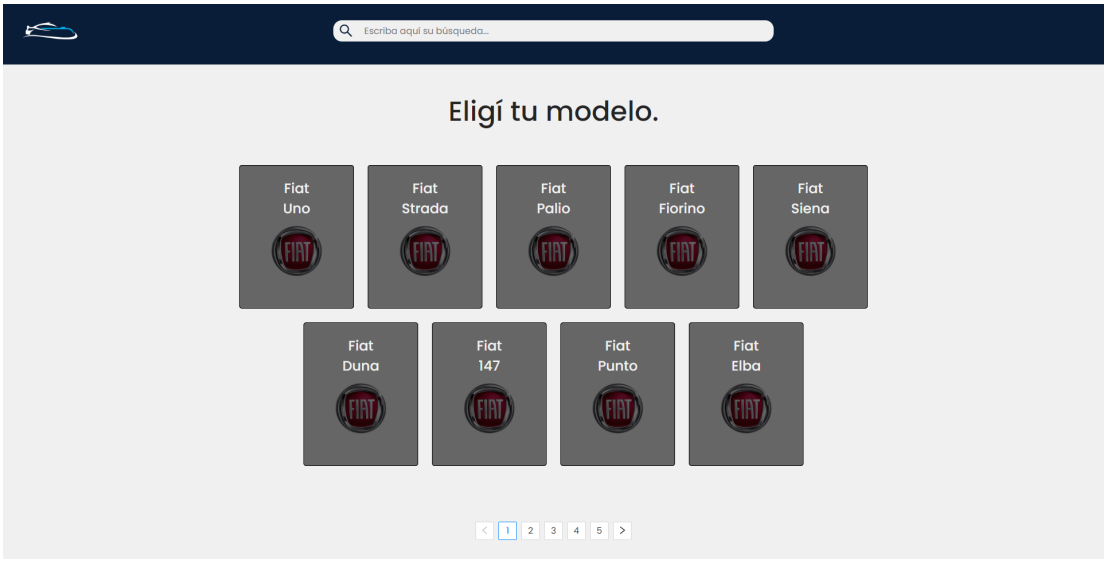
4. Funcionalidades y Uso

Se busca conseguir la mejor usabilidad, un sistema fácil de utilizar, y que devuelva resultados rápidamente.
En primer lugar, el usuario se va a encontrar con la **Pantalla Principal**, donde podrá ingresar una búsqueda que permite flexibilidad, ya que puede ingresar una marca, o un modelo en particular.

Al enviar la búsqueda, el sistema tratará de **interpretar la consulta** del usuario para devolverle lo más adecuado. Esta interpretación se lleva a cabo realizando una búsqueda en un índice separado que se creó donde se guardan las marcas y modelos. Dado el input de usuario se mira el resultado con mejor Score según ElasticSearch y si este es un Modelo se obtiene la marca y modelo normalizado (osea, como está guardado en la BD) y si este es una Marca se obtiene solo la marca normalizada.

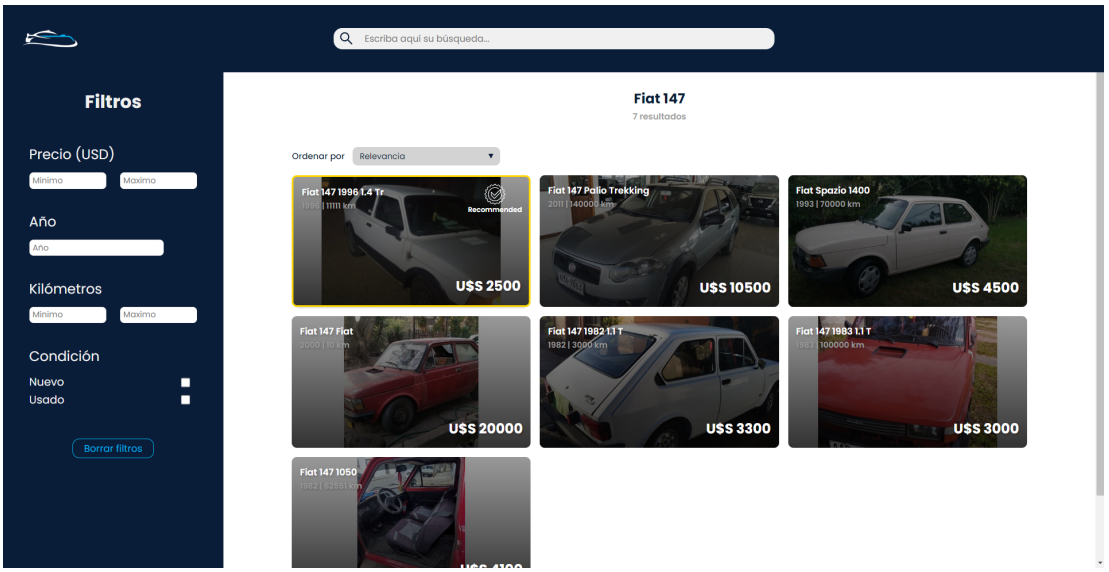
Si el sistema interpreta que la búsqueda del usuario es una marca nada más entonces se enviará al usuario a la **Página de Modelos**, específicamente para esa marca.

En este caso, si el usuario busca **“fiat”**, el resultado será el siguiente:



Resultados obtenidos de la búsqueda “fiat”

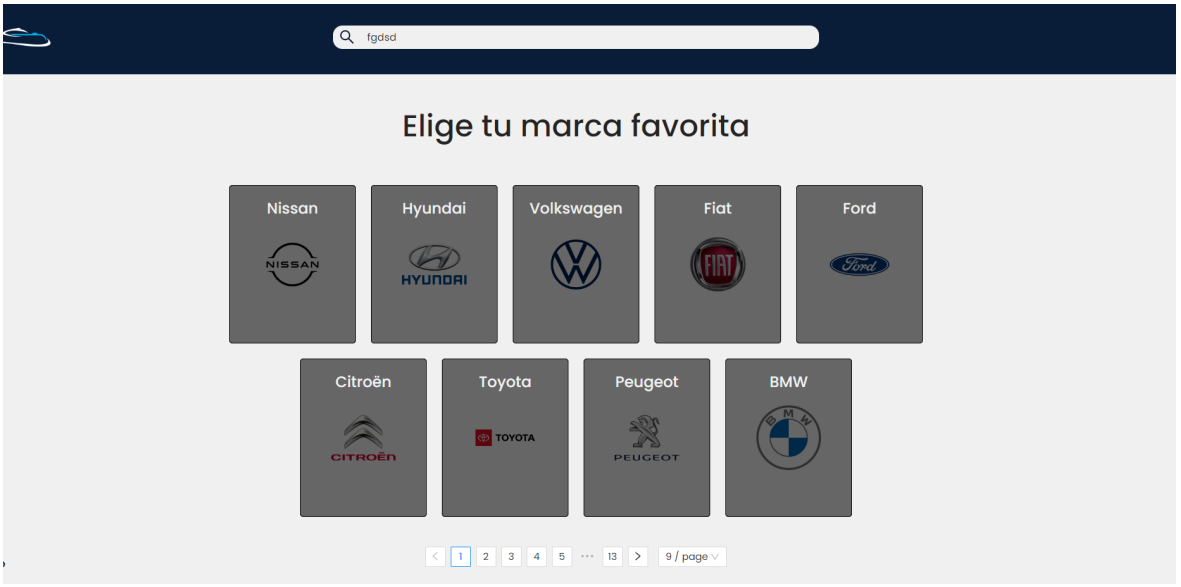
Si el sistema interpreta que la búsqueda del usuario es un modelo en específico, entonces se enviará al usuario a la pantalla de publicaciones para ese modelo.



Resultados obtenidos de la búsqueda “fiat 147”

Si al usuario le interesa alguna de estas publicaciones, puede realizar un click sobre esta y se le redirigirá a la **publicación original.**

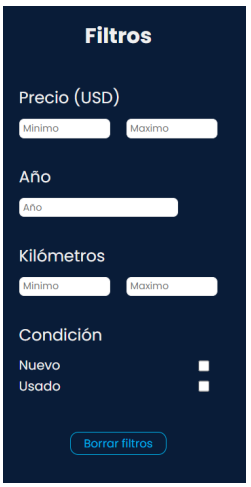
En cualquier otro caso, si el sistema no logra interpretar la búsqueda del usuario como una marca/modelo válido, entonces enviará al usuario a la **Página de Marcas** del sistema.



Resultados obtenidos de la búsqueda "fgdsd"

Filtros

Además se ofrecen distintos filtros para que el usuario pueda filtrar los resultados, acotando aún más la búsqueda. Actualmente el sistema cuenta con un filtro por rango de **kilómetros, precio**, filtrar por **condición** de nuevo o usado y buscar por un **año** en particular.

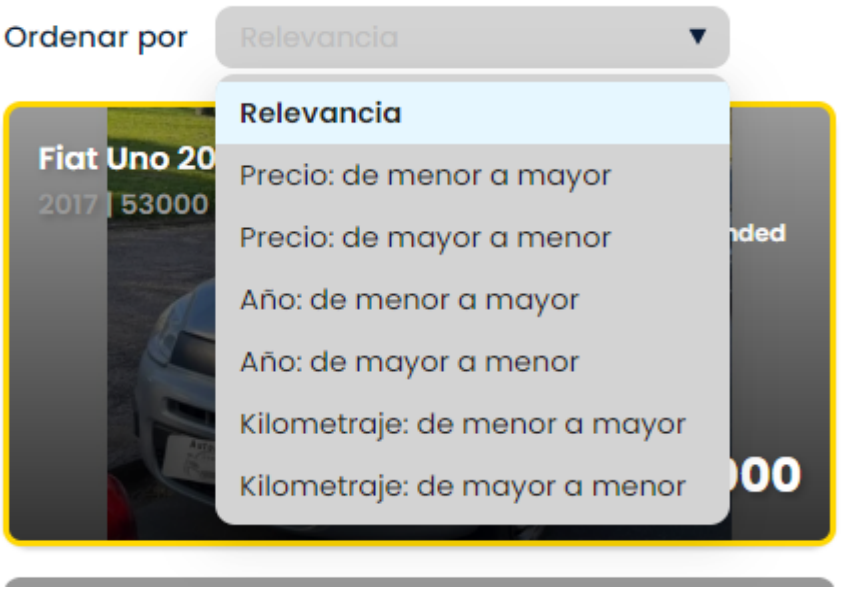


Sección de Filtros

El servidor de backend tendrá en cuenta estos filtros para construir las querys hacia la base de datos.

Ordenamiento
Se ofrece la posibilidad de que el usuario pueda ordenar los resultados de una forma distinta a la que los datos llegan. Por defecto los datos vienen ordenados por una métrica de ranking, pero la relevancia es relativa a los usuarios, y podría interesarle una publicación más que otra solamente basándose en el precio y ningúna otra métrica más.

Damos la posibilidad de que el usuario pueda ordenar los resultados de las siguientes maneras:



Ordenamiento

5. Resultados Obtenidos

Los resultados obtenidos fueron los esperados ya que se cumplieron con el diseño y funcionalidades propuestas al inicio del proyecto.
Logramos desarrollar una aplicación rápida y amigable con los usuarios gracias al diseño moderno de la misma. Esta es modular y extensible

permitiendo la fácil integración de otras fuentes de datos como nuevas características.

6. Conclusiones

La realización de este trabajo nos permitió comprender mucho más de los temas del curso ya que fueron llevados a la práctica. Para construir este sistema se tuvieron en cuenta conceptos como la construcción de índices invertidos, determinar relevancia en distintos documentos, búsqueda en grandes conjuntos de datos, entre otros.

Fue interesante enfrentarnos y resolver distintos desafíos, como el de normalizar los datos que venían de distintas fuentes, o el caso de ElasticSearch, que es una tecnología que no conocíamos y resultó muy importante para el desarrollo de esta aplicación ya que provee procesamiento en los datos que no tienen las bases más típicas o que previamente conocíamos.

La gran cantidad de datos con la que se enfrenta un usuario en la búsqueda de la compra de un vehículo ha aumentado mucho en estos años, es por esto que hay que afinar cada vez más las búsquedas, las recomendaciones, y tener en cuenta más variables para devolverle al usuario los documentos que quiere buscar , logrando así una mejor tasa de documentos relevantes obtenidos. Es por esto que creemos que esta prueba de concepto va por buen camino y sería de gran utilidad para estos usuarios.

Dejamos el código en un repositorio público para que cualquiera pueda continuar desarrollando nuevas funcionalidades.

7. Trabajo Futuro

Consideramos que todos los puntos del sistema tienen algún detalle para mejorar, desde detalles en los visuales, en la lógica o hasta en la capa de persistencia.

Sistema de Recomendaciones
Ahora mismo, el sistema se abstrae de cómo Mercado Libre u otros sitios puntúan las publicaciones, en principio, esto se hizo para que podamos implementar nuestra propia función de ranking, aunque esta es muy básica al lado de las que estos sitios implementan, esto podría mejorar mucho.

Para poder mejorar la calidad del ranking de las publicaciones podríamos tener en cuenta más variables, como por ejemplo, la reputación del vendedor, u otras prestaciones del vehículo.

También para personalizar las recomendaciones en base a un usuario se podrían rastrear las búsquedas del mismo, su navegación en las distintas páginas, u otros factores que nos sirven para crear su perfil en el sistema y poder mejorar las recomendaciones.

Algo más que se podría tener en cuenta es los precios de patente que paga cada vehículo, quizás si SUCIVE expone alguna API, se podría utilizar para completar esta información.

Escalabilidad
El tiempo acotado para el proyecto permitió que podamos alimentar la base de datos sólo con la API de Mercado Libre y CarOne, pero la modularización que se empleó en la arquitectura del sistema permiten que se pueda ampliar e integrar APIs de otros sitios fácilmente.

Además ElasticSearch nos permite una buena escalabilidad en la persistencia construyendo un cluster de nodos con los datos distribuidos por si los datos van creciendo mucho.

Interpretación de las búsquedas
El sistema actual contiene un interpretador de las búsquedas del usuario que es algo básico, hay búsquedas que el sistema no interpreta del todo bien lo que el usuario final desea buscar. Este componente del sistema podría mejorarse para brindar mejores resultados, incluso podría implementarse un ranking general para distintas marcas, en este caso sólo se comparan publicaciones de un mismo modelo.

Obtener más datos de las publicaciones originales

Las publicaciones originales de Mercado Libre u otros sitios suelen tener muchos más datos que pueden ser relevantes para el usuario final y pueden resultar en una búsqueda más exacta. Además, se podrían integrar nuevos filtros para que el usuario pueda acotar lo que busca en base a esos nuevos atributos. Un ejemplo de estos atributos podría ser el tipo de auto, si es Diesel o Nafta, entre otros.

Posibilidad de incorporar otros modelos en la búsqueda

En el estado que se encuentra el sistema se permite únicamente la búsqueda de un modelo en específico. Una futura actualización podría permitirle al usuario incorporar un multi-buscador en donde varios modelos pueden ser ingresados para así poder tener rankeado utilizando el sistema de relevancia pero aplicado a varios modelos.

8. Referencias

React: https://es.reactjs.org/
NodeJs: https://nodejs.org/en/
ElasticSearch: https://www.elastic.co/
API MercadoLibre: https://developers.mercadolibre.com.uy/
Ranking:
https://motorway.co.uk/sell-my-car/guides/how-much-does-mileage-affect-car-value
https://www.ramseysolutions.com/saving/car-depreciation
Repositorio GitHub: https://github.com/facundofleitas/wir-project