Obligatorio Webir

Izquierdo Signorelli, Nicolás
4.731.879-3
Gonzalez Pérez, Christian Marcelo
4.691.743-3
Galiano Mazzoni, Federico Ignacio
5.202.409-8

Índice

1. Introducción

Hoy en día es cada vez más común la compra de vehículos, en especial a través de internet. En la actualidad casi toda persona cuenta, como mínimo, con un smartphone o un dispositivo móvil, haciendo cada vez mas frecuente que los usuarios accedan a esta información a través de plataformas online, siendo MercadoLibre[6] y El Gallito[3] unas de las más frecuentadas.
En el presente documento se abordará el problema de mejorar la experiencia de usuario a la hora de decidir cual vehículo es el indicado según sus necesidades.

2. Problema

Tener una gran cantidad de fuentes distintas a las cuales acceder es un beneficio que hoy en día los compradores de vehículos así como de cualquier producto pueden darse. Sin embargo, cuando un usuario está buscando vehículos a través de su dispositivo móvil tiene la dificultad de tener que cambiar constantemente de app o sitio web para poder buscar, comparar precios, modelos, etc. Ya que los productos no están centralizados en un solo lugar.
Esto hace que la búsqueda se dificulte o sea tediosa, causando que los usuarios no puedan encontrar un vehículo que se ajuste a sus necesidades, o peor aún, que terminen adquiriendo uno el cual no es el más indicado.

3. Enfoque de la solución

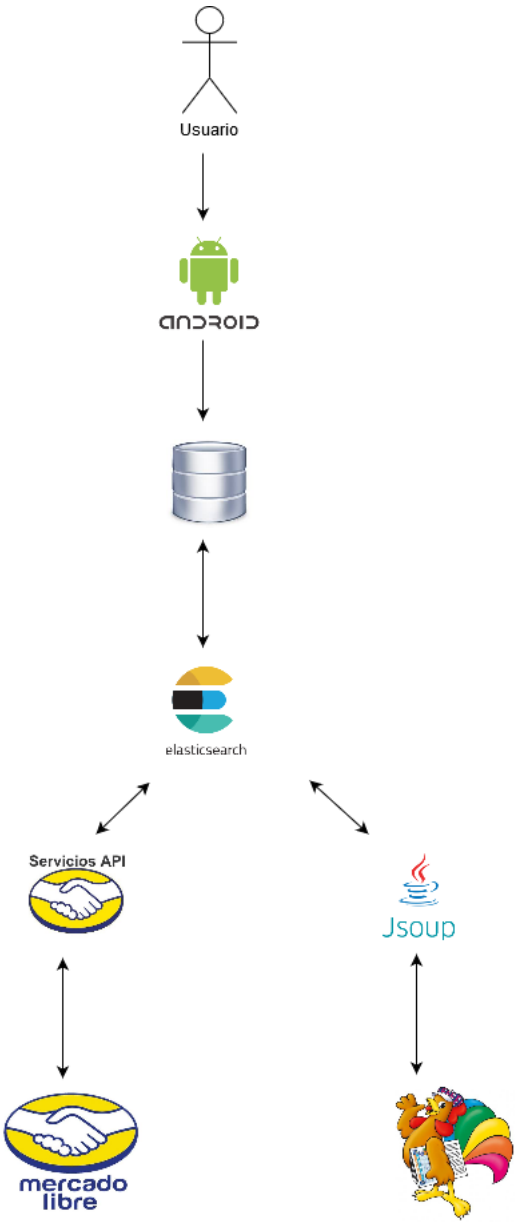
El enfoque que se plantea en esta solución, es el desarrollo de una aplicación móvil en Android que sirva como herramienta para recopilar información de venta de vehículos desde los sitios web que creemos son los más frecuentes (Mercadolibre y El Gallito) y mostrar los resultados de forma transparente al usuario en una aplicación que sea intuitiva, simple y con una interfaz amigable. Además la aplicación permitirá filtrar los resultados y realizar búsquedas sobre ellos permitiendo así al usuario cubrir un espectro más grande de vehículos en su búsqueda y brindado le una mejor experiencia.

La aplicación móvil hará peticiones mediante el protocolo REST a un servidor el cual le devolverá la información correspondiente a la búsqueda realizada por el usuario, en estas peticiones el usuario podrá ingresar filtros de

búsqueda tales como precio, marca, modelo, etc. Este servidor accederá a su base de datos documental, implementada con ElasticSearch[2], y finalmente le devolverá el resultado a la aplicación. Para cargar con datos la base de Elasticsearch, se utilizó la api de MercadoLibre y se realizó scraping sobre la web de El Gallito utilizando la biblioteca de Java JSOUP[5].

La aplicación móvil desarrollada será la encargada de hacer la búsqueda por nosotros, siendo un meta-buscador que recopile información de los dos sitios y nos brinde un conjunto de vehículos según los filtros especificados por los usuarios. Esto mejora enormemente la experiencia de usuario en lo que la búsqueda de vehículos se refiere, reduciendo el tiempo invertido en esta acción. La ventaja es que no necesita ir a varios sitios en forma paralela para comparar precios de vehículos, sino que tiene los vehículos referentes a todos los sitios en un mismo lugar y puede resolver la comparación de manera mucho más eficiente.

4. Diseño e Implementación



El sistema consta de una arquitectura cliente-servidor, donde la aplicación movil se comunica mendiante el protocolo REST con el servidor y el servidor

tiene acceso a la base de datos de ElasticSearch. El diseño e implementación de la aplicación se ha dividido en distintas etapas, cada uno de los cuales se encarga de un problema en particular. A continuación se presenta un esquema general de la solución y una breve descripción de cada una de las etapas:

4.1. Recuperación Información en la Web

En esta etapa se cargan los vehículos de los sitios web de Mercado Libre y el Gallito. En esta etapa se enfrentó el problema de cargar datos de distintas fuentes, los cuales tenían distinta estructura.
Los datos extraídos de MercadoLibre venían en un formato debidamente documentado en el sitio de su API[7], sin embargo los obtenidos de el sitio de El gallito fueron fabricados con los datos mostrados en su sitio web, los cuales se tuvieron que procesar uno a uno. Lo que se intentó es llegar a que ambas fuentes brindaran datos siguiendo una misma estructura.
En resumen, la carga de vehículos por parte de Mercado Libre se realizó mediante su API y la carga de vehículos por parte de El Gallito se realizó mediante la técnica de web scraping utilizando Jsoup. Luego se unificaron ambos conjuntos de datos y se volcaron en la base de datos ElasticSearch

4.1.1. ¿Qué es Web Scraping?

Web scraping es la técnica que se utiliza para extraer información de páginas web de forma automática. Consiste en leer el código de una página para obtener datos en bruto y transformarlos en datos estructurados que se pueden guardar en bases de datos u hojas de cálculo para analizar y extraer aquello que nos interesa. Esta técnica se puede utilizar para recolectar datos de forma masiva con lo que podríamos decir que es parte de lo que conocemos como Big Data.

4.1.2. ¿Qué es JSOUP?

JSOUP es una biblioteca Java para facilitar el Web Scraping mediante HTML. Proporciona una API para extraer y manipular datos, utilizando lo mejor de DOM, CSS y métodos similares a jquery. A continuación se muestra un pequeño ejemplo de uso:

```
Document doc = Jsoup.connect("http://en.wikipedia.org/").get();
log(doc.title());
```

```
Elements newsHeadlines = doc.select("#mp-itn b a");
for (Element headline : newsHeadlines) {
    log("%s\n\t%s",
    headline.attr("title"), headline.absUrl("href"));
}
```

El objetivo de la biblioteca Jsoup es proveer distintas herramientas que faciliten la recuperación de información en la web, y para esto provee distintas clases que serán cruciales para la recuperación de datos.
La más importante es la clase Document. En una variable del tipo Document guardaremos el HTML obtenido de hacer un Jsoup.connect a una página específica.
Teniendo el document con el HTML, ya tenemos toda la información y simplemente tenemos que navegar dentro de los div html para obtener la información que nos sea relevante.
Para obtener la información requerida utilizamos funciones de JSOUP como por ejemplo:

```
Elements precios = ele5.select("strong")
Elements imagenes=
document.getElementsByClass("img-responsiveaviso-ico-contiene")
```

En estos ejemplos el tipo Elements nos es de gran utilidad para poder guardar un array de elementos del tipo Element, que van a contener todos aquellos elementos que cumplan con lo requerido, ya sea un div que tenga un cierto nombre de clase o todas las etiquetas que tengan el prefijo strong.
Como se puede apreciar, una vez que tenemos los patrones de como se encuentran dispersos los datos en una pagina especifica es muy sencillo obtener los datos requeridos, simplemente buscándolos mediante su nombre de clase, tipo de etiqueta o mediante expresiones regulares.

4.2. ElasticSearch

Motor de búsqueda de texto completo, distribuido y con capacidad de Stemming y Fuzzy matching para mejorar la precisión de los filtros sobre las publicaciones con errores lexicográficos. Elasticsearch es un servidor de búsqueda basado en Lucene. Provee un motor de búsqueda de texto completo, distribuido y con capacidad de multi-tenant con una interfaz web RESTful

y con documentos JSON. Elasticsearch utiliza Query DSL (Lenguaje de dominio específico) para realizar las consultas a los documentos indexados. Es un lenguaje sumamente flexible y de gran alcance, además de simple, que permite conocer y explorar los datos de la mejor manera. Al ser utilizado a través de una interfaz de tipo JSON, las consultas son muy sencillas de leer y, lo más importante, de depurar. Elasticsearch está desarrollado en Java y está publicado como código abierto bajo las condiciones de la licencia Apache.
Ventajas de usar ElasticSearch:

- Multi-tenant de datos. Nos permite operar sobre distintos índices al mismo tiempo y así potenciar nuestras búsquedas.
- Acceso en tiempo real. Esta tecnología nos permite acceder de forma instantánea a los datos.
- Búsqueda de texto completo. Usar Elasticsearch hace que implementen una gran cantidad de funciones, tales como la división personalizada de texto en palabras, derivación personalizada, búsqueda personalizada, etc.
- Consultas complejas y afinación. Elasticsearch tiene una poderosa DSL basada en JSON, que permite a los equipos de desarrollo construir consultas complejas y afinarlas para recibir los resultados más precisos de una búsqueda. También proporciona una forma de clasificar y agrupar los resultados.
- Uso de facetas. Una búsqueda facetada es más sólida que una búsqueda de texto típica, lo que permite a los usuarios aplicar una cantidad de filtros a la información e incluso tener un sistema de clasificación basado en los datos. Esto permite una mejor organización de los resultados de búsqueda y permite a los usuarios determinar mejor qué información necesitan examinar.
- Escalabilidad horizontal y registro de nodos. Elasticsearch permite escalar horizontalmente, por lo que, gracias a su diseño, permite extender los recursos y equilibrar la carga entre los nodos de un cluster. Además, registra cualquier cambio realizado en registros de transacciones en múltiples nodos en el cluster para minimizar la posibilidad de

pérdida de datos. Por otro lado, estos cluster pueden detectar aquellos nodos que fallan y reorganizarlos para que los datos siempre sean accesibles.

4.3. Servidor de aplicaciones

El sistema encargado de escuchar las peticiones de la aplicación android fue implementado en JAVA. Se utilizó como servidor Wildfly-18.0.0.[8]. El mismo implementa un Web Service REST mediante la librería Jersey[4]. Recibe como entrada los filtros mediante los cuales se desea hacer la búsqueda de vehículos (Km, tipo de moneda, condición, precio, marca, etc). Una vez recibido estos datos se comunica con ElasticSearch, el cual busca los vehículos que cumplan los criterios deseados, procesa información extra de ser necesario y retornar la lista final de vehículos a la aplicación Android.

4.4. Aplicación Android

La aplicación fue desarrollada en Android Studio[1].
Para desarrollar el front-end de la aplicación se utilizaron los siguientes componentes:

- RecyclerView. Contenedor de elementos (listas) y sucesor del componente ListView.
- CardView. Implementación que nos proporciona Google del elemento visual en forma de tarjetas de información.
- Adaptadores. Acerca el modelo de datos para ser mostrados mediante un LayoutManager el cual será el responsable de posicionar cada ítem dentro del RecyclerView y de decidir cuándo reciclar las vistas de items que ya no son visibles.

Para desarrollar el back-end se utilizaron los siguientes componentes:

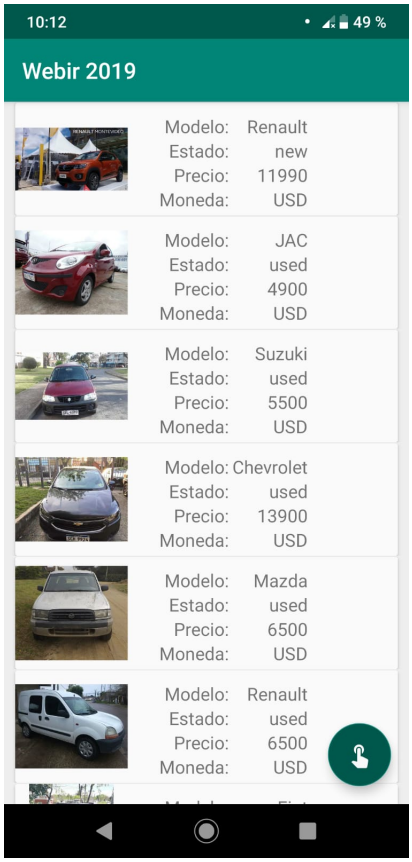
- Retrofit2. cliente REST para Android y Java, desarrollada por Square. Permite hacer peticiones GET, POST, PUT, PATCH, DELETE y HEAD, gestionar diferentes tipos de parámetros y parsear automáticamente la respuesta a un POJO.

- Dagger. Librería, actualmente de Google, para inyección de dependencia (DI). Permite modularizar la creación de objetos y encapsular sus instancias.
- RX. API que facilita el manejo de flujos de datos y eventos, a partir de una combinación de el patrón Observer, el patrón Iterator, y características de la Programación Funcional.

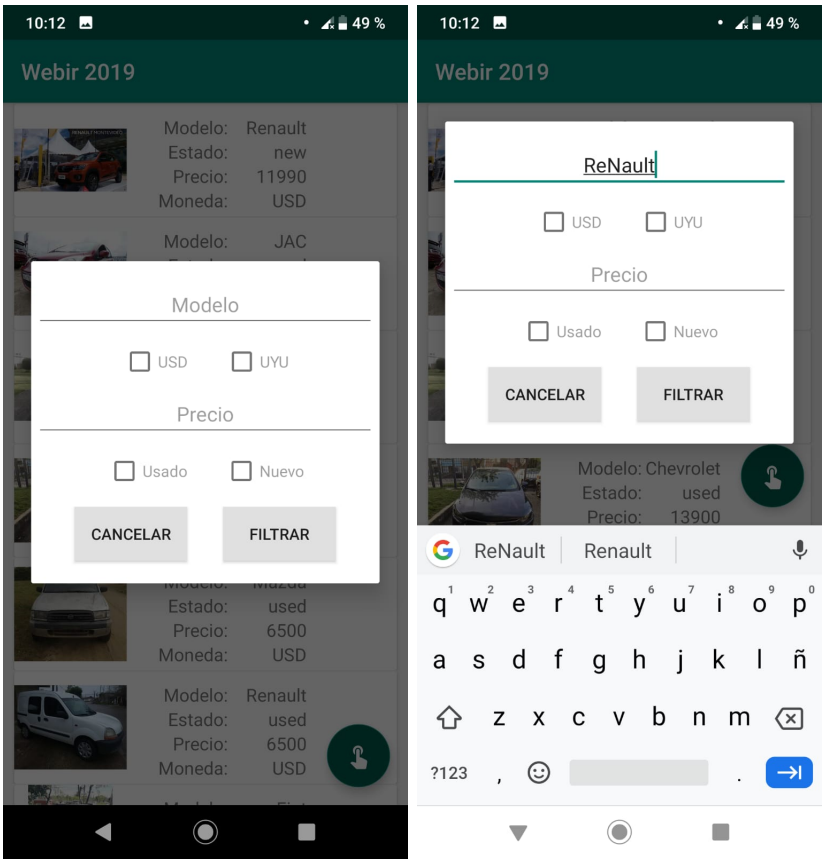
5. Evaluación, funcionalidades y uso

Se opto por un diseño minimalista, elegante y de fácil uso para que todo tipo de usuario sea capaz de utilizar satisfactoriamente la aplicación y todas las ventajas que la misma ofrece.

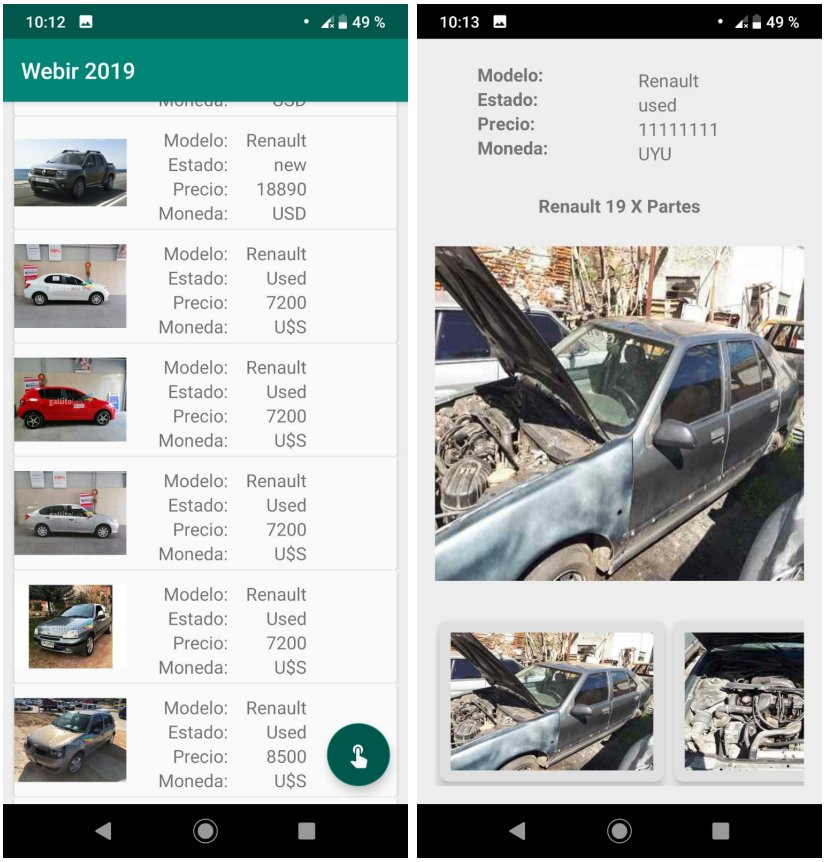
Una vez el usuario abre la aplicación se le despliega una lista de todos los vehículos.



Presionando el botón que se encuentra en la esquina inferior derecha de la pantalla se despliega un diálogo en pantalla el cual contiene los filtros para realizar una búsqueda personalizada.



Una vez elegido los filtros de búsqueda y enviada la petición al servidor se retorna la lista de vehículos y se muestran en CardViews. Cada CardView contiene una pequeña imagen del vehículo e información básica del mismo. Al tocar sobre un vehículo se accede a un menú con información más detallada del mismo, asi como la galería de imágenes proporcionada por el vendedor.



Para volver al menú con la lista de vehículos, simplemente se presiona en el centro de la galería de imágenes o en el botón para retroceder.

6. Conclusiones

Unificar la información obtenida de distintos sitios web de vehículos, nos brindó una visión más completa de la oferta que se tiene en la web. Si bien tomó un tiempo desarrollar la solución y procesar los datos obtenidos, el resultado fue una aplicación que mejora la experiencia de usuario a la hora de buscar un vehículo acorde a sus necesidades.
Se puede agregar que el uso de ElasticSearch proporciona una gran capacidad de búsqueda y filtrado de información, además de una interfaz de acceso amigable y simple de utilizar, y una buena sinergia con Java, el lenguaje de

programación utilizado.

7. Trabajo futuro

Sin duda agregar nuevos filtros a las búsquedas podría ser provechoso, por ejemplo color de auto, tipo de trasmisión, combustible, etc. Por temas de tiempo y complejidad, los filtros aplicados actualmente fueron acotados.
Sería muy rico para este proyecto que se añadan nuevas fuentes de datos de vehículos, esto brindaría una mejora en la experiencia de usuario, dándole a este más opciones entre las cuales elegir, esto es muy sencillo teniendo conocimiento de la librería JSOUP, ya que el proceso de recuperación de información mediante esta librería es muy mecánico, donde simplemente debemos buscar los patrones correspondientes a cada pagina web de la cual deseemos extraer información.
Nuestro prototipo solamente carga los datos una única vez, entendemos que la carga de datos en la base de la aplicación debería ser automatizada y se debería refrescar periódicamente para que los usuarios siempre accedan a datos actualizados.
Consideramos de interés también que los usuarios pudieran incluso comprar o contactar con los vendedores de vehículos mediante la aplicación, esto podría realizarse si se implementara una conexión entre la aplicación móvil y las fuentes de datos, evidentemente está por fuera del alcance del proyecto pero es sin duda realizable si se le destina el esfuerzo correspondiente.
Finalmente se podría agregar un sistema de login en el cual un usuario logueado pudiera guardar vehículos favoritos o mantener su historial de búsquedas, si esto se implementara habría que tener en cuenta la seguridad de la aplicación ya que los datos de los usuarios se podrían ver gravemente comprometidos.

Referencias

[1] *Android Studio*. https://developer.android.com/studio.

[2] *ElasticSearch*. https://www.elastic.co/es/.

[3] *Gallito*. https://www.gallito.com.uy/autos/automoviles.

[4] *Jersey*. https://eclipse-ee4j.github.io/jersey/.

[5] *JSoup*. https://jsoup.org/.

[6] *MercadoLibre*. https://autos.mercadolibre.com.uy/.

[7] *Mercadolibre API*. https://developers.mercadolibre.com.uy/.

[8] *WildFly*. https://wildfly.org/.