Universidad de la República

Facultad de Ingeniería

Recuperación de Información y Recomendaciones en la Web

Grupo 2 - 2024

Bruno Ferrando, Enrique Rodriguez, Franco Ferrari,
Santiago Cuadrado

Montevideo, 1 de julio de 2024

Índice general

1. Introducción

A lo largo de este proyecto intentaremos aplicar técnicas de recuperación de datos (más específicamente scrapping) para crear una aplicación que permita la visualización de datos de múltiples fuentes, de forma estructurada y que resulte útil y de fácil acceso para el usuario final. La aplicación que crearemos intentará ser una fuente centralizada de eventos, espectáculos y actividades que se encuentran disponibles en diferentes webs de venta de tickets del Uruguay, permitiendo al usuario poder comparar diferentes opciones desde un solo lugar, teniendo la información estructurada y una UI amigable.

El informe se compone de diferentes secciones: en la sección 2 definimos el problema a resolver a lo largo del proyecto, en la sección 3 veremos el enfoque y objetivo de la resolución de dicho problema y los aportes que el mismo tiene a los usuarios finales, en la sección 4 mostraremos la arquitectura del sistema, incluyendo componentes y tecnologías utilizadas, en la sección 5 paramos un minuto para ver en mas detalle el reto que nos presento la obtención y manejo de los datos, en la sección 6 describiremos las principales funcionalidades desarrolladas en el sistema. Para cerrar tenemos las secciones 7, 8 y 9 donde vemos los resultados obtenidos, las conclusiones del trabajo y lo que creemos puede ser un buen trabajo a futuro.

2. Problema

Hoy en día existen muchas páginas web que ofrecen tickets para eventos (por ejemplo: RedTickets, TickAntel, etc), cada web tiene su propia estructura y sus propios eventos, por lo que cuando un usuario desea analizar si hay algún evento de interés para él, debe navegar a través de las diferentes páginas web, entendiendo el funcionamiento de cada una y procesando/reteniendo su información. Además al estar constantemente navegando entre un sitio web y otro, es posible que se dificulte la tarea de comparación de datos básicos como pueden ser fechas, precios, lugares, etc.

3. Enfoque de la solución

La solución planteada se centra en crear una aplicación mobile que sea capaz de centralizar los datos de algunas páginas web de venta de tickets. La idea es realizar un scrapping sobre estás páginas de modo de obtener cierta información que todos los eventos tienen en común: nombre, fecha, lugar, página de venta, etc.

Además se desplegaran todos los eventos en un mapa para que el usuario pueda ver qué eventos hay cerca de él o qué eventos hay en algún lugar específico, permitiendo también realizar ciertos filtros sobre los eventos (por ejemplo filtrar por evento).

Como mencionamos anteriormente, el foco principal de la solución es la centralización de datos para evitar trabajo extra en la búsqueda de eventos por parte del usuario, también acompañado de una UX/UI que permita una experiencia agradable para el mismo.

4. Arquitectura de la solución

La arquitectura de la solución se basa en la generación de 4 componentes a nivel general. En primer lugar generamos scrapers desarrollados con utilizando la librería selenium. El objetivo de los mismos es obtener los datos asociados a los eventos en distintas plataformas. Los mismos obtendrán los datos encontrados en dichas plataformas y posteriormente enviara los datos a nuestro segundo componente, una API Rest para el almacenado de la informacion en una Base de Datos MySQL.

Nuestra API Rest fue desarrollada con Javascript utilizando el framework express y es ejecutada sobre el runtime de node. La misma es la responsable del almacenado de los datos obtenidos del scraper mencionado anteriormente y además tiene el rol de servir los datos a los clientes.

Finalmente para lo que es nuestro frontend, responsable de la interacción con el usuario y el consumo de la información obtenida, tenemos una aplicación mobile desarrollada utilizando React Native.

Mencionamos además que cada componente de backend, API, scrapers y Base de datos serán corridos utilizando docker. A su vez el backend en su totalidad sera posible de iniciar conjuntamente utilizando docker-compose y las imágenes generadas por los docker individuales de cada componente. De esta forma se facilita la ejecución de nuestra solución en cualquier ambiente donde se quiera aprovechar.

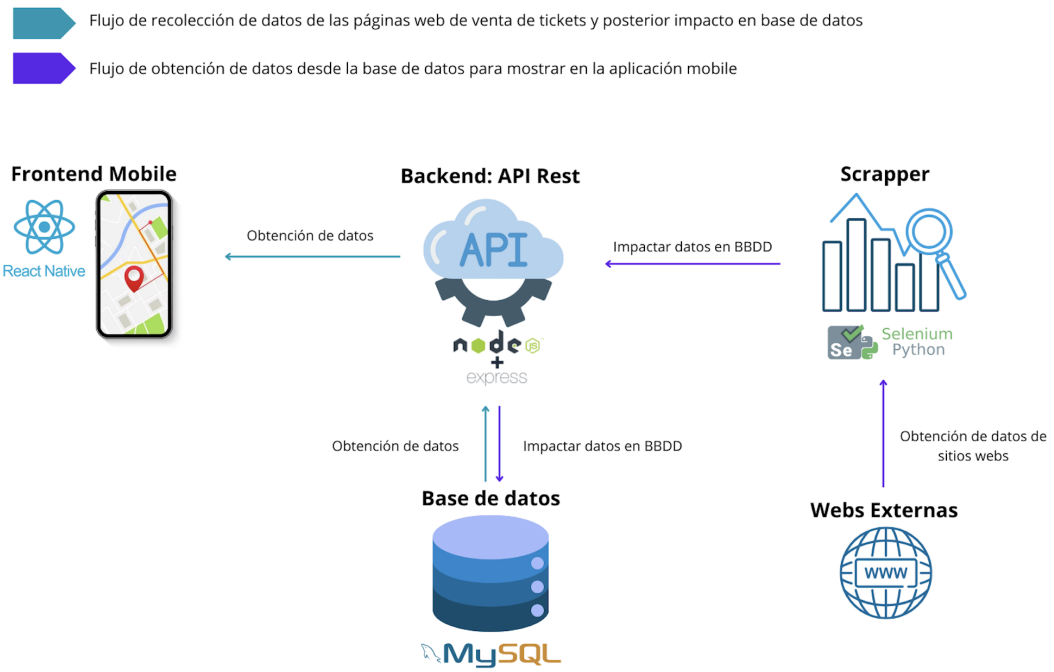

Figura 4.1: Arquitectura de la solución

5. Obtención e integración de datos

Creemos relevante realizar una sección específica para comentar la obtención e integración de los datos debido a que es el mayor problema que tuvimos que resolver y un punto clave para el funcionamiento del sistema. La obtención de los datos se realizó utilizando las bibliotecas selenium y beautiful soup.

Figura 5.1: Categorías presentes en RedTickets

Figura 5.2: Categorías presentes en Tickantel

El enfoque adoptado para obtener la información en ambos casos fue el mismo: se ingresó a todos los eventos desde cada categoría que ofrecían las páginas web de eventos. Dicho esto, surgió el primer problema, que fue la diferencia entre las categorías, como se puede apreciar en las figuras 5.1 y 5.2. En este caso, al tener categorías diferentes entre RedTickets y Tickantel, la solución fue simplemente unificar todas las categorías, quedando así categorías con eventos provenientes de una única página de eventos.

En RedTickets, la obtención de eventos fue bastante lineal, entrando por cada categoría e ingresando a cada elemento del paginador. Por otro lado, en Tickantel esto no fue así; la diferencia fue que la web cargaba eventos dinámicamente a medida que se iba haciendo scroll down, lo que hizo necesario simular este comportamiento para capturar todos los eventos y provocando que el scraper demorara más tiempo en la ejecución. Con esto en mente, se buscó la forma de agilizar este proceso para mejorar los tiempos. Esto fue posible con el uso de la funcionalidad de Python *ThreadPoolExecutor*. Lo que se implementó fue dividir la tarea del scraper, permitiendo que se ejecutara la tarea en paralelo. En este caso, existía un thread por cada categoría, lo que resultó en una disminución de los tiempos de ejecución de cerca del 50 %.

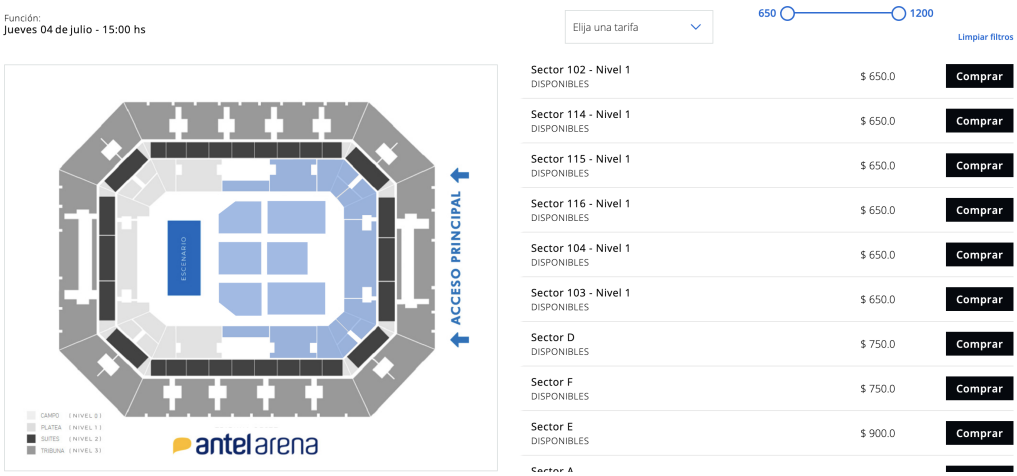

Figura 5.3: Diferencia de precios según el sector en Tickantel

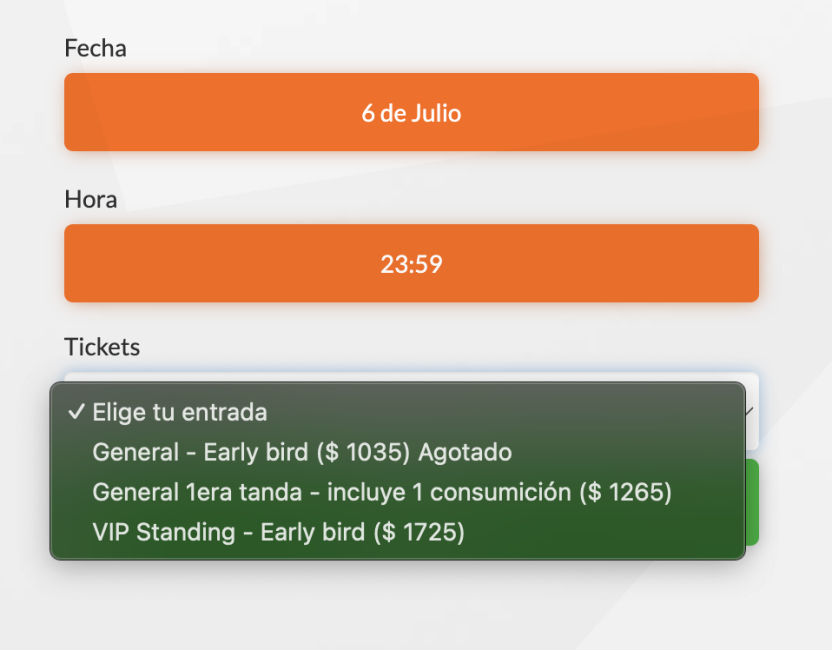

Figura 5.4: Diferencia de precios según la tanda en RedTickets

En lo que respecta a la calidad de los datos, ambas webs presentaban bastante información por cada evento, aunque hubo algunas excepciones. En las dos páginas fue difícil la obtención de los precios de los eventos. Esto se debe a la variedad de precios que tiene un evento, dependiendo del sector o tanda/lote

en el que se encuentra, como se puede apreciar en las Figuras 5.3 y 5.4 respectivamente. En este caso, se optó por no tomar los precios dentro de la información que se obtenía. En lo que respecta específicamente al *scrapeo* de RedTickets, no se tuvieron mayores problemas. En cambio, en Tickantel se presentaron dos problemas: primero, no se contaba con la descripción de los eventos, por lo tanto, estos quedaron sin dicha información. El segundo problema fue que al *parsear* la URL de la ubicación, los valores de latitud y longitud estaban intercambiados en lo que respecta al enlace de Google Maps que utiliza RedTickets. Este problema no pasó a mayores ya que simplemente fue necesario intercambiar dichos valores.

Luego de obtener los datos asociados a los eventos de cada plataforma, se envían a nuestra API para su almacenado utilizando una interfaz en común que implementa cada scraper. Esto permite que a futuro se puedan agregar nuevos scrapers asociados a distintas plataformas que puedan integrarse con el sistema de manera transparente y con una extrema facilidad.

6. Funcionalidades y uso

A continuación numeramos las distintas y principales funcionalidades que se lograron desarrollar.

6.1. Integración continua de datos

Un reto no menor a la hora de la integración de información es contar con información actualizada y de utilidad para los usuarios. Es por esto que se decidió implementar de lado del backend un *cron* o un *job* para la ejecución automatizada de los scrapers del sistema. Esto permite tener un sistema más robusto ya que el mismo sistema se encarga de obtener automáticamente nueva información sobre los eventos. La periodicidad de nuestro *job* es configurable para permitir ajustar el mismo a los distintos tiempos que pueda tener el ambiente donde se aloja.

6.2. Diseño general de la aplicación

La aplicación cuenta con tres pantallas y un menú en la parte inferior que permite desplazarse entre ellas: Las pantallas en cuestión son: un mapa, una pantalla con eventos cercanos y una última con filtros de búsqueda. El menú tiene el diseño mostrado en 6.1:

Figura 6.1: Menú de la aplicación

6.3. Mapa de eventos

La aplicación cuenta con una pantalla que permite visualizar los eventos en un mapa en su ubicación exacta. Además, si el usuario habilitó su ubicación personal para el sistema también se muestra la distancia del evento a la ubicación actual del usuario. Esta pantalla también cuenta con filtros de categorías en la parte superior con un scroll horizontal. En 6.2 se muestran imágenes representativas de dicha pantalla (donde cada marcador representa un evento):

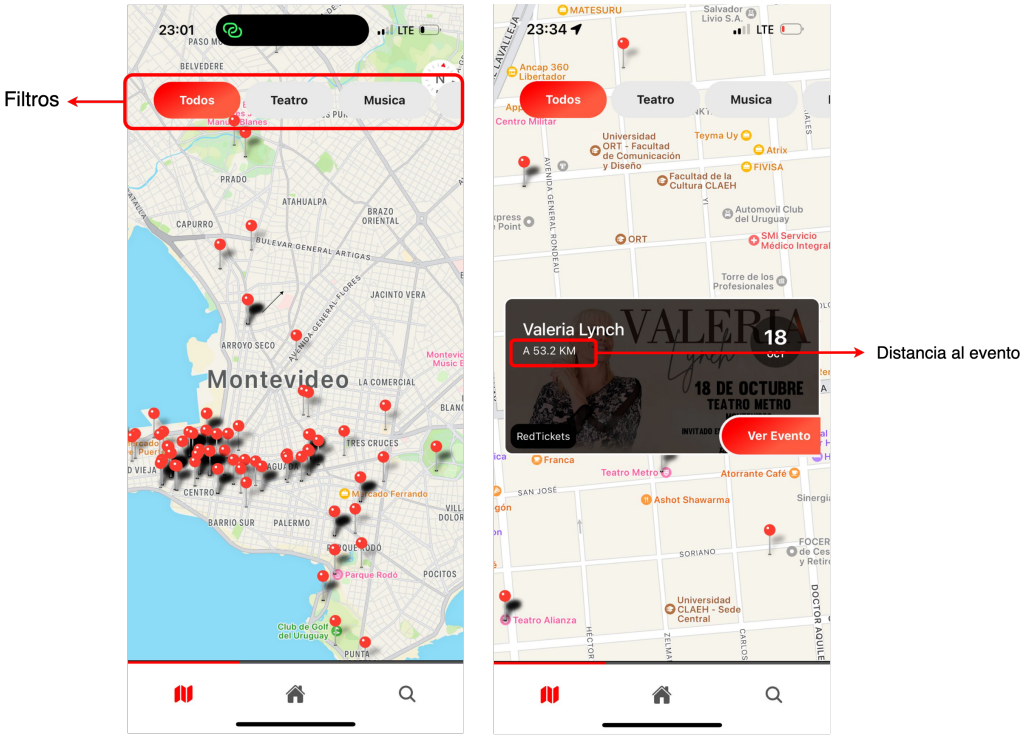

Figura 6.2: Pantalla con vista en mapa

6.4. Buscador con filtros

La aplicación posee una pantalla de búsqueda, cuya idea es permitirle al usuario utilizar filtros más específicos para encontrar los eventos que desea. Esta pantalla actualmente y dado el alcance del proyecto, solo cuenta con los filtros por categoría y un buscador por título del evento, aunque se tiene la información necesaria del lado del frontend para agregar los filtros por distancia y fechas.

En 6.3 se muestra una captura de dicha pantalla:

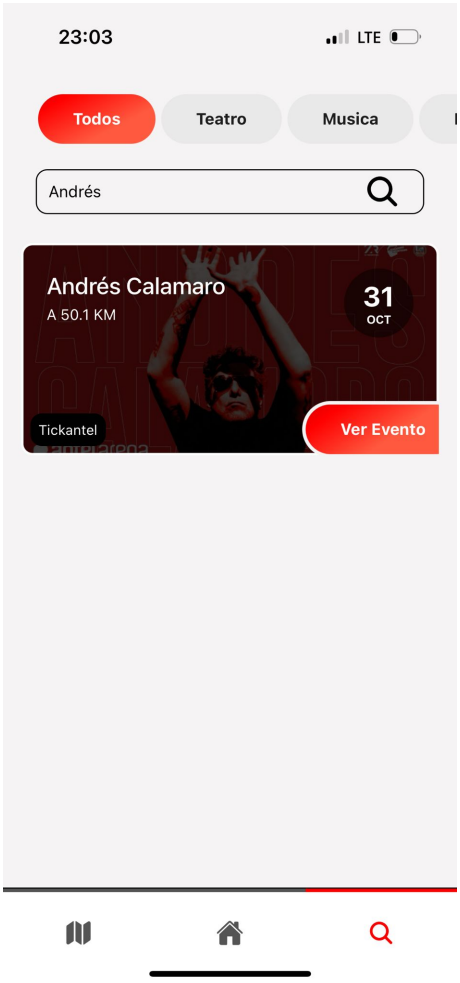

Figura 6.3: Pantalla de búsqueda

6.5. Ordenamiento por distancia

La aplicación cuenta con un "Home.[en] el cual se listan los eventos ordenados por distancia al usuario. La idea de esta pantalla es que el usuario tenga acceso a los próximos eventos cercanos a él de forma rápida al ingresar a al aplicación. En 6.4 se muestra una captura de dicha pantalla donde se puede apreciar que los eventos que se listan al comienzo tienen una distancia reducida (52/63 metros).

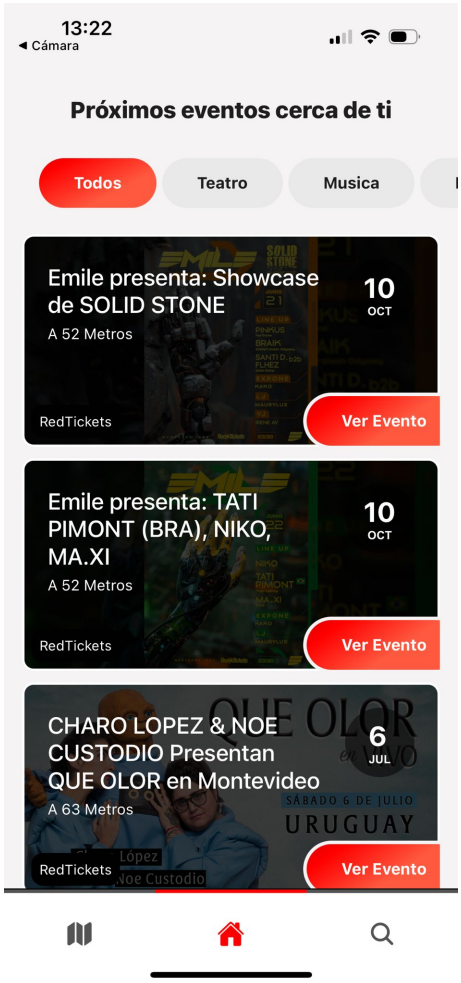

Figura 6.4: Eventos cercanos al usuario

6.6. Información básica de los eventos

Cada evento cuenta con información básica del mismo almacenada en la base de datos. Cada evento posee específicamente: la url del evento, la url de su imagen, su título, su descripción, su latitud, su longitud y sus fechas. Cada tarjeta tiene como imagen de fondo la imagen del evento. En las mismas se muestra el título, la/las fechas del evento, la distancia del usuario al evento, la fuente del evento (redtickets o tickantel) y un botón que lleva al detalle del evento. Toda esta información se muestra en 6.5

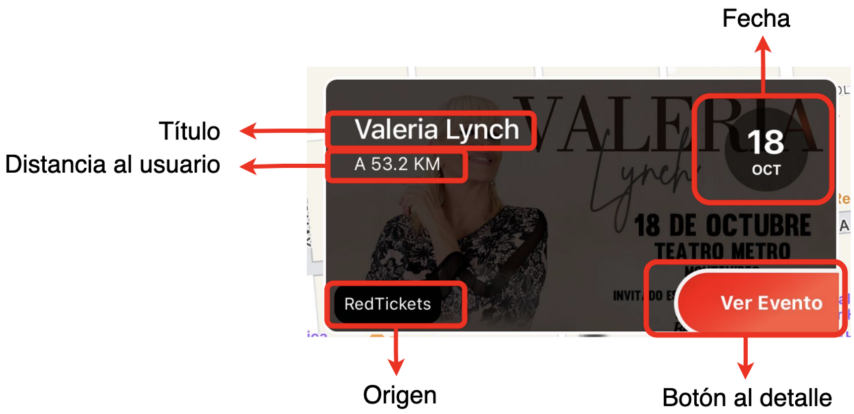

Figura 6.5: Información básica del evento

Además, como se puede ver en 6.6, si vamos al detalle del evento podemos ver la descripción del evento, la imagen del evento sin el filtro opaco, el mapa con su ubicación y también tenemos un botón que nos lleva a la página del evento para comprar los tickets.

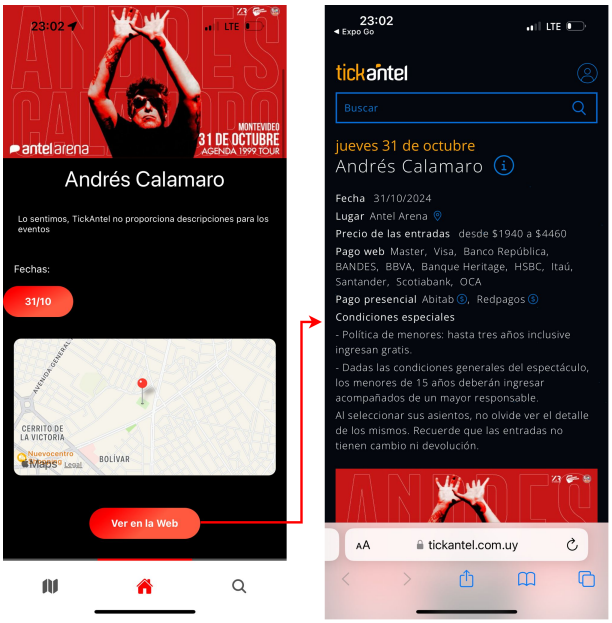

Figura 6.6: Detalle del evento

7. Evaluación y resultados

Luego de finalizado el scrapping inicial de los sitios de *TickAntel* y *Red Tickets* se obtuvieron más de 500 eventos cercanos a la ciudad de Montevideo. Los mismos se obtuvieron de manera correcta logrando recolectar toda la información necesaria para mostrarle al usuario dado el alcance del proyecto.

Además creemos que se logró un buen UX/UI dado que llegamos a un diseño sencillo e intuitivo para el usuario que logra desplegar de forma correcta toda la información necesaria para el mismo.

8. Conclusiones

Entendemos que se logro realizar un trabajo satisfactorio, donde se pudo indagar en técnicas de scrapping que el equipo no conocía. El resultado final es una aplicación que ayuda a los usuarios finales simplificando la búsqueda de eventos que pueden ser de interés para los mismos brindando herramientas como el buscador, los filtros y el mapa, permitiendo una búsqueda global sobre los sitios escrapeados, llevando al usuario a una única aplicación donde buscar eventos, centralizando la información de sitios específicos.

Vale la pena mencionar que la decisión de tener la solución dockerizada hace aún mas robusto el desarrollo y la extensión de la misma a la hora de agregarle funcionalidades ya sea a nivel de aumentar la cantidad de información con nuevos scrapers o más funcionalidades finales para los usuarios.

Dicho esto, durante el proceso de desarrollo se detectaron posibles mejoras, algunas de las mismas serán mencionadas en la sección posterior.

9. Trabajo a futuro

Si bien estamos muy contentos con el trabajo final, se identificaron mejoras que se podrían realizar a futuro con el objetivo de ampliar y mejorar la solución planteada. Algunos de los mismos son los siguientes:

- Mas plataformas admitidas
- Crear aplicación Web
- Agregado de funcionalidades adicionales
  - Perfil de usuario
  - Compras directas en plataforma
  - Histórico de búsqueda
  - Lista de deseos

Referencias

- https://hub.docker.com/
- https://hub.docker.com/_/mysql
- https://nodejs.org/en
- https://selenium-python.readthedocs.io/
- https://reactnative.dev/
- https://expressjs.com/