# RECUPERACIÓN DE INFORMACIÓN Y RECOMENDACIONES EN LA WEB

Informe – Componentes.uy


## Autores

Grupo 8:

Ignacio Vigna – 5030465-4

Darwin Araujo – 4740894-6

Juan Ignacio Betarte – 4856012-7

Pablo Hernández – 5144902-9

# 1. Introducción

Actualmente existen muchas tiendas de componentes de hardware de PC en Uruguay. Cada una con distintos precios, y dependiendo del producto a veces conviene comprar en una u otra. Por lo tanto, si eres una persona que realiza búsquedas de componentes bastante seguido, sabrás que este proceso puede ser largo y tedioso. De esta manera surgió la idea de crear componentes.uy.

# 2. Problema

La motivación surge de una necesidad de los propios integrantes del equipo, actualmente cuando deseamos comprar un componente, lo solemos buscar en varias páginas para ver en dónde está más barato. En particular, todos los integrantes del equipo tenemos dos páginas que solemos utilizar en casi cualquier búsqueda, estas son Banifox [1] y Thot Computación [2].

Otro problema relacionado surge al querer armarse una PC nueva. En este caso, debemos comprar más de un componente, por ejemplo: placa madre, gabinete, fuente, CPU, refrigeración, tarjeta de video, periféricos, etc. Esto implica hacer la suma de cada uno de los artículos en cada una de las tiendas y luego compararlos.

# 3. Enfoque de la solución

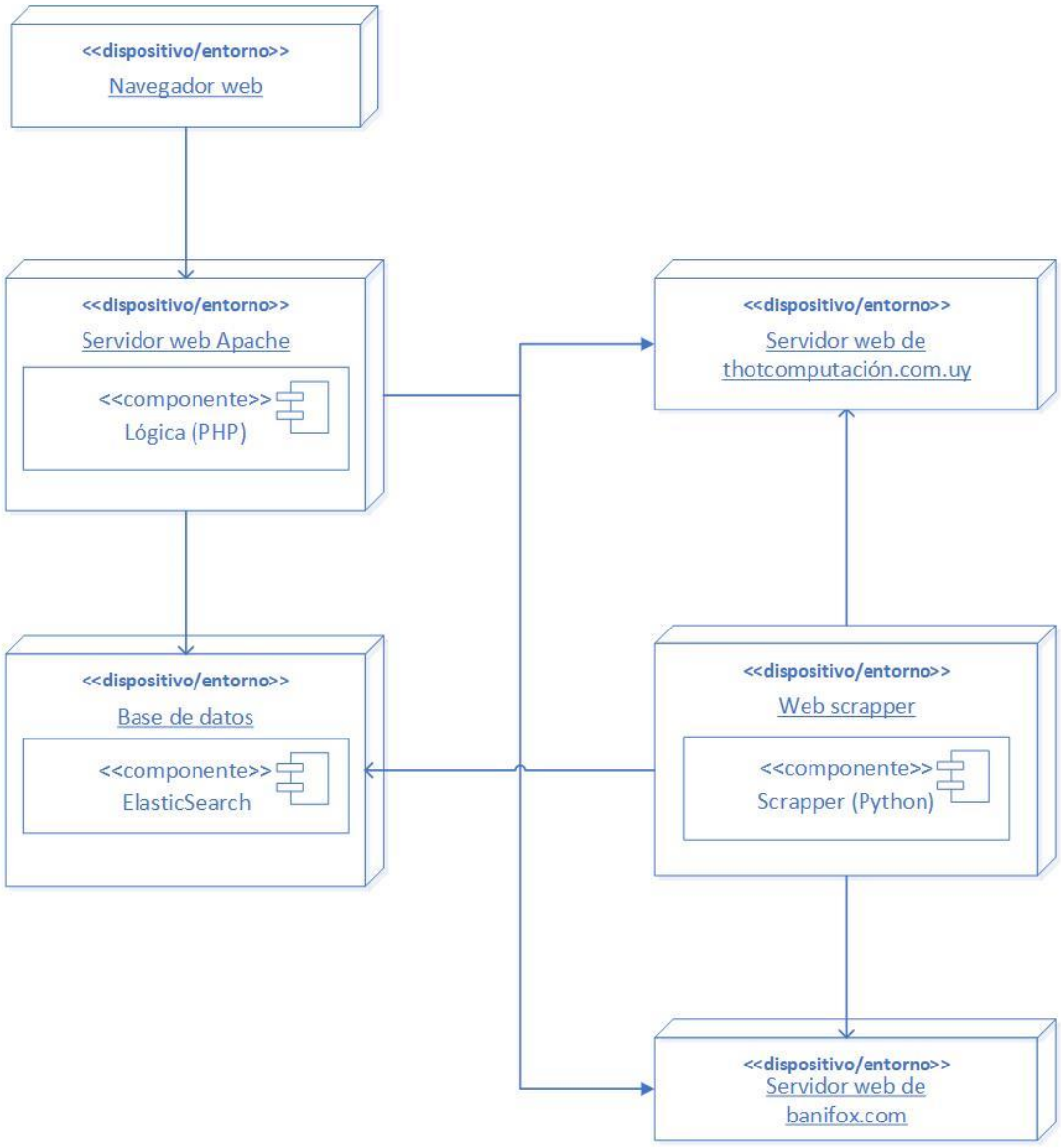
La solución que proponemos es tener un lugar centralizado donde buscar un artículo específico de computación y poder obtener los precios de las distintas páginas con una sola búsqueda. De este modo, poder encontrar cuál vende el mismo artículo, a un precio más barato.

Además, la idea es ofrecer algo similar a un carrito de compras, donde el usuario pueda seleccionar los componentes a comprar, se haga la suma de los precios según las distintas webs, y se muestre al usuario cuánto saldría comprar esa selección de componentes en cada tienda.

# 4. Diseño

Para la solución utilizamos las siguientes tecnologías:

- PHP [3]
- Python 3 [4]
- ElasticSearch [5]
- Apache [6]
- Bootstrap [7]




En el entorno de desarrollo, la base de datos ElasticSearch, el Web Scrapper y el Servidor Web ejecutarán en la misma PC. Sin embargo, dado que son componentes que interactúan a través de la red, se pueden colocar en distintos dispositivos, tal como muestra el diagrama

## 4.1. Scrapper

El módulo scrapper está programado en el lenguaje de programación Python [4].
Dado que ni Banifox [1] ni ThotComputación [2] proveen una API que permita obtener los artículos que tienen publicados, es necesario utilizar expresiones regulares que permita separar el código HTML del texto que es de nuestro interés (título del artículo, precio del artículo, imagen del artículo, categoría).
El módulo utiliza hilos, donde cada hilo se encarga de ir obteniendo los artículos de las distintas categorías. Esto hace que hacer el web scrapping sea más rápido, por ejemplo, para obtener los artículos de Banifox [1] (alrededor de 1500) no toma más de 5 minutos.
Los detalles de implementación se encuentran en la sección 6.3.

## 4.2. ElasticSearch

Utilizamos ElasticSearch [8] para almacenar y recuperar los artículos obtenidos mediante el módulo presentado en 4.1. La decisión de utilizar ElasticSearch [8] y no una base de datos SQL estándar es debido a que ElasticSearch [8] se adapta mejor al dominio de la aplicación que vamos a construir:

búsquedas en un conjunto de artículos. Las características que utilizamos de este servidor las veremos en la sección 6 donde presentamos los principales desafíos y su solución.

### 4.3. Lógica

Para la lógica utilizamos PHP [3]. Dicho lenguaje permite la creación de páginas webs dinámicas. Este lenguaje se adapta bien a los requerimientos de la aplicación: nos permitirá conectarnos a ElasticSearch [8] y hacer consultas para luego generar las páginas a ser mostradas. Además, utilizaremos la librería cURL [9] de PHP para hacer solicitudes del stock de los artículos encontrados. Los detalles de esto último lo abordamos en la sección 6.4.

### 4.4. Servidor web

En nuestro entorno de programación utilizamos el servidor web Apache [6]. Sin embargo, la arquitectura no está atada a este único servidor web: puede utilizarse cualquiera que sea compatible con PHP [3], por ejemplo, Nginx [10] o Litespeed [11].
En nuestro caso particular utilizamos Apache [6] ya que es el más sencillo de instalar en Windows en conjunto con el resto de las herramientas (basta instalar XAMPP [12]).

### 4.5. Cliente web

Para la capa de presentación utilizamos Bootstrap [7] ya que permite hacer un diseño responsive que se vea lindo de manera rápida y sencilla. Además, utilizamos jQuery [13] para hacer los pedidos de stock asíncronos (ver sección 6.4.) y para las funciones de carrito, en particular, al agregar un artículo al carrito no se recarga la página, también en el listado del carrito, lo usamos para la función de cambiar artículo.

## 5. Funcionalidades y uso

La página web funciona como la GUI de nuestra aplicación. En esta sección se describen las funcionalidades que provee.

### 5.1. Búsqueda de artículos por nombre

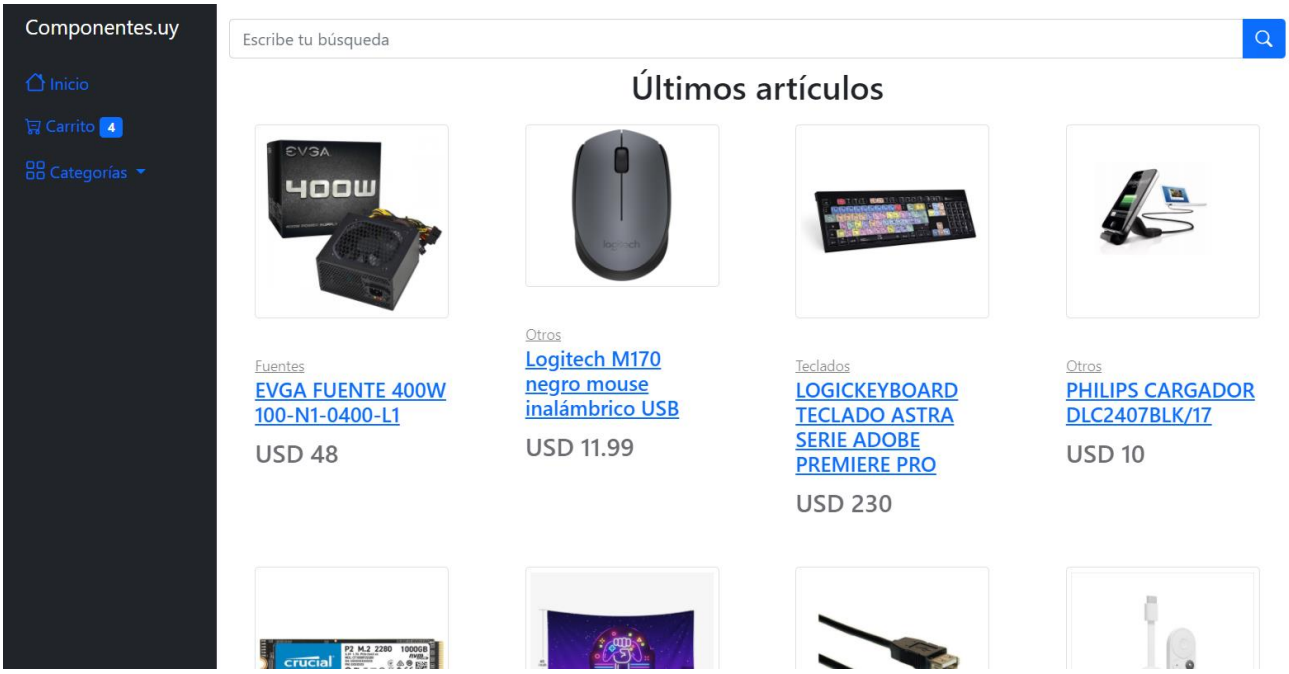



Figura 5.1.1.

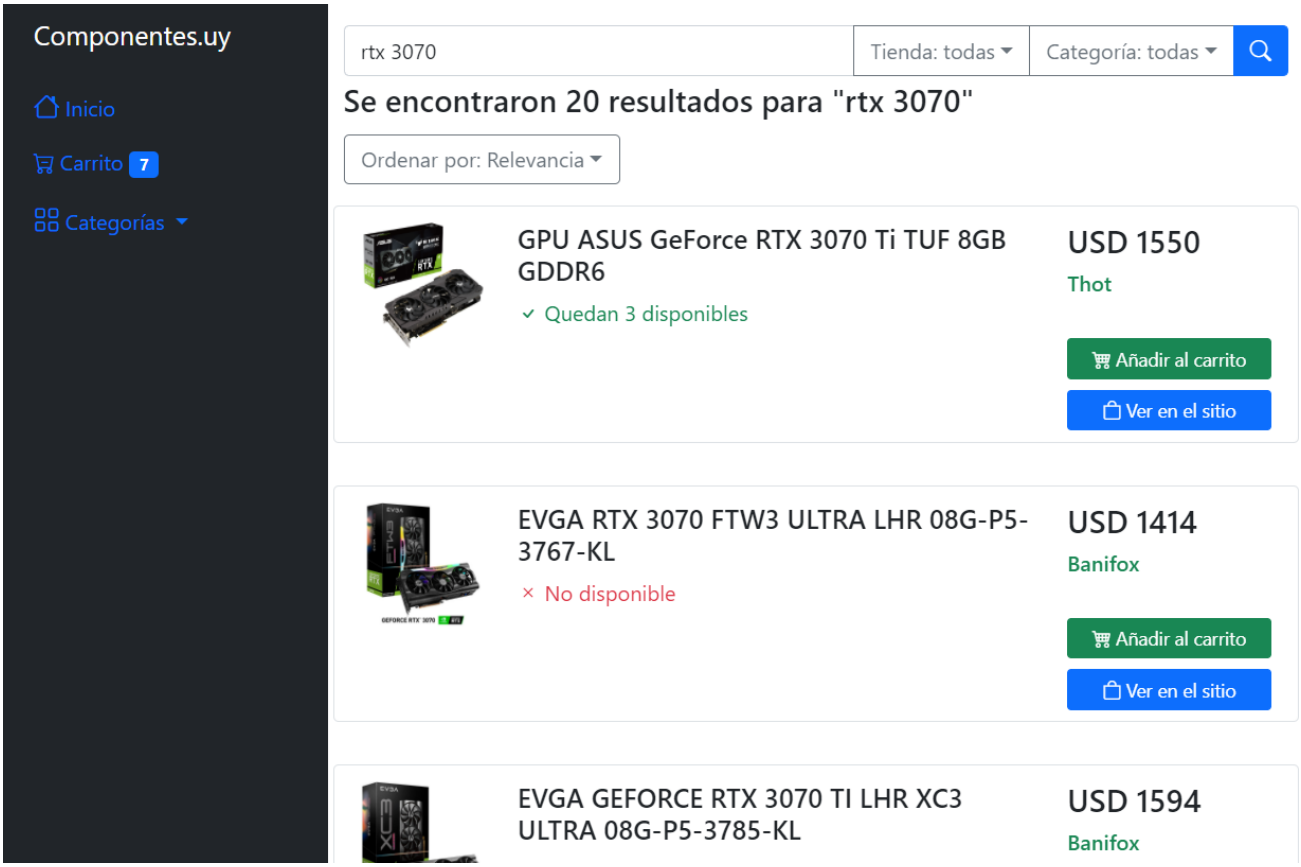



Figura 5.1.2.

Al ingresar a la página web (home), lo primero que se ve es el buscador, como puede verse en la figura 5.1.1. Este permite escribir el texto a buscar. Una vez escrito el artículo deseado, y presionado el botón de buscar o presionando el botón enter del teclado, se muestran los resultados para el texto ingresado, esto puede verse en la figura 5.1.2, en donde hemos ingresado «RTX 3070».

La GUI ofrece los siguientes filtros:

- Filtrar por tienda:
  - Todas
  - Banifox [1]
  - ThotComputación [2]
- Filtrar por categoría
  - Adaptadores
  - Almacenamiento
  - Celulares
  - Etc.
- Ordenar por:
  - Relevancia
  - Menor precio
  - Mayor precio

El criterio que seguimos para agregar estos filtros fue justamente pensar en casos de uso que hacemos nosotros cuando navegamos por páginas web de este tipo. Consideramos que el filtro más importante es poder ordenar por precio. No agregamos por ejemplo ordenar alfabéticamente, ya que no solemos utilizarlo en las páginas web que visitamos.
Para cada artículo encontrado, se muestra su título, el precio y a qué tienda corresponde. Observamos que la búsqueda se hace muy rápidamente gracias a la velocidad de ElasticSearch.

Además, luego de cargada la búsqueda, asíncronamente se va cargando si el artículo está disponible o no en la tienda a la que corresponde. Dicha consulta se hace en «tiempo real», es decir, el stock mostrado es al momento de hacer la búsqueda y consideramos que es de mucha ayuda ya que el usuario no tiene por qué salir de nuestra web para poder ver si el artículo mostrado sigue disponible.

La GUI ofrece un paginado para los casos en los que la búsqueda se corresponde con muchos artículos y además las opciones: «visitar en el sitio» y «agregar al carrito». Al presionar la primera opción se nos abrirá una nueva pestaña y en ella se nos llevará al artículo en la tienda que corresponde. Encontramos útil dicha opción para los casos en los que el usuario desea comprar el artículo. Por otra parte, la opción de «agregar al carrito» no requiere recargar la página para su ejecución. Al presionar esta opción, el botón cambiará a color rojo y pasará a ser «quitar del carrito» y se producirá una animación en el menú de la izquierda en el botón de carrito que hace alusión a que se agregó un artículo nuevo a este.

## 5.2. Carrito de compras

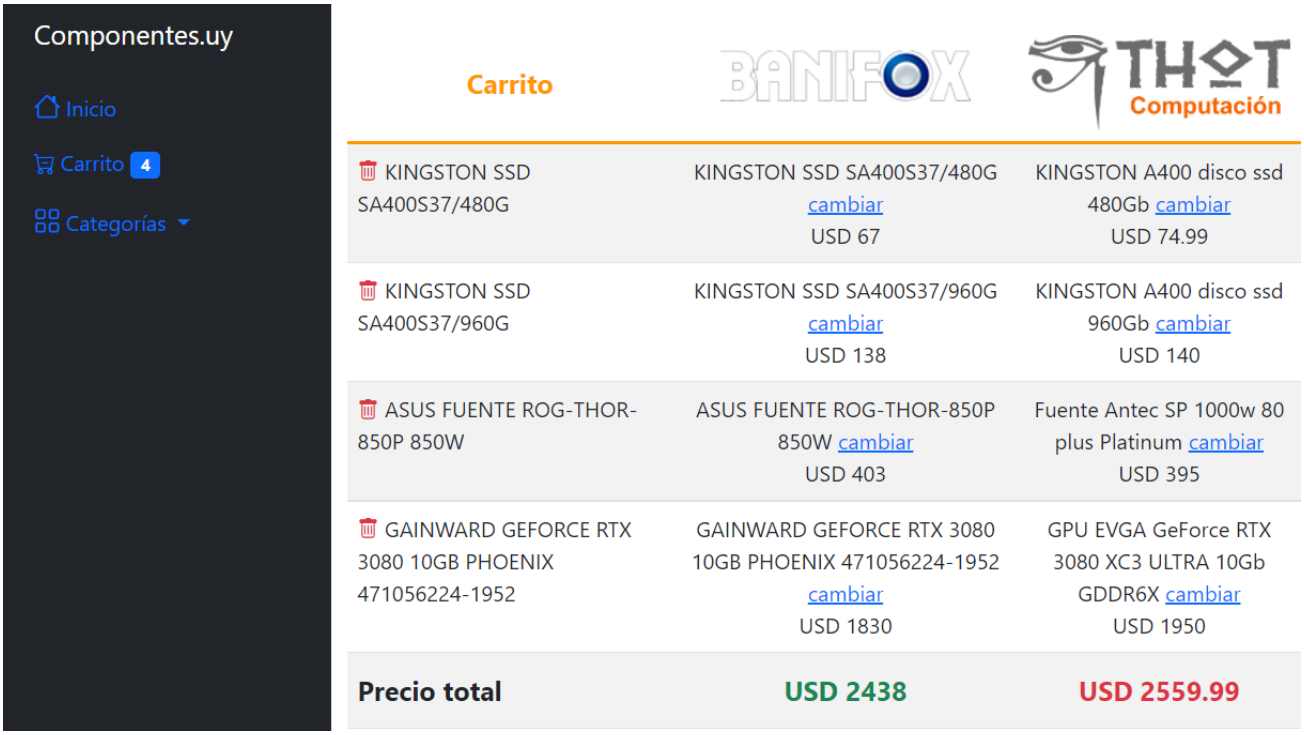



Figura 5.2.1.

Una vez que el usuario hace las búsquedas de los distintos componentes (por ejemplo, para ensamblar una computadora nueva) y agrega cada uno de ellos al carrito, puede acceder a dicho apartado.
En este apartado se muestra una tabla que contiene en la primera columna el nombre del artículo seleccionado, en la segunda columna el artículo más similar que se encuentra en Banifox [1] y en la tercera columna el artículo más similar que se encuentra en ThotComputacion [2], tal como se puede apreciar en la figura 5.2.1. Hay que aclarar que, si un usuario selecciona por ejemplo un artículo de Banifox [1], en la primera columna que recordemos corresponde a Banifox [1], se mostrará necesariamente ese artículo (ya que es el más similar, pues es el mismo artículo), análogamente ocurre con ThotComputacion [2]. En este sentido, el carrito permite dado un artículo de una tienda, encontrar el más similar (y potencial y deseablemente el mismo) en la otra tienda.
Además, ofrece algunas funcionalidades extras, entre ellas permite ver la suma de precio del total para ambas tiendas. Esto es de mucha ayuda ya que permite comparar cuánto sale comprar los componentes seleccionados en una tienda versus la otra. Para la tienda cuyo total es más barato, el precio se muestra de color verde, mientras que, para la otra, de color rojo.

Para aquellos casos en los que es no posible encontrar un artículo similar en la otra tienda, se le informa al usuario dicho problema en un texto de color rojo. De todos modos, se ofrece la opción «seleccionar artículo», esta opción permite hacer una búsqueda manual en la tienda correspondiente. Resulta una buena opción para aquellos casos en los que la búsqueda por alguna razón falla pues al menos el usuario podrá seleccionar el artículo manualmente. Además, una vez seleccionado dicho artículo, se recalcula el precio total sin necesidad de recargar la página.
Por último, el carrito de compras ofrece la opción de eliminar un artículo de este. Al hacer esto, el artículo correspondiente sale del listado y por tanto deja de ser tenido en cuenta en la suma del precio total. Si el usuario desea volver a agregarlo, tendrá que volver a hacer la búsqueda correspondiente y agregarlo manualmente.

## 5.3. Categorías

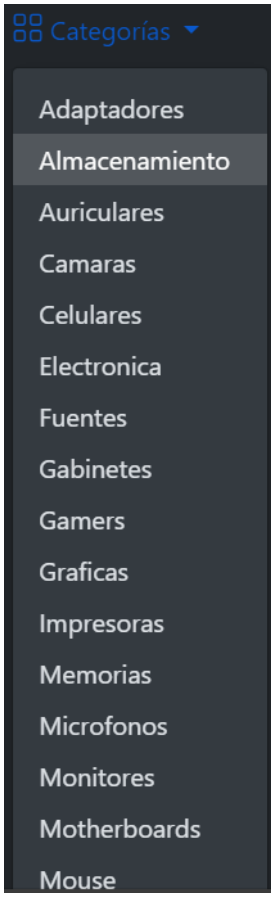



Figura 5.3.1.

En el menú se listan las categorías que corresponden a distintos tipos de hardware (periféricos, memorias, placa madre, CPU, etc.). Al presionar sobre una de estas, se listan todos los artículos que pertenecen a la categoría. Opcionalmente el usuario puede escribir qué desea buscar en dicha categoría, mostrando la misma vista que en 5.1.

# 6. Implementación

A continuación, presentamos los principales desafíos que llevó la implementación del producto y su respectiva solución.

## 6.1. Dado un artículo de una tienda, encontrar el mismo artículo en la otra

En la vista del carrito, como bien se describió se muestra para cada artículo que el usuario selecciona, el precio para dicho artículo en ambas tiendas. Esto implica dado un artículo de una tienda, buscar el mismo artículo en la otra (en caso de que exista).
Muchas veces nos ocurría que cuando se seleccionaba por ejemplo un CPU (por ejemplo, Core i5 10600), en la otra tienda ElasticSearch encontraba una PC armada con dicho CPU (ya que el título era PC Core i5 10600), y este caso claramente representaba un problema ya que el artículo encontrado no era el mismo. Observamos que la CPU pertenece en ambas tiendas a la categoría «procesadores», mientras que un equipo armado corresponde a la categoría «equipos ensamblados» o similar.
Utilizando esta observación lo que hicimos fue generar el factor común de las categorías de ambas tiendas y al momento de scrapear, guardar la categoría de cada artículo basándonos en ese factor común.
Por ejemplo, si en ThotComputación [2] la categoría de equipos armados se llama «Pcs ensamblados» y en Banifox [1] «Equipos armados», generamos una categoría con el nombre a nuestro gusto: «Equipos armados» (decidimos tomar el nombre de la categoría de Banifox [1] para este ejemplo) y le colocamos esta etiqueta al momento de scrapear los artículos tanto de la categoría «pcs ensamblados» para ThotComputacion [2] como la categoría «equipos armados» para Banifox [1].
Luego al momento de hacer la búsqueda de un artículo similar, además de pedir que coincida el título, le exigimos que el artículo similar pertenezca a la misma categoría.

Una vez solucionado lo anterior, encontramos que a veces ocurría que, dado un artículo de una tienda, ElasticSearch [8] encontraba que el artículo similar de la otra era uno que solo coincidía porque, por ejemplo, coincidía la marca, no así el modelo del artículo (imaginemos el caso de Mouse Logitech G203 y Mouse Logitech G903, si bien el título es muy similar -solo cambia un número-, al tratarse de distinto modelo ocurre que uno vale 30 USD mientras que el otro 108 USD). Esto claramente representaba un problema ya que comparar artículos tan diferentes de gama (y precio) no tiene sentido. Además, en general vimos que, en ambas tiendas, cuando se trata del mismo artículo, la diferencia de precio por lo general es a lo sumo de un 5-10% entre ambos.
Dado lo anterior, cuando buscamos un artículo similar le exigimos que el precio del artículo similar buscado tenga una cota inferior del precio del artículo seleccionado menos el 30% de su precio, y como cota superior el precio del artículo seleccionado más el 30% de su precio. Esto lo permite hacer ElasticSearch mediante el uso de rangos en las búsquedas [14].

En resumen, estas mejoras fueron de utilidad para:

- Que el artículo similar encontrado necesariamente esté en la misma categoría: no tiene sentido comparar el precio de un CPU con el precio de un equipo armado con esa CPU.
- Que la comparación se haga con artículos de precios medianamente similares, en consecuencia, de gamas similares. Evitando el caso mostrado del «Mouse Logitech G203» [15] vs «Mouse Logitech G903» [16]. Notar que en este caso tiene más sentido comparar con cualquier otro mouse de un precio similar que el G203 (por ejemplo, el Logitech G PRO que vale 50 USD [17]), a compararlo con un G903. Es decir, a pesar de que el título no coincide tanto, es mejor resultado que otro que el título es casi el mismo.

Si todo lo anterior falla, tal como se mostró en la sección 5.2. se permite al usuario cambiar el artículo similar encontrado y elegir otro manualmente. Al usuario al presionar allí, se le abrirá una ventana modal que contiene un listado de artículos para los cuales el precio es similar y su título también, permitiendo cambiar este último. En este caso, no se obliga a que los artículos listados estén en la misma categoría. Esto es debido a que podría haber algún error y que, para un determinado artículo, las tiendas lo publiquen en categorías distintas. Además, observar que se le da al usuario la libertad de cambiar el

título por el cual se está buscando el artículo similar. Esto es útil para aquellos casos en los que el artículo tiene un nombre alternativo y se encuentra por ese nombre en la otra tienda.

Es necesario destacar que en todos los casos los artículos mostrados siempre respetan la condición del precio definida antes, es decir, no se listan artículos que sobrepasen un umbral máximo y mínimo de precio, basándonos en el precio del artículo original que el usuario selecciono y agregó a su carrito.

## 6.2. Mejorar las búsquedas por texto: existencia de sinónimos

Para el hardware al igual que para muchos otros campos, existen sinónimos. Por ejemplo, «CPU» es lo mismo que «procesador». En este sentido, si un usuario busca por CPU sería positivo que se listen los artículos que tienen tanto CPU como procesador en su título y viceversa.
Para lograr esto, configuramos un diccionario en ElasticSearch [5].
Dicho diccionario se posible cargarlo mediante un archivo en formato txt y ejecutando una consulta encargada de hacer el refresh.
A efectos del curso no cargamos excesivamente dicho diccionario ya que tomaría demasiado tiempo pensar en todos los sinónimos posibles. Simplemente mostramos que esto era posible hacerlo usando ElasticSearch a través de la configuración de este diccionario.
Adicionalmente se generó un analizador personalizado en ElasticSearch, en el que se utilizó para agregar la opción de los sinónimos, se agregaron filtros por idioma para quitar las palabras que no aportan significado en las búsquedas (por ejemplo, de, la, the, of, and), también se normalizan los tokens generados con lowercase y se generó un tokenizer personalizado.
El tokenizer fue agregado al analizador para solucionar el problema en el cual productos de ambas tiendas la numeración y las siglas, en una estaban separadas y en otra no (por ejemplo RTX3090, RTX 3090 y RTX3060), en un caso como los ejemplos mencionados es importante tomar en cuenta el token RTX, pero por defecto ElasticSearch genera RTX3090 en lo que se desperdicia la información frente a la búsqueda, entonces se generaron expresiones regulares para mitigar el problema, inicialmente se investigó la utilización de búsqueda difusa, si bien en un principio parece una solución simple y atractiva, la realidad es que en el contexto del proyecto empeora las búsquedas esto se debe a que por ejemplo G903 es un mouse y G915 es un teclado, entonces la variación de un par de caracteres aunque el resto de los token de la búsqueda coincidan con los productos, se estaría perdiendo mucha precisión.

## 6.3. Obtener artículos de las tiendas para ser procesados en ElasticSearch

Tal como se discutió en la sección de arquitectura, los Scrapper los programamos usando Python [4]. Se observa que tanto ThotComputación [2] como Banifox [1] no proveen una API que permita obtener los artículos de forma amigable. Debido a esto, fue necesario hacer uso de expresiones regulares que permitan separar el HTML del texto de las distintas secciones que nos interesaba obtener la información para guardarse en ElasticSearch [8].
Esto impacta en el tiempo que toma obtener los artículos de las tiendas. En el caso de Banifox [1], obtener todos los artículos (alrededor de 1500) toma aproximadamente 5 minutos.

## 6.4. Obtención del stock en «tiempo real»

Una desventaja que tiene el enfoque que dimos a la aplicación, es que, se ejecutan los scrapper y esos artículos quedan en nuestra «base de datos» (ElasticSearch [8]). Esto implica que, si en alguna de las tiendas se borra algún artículo, esto no se va a ver reflejado en nuestra aplicación hasta que volvamos a ejecutar los scripts de carga (scrappers). Por lo tanto, consideramos una buena idea mostrar el stock real de cada artículo que se le muestra al usuario. Con stock real nos referimos al stock que hay en la tienda para ese artículo al momento en el que el usuario está haciendo la búsqueda. Hay que destacar que ThotComputación [2] para todos los artículos que tiene publicados indica el número de stock que hay, o en caso contrario pone «consultar stock». El caso de Banifox [1] es diferente: solo muestra un mensaje cuando no hay stock.

Debido a esto, estos casos los tratamos de forma diferente:

- Para un artículo de Banifox [1], mostramos «disponible» si la página no informa que falta stock. En caso contrario, mostramos «no disponible».
- Para un artículo de ThotComputación [2], mostramos la cantidad de stock disponible cuando hay (es decir, cuando en la página del artículo figura la cantidad de stock que posee) y, en cambio, cuando dice «consultar stock» mostramos el mensaje «no se pudo obtener el stock».

Como bien mencionamos en la sección 6.3. tanto Banifox [1] como ThotComputacion [2] no poseen una API que permita obtener los artículos, mucho menos obtener su stock. Esto implica que para cada artículo nuestro módulo «lógica» debe conectarse a la página a la que el artículo pertenece y aplicar expresiones regulares para obtener la sección que nos interesa. Esto impacta fuertemente en la velocidad de las búsquedas, pues si al momento de buscar hacemos una de estas llamadas, para cada artículo, la búsqueda tarda mucho más tiempo, sobre todo para el caso de ThotComputación [2] donde el tiempo de respuesta de su servidor no es bueno. La solución que encontramos para no enlentecer las búsquedas fue primero mostrar los resultados para la búsqueda del usuario, y una vez cargado, utilizando jQuery [13] obtener para cada artículo el stock de forma asíncrona. Esto permite listar los artículos para la búsqueda del usuario muy rápidamente y luego en la medida que es posible listar el stock para cada resultado sin afectar la agilidad de las búsquedas.

# 7. Posibles mejoras a futuro

## 7.1. Agregar caché a las consultas de stock

Como consultar el stock para los artículos lleva tiempo, y además es una operación costosa, sería bueno que cada consulta se mantenga en caché durante una hora. De este modo si bien la información que mostramos no es en tiempo real, nuestra web soportaría mayor cantidad de usuarios al tiempo que da información relativamente actualizada (con máximo una hora de desfasaje del stock real). Su implementación podría realizarse de manera muy sencilla instalando Memcached [18] y almacenando los resultados de cada consulta con un expire de una hora.

## 7.2. Agregar más sinónimos al diccionario

Se podrían agregar mayor cantidad de sinónimos al diccionario [5], de este modo, mejorar las búsquedas de los usuarios aún más.

## 7.3. Mejoras en el carrito

En el carrito sería bueno tener las siguientes funcionalidades:

- Ver imagen del artículo seleccionado.
- Tener la posibilidad de elegir la cantidad de cada artículo.
- Poder ordenar los artículos del carrito por distintos filtros, por ejemplo, su precio.
- Agregar un botón que permita vaciar completamente el carrito (sin necesidad de borrar artículo por artículo).

## 7.4. Mejoras en el listado de la búsqueda

- Permitir ordenar alfabéticamente.
- Filtrar por rango de precio.
- Elegir moneda (USD, UYU).
- Imágenes guardas localmente y no remotamente.
- Historial de búsquedas.
- Ofrecer sugerencias mientras el usuario va escribiendo la búsqueda.

## 7.5. Otras mejoras

- Agregar mayor cantidad de categorías.
- Consulta de stock en tiempo real para los artículos mostrados en el home.
- Obtener la descripción de los artículos scrapeados.

# 8. Evaluación y resultados

Hemos conseguido el objetivo inicial del proyecto: poder comparar precios entre las dos tiendas que más utilizamos para realizar compras en lo que respecta a hardware de PC. Esta comparación es posible gracias a la incorporación del carrito.

| Carrito | BANIFOX | THOT Computación |
|---|---|---|
| KINGSTON SSD SA400S37/480G | KINGSTON SSD SA400S37/480G cambiar<br>USD 67 | KINGSTON A400 disco ssd 480Gb cambiar<br>USD 74.99 |
| ASUS FUENTE ROG-THOR-850P 850W | ASUS FUENTE ROG-THOR-850P 850W cambiar<br>USD 403 | Fuente Antec SP 1000w 80 plus Platinum cambiar<br>USD 395 |
| KINGSTON FURY MEMORIA DDR4 KF432C16BBA/8 3200MHZ 8GB | KINGSTON FURY MEMORIA DDR4 KF432C16BBA/8 3200MHZ 8GB cambiar<br>USD 67 | Memoria Kingston Fury Beast 8GB DDR4 2666Mhz cambiar<br>USD 55 |
| INTEL PROCESADOR I7-11700 LGA 1200 | INTEL PROCESADOR I7-11700 LGA 1200 cambiar<br>USD 540 | CPU Intel Core i7 11700 Rocket Lake 1200 cambiar<br>USD 521.99 |
| Asus ROG Strix B460-H Gaming | ASUS PLACA TUF GAMING B460M-PLUS WI-FI cambiar<br>USD201 | Asus ROG Strix B460-H Gaming cambiar<br>USD 232 |
| GAINWARD GEFORCE RTX 3080 10GB PHOENIX 471056224-1952 | GAINWARD GEFORCE RTX 3080 10GB PHOENIX 471056224-1952 cambiar<br>USD 1830 | GPU EVGA GeForce RTX 3080 XC3 ULTRA 10Gb GDDR6X cambiar<br>USD 1950 |
| **Precio total** | **USD 3108** | **USD 3228.98** |

Figura 8.1.

En la figura 8.1. se puede apreciar un caso de uso de la aplicación: se seleccionan distintos componentes de PC (memoria RAM, almacenamiento, placa madre, tarjeta de video, procesador, gabinete, fuente de poder) y se puede ver el precio que sale comprar todos los componentes en una tienda versus la otra, indicándose de manera gráfica en cuál tienda nos sale más barato.

Como también se puede apreciar en la figura 8.1. el buscador de artículos similares funciona bien. Si bien no es perfecto, por todas las posibles mejoras mencionadas en la sección 7, sí es bastante funcional, así como está. Además, en caso de que el artículo encontrado no sea el adecuado, siempre el usuario puede seleccionar otro manualmente por lo que de todos modos podrá efectuar la comparación de precios de manera satisfactoria.

# 9. Conclusiones

Consideramos que la prueba de concepto que realizamos está casi apta para ser puesta en producción. Solo faltaría la adición de Memcached a la consulta de stock de los artículos mencionada en la sección 7.1 y un Cron Job que se encargue de ejecutar los scrapper por ejemplo a la media noche todos los días, a modo de mantener el stock actualizado automáticamente.

En cuanto a mantenibilidad, hay que destacar que tanto ThotComputación [2] como Banifox [1] si modifican su código HTML, por ejemplo, su diseño, o sus URLs, entonces puede ocurrir que los scrapper dejen de operar correctamente. De hecho, esto nos pasó unos días previos a la entrega: Banifox [1] cambió unos detalles de su interfaz y hubo que hacer unos ajustes rápidos para dejar el scrapper operativo nuevamente.

Lo mejor en estos casos sería tener un convenio con Banifox [1] y ThotComputacion [2] a través del cual nos brindaran por ejemplo una API REST para obtener sus artículos, sin embargo, al menos de momento no ofrecen algo así. Como solución ante este problema y pensando en su puesta en producción, podríamos hacer que el Cron Job [19] nos envíe un mail con la cantidad de artículos scrapeados al terminar su ejecución, de este modo, si falla nos enteraríamos de manera temprana y podríamos hacer las correcciones necesarias.

Por último, en lo que respecta a los objetivos del curso, hemos utilizado la información que teníamos y pensando en el dominio de la aplicación (componentes de pc) hemos podido mejorar las búsquedas. Estas mejoras se da gracias a la incorporación de:

- Diccionario de sinónimos.
- Factorización de las categorías de ambas webs y uso de dichas categorías en la búsqueda de artículos similares.
- Exigir una cota inferior y superior de precio del producto similar, basado en el porcentaje del precio del producto buscado.
- Algunas configuraciones extras en ElasticSearch (uso de un tokenizador y analizador personalizado).

# 10. Referencia


[1] Banifox, «Banifox.com,» [En línea]. Available: https://www.banifox.com/.

[2] ThotComputacion, «ThotComputacion.com.uy,» [En línea]. Available: https://thotcomputacion.com.uy/.

[3] PHP , [En línea]. Available: https://www.php.net/.

[4] Python 3, «Python 3.0 Release,» [En línea]. Available: https://www.python.org/download/releases/3.0/.

[5] ElasticSearch, «Aumento del poder de ElasticSearch con sinónimos,» [En línea]. Available: https://www.elastic.co/es/blog/boosting-the-power-of-elasticsearch-with-synonyms.

[6] APACHE, «HTTP Server Project,» [En línea]. Available: https://httpd.apache.org/.

[7] Bootstrap, «Bootstrap 5.0,» [En línea]. Available: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

[8] ElasticSearch, «https://www.elastic.co/es/,» [En línea].

[9] PHP, «cURL PHP,» [En línea]. Available: https://www.php.net/manual/es/book.curl.php.

[10] nginx , «High performance load,» [En línea]. Available: https://www.php.net/manual/es/book.curl.php.

[11] LiteSpeedTech, [En línea]. Available: https://www.litespeedtech.com/.

[12] xampp, «Descargar XAMPP,» [En línea]. Available: https://www.apachefriends.org/es/download.html.

[13] jQuery, «jQuery v3.6.0,» [En línea]. Available: https://www.apachefriends.org/es/download.html.

[14] ElasticSearch , «Range query,» [En línea]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-range-query.html.

[15] Banifox, «LOGITECH MOUSE G203 LIGHTSYNC WHITE 910-005794,» [En línea]. Available: https://www.banifox.com/logitech-mouse-g203-lightsync-white-910-005794/art_101111/.

[16] Banifox, «LOGITECH G MOUSE G903 LIGHTSPEED,» [En línea]. Available: https://www.banifox.com/logitech-g-mouse-g903-lightspeed/art_101013/.

[17] Banifox, «LOGITECH G MOUSE G PRO 910-005536,» [En línea]. Available: https://www.banifox.com/logitech-g-mouse-g-pro-910-005536/art_100742/.

[18] Memcached.org, «Memcached - a distributed memory object caching system,» [En línea]. Available: https://memcached.org/.

[19] man7, «Linux manual page,» [En línea]. Available: https://man7.org/linux/man-pages/man5/crontab.5.html.

[20] https://jquery.com/, «jQuery,» [En línea]. Available: https://thotcomputacion.com.uy/.

[21] ElasticSearch, «Tokenizer reference,» [En línea]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenizers.html.

[22] Banifox, «Mouse Logitech G203 - Banifox,» [En línea]. Available: https://www.apachefriends.org/es/download.html.