Proyecto Final
Recuperación de Información y Recomendaciones en la Web

Carmela Beiro - Vicente Fava - Maite Mañana - Ignacio Villaverde

5 de diciembre de 2014

Resumen

Este informe tiene como objetivo introducir y brindar conocimiento acerca del proyecto final realizado para la asignatura Recuperación de Información y Recomendaciones en la Web. El documento no solo incluye información sobre lo que fue realizado sino que también plantea posible trabajo a futuro de forma de continuar la investigación y el desarrollo del producto obtenido.

Índice

1. Introducción

Este informe planea presentar una solución al problema de realizar búsquedas sobre sitios de ofertas online, específicamente en los sitios `http://www.notelapierdas.com.uy` y `http://www.woow.com.uy`. Dichas páginas cuentan con diversas ofertas, sobre diferentes rubros, las cuales son lanzadas con una frecuencia diaria y tienen una fecha de expiración, en múltiples casos no mayor a 24 horas, mientras que en otros casos el cierre de una oferta se da por falta de stock. Por lo general, se publican cientos de ofertas y se renuevan diariamente. Por lo tanto, consideramos importante contar con la capacidad de realizar búsquedas avanzadas, filtrar resultados y optimizar las mismas de manera de obtener el máximo provecho posible.

Este trabajo está centrado en la realización de búsquedas de ofertas, las cuales pueden realizarse según el título de la misma, contenido de la descripción, entre otros. A su vez, se pretende contar con la capacidad de filtrar los resultados, tanto por rubro, como por precio total o porcentaje de ahorro obtenido.

Finalmente, se presentará la solución mediante un sitio web sencillo, intuitivo y de fácil uso el cual contará con las funcionalidades definidas anteriormente. Por otra parte, debido al objetivo principal de la entrega, se pretende que el mismo presente los resultados de forma clara y directa, de manera de realizar tanto la búsqueda como la presentación de los resultados lo más eficaz y eficiente posible.

2. Presentación de la solución

Lo que se quiere es centralizar todas las ofertas en un solo sitio, donde el usuario ingrese y genere búsquedas personalizadas según distintos parámetros. Entonces, de forma de no solo agilizar el desarrollo sino también de obtener un producto de calidad se pensó en una aplicación web con una arquitectura de sistemas basada en el patrón MVC (modelo vista controlador). Este patrón separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. El Modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia, la Vista, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos interacción con éste y el Controlador, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

Sin embargo esto no soluciona el problema planteado. Hasta ahora lo que se tiene es la arquitectura principal de la solución, pero ¿cómo se centralizan todas las ofertas en un solo sitio común? La solución encontrada fue utilizar la técnica Scrapping. Esta técnica también conocida como Web Scrapping, es utilizada para extraer los datos de sitios web, simulando la interacción con un usuario real (humano). El objetivo es generar data estructurada a partir de elementos del DOM HTML, parseando la web para luego almacenar los resultados en una base de datos propia, y así manipularla a gusto. Entonces, lo que se necesita es optimizar de alguna forma las búsquedas sobre la base de datos ya generada. Para ello, se debe utilizar un motor de búsquedas, software que permite realizar búsquedas por palabras claves o con árboles jerárquicos por temas. De esta forma el usuario final podrá ingresar en la web publicada, cualquier palabra, precio,

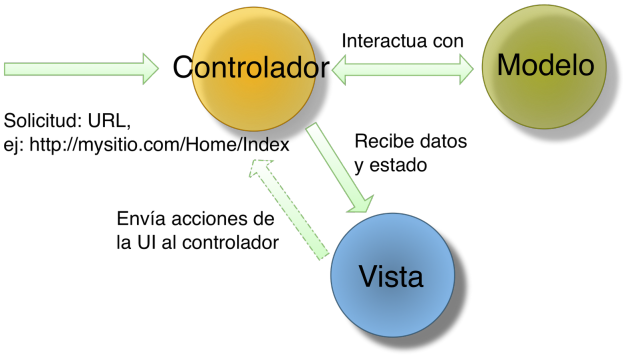

Figura 1: Diagrama de la arquitectura MVC.

etc, que considere adecuado para obtener sus ofertas deseadas de forma óptima. En particular se optó por Apache Solr, un motor de búsqueda open source, que funciona como bloque independiente. Es decir, no se encuentra fuertemente acoplado a nuestro sistema MVC, sino que corre en un servidor aparte. Entre sus ventajas, se encuentra el indexado, mejorando la performance en lo que refiere a las búsquedas.

3. Uso de herramientas

A continuación se listan las herramientes utilizadas junto a una breve descripción de las mismas.

3.1. Ruby on Rails

Ruby on rails es un framework web open source escrito en el lenguaje de programación Ruby, lenguaje de programación orientado a objetos, ideado para programar de forma rápida y ágil. Rails sigue el paradigma del modelo vista controlador (MVC), pero además, una gran ventaja brindada por el framework y uno de los motivos por el cual fue utilizado en esta ocasión, es que permite utilizar gemas, las cuales se asemejan a una librería o plugin, para cubrir ciertas necesidades. Estas gemas son librerías reutilizables, habilitando su uso a múltiples usuarios sin la necesidad de re-implementar código.

3.2. Nokogiri

Nokogiri es una gema que permite hacer búsquedas usando selectores CSS y XPath. Esto es de gran ayuda a la hora de parsear los distintos sitios web que publican sus ofertas. De este modo el uso de la técnica “Scrapping” se hace con mayor eficiencia y velocidad.

3.3. Apache Solr

Apache Solr es una servidor de búsquedas open source NoSQL basado en la librería Java Lucene. Entre sus principales funcionalidades se encuentran las búsquedas de texto completo, resaltado de resultados,búsquedas por facetas, clustering dinámico, integración con bases de datos, manejo de documentos ricos (como Word y PDF) y búsquedas geoespaciales. El servidor cuenta con una API del estilo Rest para indexar documentos y luego realizar consultas. Sin embargo encontramos más conveniente y sencillo utilizar la gema de Ruby on Rails Sunspot, que expone las principales funcionalidades de Solr para utilizarlas dentro del mismo lenguaje.

3.4. Will_paginate

Will_paginate es una librería de paginación que se integra con Ruby mediante la gema will_paginate. La misma provee una API para la realización de consultas paginadas sobre ActiveRecord de manera altamente performante. Además, cuenta con helpers utilizados para realizar el renderizado de links de paginado en aplicaciones web Rails, entre otras.

4. Implementación final

En la siguiente sección se hace un analisis sobre las distintas dificultades encontradas a lo largo del proyecto, incluyendo luego las distintas funcionalidades con las cuales la aplicación cuenta finalmente.

4.1. Dificultades encontradas

La implementación final mantuvo los requisitos planteados en la sección "Presentación de la solución". Sin embargo, a lo largo del desarrollo se encontraron varios inconvenientes que llevaron a disminuir el alcance del proyecto. En un principio se pensó en implementar el parser de tan solo un sitio, para así avanzar con el desarrollo de la aplicación y una vez terminado, o prácticamente resuelto el objetivo, implementar parsers para al menos 5 sitios más. Se optó por comenzar con el sitio `http://www.woow.com.uy/`, el cual permitió desarrollar un parser adecuado, de modo de almacenar sus ofertas en nuestra base de datos. El problema se dió cuando WooW actualizó su sitio, dejando inútil el parser generado, pero además, dificultando la creación de un nuevo parser ya que utiliza AngularJS como framework javascript. AngularJS es un framework javascript open source mantenido por Google. Lo importante es que permite extender el documento HTML a través del uso de un lenguaje propio, para así, al cargar la página, encargarse de renderizarla a su gusto. Esto impacta fuertemente sobre la forma de obtener la información de cada oferta, ya que cuando la gema nokogiri es utilizada, obtiene el documento HTML sin el renderizado previo que el código javascript perteneciente a AngularJS genera. Por lo tanto, en lugar de obtener un HTML básico, ya conteniendo las ofertas, se obtiene algo del tipo:

```
<div ng-repeat='offer in offers'>
   .
   .
```

```
    .
</div>
```

Debido a esto, se optó por analizar el código fuente del sitio, para así lograr reproducir los distintos requests que genera el cliente del sitio para recoger las ofertas del backend. De este modo, la respuesta obtenida fue en forma de json, lo cual facilitó el pasaje a data interpretable por la aplicación MVC propia. Sin embargo, el costo que significó fue alto, lo cual llevó a que un parser más fuera añadido a la aplicación, disminuyendo el alcance planteado anteriormente. Este parser fue el del sitio `http://www.notelapierdas.com.uy/`, el cual sí cuenta con la técnica Web Scrapping aplicada.

4.2. Manejo de la base de datos

El proyecto cuenta además con un administrador para ser utilizado por los desarrolladores y el encargado del mantenimiento del sitio, en donde mediante una interfaz gráfica se puede editar y consultar información de la base de datos de manera sencilla y rápida.

4.3. Parser

De la página Woow se obtuvieron los siguientes datos de una oferta :

- El título
- Los detalles del producto
- La descripción del producto
- El precio
- El porcentaje ahorrado
- La cantidad de productos comprados hasta el momento

De la página Notelapierdas se obtuvieron los datos mencionados a continuación

- El título
- Los detalles del producto
- El precio
- El porcentaje ahorrado

4.4. Funcionalidades del sitio

Como resultado del trabajo realizado, la aplicación final cuenta con las siguientes funcionalidades:

- Realizar una búsqueda utilizando palabras claves.
- Filtrar ofertas por precio, escogiendo dentro de un rango ya preestablecido.
- Visitar la oferta encontrada en su sitio original.

5. Conclusiones

5.1. Resultados obtenidos

Luego de finalizada la etapa de implementación, se hicieron multiples tests, analizando los resultados obtenidos a través de la web, simulando el comportamiento de un usuario final, y los resultados de ejecutar consultas SQL directas a la base de datos, monitoreando el tiempo de respuesta. Es importante aclarar que estas consultas se asemejan a consultas que haría un usuario final, por ejemplo:

```
(SELECT * FROM deals WHERE title LIKE '%' || 'Punta del Este' || '%')
    UNION (SELECT * FROM deals WHERE info1 LIKE '%' || 'Punta del Este'
    || '%')
UNION (SELECT * FROM deals WHERE info2 LIKE '%' || 'Punta del Este' ||
    '%')
```

Para la consulta anterior, en un escenario típico donde en la base de datos, la tabla Ofertas continene 3100 filas, el tiempo de respuesta de ejecutar la consulta SQL directa a traves del manejador fue de 18 milisegundos. Lo cual no permite enfatizar el uso de Solr, ya que sin él, el resultado es excelente, o mejor dicho, excelente desde el punto de vista del usuario final. Por ello, se incrementó el tamaño de la base de datos ejecutando multiples veces el script de parseo de los sitios. Una vez alcanzado un tamaño de 52000 filas, es decir, casi un 1700 por ciento de su tamaño típico, el tiempo de respuesta obtenido fue de aproximadamente 1 segundo. Nuevamente, tomando en cuenta que es un escenario atípico, y aún asi, obteniendo un buen resultado, se hace difícil demostrar la necesidad del uso de Solr.

5.2. Conclusión final

En el presente proyecto se implementó un sitio de búsquedas de ofertas online sobre dos diferentes sitios de ofertas. Para el mismo se utilizó la herramienta Apache Solr, de manera de obtener un resultado eficiente y eficaz. La búsqueda se realizó por título, descripcion, página a la que pertenece la oferta, entre otros. Además se agregó la posibilidad de realizar un filtrado por diferentes rangos de precios.

Dadas las dimensiones de los datos, el uso de la herramienta no presenta un beneficio directo al momento de realizar la búsqueda, ya que la cantidad de entradas a indexar es relativamente pequeña. Sin embargo, este proyecto no deja de ser un prototipo con gran potencial de trabajo a futuro, por lo que cuenta con las características necesarias para que sea escalable y mantenible. Se daría un mejor aprovechamiento del uso de Solr en el caso que se contara con ofertas parseadas de cientos de páginas diferentes, teniendo la capacidad de realizar búsquedas sobre grandes volumenes de datos en milisegundos, otorgando una experiencia de usuario más que placentera.

Por otra parte, se implementó un filtrado, de manera de contar con la posibilidad de realizar búsquedas menos especificas (en comparación con las búsquedas por palabras claves) pero realmente útiles y populares, como es la búsuqeda por rango de precios. Este filtrado se puede realizar en el total de ofertas, o sobre una

búsqueda en particular. Esto da la posibilidad de refinar aún más los resultados obtenidos y así brindarle mås potencial al sitio.

Finalmente, se cuenta con una vista previa de la oferta y la posibilidad de visitar la publicación original de la misma. En la misma se muestran los datos extraídos mediante los parsers implementados, como descripción de la oferta, precio, ahorro obtenido, etc.

6. Trabajo a futuro

Existen varios puntos sobre los cuales mejorar el trabajo realizado. Uno de ellos es el incorporar una amplia cantidad de sitios web a la lista de sitios a parsear. De esta forma el número de ofertas almacenadas en nuestra base de datos sería muy amplio, mejorando la calidad del servicio hacia los usuarios finales. Sin embargo, es importante aclarar que el aumentar la lista de sitios a parsear, no solo incrementa el costo a la hora de generar los distintos parsers, sino que que con el continuo avance de la tecnología y la competencia comercial de las distintas empresas, los sitios son actualizados frecuentemente. Esto implica que los parsers pasen a ser inútiles y en el peor de los casos, se deben re-implementar desde cero.

Otra posible mejora, es el manejo de sugerencias a los usuarios. Esta mejora pertenece básicamente al área de usabilidad y cuidado del consumidor y no tanto en la performance. La idea implica el almacenar las últimas búsquedas realizadas por el usuario, para así combinar ya sea palabras claves o similitudes por categoría (Vehículos, Viajes, etc) y publicar a través de la UI una lista de posibles ofertas de interés. Además se podrían contemplar las faltas de ortografía de los usuarios y mediante la configuración de Solr tratar de realizar la búsqueda correcta o sugerirsela al usuario. Por otra parte, existe la posibilidad de configurar Solr para que utilice el modelo de ngramas para realizar autocompletado y busquedas con strings parciales de lo que se quiere encontrar.

Por último, como en toda herramienta de software, la performance siempre es mejorable. El motor de búsquedas puede ser mejorado, o mejor dicho, se puede hacer un mejor uso del mismo. Ya sea utilizando a pleno las funcionalidades ya en uso, o incorporando la mayoría de ellas. A su vez, se puede extender la lista de campos por los cuales actualmente se realizan filtros y búsquedas avanzadas, como por ejemplo la categoría de la oferta(Gastronomía, Viajes, Estética).

Referencias

[1] `http://rubyonrails.org/`

[2] `http://www.nokogiri.org/`

[3] `http://lucene.apache.org/solr/`

[4] `http://asciicasts.com/episodes/278-search-with-sunspot`

[5] `https://github.com/mislav/will_paginate`