Memoria práctica RI

Mauro Paredes Romero (mauro.paredes@udc.es)
Iván Janeiro Tato (ivan.janeiro.tato@udc.es)

10 de diciembre de 2021

Índice

Índice de figuras

1. Desarrollo del proyecto

El desarrollo del proyecto se ha dividido en 3 partes fundamentales:

- *Webcrawling* con Scrapy.
- Configuración e introducción datos en Apache Solr.
- Implementar una aplicación que use el *search engine* de Solr.

Comentar a mayores que parte de la configuración de Solr se realizó paralelamenten a la implementación de la aplicación. Pero por motivos de claridad, en esta memoria se tratará como si se hubiera realizado toda la configuración primero.

1.1. *Webcrawling* con Scrapy.

Como primer intento de *webcrawler* intentamos crawlear una pagina que contenía datos sobre las actualizaciones que se iban realizando sobre el juego League of Legends. Pero debido a cómo estaba creada la página, repetición de clases y componentes html, nos era imposible sacar una cantidad de datos consistente y organizada.

La decisión tomada fue cambiar y hacer el *webcrawler* de una página más sencilla que contenía un top histórico de películas y otro de series. Aunque organizados de manera más sencilla sus datos, si nos encontramos algún problema a solucionar.

En estas 2 páginas para los tops no existía paginación, sino que los datos se recuperaban de manera dinámica. Esto supone que en la aplicación, cada vez que un usuario pulsa el botón de **cargar más**, lanza una petición POST al backend para recuperar las siguientes películas/series. Todo esto sin modificar la url. La solución fue emplear una librería que nos permitiera realizar estas peticiones para recuperar los datos.

1.1.1. Código *webcrawler*

Lo que queremos recuperar de cada elemento, ya sea serie o película, es lo siguiente:

- **title:** título de la película/serie.
- **ranking:** campo que especifíca si es película o serie.
- **position:** posición dentro del top.
- **year:** año de su estreno.
- **director:** lista de directores.
- **cast:** lista de actores del reparto.

- **avg_rating:** media de la valoración.
- **rate_count:** número total de valoraciones.
- **poster:** url de la portada de la película, para posteriormente ser usada.

También configuramos el *webcrawler* para que guardara los resultados en una BD. Aunque finalmente no nos hizo falta ya que subimos directamente los datos a Solr mediante un POST.

1.2. Configuración e introducción de datos en Apache Solr.

Para el correcto funcionamiento de la aplicación es necesario cambiar la configuración por defecto que nos da Apache Solr al crear un core. La configuración que tuvimos que añadir/modificar fue la siguiente:

- Añadir los campos al **managed-schema**.
- Subir los datos al core de Solr.
- Arreglar el problema del **CORS**.
- Cambiar campo default en el **solrconfig.xml**.
- Añadir el **solr.SuggestComponent** en el **solrconfig.xml**.
- Añadir **requestHandler** para **solr.SuggestComponent**.

Hay que tener en cuenta que el core inicialmente lo creamos con la configuración **default** de Apache Solr. Esta configuración la fuimos modificando cada vez que necesitábamos solucionar un problema o añadir una nueva funcionalidad.

```
    ./solr create_core -c riws-filmaffinity -d _default
```

1.2.1. Añadir los campos al managed-schema

El primer paso necesario es crear los campos en el schema de Solr para este pueda guardar los datos y posteriormente hacer búsquedas sobre estos. Para ello es necesario modificar el **managed-schema**.

```

<field name="ranking" type="string" indexed="true" stored="true"/>
<field name="position" type="pint" indexed="true" stored="true"/>
<field name="title" type="text_general" indexed="true" stored="true
    "/>
<field name="year" type="pint" indexed="true" stored="true"/>
<field name="director" type="text_general" indexed="true" stored="
    true" multiValued="true"/>
<field name="cast" type="text_general" indexed="true" stored="true"
     multiValued="true"/>
<field name="avg_rating" type="pfloat" indexed="true" stored="true"
    />
<field name="rate_count" type="pint" indexed="true" stored="true"/>
<field name="poster" type="string" indexed="true" stored="true"/>
```

1.2.2. Subir los datos al core de Solr

Para subir los datos, simplemente hacemos un POST del archivo .json que nos genera el *webcrawler*, de esta forma se subiran y guardaran en el core los datos crawleados.

```
    /RIWS/solr-8.4.1$ bin/post -c riws-filmaffinity example/fa/
    filmaffinity-data.json
```

1.2.3. Arreglar el problema del CORS

El Intercambio de Recursos de Origen Cruzado (**CORS**) es un mecanismo que utiliza cabeceras HTTP adicionales para permitir que un textituser agent obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al que pertenece.

Al intentar acceder desde nuestra aplicación web al motor de búsqueda nos indicaba que no existía este control de acceso al recurso solicitado. Para ello hemos tenido que modificar el archivo web.xml, localizado en /solr-8.4.1/server/solr-webapp/webapp/WEB-INF.

```
<filter>
  <filter-name>cross-origin</filter-name>
  <filter-class>org.eclipse.jetty.servlets.CrossOriginFilter</
    filter-class>
  <init-param>
    <param-name>allowedOrigins</param-name>
    <param-value>*</param-value>
  </init-param>
  <init-param>
    <param-name>allowedMethods</param-name>
    <param-value>GET,POST,OPTIONS,DELETE,PUT,HEAD</param-value>
  </init-param>
  <init-param>
    <param-name>allowedHeaders</param-name>
    <param-value>origin, content-type, accept</param-value>
  </init-param>
</filter>
```

1.2.4. Cambiar campo default en el solrconfig.xml

Este cambio fue necesario para indexar el campo **title** por defecto en la query, esto es necesario ya que sin eso la indexación de los datos nos dará un error una vez tengamos añadido el **solr.SuggestComponent**.

```

<requestHandler name="/select" class="solr.SearchHandler">
<!-- default values for query parameters can be specified, these
     will be overridden by parameters in the request
  -->
<lst name="defaults">
  <str name="echoParams">explicit</str>
  <int name="rows">10</int>

     <str name="df">title</str>
```

1.2.5. Añadir el solr.SuggestComponent en el solrconfig.xml

Se añade la configuración del suggester de Solr implementado en la clase **solr.SuggestComponent**

```
<searchComponent name="suggest" class="solr.SuggestComponent">
 <lst name="suggester">
  <str name="name">filmaffinitySuggester</str>
  <str name="lookupImpl">FuzzyLookupFactory</str>
  <str name="dictionaryImpl">DocumentDictionaryFactory</str>
  <str name="field">title</str>
  <str name="suggestAnalyzerFieldType">text_general</str>
  <str name="weightField">(avg_rating / rate_count)</str>
  <str name="suggestAnalyzerFieldType">string</str>
  <str name="buildOnStartup">true</str>
 </lst>
</searchComponent>
```

Dentro del **searchComponent**, se le estan indicando varias cosas de las cuales destacamos:

- **name**: nombre del diccionario de sugerencias.
- **field**: campo sobre el que se va a realizar las sugerencias.
- **suggestAnalyzerFieldType**: tipo de dato de la sugerencia.
- **weightField**: un campo o expresion del que se va a sacar el peso que ordenará las sugerencias extraidas.

1.2.6. Añadir requestHandler para solr.SuggestComponent

Para poder realizar peticiones al **solr.SuggestComponent** que hemos añadido, es necesario crear un **requestHandler** para poder realizar peticiones desde nuestra aplicacion.

```
<requestHandler name="/suggest" class="solr.SearchHandler"
    startup="lazy">
  <lst name="defaults">
    <str name="suggest">true</str>
    <str name="suggest.count">10</str>
  </lst>
  <arr name="components">
    <str>suggest</str>
  </arr>
</requestHandler
```

1.3. Implementar una aplicación que use el *search engine* de Solr.

El último paso del desarrollo ha sido desarrollar una aplicación web con una interfaz que interactúe con Solr. Para su creación hemos empleado el **Angular**

CLI desde la línea de comandos, al igual que los componentes que hemos ido añadiendo (sidenav, filmaffinit-list, etc.).

La aplicación dispone de un *sidenav* para poder navegar entre el listado de películas y series que, en ambos casos, nos llevarán al listado correspondiente. En este listado se puede apreciar el conjunto de filtros por los que se permitirá buscar y ver en tiempo real los resultados deseados.

En cuanto a la lógica que procesa las peticiones a Solr existe un servicio (FilmaffinityService) que gestionará todas las funcionalidades gracias a dos únicas llamadas. Estas formarán las *querys* solicitadas por los usuarios.

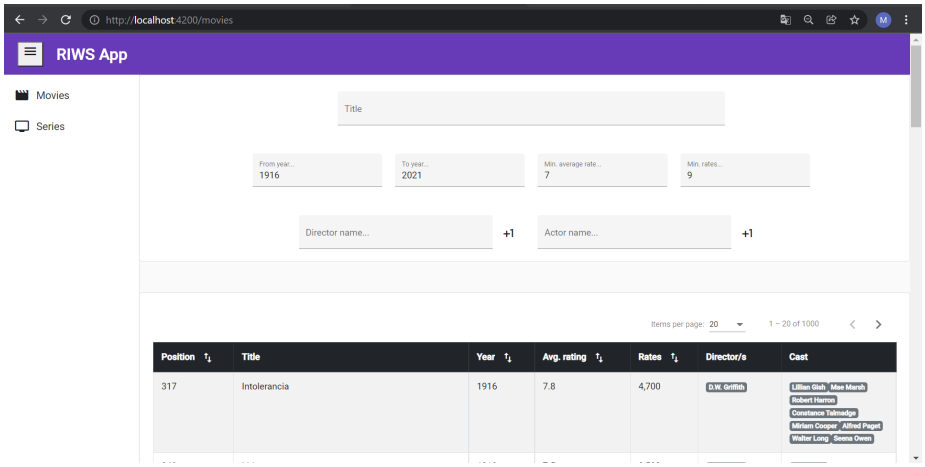

Figura 1: Página principal de la aplicación

2. Funcionalidades implementadas

En la **aplicación web**, implementamos diferentes funcionalidades con respecto a la búsqueda de las series y las películas. Estas son:

- **Separación entre películas y series:** un aspecto importante a destacar, es que las búsquedas van agrupadas en base a películas o series. De esta forma separamos los 2 tops, aunque cuentan con el mismo tipo de filtro.
- **Filtrado de resultados:** existe un filtro con diferentes opciones (título, fecha mínima, fecha máxima...) a medida que se escribe en el filtro se realiza la búsqueda sin necesidad de darle a ENTER o a algún botón.
- **Sugerencias y texto predictivo:** El título, directores y actores cuentan con texto predictivo en la búsqueda, esto quiere decir que se comparará la palabra que se escribe con lo posibles resultados de esa palabra. A mayores, en el campo de texto se realizan sugerencias de búsqueda en base al texto escrito y ordenadas por popularidad.
- **Ordenación por campos:** En la aplicación se pueden ordenar los resultados obtenidos por posición, año de estreno, media de valoraciones y total valoraciones. Tanto de manera ascendente como descendente.
- **Paginación y tamaño de búsqueda:** Por último,los resultados obtenidos serán paginados, y a mayores de esto, también se puede modificar el tamaño de la búsqueda entre 20, 50 y 100 resultados.

3. Tecnologías usadas (motivación y uso)

Las tecnologías utilizadas en esta práctica se dividen en qué uso o función principal desempeñaron:

- ***Webcrawler:*** Como *webcrawler* utilizamos **Scrapy** por 2 movivos principales: el primero de ellos es su sencilla sintaxis y la facilidad de aprender y entender el paradigma. La segunda es debido a su amplia documentación y su extenso uso, facilitando la búsqueda de soluciones dado un determinado error.

  En un principio habíamos empezado a trabajar con Apache Nutch para realizar el *crawling*, pero nos resultó una tarea más atractiva y sencilla de realizar con Scrapy.

- **Motor de búsqueda:** En este caso optamos por usar **Apache Solar**, debido a la distinta información que teniamos de él, en un principio, frente a otras opciones (p.ej Elastic Search).

- **Aplicación Web:** Para la aplicación decidimos que la mejor opción para nosotros sería desarrollarla en **Angular**. Esto se debe a que nos sentimos más cómodos en Angular que en otros frameworks (p.ej React o VueJS).

  Además de que consideramos que es un framework que dispone de una gran cantidad de documentación propia para su desarrollo, sobretodo acerca de sus componentes.