

**PRÁCTICA 6 - Implementación de un
Sistema de Recuperación de Información
utilizando Lucene.
Indexación de Facetas.
MANUAL DE USUARIO**



**UNIVERSIDAD
DE GRANADA**

Integrantes:

- Rafael Luque Framit
- Cristóbal Jiménez Álvarez

1. Indexación

En primer lugar, para poder realizar el proceso de Indexación hemos creado la clase *Indexador_P4.java* la cual almacena la información relacionada a los archivos CSV de los episodios de los Simpsons.

Por un lado, nos encontramos con los archivos de tipo **Guiones** los cuales son archivos CSV que almacenan la información de cada diálogo dicho por un personaje línea a línea. En estos tipos de archivos hemos ido extrayendo los diferentes campos a través de una biblioteca capaz de manejar archivos CSV de forma cómoda y eficiente. Esta información se guarda en la variable *dialogsReader* de tipo *CSVReader*, después en un bucle *while* vamos a ir almacenando cada campo de cada línea, estableciendo así los diferentes campos de indexación de Guiones:

- **episode_id**: El número de episodio.
- **number**: Entero que representa la posición (orden) del diálogo en el episodio
- **timestamp_in_ms**: tiempo desde el inicio del capítulo
- **raw_character_text**: Personaje que habla
- **raw_location_text**: Ubicación del discurso
- **spoken_words**: El diálogo en sí.

El índice creado genera varios ficheros los cuales se van a almacenar en el directorio *./indexGuiones*. Además de ser **“raw_location_text”** un campo de indexación, lo hemos establecido como *Faceta*, para así poder posteriormente filtrar los resultados de la búsqueda a partir de estos campos.

Por otro lado, nos encontramos con los archivos de tipo **CapitulosUnidos** los cuales son archivos CSV que almacenan la información de cada diálogo dicho por un personaje en una misma celda. En estos tipos de archivos hemos ido extrayendo los diferentes campos a través de una biblioteca capaz de manejar archivos CSV de forma cómoda y eficiente. Esta información se guarda en la variable *capitulosReader* de tipo *CSVReader*, después en un bucle *while* vamos a ir almacenando cada campo de cada línea, estableciendo así los diferentes campos de indexación de CapitulosUnidos:

- **episode_id**: El número de episodio.
- **spoken_words**: Todos los diálogos de los que constan el episodio
- **raw_character_text**: Listado de todos los personajes que intervienen en el capítulo, en orden alfabético.
- **imdb_rating**: La valoración media del capítulo en IMDB (Internet Movie Database)
- **imdb_votes**: Número de votos recibidos
- **number_in_season**: número del episodio en la temporada (igual que *episode_id*)
- **original_air_date**: Fecha de emisión original
- **original_air_year**: Año de emisión
- **season**: Temporada del capítulo
- **title**: Título del capítulo
- **us_viewers_in_millions**: Número de espectadores en US (en millones)
- **views**: Espectadores totales

El índice creado genera varios ficheros los cuales se van a almacenar en el directorio *./indexCapitulosUnidos*. Además de ser **“season”** un campo de indexación, lo hemos establecido como *Faceta*, para así poder posteriormente filtrar los resultados de la búsqueda a partir de estos campos.

```

379 public void configurarFaceta1() throws IOException {
380     taxoDir = FSDirectory.open(Paths.get(facetPathGuiones));
381     fconfig1 = new FacetsConfig();
382     fconfig1.setMultiValued("raw_location_text", true);
383     taxoWriter1 = new DirectoryTaxonomyWriter(taxoDir);
384 }
385 public void configurarFaceta2() throws IOException {
386     taxoDir = FSDirectory.open(Paths.get(facetPathCapitulosUnidos));
387     fconfig2 = new FacetsConfig();
388     fconfig2.setMultiValued("season", true);
389     taxoWriter2 = new DirectoryTaxonomyWriter(taxoDir);
390 }
391 public void configurarIndice1(Similarity similarity) throws IOException {
392     PerFieldAnalyzerWrapper analizadorCampos1 = analizadorPorCampo1();
393     IndexWriterConfig config1 = new IndexWriterConfig(analizadorCampos1);
394     config1.setSimilarity(similarity);
395     if (create) {
396         config1.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
397     } else { config1.setOpenMode(IndexWriterConfig.OpenMode.CREATE OR APPEND); }
398     Directory indexDir1 = FSDirectory.open(FileSystems.getDefault().getPath(indexPathGuiones));
399     indexWriterGuiones = new IndexWriter(indexDir1, config1);
400 }
401 public void configurarIndice2(Similarity similarity) throws IOException {
402     PerFieldAnalyzerWrapper analizadorCampos2 = analizadorPorCampo2();
403     IndexWriterConfig config2 = new IndexWriterConfig(analizadorCampos2);
404     config2.setSimilarity(similarity);
405     if (create) {
406         config2.setOpenMode(IndexWriterConfig.OpenMode.CREATE);
407     } else { config2.setOpenMode(IndexWriterConfig.OpenMode.CREATE OR APPEND); }
408     Directory indexDir2 = FSDirectory.open(FileSystems.getDefault().getPath(indexPathCapitulosUnidos));
409     indexWriterCapitulosUnidos = new IndexWriter(indexDir2, config2);
410 }

```

Figura 1: Configuraciones de los índices/facetos.

Para añadir las facetos al índice hemos utilizado las siguientes líneas de código:

```

doc.add(new FacetField("season", discursoLinea[9]));
doc.add(new FacetField("raw_location_text", discursoLinea[5]));

```

Figura 2: Añadir los campos como facetos.

2. Realización de la búsqueda

El objetivo de esta práctica ha sido seguir profundizando en Lucene como herramienta para construir un SRI. En este caso, asumimos que ya tenemos un índice creado, sobre el que un usuario desea encontrar los documentos relevantes a una necesidad de información. Lucene nos permite realizar una gran variedad de consultas (query).

En primer lugar, hemos creado dos IndexReader, uno para Guiones y otro para Capítulos Unidos, este se encarga de ver el contenido de los índices. En segundo lugar, hacemos uso del IndexSearcher, el cual es la clase encargada de realizar la búsqueda sobre el índice, dispone de distintos métodos para realizar la búsqueda y el conjunto de documentos más relevante los devuelve en un objeto de la clase TopDocs.

Como hemos visto en el guión de la práctica y en documentación complementaria, con el IndexSearcher podremos, entre otros:

- Consultar la medida de similitud (modelo de recuperación) `searcher.getSimilarity()`. Por defecto es BM25Similarity.
- Modificar la medida de similitud `searcher.setSimilarity(Similarity sim)`

- Obtener un Document searcher.doc(int docID) del índice.
- Realizar una consulta: searcher.search(Query Q,int N) devolviendo los top N documentos relevantes a Q considerando la medida de similitud.
- Explicar cómo se ha obtenido el escore de un documento ante la consulta searcher.explain(Query q, int doc)

El proceso de búsqueda puede llegar a ser muy complejo ya que se pueden realizar consultas usando desde una única palabra, hasta búsquedas por campos o aplicando lógica booleana sobre varios de estos campos. La clase **búsqueda** que hemos implementado lee desde el directorio que contiene nuestros índices de Guiones y Capítulos Unidos y asigna una medida de similitud, por defecto es BM25, para establecer cómo se comparan los documentos.

```

340 public String indexSearch(boolean esGuiones, String consulta) {
341     IndexReader reader = esGuiones ? indexReaderGuiones : indexReaderCapitulosUnidos;
342     try {
343         searcher = new IndexSearcher(reader);
344         searcher.setSimilarity(new BM25Similarity());
345         fconfig = new FacetsConfig();
346         fcollector = new FacetsCollector();
347         String[] campos = esGuiones ?
348             new String[]{"raw_character_text", "spoken_words", "raw_location_text", "number", "episode_id", "file"} :
349             new String[]{"episode_id", "spoken_words", "raw_character_text", "imdb_rating", "number_in_season",
350                 "original_air_date", "season", "title", "views", "file"};
351
352         MultiFieldQueryParser parser = new MultiFieldQueryParser(campos, new StandardAnalyzer());
353         String line = consulta.trim();
354         if (line.length() == 0) {
355             return "Consulta vacía";
356         }
357         query = parser.parse(line);
358         ArrayList<String> result = searchAndPrintResults(searcher, query);
359         return String.join("\n", result);
360     } catch (IOException | ParseException e) {
361         e.printStackTrace();
362         return "Error en la búsqueda";
363     }
364 }

```

Figura 3: Método indexSearch para la búsqueda.

3. Búsqueda genérica

Esta es la búsqueda estándar de cualquier sistema de recuperación de información; es decir, la búsqueda que haría Google por defecto. Para ello, en nuestra interfaz debemos introducir lo que queremos buscar y el sistema hará una búsqueda general en el índice como explicamos a continuación.

Primero debemos inicializar el parser a utilizar, que en este tipo de búsqueda será un StandardAnalyzer sobre aquellos campos que queramos trabajar, tanto Guiones como Capítulos Unidos, que almacenan toda la información de la que disponemos de cada una de los documentos.

Para que nuestro sistema funcione verdaderamente como un buen sistema recuperador de información, tenemos que estar preparados para cualquier posible consulta por muy mal estructurada que pueda estar. Para ello, separamos los tokens de la consulta por espacios.

Por ejemplo, si la búsqueda introducida es "Bart Halloween", primero parseamos la palabra Bart y posteriormente la palabra Halloween. Así, nos aseguramos de hacer el match correcto en las ocurrencias de nuestro índice. Además, por supuesto también recuperamos aquellas películas que únicamente hagan match con Bart y las que únicamente hagan match con Halloween.

Finalmente, hacemos la búsqueda de la consulta y llamamos al método *searchAndPrintResults* pasándole por parámetro tanto el *search* como el *query*. Por último, el método devuelve ese ArrayList de String con los resultados obtenidos.

Para la recogida y muestra de resultados hemos implementado el método *searchAndPrintResults*, el cual devuelve un ArrayList de String el cual llevará los resultados obtenidos en forma de String. Hacemos uso de una variable *TopDocs* y un vector *ScoreDocs[]*. En un bucle for recorremos este vector y añadimos al ArrayList la información de cada documento. El código final sería:

```
454 private ArrayList<String> searchAndPrintResults(IndexSearcher searcher, Query query) throws IOException {
455     ArrayList<String> resultList = new ArrayList<>();
456     results = searcher.search(query, DOCUMENTOS);
457     ScoreDoc[] hits = results.scoreDocs;
458     long numTotalHits = results.totalHits.value;
459     resultList.add(numTotalHits + " documentos encontrados");
460     for (int j = 0; j < hits.length; j++) {
461         Document doc = searcher.doc(hits[j].doc);
462         // Construir la cadena con la información del documento
463         Arrays.stream(doc.getFields().toArray())
464             .forEach(field -> {
465                 resultList.add(((IndexableField) field).name() + ": " + doc.get(((IndexableField) field).name()));
466             });
467         resultList.add("-----");
468     }
469     return resultList;
470 }
```

Figura 4: Método para mostrar los resultados.

4. Búsqueda Booleana Guiones

Esta búsqueda se basa en realizar una consulta sobre varios campos de forma simultánea para poder obtener unos resultados más precisos. En este tipo de búsquedas, el usuario debe tener cierto conocimiento sobre lo que busca, pues debe introducir los datos correctos en cada uno de los campos.

Hemos decidido que los campos del índice Guiones sobre los que se pueden hacer búsquedas sean:

- **episode_id**: El número de episodio
- **spoken_words**: El diálogo en sí.
- **raw_location_text**: Ubicación del discurso.
- **raw_character_text**: Personaje que habla.
- **number**: Entero que representa la posición (orden) del diálogo en el episodio.

Para este tipo de búsquedas, la estructura es parecida a la anterior, pero esta vez vamos viendo campo a campo si hay un valor para hacer la búsqueda y en caso afirmativo lo añadimos a la consulta.

Por último, realizamos una única consulta *BooleanClause* a partir de lo anterior e indicamos con la cláusula *MUST* o *SHOULD* que queremos una búsqueda booleana equivalente a *AND* o *OR* respectivamente. Finalmente, llamamos al método *searchAndPrintResults* pasándole por parámetro tanto el *search* como el *query*. Por último, el método devuelve ese *ArrayList* de *String* con los resultados obtenidos.

```

158 |         if (campos[4].compareTo("") != 0) {
159 |             int[] values = Arrays.stream(campos[4].split(" "))
160 |                 .mapToInt(Integer::parseInt)
161 |                 .toArray();
162 |             Query querynumber = IntPoint.newSetQuery("number", values);
163 |             consultas.add(querynumber);
164 |         }
165 |
166 |         for (Query consulta : consultas) {
167 |             BooleanClause.Occur occurType = BooleanClause.Occur.MUST; // Por defecto, AND
168 |             if (operador.equalsIgnoreCase("OR")) {
169 |                 occurType = BooleanClause.Occur.SHOULD; // Si se especifica OR
170 |             }
171 |             bqbuilder.add(new BooleanClause(consulta, occurType));
172 |         }
173 |         query = bqbuilder.build();
174 |         ArrayList<String> result = searchAndPrintResults(searcher, query);
175 |         return String.join("\n", result);
176 |     }

```

Figura 5: Parte de código de *ConsultaBooleanaGuiones*.

5. Búsqueda Booleana Capítulos Unidos

Esta búsqueda se basa en realizar una consulta sobre varios campos de forma simultánea para poder obtener unos resultados más precisos. En este tipo de búsquedas, el usuario debe tener cierto conocimiento sobre lo que busca, pues debe introducir los datos correctos en cada uno de los campos.

Hemos decidido que los campos del índice Capítulos Unidos sobre los que se pueden hacer búsquedas sean:

- **episode_id**: El número de episodio.
- **spoken_words**: El diálogo en sí.
- **raw_character_text**: Personaje que habla.
- **imdb_rating**: La valoración media del capítulo en IMDB (Internet Movie Data Base).
- **number_in_season**: número del episodio en la temporada (igual que *episode_id*).
- **original_air_date**: Fecha de emisión original.
- **season**: Temporada del capítulo.
- **title**: Título del capítulo.
- **views**: Espectadores totales.

Para este tipo de búsquedas, la estructura es parecida a la anterior, pero esta vez vamos viendo campo a campo si hay un valor para hacer la búsqueda y en caso afirmativo lo añadimos a la consulta. Para algunos campos, como *views* y *imdb_votes*, hemos puesto que busque con un rango de error, para que se muestren resultados cercanos a los del valor de la búsqueda.

Por último, realizamos una única consulta *BooleanClause* a partir de lo anterior e indicamos con la cláusula *MUST* o *SHOULD* qué queremos una búsqueda booleana equivalente a *AND* o *OR* respectivamente. Finalmente, llamamos al método *searchAndPrintResults* pasándole por parámetro tanto el *search* como el *query*. Por último, el método devuelve ese *ArrayList* de *String* con los resultados obtenidos. Resultando el siguiente código:

```

294     if (campos[8].compareTo("") != 0) {
295         int[] viewsValues = Arrays.stream(campos[8].split(" "))
296             .mapToInt(Integer::parseInt)
297             .toArray();
298
299         BooleanQuery.Builder booleanQueryBuilder = new BooleanQuery.Builder();
300         for (int viewsValue : viewsValues) {
301             int minRange = viewsValue - 5000;
302             int maxRange = viewsValue + 5000;
303             Query rangeQuery = IntPoint.newRangeQuery("views", minRange, maxRange);
304             booleanQueryBuilder.add(rangeQuery, BooleanClause.Occur.SHOULD);
305         }
306         Query finalQuery = booleanQueryBuilder.build();
307         consultas.add(finalQuery);
308     }
309
310     for (Query consulta : consultas) {
311         BooleanClause.Occur occurType = BooleanClause.Occur.MUST; // Por defecto, AND
312         if (operador.equalsIgnoreCase("OR")) {
313             occurType = BooleanClause.Occur.SHOULD; // Si se especifica OR
314         }
315         bqbuilder.add(new BooleanClause(consulta, occurType));
316     }
317     query = bqbuilder.build();
318     ArrayList<String> result = searchAndPrintResults(searcher, query);
319     return String.join("\n", result);
320 }

```

Figura 6: Parte de código de *ConsultaBooleanaCapitulosUnidos*.

6. Muestra de Facetas

Una vez realizada la consulta sobre el índice, el Sistema de Recuperación de Información debe permitir un filtrado de los resultados obtenidos. En este momento, la herramienta que nos proporciona el uso de las facetas es la más útil y la que utilizaremos en nuestro proyecto. La búsqueda por facetas requiere que se añadan a un directorio en tiempo de indexación, cosa que ya hemos explicado antes en la indexación.

En la clase *Búsqueda* y al igual que hicimos con el índice, deberemos indicar el directorio donde están almacenadas las facetas que hemos creado en tiempo de indexación. Además crearemos tres variables que contienen los objetos *TaxonomyReader* al cual asignamos el directorio donde se encuentran las facetas y otros dos *FacetsConfig* y *FacetsCollector* que usaremos más adelante.

Para mostrar las facetas, hemos creado un método llamado *mostrarFacetas* que usa una consulta y a partir de ella obtiene las facetas de la colección de archivos que se adecuen a

dicha consulta. Las facetas contienen información sobre los campos “season” si es para CapítulosUnidos y “raw_location_text” si es para Guiones.

Guardaremos en allDims la lista de todas las facetas obtenidas para la consulta aplicada a los documentos y cada faceta será un objeto FaceResult que contiene la información específica de cada faceta. La variable FastTaxonomyFacetCounts almacenará el total de ocurrencias para cada faceta para esa consulta determinada.

Finalmente, se devuelve el ArrayList donde muestran cada uno de las facetas junto con su valor para la consulta realizada sobre la colección de documentos indexados.

```
348 public ArrayList<String> MostrarFacetas(boolean esGuiones) {
349     vector_facetas = new String[4 * 100];
350     map_faceta_season = new HashMap<String, String>();
351     ArrayList<String> l = new ArrayList<String>();
352     ddq = new DrillDownQuery(fconfig, query);
353     categorias = new ArrayList<String>();
354     int i = 0;
355     try {
356         //FacetsCollector fcl = new FacetsCollector();
357         tdc = FacetsCollector.search(searcher, ddq, 10, fcollector);
358         Facets fcCount=null;
359         if (!esGuiones){fcCount = new FastTaxonomyFacetCounts(taxoReaderCapitulosUnidos, fconfig, fcollector);}
360         else{fcCount = new FastTaxonomyFacetCounts(taxoReaderGuiones, fconfig, fcollector);}
361
362         List<FacetResult> allDims = fcCount.getAllDims(100);
363         // Para cada categoria mostramos el valor de la etiqueta y su numero de ocurrencias
364         for (FacetResult fr : allDims) {
365             categorias.add(fr.dim);
366             int cont = 0;
367             // Almacenamos cada etiqueta en un vector de 3*TOP casillas para guardar todas las que mostramos
368             for (LabelAndValue lv : fr.labelValues) {
369                 vector_facetas[i] = new String(fr.dim + " (#n)-> " + lv.label + " (" + lv.value + ")");
370                 map_faceta_season.put(lv.label, fr.dim);
371                 l.add(fr.dim + ": " + lv.label + " (" + lv.value + ")");
372                 cont++;
373                 i++;
374             }
375         }
376     } catch (IOException e) {
377         System.out.println("Error al mostrar facetas. ");
378     }
379     return new ArrayList<>(l);
380 }
```

Figura 7: Método para mostrar las Facetas.

7. Filtrar por Facetas

Una vez recopiladas las facetas, podremos realizar un filtrado de la búsqueda haciendo uso de ellas. Hemos creado un método llamado FiltrarPorFacetas que recibe por parámetro un String con el valor de la faceta. El objetivo es realizar un filtrado por DrillDown sobre los documentos resultantes de una consulta para las categorías seleccionadas.

Esto lo podemos llevar a cabo usando el objeto DrillDownQuery al que le pasaremos la consulta inicial y todas las facetas y luego haremos el filtrado sobre las facetas seleccionadas por el usuario.

Una vez añadidas al DrillDownQuery todas las facetas, llamaremos al método FacetsCollector.search al que le pasaremos el nuevo objeto DrillDownQuery y nos devolverá un nuevo objeto TopDocs con el filtro aplicado sobre los documentos que habíamos obtenido a partir de la primera consulta.


```

383 public String FiltrarPorFacetas(String valorFaceta) {
384     // Inicializamos el DrillDownQuery con la consulta realizada
385     DrillDownQuery ddq = new DrillDownQuery(fconfig, query);
386     ArrayList<String> result = new ArrayList<>();
387     try {
388         // Dividir la entrada en tres partes
389         String[] partes = valorFaceta.split(": |\\(");
390         partes[2] = partes[2].substring(0, partes[2].length() - 1); //quitar )
391         // Limpiar los espacios en blanco alrededor de las partes
392         for (int i = 0; i < partes.length; i++) {
393             partes[i] = partes[i].trim();
394         }
395         // Mostrar las partes
396         for (String parte : partes) {
397             System.out.println(parte);
398         }
399         // Realizamos operación AND entre la dimensión y el valor de la faceta
400         ddq.add(partes[0], partes[1]);
401         // Volvemos a hacer la búsqueda con el nuevo ddq que contiene las facetas.
402         FacetsCollector fcollector = new FacetsCollector();
403         TopDocs tdc = FacetsCollector.search(searcher, ddq, 10, fcollector);
404         totalHits = tdc.totalHits.value;
405
406         // Mostrar resultados (o haz lo que necesites con ellos)
407         result = PrintFilteredResults(tdc);
408         return String.join("\n", result);
409     } catch (IOException e) {
410         System.out.println("Error al filtrar facetas. ");
411     }
412     return String.join("\n", result);
413 }

```

Figura 8: Método para filtrar por Facetas.

8. Posibles Facetas

Durante la práctica planteamos la posibilidad de realizar más tipos de facetas para otros campos como por ejemplo:

- **Facetas Jerárquicas:** planteamos realizar una faceta jerárquica para el campo "raw_character_text" tanto para Guiones como para CapítulosUnidos. Esta jerarquía podría haber seguido la siguiente distribución. En una primera línea del árbol pondremos "Personajes", en la siguiente línea del árbol habría 3 diferentes categorías: "Principales", "Secundarios" y "Extras". Y por último, en la siguiente línea del árbol meteremos los distintos personajes como por ejemplo: "Marge Simpson" como principal, "Milhouse" como secundario y "Lady Gaga" como extra. Esta distribución finalmente no ha podido realizarse debido a la forma de los datos y la distribución de los archivos CSV.
- **Facetas por rangos:** planteamos realizar facetas por rangos para los campos "imdb_rating" y para "views" para CapítulosUnidos. Esta jerarquía hubiera consistido para cada campo realizar las facetas siguiendo un array con los diferentes rangos de valores que hubieran seguido estos campos. Esta distribución finalmente no la hemos realizado debido a que ya trabajamos con rangos en estos campos a la hora de indexarlos.

9. Interfaz gráfica de usuario

Para esta práctica, hemos diseñado una interfaz que consideramos adecuada que integra todas las posibles funcionalidades explicadas anteriormente. Esta interfaz no es definitiva en nuestro sistema de recuperación de información ya que más adelante deberemos implementar el código para las facetas y por tanto deberá cambiar la interfaz. Dicha interfaz tiene el siguiente aspecto en el momento de apertura:

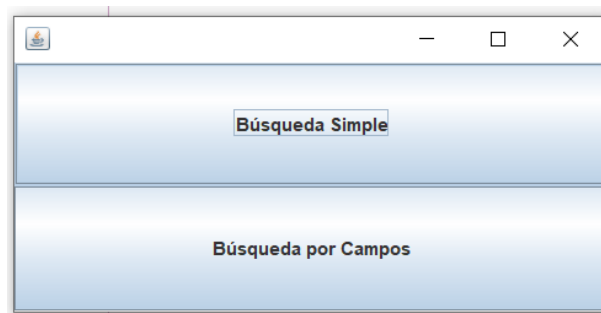


Figura 9: Pantalla de inicio de la interfaz.

Desde aquí, tenemos dos menús disponibles:

- Búsqueda Simple: menú para búsquedas simples sobre Guiones y Capítulos Unidos.
- Búsqueda por Campos: menú para búsquedas por campos (booleanas).

En primer lugar, creamos un JPanel con sus características correspondientes. Seguidamente creamos el JButton para Búsqueda Simple y otro para Búsqueda por campos. Si el usuario pulsa uno de los dos llamará al método abrirBusquedaForm pasándole por parámetro false o true respectivamente. Aquí el código:

```
73 public MainFrame() {
74     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
75     setSize(400, 200);
76     // Crear un JPanel con un layout de cuadrícula
77     JPanel panel = new JPanel(new GridLayout(2, 1));
78     // Botón "Búsqueda Simple"
79     JButton simpleSearchButton = new JButton("Búsqueda Simple");
80     simpleSearchButton.addActionListener(new ActionListener() {
81         @Override
82         public void actionPerformed(ActionEvent e) {
83             // Abrir el formulario de búsqueda simple
84             abrirBusquedaForm(false);
85         }
86     });
87     // Botón "Búsqueda por Campos"
88     JButton fieldSearchButton = new JButton("Búsqueda por Campos");
89     fieldSearchButton.addActionListener(new ActionListener() {
90         @Override
91         public void actionPerformed(ActionEvent e) {
92             // Abrir el formulario de búsqueda por campos
93             abrirBusquedaForm(true);
94         }
95     });
96     // Agregar botones al panel
97     panel.add(simpleSearchButton);
98     panel.add(fieldSearchButton);
99     // Agregar el panel al JFrame
100     getContentPane().add(BorderLayout.CENTER, panel);
101     // Hacer visible el JFrame
102     setVisible(true);
103 }
```

Figura 10: Código inicial del MainFrame.

En segundo lugar, creamos el método `abrirBusquedaForm` el cual abre los índices, crea las instancias de búsqueda y por último crea el Form correspondiente a lo que se le haya pasado por parámetro, si es `false` lo crea para una búsqueda simple y si es `true` lo crea para una búsqueda por campos. El código sería el siguiente:

```

105 // Método para abrir el formulario de búsqueda
106 private void abrirBusquedaForm(boolean campos) {
107     try {
108         FSDirectory guionesDirectory = FSDirectory.open(Paths.get("./indexGuiones"));
109         FSDirectory capitulosUnidosDirectory = FSDirectory.open(Paths.get("./indexCapitulosUnidos"));
110         TaxonomyReader taxoReaderGuiones = new DirectoryTaxonomyReader(FSDirectory.open(Paths.get("./facetGuiones")));
111         TaxonomyReader taxoReaderCapitulosUnidos = new DirectoryTaxonomyReader(FSDirectory.open(Paths.get("./facetCapitulosUnidos")));
112         // Crear instancias de Búsqueda
113         IndexReader guionesReader = DirectoryReader.open(guionesDirectory);
114         IndexReader capitulosUnidosReader = DirectoryReader.open(capitulosUnidosDirectory);
115         Analyzer analyzer = new StandardAnalyzer();
116         busqueda busqueda = new busqueda(guionesReader, capitulosUnidosReader, taxoReaderGuiones, taxoReaderCapitulosUnidos, analyzer);
117         // Crear el formulario y mostrarlo
118         if(campos == false)
119             SwingUtilities.invokeLater(() -> new BusquedaForm(busqueda));
120         if (campos == true)
121             SwingUtilities.invokeLater(() -> new BusquedaFormCampos(busqueda));
122     } catch (IOException e) {
123         e.printStackTrace();
124     }
125 }

```

Figura 9: Método `abrirBusquedaForm`.

10. BusquedaForm (Búsquedas Simples)

La interfaz para búsquedas simples tiene el siguiente aspecto en el momento de la apertura:

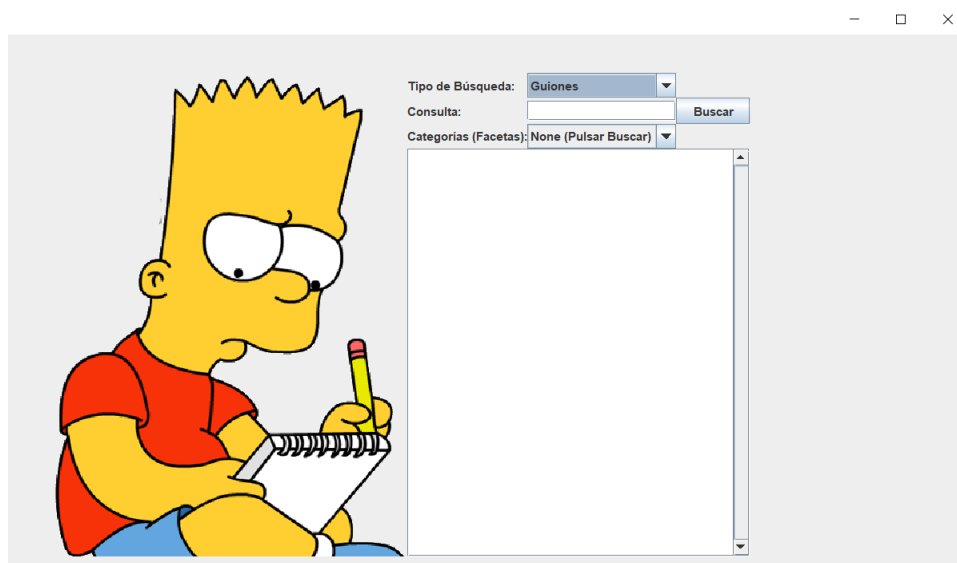


Figura 10: Pantalla de inicio de búsquedas simples.

Hemos creado primero un `JPanel` con una imagen de Bart Simpson de fondo, hemos añadido dos `JLabel` que contienen “Tipo de Búsqueda”, “Consulta” y Categorías (Facetas). Además, hemos añadido un `JComboBox<String>` el cual contiene Guiones y Capitulos unidos, así podemos seleccionar sobre qué índice queremos realizar la búsqueda simple. También, hemos añadido otro `JComboBox<String>` para seleccionar entre las diferentes facetas disponibles.

Por otro lado, añadimos un JTextField para poder escribir la consulta, por último añadimos el botón para buscar. Debajo de todo esto hemos añadido un JTextArea el cual mostrará los resultados obtenidos en la búsqueda.

Todo esto anterior, lo hemos podido situar y modificar con los ajustes de gridx, gridy y gridheight entre otros. Al pulsar el botón de buscar, se llama a la acción buscarButton el cual consulta que tipo de búsqueda queremos realizar y llama al método correspondiente según la selección, por último muestra el resultado en el JTextArea.

```

125 // Resultado
126 c.gridx = 1; // Cambiar a la columna 1
127 c.gridy = 3; // Cambiar a la fila 2
128 c.gridwidth = 3; // Ocupar tres columnas
129 c.gridheight = GridBagConstraints.REMAINDER; // Ocupar el resto de las filas
130 c.fill = GridBagConstraints.BOTH; // Ocupar tanto horizontal como verticalmente
131 resultadoTextArea = new JTextArea();
132 resultadoTextArea.setEditable(false);
133 resultadoTextArea.setLineWrap(true);
134 resultadoTextArea.setRows(20);
135 JScrollPane scrollPane = new JScrollPane(resultadoTextArea);
136 scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
137 panel.add(scrollPane, c);
138
139 // ActionListener para el JComboBox de categorías
140 categoriasComboBox.addActionListener(new ActionListener() {
141     @Override
142     public void actionPerformed(ActionEvent e) {
143         // Obtener la categoría seleccionada
144         String categoriaSeleccionada = (String) categoriasComboBox.getSelectedItem();
145
146         // Verificar si se seleccionó "None"
147         if ("None (Pulsar Buscar)".equals(categoriaSeleccionada)) {
148             // Realizar acciones específicas cuando se selecciona "None"
149             // Puedes dejarlo vacío o mostrar un mensaje, según tus necesidades
150             System.out.println("No se realizará ninguna acción");
151         } else {
152             // Realizar acciones adicionales según la categoría seleccionada
153             // Puedes llamar a tu función FiltrarPorFacetas aquí
154             String resultado = busqueda.FiltrarPorFacetas(categoriaSeleccionada);
155             actualizarResultado(resultado);
156         }
157     }
158 });

```

Figura 11: Parte del código de Búsqueda Simple.

11. BusquedaFormCampos (Búsqueda por campos)

Primero de todo, se nos abre una ventana para escoger entre que índice queremos realizar la búsqueda, si Guiones o Capítulos Unidos. La interfaz para búsquedas por campos tiene el siguiente aspecto en el momento de la apertura:

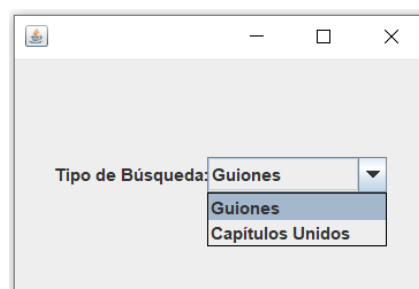


Figura 13: Pantalla de inicio de búsquedas por campos.

Para esta parte de la interfaz también primero hemos creado un JPanel en el que incluiremos un JComboBox<String> el cual marcará si queremos hacer la búsqueda por Guiones o por Capítulos Unidos. Dependiendo de lo que se marque realizará una acción u otra.

En el caso de que marque Guiones O Capítulos Unidos se abrirá una nueva ventana en la cual hemos creado primero un JPanel con una imagen de Bart Simpson de fondo y hemos añadido JLabels y JTextFields para añadir los distintos campos de búsqueda que tiene el índice Guiones o Capítulos Unidos. También, hemos añadido otro JComboBox<String> para seleccionar entre las diferentes facetas disponibles.

Por otro lado, hemos añadido un JButton y un JTextArea. Todo esto anterior, lo hemos podido situar y modificar con los ajustes de gridx, gridy y gridheight entre otros. Además, añadimos un JComboBox<String> con los operadores AND y OR dependiendo del tipo de consulta que queramos realizar. Al pulsar el botón de buscar, se llama a la acción buscarButton el cual consulta que tipo de búsqueda queremos realizar y llama al método correspondiente según la selección, por último muestra el resultado en el JTextArea. El código sería el siguiente en el caso de Guiones:

```

171 // Resultado
172 c.gridx = 1;
173 c.gridy = camposGuiones.length + 4; // Ajustar según la cantidad de campos
174 c.gridwidth = 3;
175 c.fill = GridBagConstraints.BOTH;
176 resultadoTextArea = new JTextArea();
177 resultadoTextArea.setEditable(false);
178 resultadoTextArea.setLineWrap(true);
179 resultadoTextArea.setRows(20);
180 JScrollPane scrollPane = new JScrollPane(resultadoTextArea);
181 scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
182 panel.add(scrollPane, c);
183
184 // ActionListener para el JComboBox de categorías
185 categoriasComboBox.addActionListener(new ActionListener() {
186     @Override
187     public void actionPerformed(ActionEvent e) {
188         // Obtener la categoría seleccionada
189         String categoriaSeleccionada = (String) categoriasComboBox.getSelectedItem();
190
191         // Verificar si se seleccionó "None"
192         if ("None (Pulsar Buscar)".equals(categoriaSeleccionada)) {
193             // Realizar acciones específicas cuando se selecciona "None"
194             // Puedes dejarlo vacío o mostrar un mensaje, según tus necesidades
195             System.out.println("No se realizará ninguna acción");
196
197         } else {
198             // Realizar acciones adicionales según la categoría seleccionada
199             // Puedes llamar a tu función FiltrarPorFacetas aquí
200             String resultado = busqueda.FiltrarPorFacetas(categoriaSeleccionada);
201             actualizarResultado(resultado);
202         }
203     }
204 });

```

Figura 12: Parte del código de Búsqueda Booleana por Campos de Guiones.

La pantalla de la interfaz para Guiones quedaría abierta de la siguiente manera:

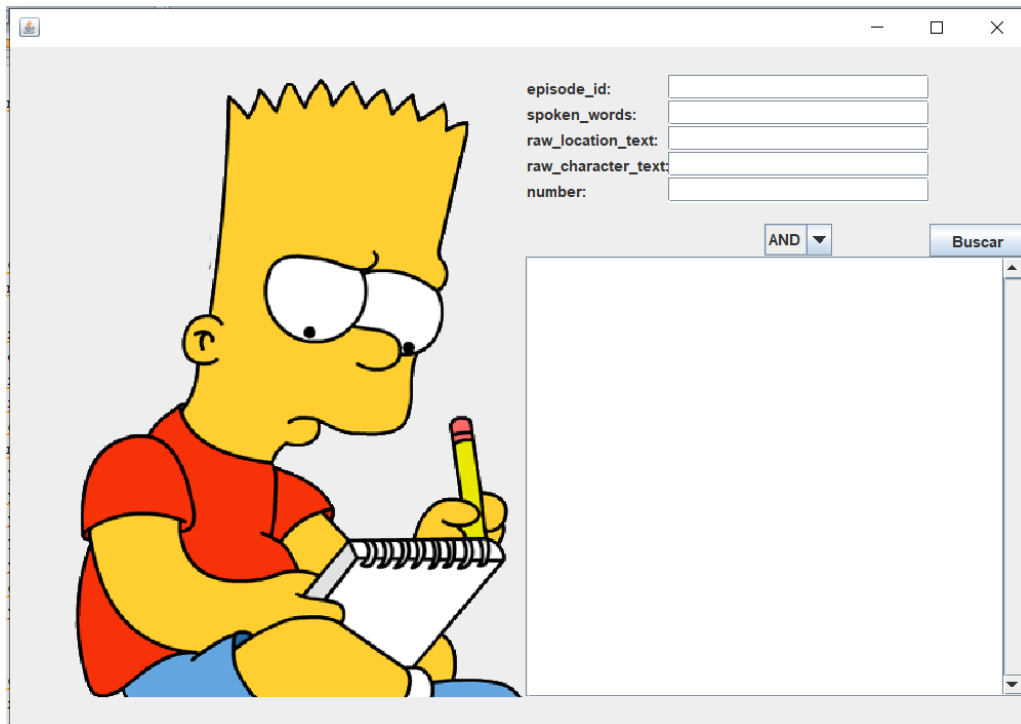


Figura 13: Pantalla de inicio de búsquedas por campos de Guiones.

La pantalla de la interfaz para Capítulos Unidos quedaría abierta de la siguiente manera:

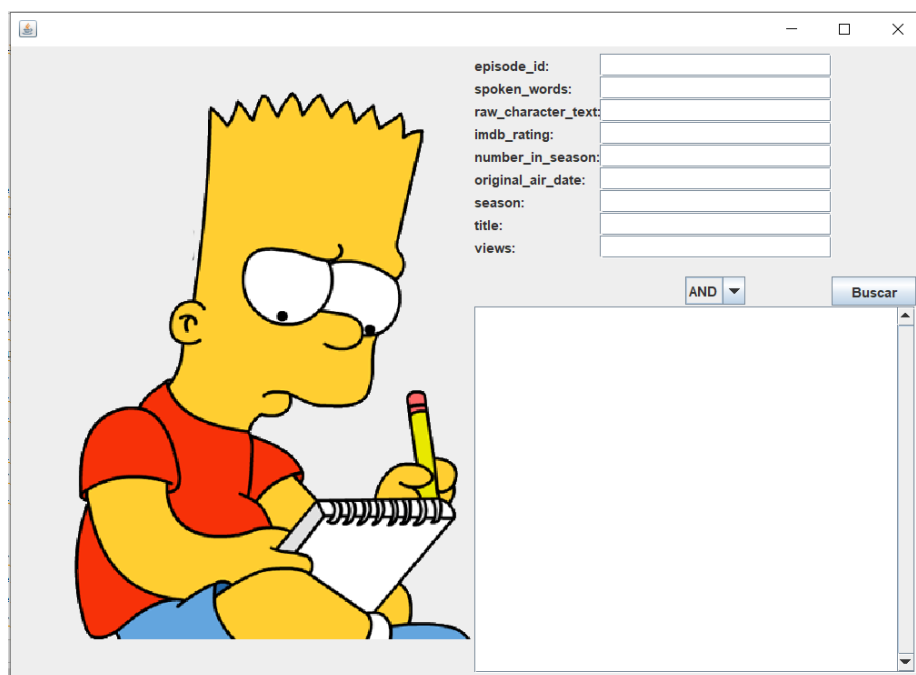


Figura 14: Pantalla de inicio de búsquedas por campos de Capítulos Unidos.

12. Trabajo en Grupo

El trabajo lo hemos repartido en primera instancia de la siguiente manera:

- **Cristóbal Jiménez Álvarez:** búsqueda simple, muestra y filtro de facetas y toda la interfaz.
- **Rafael Luque Framit:** indexación de los índices y las facetas, búsqueda booleana por campos y toda la memoria.

No obstante hemos mantenido el contacto durante todo el desarrollo de la práctica y hemos colaborado conjuntamente en la elaboración del proyecto haciendo un continuo seguimiento del trabajo realizado. Estamos muy orgullosos y contentos con el resultado final de nuestra práctica.

12. Manual de Usuario

En primer lugar, al momento de la apertura de la interfaz nos encontramos con la siguiente ventana:

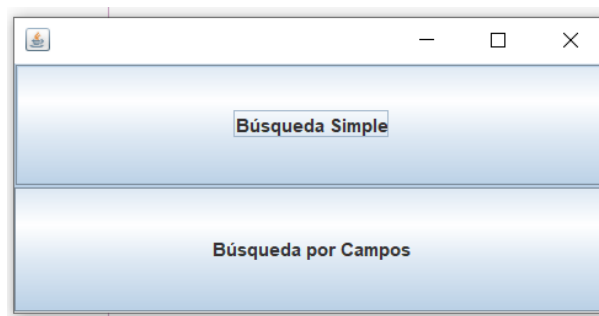
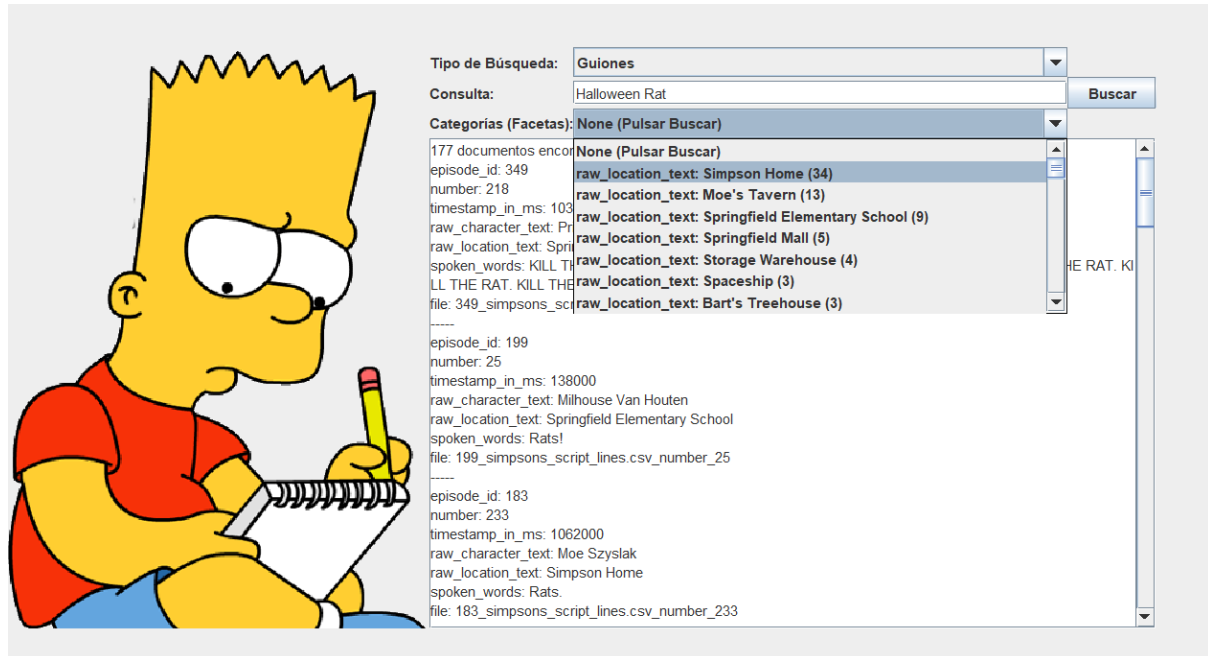


Figura 9: Pantalla de inicio de la interfaz.

Búsqueda Simple

Guiones

Si elegimos hacer una **Búsqueda Simple (sin campos)** , se nos dará la opción de utilizar el índice de **Guiones** o de **Capítulos Unidos** en el “**Tipo de Búsqueda**”.



Tipo de Búsqueda: **Guiones**

Consulta: **Buscar**

Categorías (Facetas): **None (Pulsar Buscar)**

177 documentos encontrados

raw_location_text	Count
Simpson Home	34
Moe's Tavern	13
Springfield Elementary School	9
Springfield Mall	5
Storage Warehouse	4
Spaceship	3
Bart's Treehouse	3

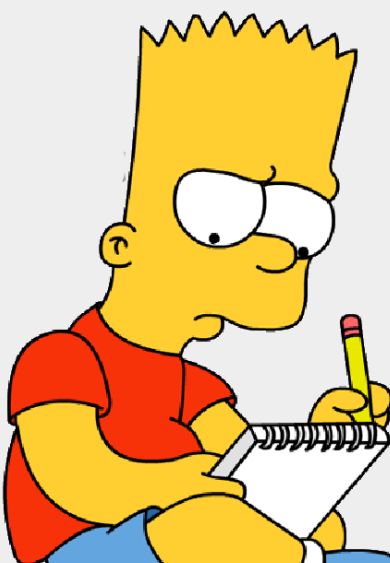
episode_id: 349
number: 218
timestamp_in_ms: 103
raw_character_text: Pr
raw_location_text: Spr
spoken_words: KILL THE
LL THE RAT. KILL THE
file: 349_simpsons_script_lines.csv_number_218

episode_id: 199
number: 25
timestamp_in_ms: 138000
raw_character_text: Milhouse Van Houten
raw_location_text: Springfield Elementary School
spoken_words: Rats!
file: 199_simpsons_script_lines.csv_number_25

episode_id: 183
number: 233
timestamp_in_ms: 1062000
raw_character_text: Moe Szyslak
raw_location_text: Simpson Home
spoken_words: Rats.
file: 183_simpsons_script_lines.csv_number_233

Debes **escribir tu Consulta** y **buscar los resultados**, y después podrás **filtrar** por las facetas correspondientes al índice que estemos usando.

Si usamos el índice de **Guiones**, podremos filtrar por la **categoría “Localización”**, pudiendo así elegir que sólo se muestren resultados donde la ubicación de los personajes sea una en particular.



Tipo de Búsqueda: **Guiones**

Consulta: Halloween Rat

Categorías (Facetas): raw_location_text: Spaceship (3)

3 documentos encontrados

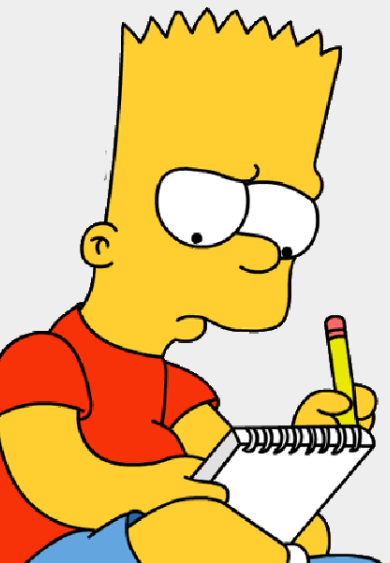
episode_id: 562
number: 85
timestamp_in_ms: 357000
raw_character_text: Homer Simpson
raw_location_text: Spaceship
spoken_words: What the? This isn't Halloween.
file: 562_simpsons_script_lines.csv_number_85

episode_id: 314
number: 16
timestamp_in_ms: 65000
raw_character_text: Kang
raw_location_text: Spaceship
spoken_words: Pathetic humans! They're showing a Halloween episode in November.
file: 314_simpsons_script_lines.csv_number_16

episode_id: 314
number: 17
timestamp_in_ms: 71000
raw_character_text: Kodok
raw_location_text: Spaceship
spoken_words: Who's still thinking about Halloween? We've already got our Christmas decorations up!
file: 314_simpsons_script_lines.csv_number_17

Capítulos Unidos

En cambio, si usamos el índice de **Capítulos Unidos**, podremos filtrar por la **categoría “Season”**, pudiendo así elegir que sólo se muestren resultados donde la temporada sea la indicada.



Tipo de Búsqueda: **Capítulos Unidos**

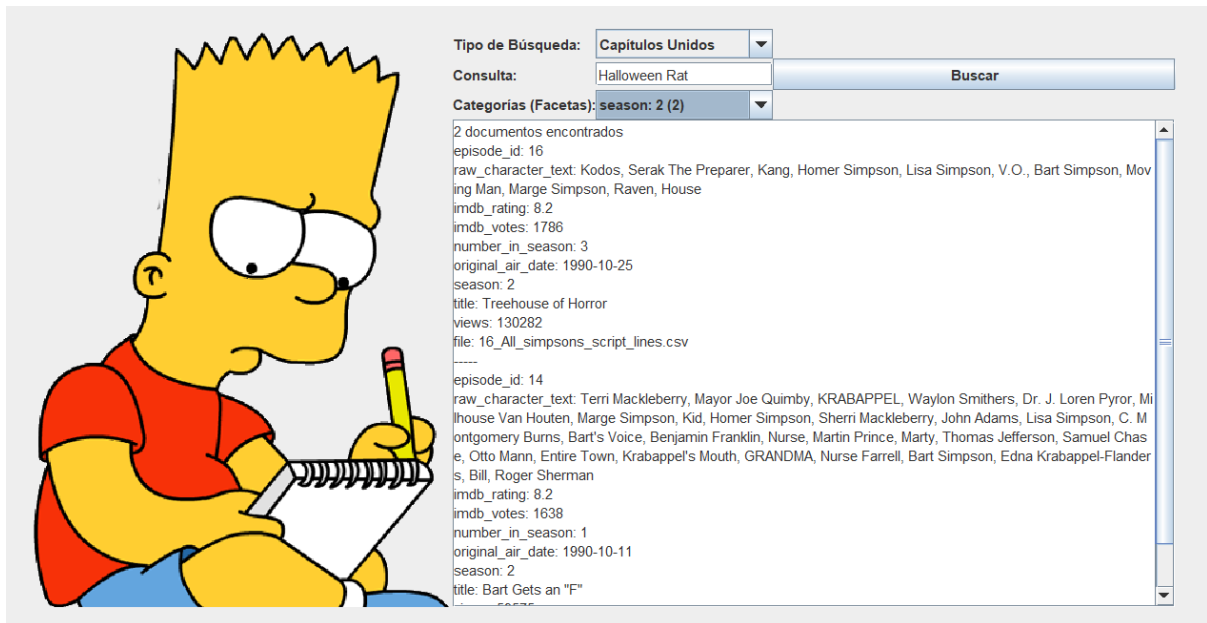
Consulta: Halloween Rat

Categorías (Facetas): **None (Pulsar Buscar)**

113 documentos encontrados

episode_id: 515
raw_character_text: Ralph Wiggum, Bart Simpson, Future Boy, Nelson Muntz, T-Rex, Terence, YO-YO, Colin Meloy, Homer Simpson, Futuristic Marge Simpson, C. Montgomery Burns, Sideshow Mel, Lenny Leonard, Moe Szyslak, Pretentious Ralph Simpson, Edna Krabappel
imdb_rating: 7.2
imdb_votes: 493
number_in_season: 7
original_air_date: 2012-12-09
season: 24
title: The Day the Earth Stood Cool
views: 40649
file: 515_All_simpsons_script_lines.csv

episode_id: 360
raw_character_text: Mayor Joe Quimby, Carl Carlson, Blue-Haired Lawyer, Waylon Smithers, Dracula Hibbert, Homer's Head, Chief Wiggum, Old Jewish Man, Hans Moleman, Milhouse Van Houten, Agnes Skinner, Apu Nahasapeemapetilon, Marge Simpson, Nelson Muntz, Grampa Gorilla, Homer Simpson, David, Angels, Dr. Julius Hibbert, Lisa Simpson, Seymour Skinner, Technician, C. Montgomery Burns, Werewolf Bart, Sideshow Mel, Groundskeeper Willie, Apu Robot, Bart's Thoughts, Moe Pacifier, Lenny Leonard, Krusty the Clown, Moe Szyslak, TV Announcer, Ned Flanders, Selma Bouvier, Martin Prince, Dennis Rodman Pacifier, Old Robot, Robot, Silver-haired Priest, Lenny Pacifier, Barney Gumble, Witch, Terry Bradshaw, Einstein Lisa, Bart Simpson, Disco Stu, Bunny, Comic Book Guy, Female Robot, Patty Bouvier

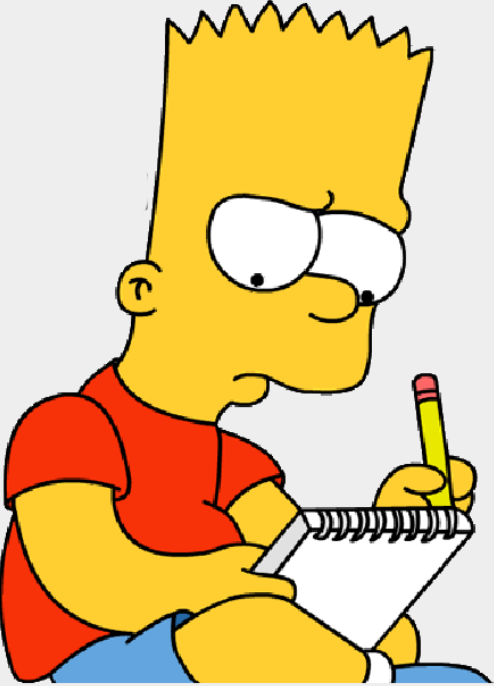


En todas las pantallas de esta interfaz, **podrás cambiar el filtro** de categoría **constantemente**, y **se refrescará el resultado** de la búsqueda obtenido. De querer volver a tener el resultado global, puedes pulsar el Botón “Buscar”.

Búsqueda Booleana

Guiones

Para buscar en Búsquedas Booleanas por Guiones deberás rellenar los campos por los que deseas buscar así como elegir si quieres una consulta de tipo AND o OR. Una vez que busques la consulta te aparecerán las posibles facetas junto con su valor.



episode_id:

spoken_words:

raw_location_text:

raw_character_text:

number:

Categorías (Facetas): **None (Pulsar Buscar)** ▼

5 documentos encontrados

episode_id: 16
number: 207
timestamp_in_ms: 973
raw_character_text: Bart Simpson
raw_location_text: Bart's Treehouse
spoken_words: Hey, Poindexter. It's Halloween, put the book away.
file: 16_simpsons_script_lines.csv_number_207


episode_id: 469
number: 46
timestamp_in_ms: 278000
raw_character_text: Bart Simpson
raw_location_text: Simpson Car
spoken_words: As long as I can dig you up and stick you on the front porch every Halloween.
file: 469_simpsons_script_lines.csv_number_46

episode_id: 298
number: 214

raw_location_text: Simpson Home (1)
raw_location_text: Simpson Car (1)
raw_location_text: Bart's Treehouse (1)
raw_location_text: OUTLOOK (1)
raw_location_text: EFCOT Center (1)

Buscar

Si eliges alguna faceta te mostrará los resultados solo de esa faceta.



episode_id:

spoken_words:

raw_location_text:

raw_character_text:

number:

Categorías (Facetas): **raw_location_text: Simpson Car (1)** ▼

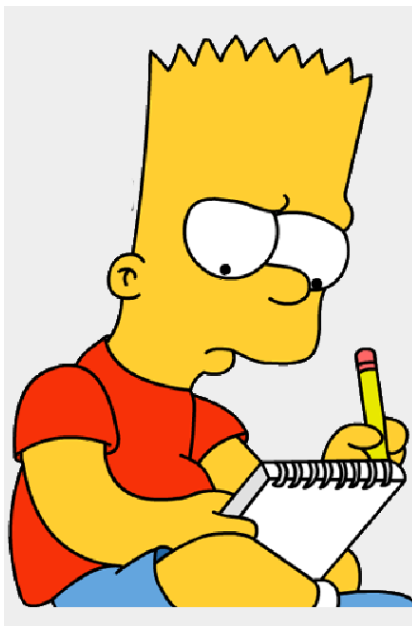
AND ▼

Buscar

1 documentos encontrados

episode_id: 469
number: 46
timestamp_in_ms: 278000
raw_character_text: Bart Simpson
raw_location_text: Simpson Car
spoken_words: As long as I can dig you up and stick you on the front porch every Halloween.
file: 469_simpsons_script_lines.csv_number_46

Si realizas una consulta de tipo OR es normal que te salgan más resultados ya que es menos estricta.



episode_id:

spoken_words:

raw_location_text:

raw_character_text:

number:

Categorías (Facetas):

Halloween

Milhouse

None (Pulsar Buscar)

1753 documentos encontrados

episode_id: 230

number: 104

timestamp_in_ms: 471

raw_character_text: Milhouse

raw_location_text: Springfield

spoken_words: He would on Halloween.

file: 230_simpsons_script_lines.csv_number_104

episode_id: 359

number: 248

timestamp_in_ms: 1147000

raw_character_text: Milhouse Van Houten

raw_location_text: Cliff

spoken_words: Yeah. What are you going as for Halloween?

file: 359_simpsons_script_lines.csv_number_248

episode_id: 532

number: 7

timestamp_in_ms: 201000

raw_location_text: Simpson Home (279)

raw_location_text: Springfield Elementary School (261)

raw_location_text: Bart's Treehouse (97)

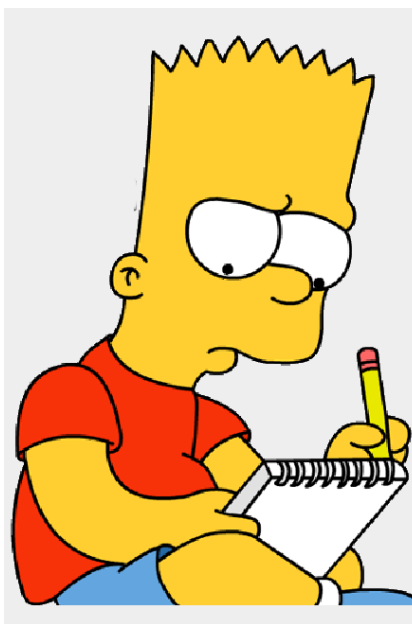
raw_location_text: Van Houten Home (75)

raw_location_text: The Android's Dungeon & Baseball Card Shop (61)

raw_location_text: Springfield (30)

raw_location_text: Springfield Street (29)

Buscar



episode_id:

spoken_words:

raw_location_text:

raw_character_text:

number:

Categorías (Facetas):

Halloween

Milhouse

raw_location_text: Auditorium (5)

OR

Buscar

5 documentos encontrados

episode_id: 152

number: 2

timestamp_in_ms: 41000

raw_character_text: Milhouse Van Houten

raw_location_text: Auditorium

spoken_words: Otto, wake up!

file: 152_simpsons_script_lines.csv_number_2

episode_id: 413

number: 101

timestamp_in_ms: 458000

raw_character_text: Milhouse Van Houten

raw_location_text: Auditorium

spoken_words: But some of us have After-School Care!

file: 413_simpsons_script_lines.csv_number_101

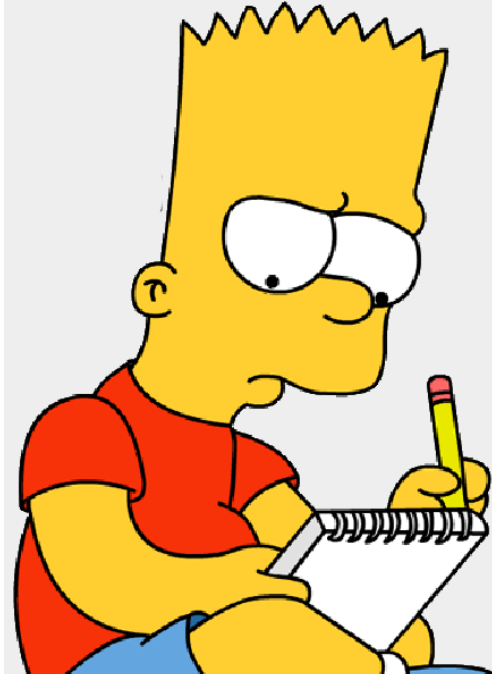
episode_id: 74

number: 253

timestamp_in_ms: 1116000

CapitulosUnidos

Para buscar en Búsquedas Booleanas por CapitulosUnidos deberás rellenar los campos por los que deseas buscar así como elegir si quieres una consulta de tipo AND o OR. Una vez que busques la consulta te aparecerán las posibles facetas junto con su valor.



episode_id:

spoken_words:

raw_character_text:

imdb_rating:

number_in_season:

original_air_date (YYYY-MM-DD):

season:

title:

views:

Halloween

Milhouse

Categorías (Facetas):

None (Pulsar Buscar)

None (Pulsar Buscar)

season: 18 (3)

season: 17 (2)

season: 23 (2)

season: 6 (2)

season: 15 (1)

season: 14 (1)

season: 11 (1)

Buscar

19 documentos encontrados

episode_id: 405

raw_character_text: Chief Wiggum, Agnes Skinner, Apu Nahasapeemom, Homer Simpson, MEN IN CROWD, G. Simpson, Seymour Skinner, C. Montgomery Burns, Lenny Leonard, Krusty the Clown, Rev. Timothy Lovejoy, Kang, General, Moe Szyslak, Ned Flanders, Kearney Zyzwicz, Agent #2, Homer's Thoughts, Bart Simpson

imdb_rating: 7.1

imdb_votes: 687

number_in_season: 5

original_air_date: 2007-11-04

season: 19

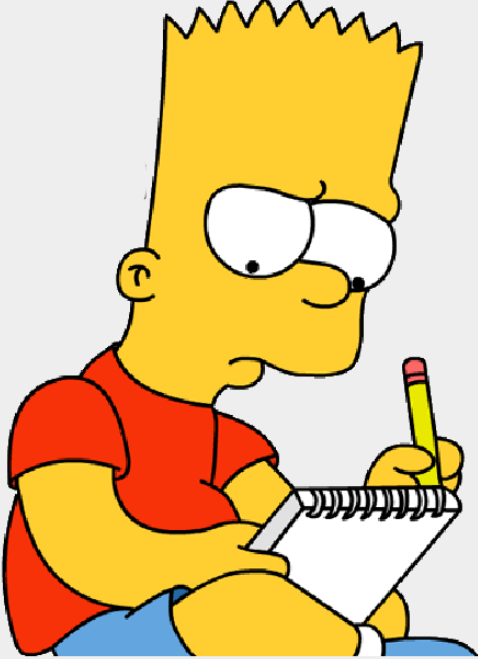
title: Treehouse of Horror XVIII

views: 69957

file: 405_All_simpsons_script_lines.csv

episode_id: 489

raw_character_text: Taxi Dragon, Chief Wiggum, Rabbi Hyman Krustofsky, Avatar Jimbo, KAMALA, Avatar Dolph, Marge Simpson, Kodos, Homer Simpson, Grampa Simpson



episode_id:

spoken_words:

raw_character_text:

imdb_rating:

number_in_season:

original_air_date (YYYY-MM-DD):

season:

title:

views:

Categorías (Facetas):

AND

1 documentos encontrados

episode_id: 314

raw_character_text: Mayor Joe Quimby, Miss Hoover, Swedish Scientist, Wino, Milhouse Van Houten, Agnes Skinner, Marge Simpson, Nelson Muntz, Kodos, Homer Simpson, Grampa Simpson, Lisa Simpson, Seymour Skinner, Johnny Tightlips, Sideshow Mel, Jennifer Garner, Delivery Boy, Kang, Moe Szyslak, Ned Flanders, Mailman, Kids, Jasper Beardsly, Announcer, Oscar De La Hoya, Death, Dudley Herschbach, Professor Jonathan Frink, Frink Senior, Young Professor Frink, Frankie the Squealer, Bart Simpson, Comic Book Guy, God

imdb_rating: 7.4

imdb_votes: 738

number_in_season: 1

original_air_date: 2003-11-02


season: 15

title: Treehouse of Horror XIV

views: 68381

file: 314_All_simpsons_script_lines.csv

Si realizas una consulta de tipo OR es normal que te salgan más resultados ya que es menos estricta.



episode_id:

spoken_words:

raw_character_text:

imdb_rating:

number_in_season:

original_air_date (YYYY-MM-DD):

season:

title:

views:

Categorías (Facetas):

336 documentos encontrados

episode_id: 405

raw_character_text: Chief Wiggum, Agent K, Krusty the Clown, Lenny Leonard, Ned Flanders, Kearney Zzyzwick, Agent #2, Homer's Thoughts, Bart Simpson

imdb_rating: 7.1

imdb_votes: 687

number_in_season: 5

original_air_date: 2007-11-04

season: 19

title: Treehouse of Horror XVIII

views: 69957

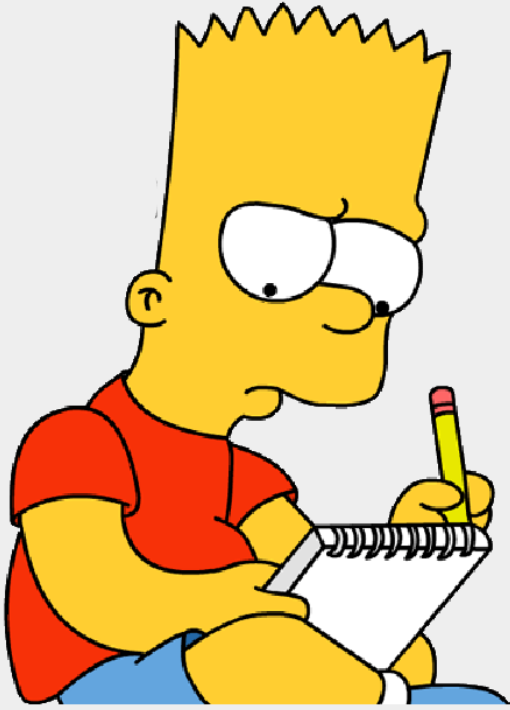
file: 405_All_simpsons_script_lines.csv

episode_id: 489

raw_character_text: Taxi Dragon, Chief Wiggum, Rabbi Hyman Krustofsky, Avatar Jimbo, KAMALA, Avatar Dolph, Marge Simpson, Kodos, Homer Simpson, Grampa Simpson

Configuraciones adicionales

- En cada campo puedes introducir **varios valores**, el programa te los reconocerá como individuales y podrás hacer la búsqueda sin problema.
- Los imdb rating mostrarán resultados con ± 0.2 puntos de margen, para encontrar resultados semejantes al que buscas, de lo contrario saldrían muy pocos probablemente.
- En views también hay un rango de ± 5000 views, para ser más permisivos.
- Para introducir las fechas debes seguir el esquema que se propone en la interfaz (YYYY-MM-DDDD)



episode_id: 5 12 1 7

spoken_words:

raw_character_text:

imdb_rating: 9 8

number_in_season:

original_air_date (YYYY-MM-DD):

season:

title:

views: 40000 60000

Categorías (Facetas): None (Pulsar Buscar) ▼

AND ▼

Buscar

2 documentos encontrados

episode_id: 7

raw_character_text: Homer's Canyon Echo, Salesman, Man #2, Cowboy Bob, Camper Mother, Beautiful French Scientist, Man #1, Marge Simpson, Newsman, Homer Simpson, Lisa Simpson, Dr. Marvin Monroe, German Scientist, Camper Father, English Scientist, Ned Flanders, Reporter #1, Rod Flanders, Reporter #2, Map Boy, Game Warden, Bart Simpson, NED, Tv Announcer #2, Flunky, Photographer

imdb_rating: 7.9

imdb_votes: 1638

number_in_season: 7

original_air_date: 1990-02-18

season: 1

title: The Call of the Simpsons

views: 57793

file: 7_All_simpsons_script_lines.csv

episode_id: 5

raw_character_text: Receptionist, Herman Hermann, Milhouse Van Houten, Marge Simpson, Small Boy, Nelson Muntz, Kid, Homer Simpson, Grampa Simpson, Lisa Simpson, Seymour Skinner, All, Homer's Image, Weasel #1, Otto Mann, ALL AT THE TABL