



UNIVERSIDAD DE GRANADA

PRÁCTICA 1: DESARROLLO DE UN SISTEMA DE RECUPERACIÓN DE INFORMACIÓN CON LUCENE

Autor

Juan Manuel Castillo Nievas



MÁSTER PROFESIONAL EN INGENIERÍA INFORMÁTICA 2020/2021

Granada, 25 de marzo de 2021

Índice

1. Descripción inicial	2
2. Colección de documentos	2
2.1. Parseador de documentos	4
3. Indexador	6
3.1. Constructor Indexer	7
3.2. Método parseStopWords	7
3.3. Método indexFiles	8
3.4. Método main	9
4. Buscador	9
4.1. Constructor Searcher	10
4.2. Método search	10
4.3. Método getDocument	11
4.4. Método main	11
5. Interfaz gráfica	12
5.1. Interfaz línea de comandos	12
5.2. Interfaz web	12
5.3. Procesando las búsquedas del usuario	13
6. Manual de usuario	15
6.1. Línea de comandos	15
6.2. Aplicación web	20
6.2.1. Estructura de directorios	20
6.2.2. Ejecución	20
7. Referencias	24

1. Descripción inicial

Durante esta práctica se ha desarrollado un **Sistema de Recuperación de Información** con la biblioteca **Lucene 8.8.1** [1] disponible para Java. Este sistema se compone de dos programas:

- **Indexador:** que recibe como parámetros la ruta de la colección documental a indexar, el fichero de palabras vacías a emplear y la ruta donde alojar los índices, y se encarga de crear los índices oportunos y ficheros auxiliares necesarios para la recuperación.
- **Motor de búsqueda:** que recibe como parámetro la ruta donde están alojados todos los índices y ofrece al usuario una interfaz gráfica en la que puede realizar una consulta de texto y obtener los documentos relevantes a dicha consulta.

Como **IDE** de desarrollo para esta aplicación se ha usado **Apache Netbeans 12.3** [2].

Para la contrucción del indexador y del buscador se ha seguido una guía inicial para entender los conceptos más básicos [3].

2. Colección de documentos

Entre todas las colecciones de documentos posibles que se han facilitado, en este sistema se ha usado la colección de documentos **Medline** [4], cuyo contenido está relacionado con el ámbito de la medicina.

Esta colección viene en un archivo llamado **MED.ALL** y su contenido se muestra en la Figura 1. Tal y como se ve, no son documentos independientes, si no que es un mismo archivo de texto que contiene a todos los documentos. Cada documento viene separado por una línea que siempre sigue el formato “*IX*”, siendo *X* el número del documento. El nombre de cada documento sigue el formato *documentX.txt*.

Esta colección de documentos se ha subido a la nube y se encuentra disponible en el siguiente enlace:

https:
[//drive.google.com/drive/folders/1x8TSBtTloATP2bZpwP90ZnEIK1XQ2ux3?usp=sharing](https://drive.google.com/drive/folders/1x8TSBtTloATP2bZpwP90ZnEIK1XQ2ux3?usp=sharing)

```
MED.ALL x
1 .I 1
2 .W
3 correlation between maternal and fetal plasma levels of glucose and free
4 fatty acids .
5 correlation coefficients have been determined between the levels of
6 glucose and ffa in maternal and fetal plasma collected at delivery .
7 significant correlations were obtained between the maternal and fetal
8 glucose levels and the maternal and fetal ffa levels . from the size of
9 the correlation coefficients and the slopes of regression lines it
10 appears that the fetal plasma glucose level at delivery is very strongly
11 dependent upon the maternal level whereas the fetal ffa level at
12 delivery is only slightly dependent upon the maternal level .
13 .I 2
14 .W
15 changes of the nucleic acid and phospholipid levels of the livers in the
16 course of fetal and postnatal development .
17 we have followed the evolution of dna, rna and pl in the livers of rat
18 foeti removed between the fifteenth and the twenty-first day of
19 gestation and of young rats newly-born or at weaning . we can observe
20 the following facts..
21 1. dna concentration is 1100 ug p on the 15th day, it decreases from
22 the 19th day until it reaches a value of 280 ug 5 days after weaning .
23 2. rna concentration is 1400 ug p on the 15th day and decreases to 820
24 during the same period .
25 3. pl concentration is low on the 15th day and during foetal life (700
26 ug) and increases abruptly at birth .
27 4. the ratios rna cyto/dna and pl cyto/dna increase regularly from the
28 18th day of foetal life .
29 5. nuclear rna and pl contents are very high throughout the
30 development .
```

Figura 1: Estructura del archivo **MED.ALL**

2.1. Parseador de documentos

Para obtener todos los documentos por separado, es decir, un archivo de texto por cada documento, se ha parseado el archivo **MED.ALL**. Para ello, se ha creado la clase **Parser.java** cuyos atributos y métodos son los siguientes:

```
1 public class Parser {
2     private File currentFile;
3     private ArrayList<String> fileText;
4
5     public Parser(String filename) {
6         ...
7     }
8
9     private void processLine(String line){
10        ...
11    }
12 }
```

Atributos:

- **currentFile**: es el documento actual en el que se está escribiendo. Por ejemplo, si se ha detectado una línea con el formato *.I40*, se crea un archivo llamado *document40.txt* y este atributo pasa a ser dicho archivo sobre el cual se escribirán las siguientes líneas.
- **fileText**: es una cadena de caracteres que será cada línea de texto del documento **MED.ALL**. Dependiendo del formato de la línea, se creará un archivo o se escribirá esa línea sobre el archivo.

Métodos:

- **Parser(String filename)**: el constructor de la clase que recibe como argumentos el nombre del archivo a parsear, que en este caso será **MED.ALL**. El constructor se encarga de leer cada línea del archivo y llamar al método *processLine* para procesar dichas líneas.
- **processLine(String line)**: se encarga de procesar cada línea del archivo. Dependiendo del formato de la línea, se hará una cosa u otra:
 - Si la línea empieza por **.I**, se coge el número que va acompañado de la línea, llamado *X*, y se crea un archivo llamado *documentX.txt* y se actualiza el atributo **currentFile**.
 - Si la línea empieza por **.W**, se ignora la línea ya que son las líneas que van a continuación de las **.I** y que no sirven de nada.
 - De lo contrario, dicha línea se escribe en el fichero **currentLine**.

Como resultado se obtiene un directorio que contiene todos los documentos (ver Figura 2) en el que cada documento tiene su contenido (ver Figura 3).

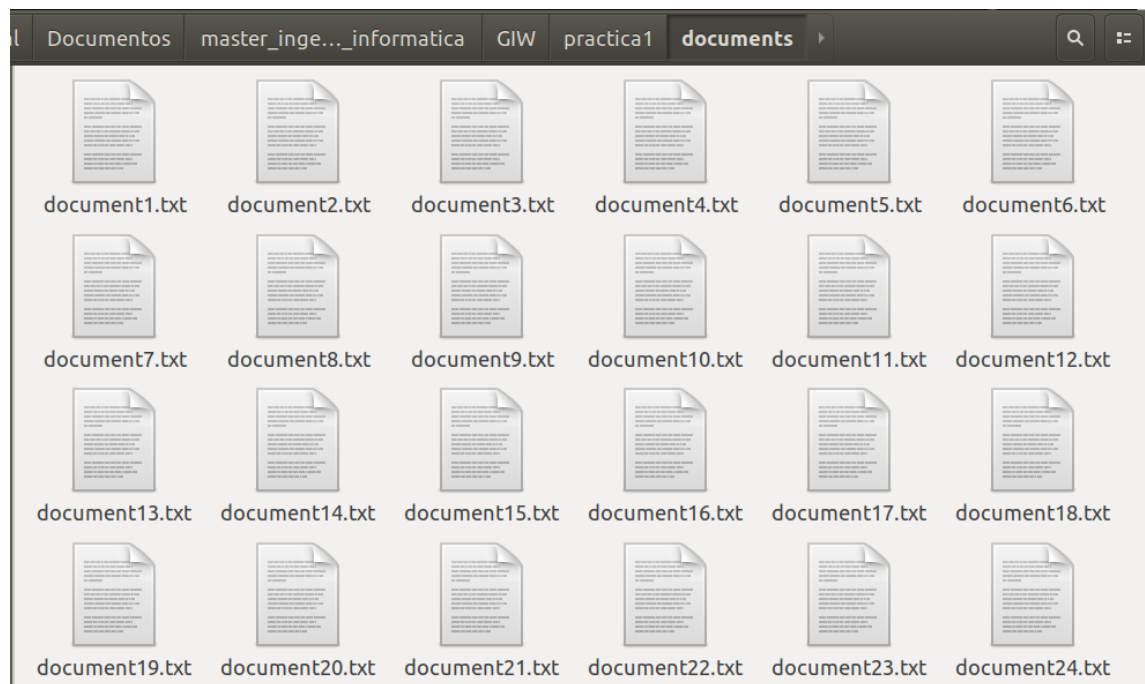


Figura 2: Directorio de documentos creado

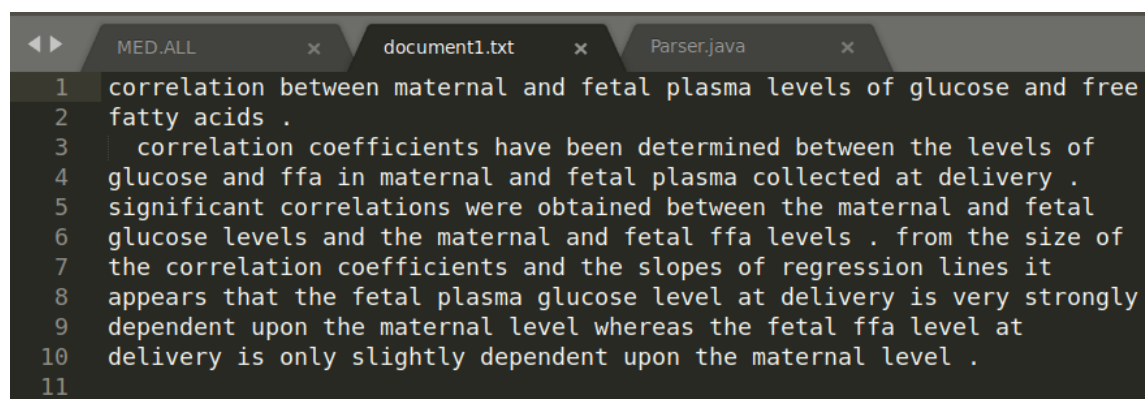


Figura 3: Archivo creado **document1.txt**

La clase **Parser** es ejecutable y su método main es el siguiente:

```
1 public class Main {
2     public static void main(String[] args){
3         Parser parser = new Parser("src/main/webapp/resources/MED.ALL");
4     }
5 }
```

Simplemente se llama al parseador con el *path* del archivo **MED.ALL**, que es el que se va a parsear.

3. Indexador

El indexador se encuentra en la clase **Indexer.java** y contiene la siguiente estructura:

```
1 public class Indexer {
2     private IndexWriter iwriter;
3     private String documentsDirectoryPath;
4     private String fileStopWords;
5     private String indexDirectoryPath;
6     private CharArraySet stopWords;
7
8     public Indexer(String documentsDirectoryPath, String fileStopWords, String
9         indexDirectoryPath){
10         ...
11     }
12
13     private CharArraySet parseStopWords(){
14         ...
15     }
16
17     public void indexFiles(){
18         ...
19     }
20
21     public static void main(String[] args) {
22         ...
23     }
24 }
```

Atributos:

- **iwriter**: el que escribe los índices
- **documentsDirectoryPath**: el directorio donde se encuentran los documentos que se van a indexar

- **fileStopWords**: el archivo que contiene las palabras vacías
- **indexDirectoryPath**: el directorio donde se almacenarán los índices
- **stopWords**: a Lucene no se le puede pasar un archivo como *stop words*, si no que hay que parsear el archivo y transformarlo en un objeto de tipo **CharArraySet**

3.1. Constructor Indexer

El constructor recibe como parámetros el directorio que contiene los documentos a indexar, el archivo de palabras vacías y el directorio donde se almacenarán los índices.

```

1 public Indexer(String documentsDirectoryPath, String fileStopWords, String
   indexDirectoryPath) throws IOException {
2
3     ...
4
5     Directory directory = FSDirectory.open(Paths.get(indexDirectoryPath));
6
7     this.parseStopWords();
8
9     EnglishAnalyzer analyzer = new EnglishAnalyzer(stopWords);
10    IndexWriterConfig config = new IndexWriterConfig(analyzer);
11    iwriter = new IndexWriter(directory, config);
12
13    if(DirectoryReader.indexExists(directory)){
14        iwriter.deleteAll();
15    }
16 }
```

- **Línea 5**: se abre el directorio que aloja los índices (o se crea si no existe)
- **Línea 7**: se llama a la función que convierte el fichero de las palabras vacías en un **CharArraySet**, de la que se habla más adelante
- **Líneas 9-10-11**: se construye un analizador con las palabras vacías dadas y se construye el índice que escribirá los ficheros auxiliares necesarios
- **Línea 13**: si existen índices en el directorio se eliminan

3.2. Método parseStopWords

Este método es muy simple: abre el archivo que contiene las palabras vacías, crea un array que contiene todas las líneas y las convierte al tipo **CharArraySet** usando su constructor.

```

1 private CharArraySet parseStopWords() throws FileNotFoundException {
2     Scanner sc = new Scanner(new File(fileStopWords));
3     List<String> lines = new ArrayList<String>();
4     while (sc.hasNextLine()) {
5         lines.add(sc.nextLine());
6     }
7
8     this.stopWords = new CharArraySet(lines, true);
9     return stopWords;
10 }

```

3.3. Método indexFiles

Este método se encarga de indexar todos los documentos. Coge todos los archivos que se encuentran en el directorio de los documentos (**documentsDirectoryPath**) y los recorre todos, haciendo lo siguiente en cada iteración:

- **Línea 5:** crea un documento usando el constructor por defecto
- **Líneas 7-8-9:** se crean tres campos que van a estar dentro del documento:
 - **content:** es el contenido del documento, todo el texto
 - **name:** el nombre del archivo
 - **filepath:** el *path* del archivo donde está almacenado
- **Líneas 11-12-13:** añade los tres campos creados al documento
- **Línea 14:** se crea el índice del documento

```

1 public void indexFiles() throws IOException {
2     File[] files = new File(documentsDirectoryPath).listFiles();
3     for(File file: files){
4
5         Document document = new Document();
6
7         TextField bodyField = new TextField("content", new FileReader(file));
8         TextField fileNameField = new TextField("name", file.getName(), Field.Store.YES);
9         TextField filePathField = new TextField("filepath", file.getCanonicalPath(),
10             TextField.Store.YES);
11
12         document.add(bodyField);
13         document.add(fileNameField);
14         document.add(filePathField);
15         iwriter.addDocument(document);
16     }
17     iwriter.close();
18 }

```

3.4. Método main

Este método toma 3 argumentos inicialmente: el directorio de documentos, el fichero de palabras vacías, y el directorio donde se alojarán los índices. A continuación, crea el indexador y llama al método **indexFiles** para crear los índices.

```
1 public static void main(String[] args) throws IOException {
2     if(args.length != 3){
3         System.out.println("You need to pass 3 arguments, the documents directory path,
4             stop words file, and index directory path\n"
5             + "Example: /home/user/documents /home/user/stop_words.txt
6             /home/user/index");
7         System.exit(-1);
8     }
9
10    Indexer indexer = new Indexer(args[0], args[1], args[2]);
11
12    indexer.indexFiles();
13 }
```

4. Buscador

En la Sección 3 se acaba de ver cómo se indexan los documentos, pero un indexador no tiene sentido sin un buscador que permita buscar en esos documentos indexados.

La clase **Searcher.java** se encarga de manejar las búsquedas que realiza el usuario y tiene la siguiente estructura:

```
1 public class Searcher {
2     private QueryParser queryParser;
3     private IndexSearcher indexSearcher;
4
5     public Searcher(String indexDirectoryPath) {
6         ...
7     }
8
9     public TopDocs search(String searchQuery){
10        ...
11    }
12
13    public Document getDocument(ScoreDoc scoreDoc){
14        ...
15    }
16
17    public static void main(String[] args) {
18        ...
19    }
20 }
```

```
19     }  
20 }
```

Atributos:

- **queryParser:** se encarga de traducir la búsqueda del usuario en una búsqueda más concreta
- **indexSearcher:** se encarga de realizar la búsqueda en los índices creados en el indexador

4.1. Constructor Searcher

El constructor recibe como parámetros el directorio donde se encuentran almacenados los índices y la lista de palabras vacías.

```
1 public Searcher(String indexDirectoryPath, CharArraySet stopWords) throws IOException {  
2     Directory indexDirectory = FSDirectory.open(Paths.get(indexDirectoryPath));  
3     IndexReader reader = DirectoryReader.open(indexDirectory);  
4     this.indexSearcher = new IndexSearcher(reader);  
5     this.queryParser = new QueryParser("content", new EnglishAnalyzer(stopWords));  
6 }
```

- **Línea 2:** abre el directorio que contiene los índices
- **Línea 3:** accede a los índices
- **Líneas 4-5:** se asignan los objetos creados en las dos anteriores líneas a los atributos de la clase

4.2. Método search

Este método recibe como entrada la búsqueda que realiza el usuario y se encarga de transformar dicha consulta en una consulta más concreta de forma que el buscador lo entienda, y devuelve los mejores resultados encontrados. En este caso se ha puesto que devuelva los 2.000 mejores resultados, pero en realidad podría ser cualquier número superior a 1.033 que es el máximo número de documentos que se tienen.

```
1 public TopDocs search(String searchQuery) throws IOException, ParseException {  
2     Query query = queryParser.parse(searchQuery);  
3     return indexSearcher.search(query, 2000);  
4 }
```

4.3. Método getDocument

Otro método también muy simple que se encarga de devolver el documento que está en una determinada posición.

```
1 public Document getDocument(ScoreDoc scoreDoc) throws CorruptIndexException, IOException {
2     return indexSearcher.doc(scoreDoc.doc);
3 }
```

4.4. Método main

Este método toma el primer argumento (que debe ser el directorio donde se encuentran alojados los índices). Después pide al usuario que introduzca su búsqueda. A continuación, se muestran todos los resultados encontrados al usuario. Por último, se le pide al usuario que introduzca un número para mostrar un documento.

Esto se verá con un ejemplo en la Sección 6.

```
1 public static void main(String[] args) throws IOException, ParseException {
2     if(args.length != 1){
3         System.out.println("You need to pass 1 argument, the index directory path\n"
4             + "Example: /home/user/index");
5         System.exit(-1);
6     }
7
8     Searcher searcher = new Searcher(args[0]);
9
10    Scanner myObj = new Scanner(System.in); // Create a Scanner object
11    System.out.println("What are you looking for?");
12
13    String searchQuery = myObj.nextLine(); // Read user input
14
15    TopDocs hits = searcher.search(searchQuery);
16
17    System.out.println("\n"+hits.totalHits + " documents found:");
18
19    Integer number = 0;
20
21    for(ScoreDoc scoreDoc : hits.scoreDocs) {
22        Document doc = searcher.getDocument(scoreDoc);
23
24        System.out.println(Integer.toString(number)+" : "+doc.get("name"));
25
26        number += 1;
27    }
28
29    while(true){
```

```

30     myObj = new Scanner(System.in); // Create a Scanner object
31     System.out.println("\nWhich document do you want to open? Introduce the number on
        the left (E.g: 50)");
32
33     String documentNumber = myObj.nextLine(); // Read user input
34
35     number = 0;
36
37     for(ScoreDoc scoreDoc : hits.scoreDocs) {
38         number += 1;
39
40         if(number == Integer.parseInt(documentNumber)){
41             Document doc = searcher.getDocument(scoreDoc);
42             String line;
43             BufferedReader brTest = new BufferedReader(new
                FileReader(doc.get("filepath")));
44             while ((line = brTest.readLine()) != null) {
45                 System.out.println(line);
46             }
47         }
48     }
49 }
50 }

```

5. Interfaz gráfica

Se han creado dos interfaces gráficas: una interfaz que ve el usuario cuando ejecuta la clase **Searcher.java** desde la **línea de comandos**, y otra interfaz web que requiere ejecutar el proyecto entero en un IDE como es **Netbeans**.

5.1. Interfaz línea de comandos

Esta interfaz es muy simple y se lleva acabo a través de la ejecución del archivo **Searcher.jar**. Mediante la línea de comandos se puede usar la instrucción `java -jar Searcher.jar`. Su manual de uso se detalla en la siguiente Sección.

5.2. Interfaz web

Para la interfaz gráfica del buscador se ha creado una interfaz gráfica en la web, creando para ello una **aplicación web en Java con Maven**.

Se ha creado el archivo **index.html** usando como plantilla una ya creada con **Bootstrap** [5]. Dicha plantilla se encuentra en **Colorlib** [6] y se puede ver en la Figura 4.

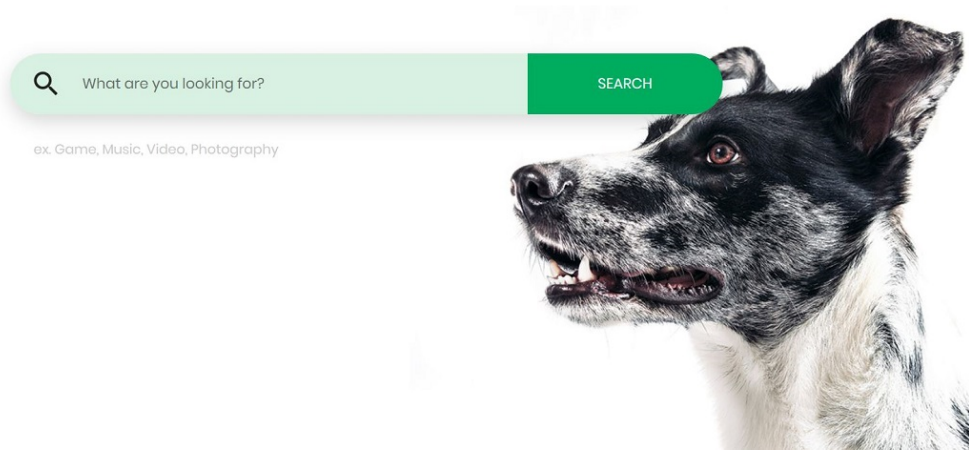


Figura 4: Página de inicio del buscador

5.3. Procesando las búsquedas del usuario

Para procesar las peticiones del usuario se ha creado la clase **SearchServlet.java**, que como bien indica su nombre es un *servlet* para procesar las peticiones HTTP en Java.

El método **processRequest** se encarga de procesar las peticiones **GET** y **POST**. En este caso se procesarán las peticiones **POST** ya que se procesan las búsquedas del usuario a través del formulario. El código asociado a este método se encuentra a continuación y su explicación viene debajo de este.

```
1  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException, ParseException {
2
3      String query = request.getParameter("searchQuery");
4
5      Indexer indexer = new
        Indexer(this.getServletContext().getRealPath(".") + DOCUMENTS_DIRECTORY_PATH,
6              this.getServletContext().getRealPath(".") + FILE_STOP_WORDS,
7              this.getServletContext().getRealPath(".") + INDEX_DIRECTORY_PATH);
8
9      indexer.indexFiles();
10
11     Searcher searcher = new
        Searcher(this.getServletContext().getRealPath(".") + INDEX_DIRECTORY_PATH,
            indexer.getStopWords());
```

```

12
13 TopDocs hits = searcher.search(query);
14
15 PrintWriter out = response.getWriter();
16 out.println("<!DOCTYPE html>");
17
18 ...
19
20 try {
21     for(ScoreDoc scoreDoc : hits.scoreDocs) {
22         Document doc = searcher.getDocument(scoreDoc);
23
24         BufferedReader brTest = new BufferedReader(new
25             FileReader(this.getServletContext().getRealPath(".") + DOCUMENTS_DIRECTORY_PATH + doc.get("name")));
26         String text = brTest.readLine();
27
28         out.println("<a class=\"link\"
29             href=\"http://localhost:8080/Search/resources/documents/\" + doc.get("name") + \"\"
30             target=\"_blank\" rel=\"noopener noreferrer\"><span style=\"font-size:
31             30px\">"+Integer.toString(number)+".</span> "+text+"</a>");
32         number += 1;
33     }
34 } finally {
35     out.println("</div>");
36     out.println("</body>");
37     out.println("</html>");
38     out.close();
39 }

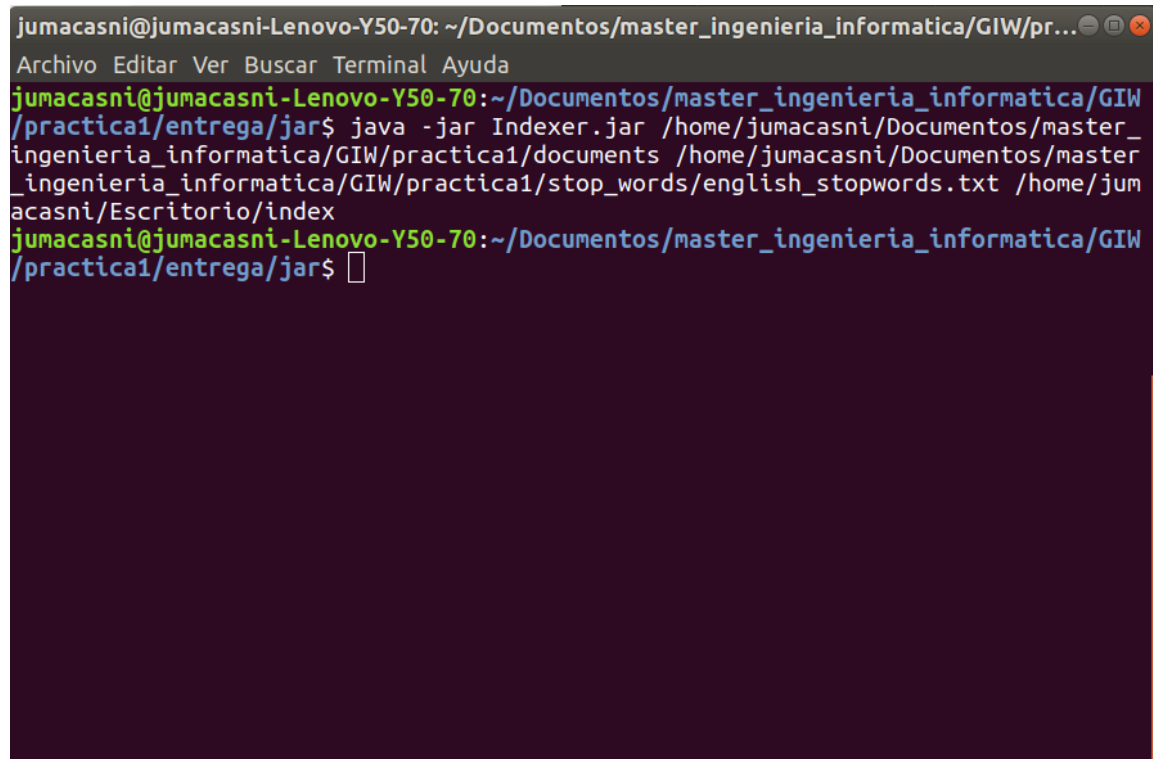
```

- **Línea 3:** se coge la búsqueda que ha introducido el usuario en el formulario
- **Línea 5-6-7:** se crea el indexador pasando como argumento el directorio donde están los documentos, el archivo con la palabras vacías y el directorio donde se almacenarán los índices
- **Línea 9:** se llama al método **indexFiles** del indexador que crea los índices
- **Línea 11:** se crea el buscador pasando como argumento el directorio que aloja los índices y el archivo de palabras vacías
- **Línea 13:** se llama al método **search** del buscador que obtiene los documentos más relevantes a partir de la búsqueda introducida por el usuario
- **Línea 15-16-18:** se escribe la plantilla en HTML con una ligera modificación con respecto a la pantalla de inicio que se mostrará después
- **Líneas 21-28:** un bucle que recorre todos los documentos relevantes encontrados y se escribe en el documento HTML un enlace por cada resultado obtenido que redirige al usuario al documento

6. Manual de usuario

6.1. Línea de comandos

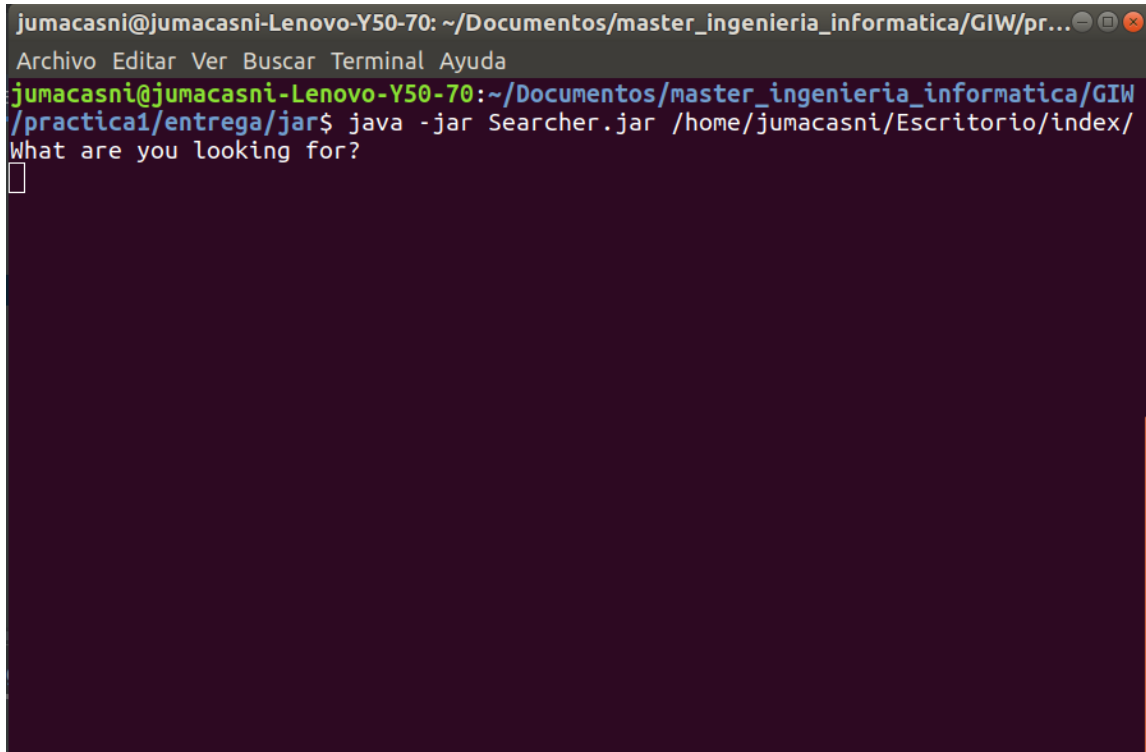
Lo primero que se debe hacer es ejecutar el jar **Indexer.jar** para crear los índices de los documentos. Para ello, se debe ejecutar la instrucción `java -jar <directorio_documentos><fichero_stopwords><directorio_`. En la Figura 5 se muestra un ejemplo realizado en mi propio ordenador. Esto dará como resultado los índices en el directorio `/home/jumacasni/Escritorio/indices`.

A screenshot of a terminal window with a dark background. The window title is "jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...". The menu bar shows "Archivo", "Editar", "Ver", "Buscar", "Terminal", and "Ayuda". The terminal text shows the user at the prompt "jumacasni@jumacasni-Lenovo-Y50-70:~/Documentos/master_ingenieria_informatica/GIW/practica1/entrega/jar\$" typing the command "java -jar Indexer.jar /home/jumacasni/Documentos/master_ingenieria_informatica/GIW/practica1/documents /home/jumacasni/Documentos/master_ingenieria_informatica/GIW/practica1/stop_words/english_stopwords.txt /home/jumacasni/Escritorio/index". The command is executed, and the prompt returns to "jumacasni@jumacasni-Lenovo-Y50-70:~/Documentos/master_ingenieria_informatica/GIW/practica1/entrega/jar\$".

```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
jumacasni@jumacasni-Lenovo-Y50-70:~/Documentos/master_ingenieria_informatica/GIW
/practica1/entrega/jar$ java -jar Indexer.jar /home/jumacasni/Documentos/master_
ingenieria_informatica/GIW/practica1/documents /home/jumacasni/Documentos/master
_ingenieria_informatica/GIW/practica1/stop_words/english_stopwords.txt /home/jum
acasni/Escritorio/index
jumacasni@jumacasni-Lenovo-Y50-70:~/Documentos/master_ingenieria_informatica/GIW
/practica1/entrega/jar$
```

Figura 5: Creación de índices con **Indexer.jar**

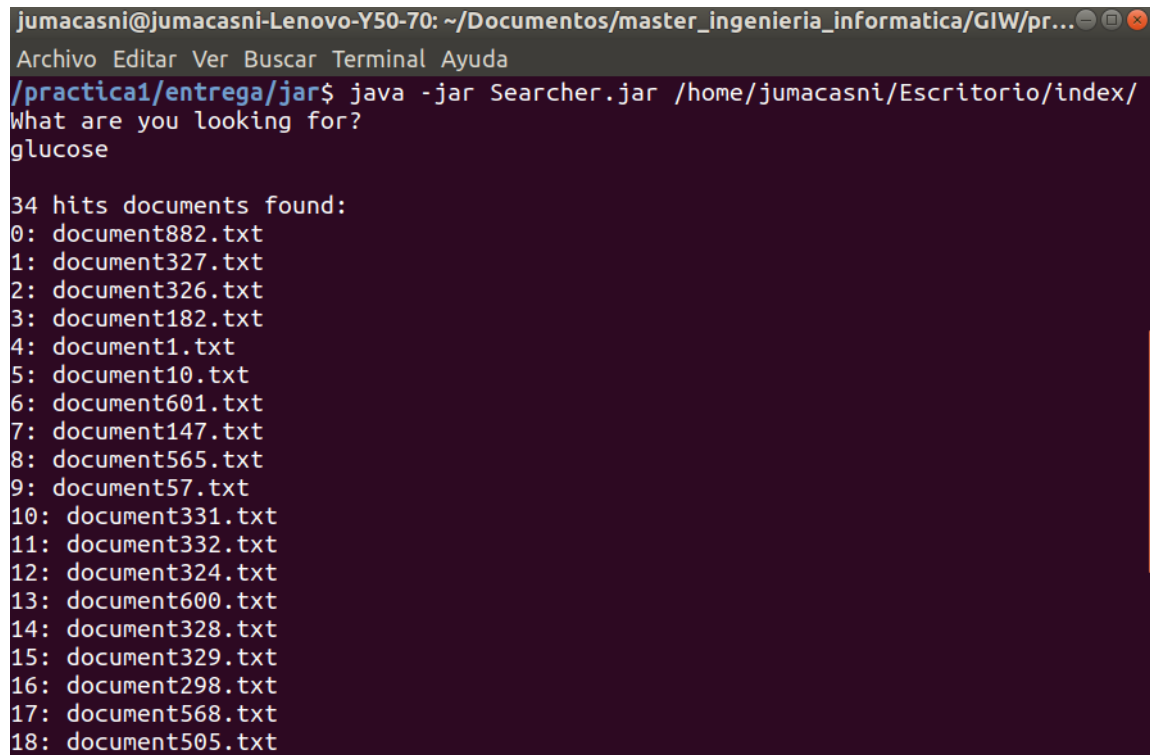
Lo siguiente sería ejecutar la interfaz gráfica mediante la línea de comandos. Para ello, se debe ejecutar la instrucción `java -jar Searcher.java <directorio_indices>`. En la Figura 6 se muestra un ejemplo. El buscador nos pedirá que introduzcamos una búsqueda.



```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
jumacasni@jumacasni-Lenovo-Y50-70:~/Documentos/master_ingenieria_informatica/GIW
/practica1/entrega/jar$ java -jar Searcher.jar /home/jumacasni/Escritorio/index/
What are you looking for?
█
```

Figura 6: Ejecución de la interfaz gráfica en la línea de comandos

En este ejemplo se ha buscado la palabra *glucose*, cuya búsqueda da 34 resultados entre todos los documentos (ver Figura 7).

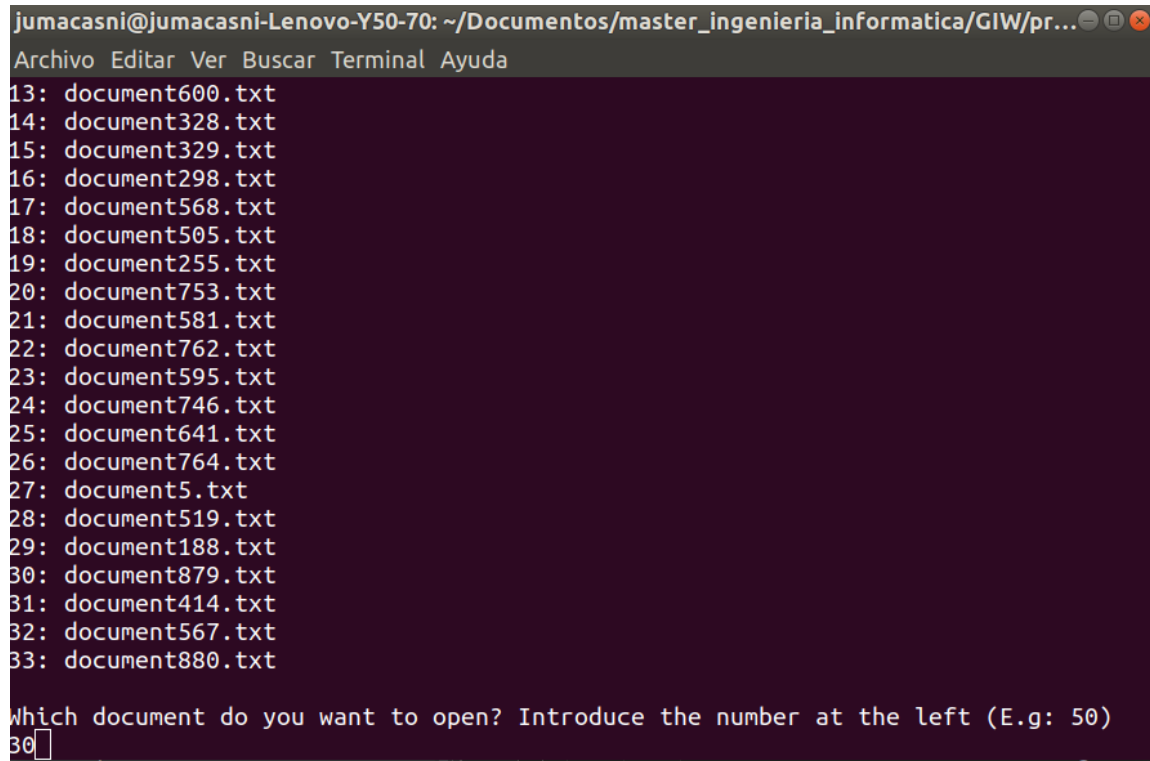
A terminal window with a dark background and light-colored text. The window title is 'jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...'. The menu bar shows 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The prompt is '/practica1/entrega/jar\$'. The user has entered 'java -jar Searcher.jar /home/jumacasni/Escritorio/index/' and the program has responded with 'What are you looking for?'. The user has entered 'glucose'. The program has responded with '34 hits documents found:' followed by a list of 19 document names, each preceded by a number from 0 to 18. The list ends with '18: document505.txt'.

```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
/practica1/entrega/jar$ java -jar Searcher.jar /home/jumacasni/Escritorio/index/
What are you looking for?
glucose

34 hits documents found:
0: document882.txt
1: document327.txt
2: document326.txt
3: document182.txt
4: document1.txt
5: document10.txt
6: document601.txt
7: document147.txt
8: document565.txt
9: document57.txt
10: document331.txt
11: document332.txt
12: document324.txt
13: document600.txt
14: document328.txt
15: document329.txt
16: document298.txt
17: document568.txt
18: document505.txt
```

Figura 7: Ejecución de la interfaz gráfica en la línea de comandos

A continuación se nos pedirá abrir el documento que se quiera. Por ejemplo, si el usuario desea abrir el documento llamado *document879.txt*, el usuario debe introducir el número de la izquierda, **30** (ver Figura 8).

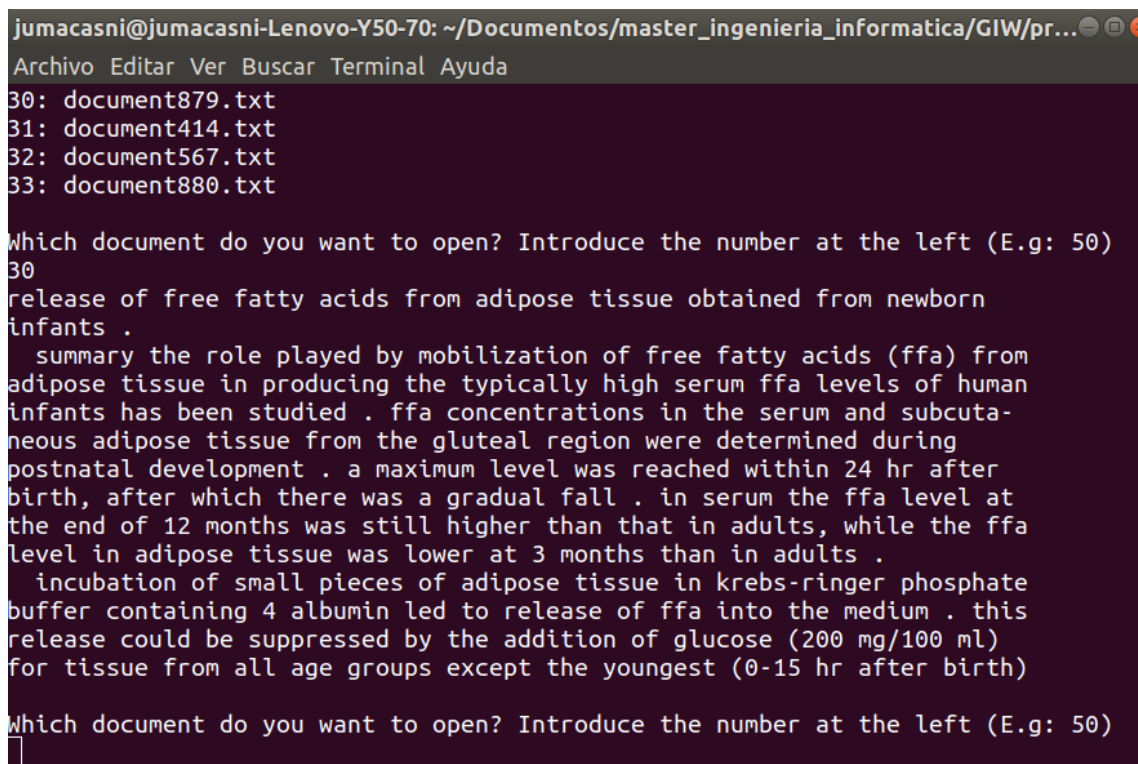


```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
13: document600.txt
14: document328.txt
15: document329.txt
16: document298.txt
17: document568.txt
18: document505.txt
19: document255.txt
20: document753.txt
21: document581.txt
22: document762.txt
23: document595.txt
24: document746.txt
25: document641.txt
26: document764.txt
27: document5.txt
28: document519.txt
29: document188.txt
30: document879.txt
31: document414.txt
32: document567.txt
33: document880.txt

Which document do you want to open? Introduce the number at the left (E.g: 50)
30
```

Figura 8: Ejecución de la interfaz gráfica en la línea de comandos

Como resultado se mostrará el contenido del documento deseado (ver Figura 9).



```
jumacasni@jumacasni-Lenovo-Y50-70: ~/Documentos/master_ingenieria_informatica/GIW/pr...
Archivo Editar Ver Buscar Terminal Ayuda
30: document879.txt
31: document414.txt
32: document567.txt
33: document880.txt

Which document do you want to open? Introduce the number at the left (E.g: 50)
30
release of free fatty acids from adipose tissue obtained from newborn
infants .
summary the role played by mobilization of free fatty acids (ffa) from
adipose tissue in producing the typically high serum ffa levels of human
infants has been studied . ffa concentrations in the serum and subcuta-
neous adipose tissue from the gluteal region were determined during
postnatal development . a maximum level was reached within 24 hr after
birth, after which there was a gradual fall . in serum the ffa level at
the end of 12 months was still higher than that in adults, while the ffa
level in adipose tissue was lower at 3 months than in adults .
incubation of small pieces of adipose tissue in krebs-ringer phosphate
buffer containing 4 albumin led to release of ffa into the medium . this
release could be suppressed by the addition of glucose (200 mg/100 ml)
for tissue from all age groups except the youngest (0-15 hr after birth)

Which document do you want to open? Introduce the number at the left (E.g: 50)

```

Figura 9: Ejecución de la interfaz gráfica en la línea de comandos

6.2. Aplicación web

El proyecto se encuentra en el archivo llamado **Proyecto_web.zip** que es una aplicación web de Java y se puede importar a un **IDE**, en mi caso **Apache Netbeans 12.3**.

6.2.1. Estructura de directorios

Los archivos están estructurados de la siguiente forma:

- En la ruta `/src/main/java/com/juanmanuelcastillonievas/search/*` se encuentran las clases Java utilizadas
- En la ruta `/src/main/webapp/` se encuentran los elementos necesarios para la web: css, js, html, etc.
- En la ruta `/src/main/webapp/resources` hay cuatro elementos:
 - Directorio `/documents`: directorio donde se almacenan los documentos
 - Directorio `/index`: directorio donde se almacenan los índices
 - Directorio `/stop_words`: directorio que contiene el archivo con las palabras vacías
 - Archivo **MED.ALL**: archivo que contiene todos los documentos y que se debe parsear para sacar los documentos individuales

6.2.2. Ejecución

Lo primero que se debe hacer es ejecutar la clase **Parser.java** para parsear el archivo **MED.ALL** y crear los documentos.

Una vez que los documentos están creados, se ejecuta el proyecto web Java que abrirá en el navegador la url <http://localhost:8080/Search/> y ya se podrán realizar búsquedas.

En la Figura 10 se muestra la **búsqueda que realiza el usuario** en la página de inicio, que en este caso se ha usado la consulta **“glucose”**. Cuando el usuario pulsa el botón de buscar, se mostrarán los **resultados más relevantes** en forma de enlaces (ver Figura 11). Por último, en la Figura 12 se muestra el **primer documento** que se ha encontrado con esta búsqueda.

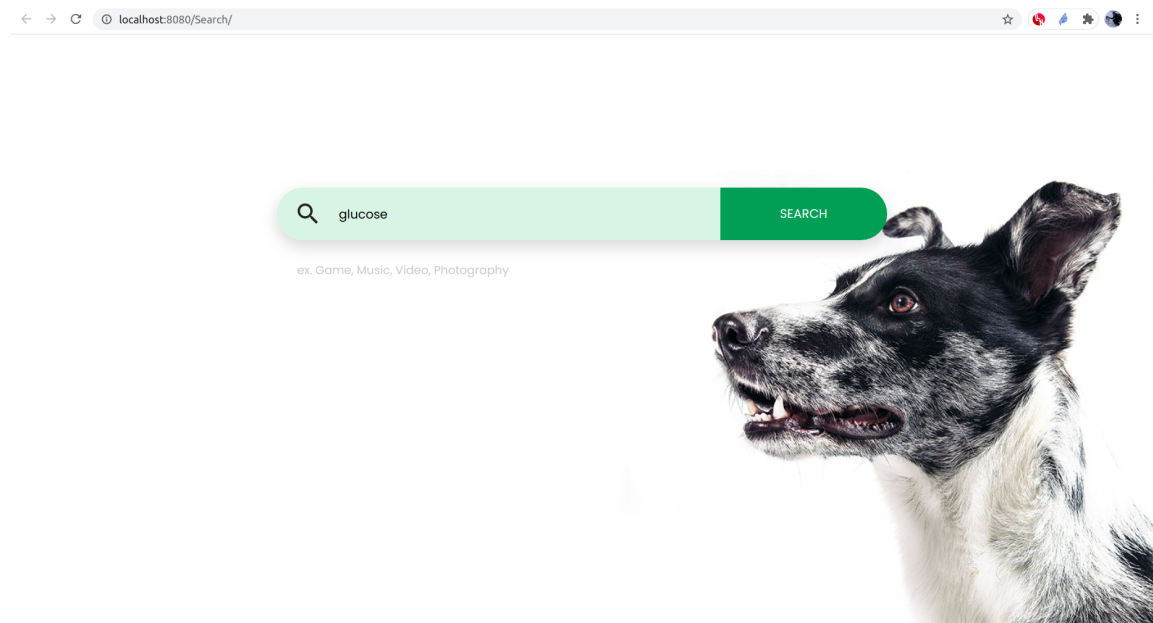


Figura 10: Usuario introduce la búsqueda

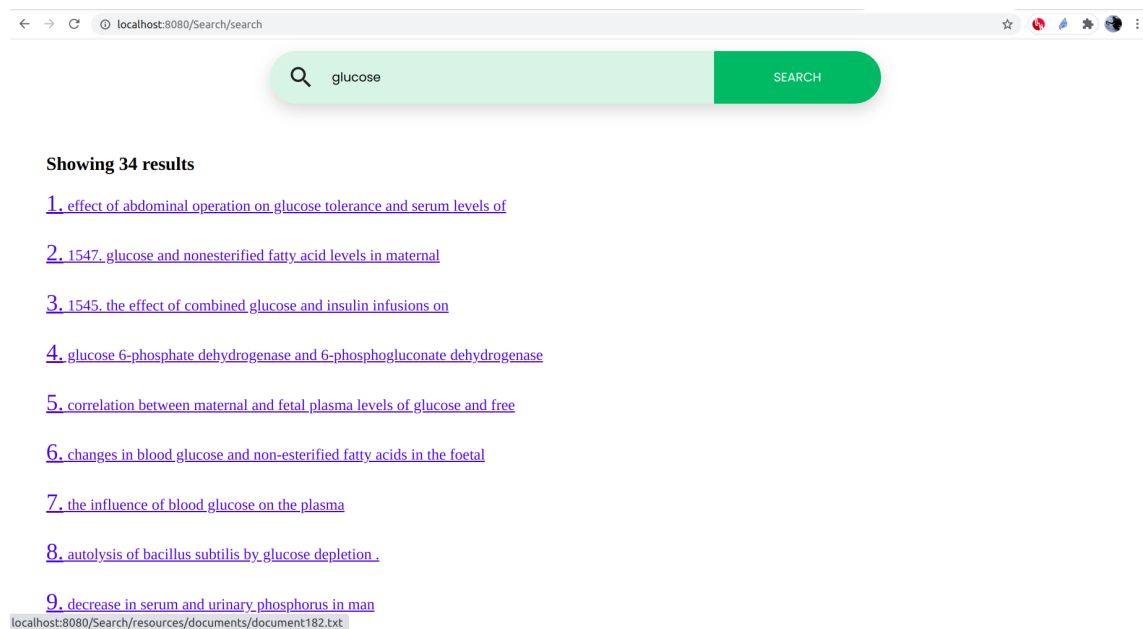


Figura 11: Resultados más relevantes



Figura 12: Primer documento encontrado

7. Referencias

- [1] Apache lucene™ 8.8.1. <https://lucene.apache.org/>, 2021.
- [2] Apache netbeans 12.3. <https://netbeans.apache.org/download/nb123/nb123.html>, 2021.
- [3] Digital Guide IONOS. Apache lucene: búsqueda libre para tu sitio web. <https://www.ionos.es/digitalguide/servidores/configuracion/apache-lucene/>, 2019.
- [4] Glasgow Information Retrieval Group. Medline collection. http://ir.dcs.gla.ac.uk/resources/test_collections/medl/.
- [5] Bootstrap. Build fast, responsive sites with bootstrap. <https://getbootstrap.com/>, 2021.
- [6] Colorlib. Colorlib search form v23. <https://colorlib.com/wp/template/colorlib-search-23/>, 2021.