



Recuperación de la información y web  
semántica

Máster Universitario en Ingeniería  
Informática

2015/16

**PRÁCTICA LIBRE**

**FACEBOOK CRAWLER && OFFLINE SEARCHER**

Daniel Ruiz Pérez  
Bruno Santiago Vázquez

## Índice

1. Introducción y estado del arte .....	3
2. Desarrollo .....	4
2.1. Obtención de los datos .....	4
2.2. Parseo y almacenamiento de los datos.....	6
2.3. Indexado de los datos .....	7
2.4. Consultas sobre los datos .....	9
3. Ejecución .....	11
4. Conclusiones y desarrollos futuros.....	16

# 1. Introducción y estado del arte

Hoy en día las redes sociales tienen un importantísimo peso en la forma en la que nos relacionamos. Tanto que algunos incluso hablan de una revolución en la forma en la que nos relacionamos que ha llegado para quedarse. A pesar de todas las características, funcionalidades y facilidades que ofrecen las principales (Facebook, Twitter, LinkedIn...) todavía existen multitud de posibilidades que no se han explorado, y se empieza a hacerlo a través de plugins y extensiones tanto oficiales como no oficiales.

Es por ello que el presente trabajo se va a centrar en Facebook, ofreciendo características inexistentes hasta el momento.

Inicialmente la idea que se barajaba era realizar una aplicación que, dados dos usuarios de Facebook obtuviera el camino mínimo de amigos necesarios para enlazarlos.

Después de una extensa búsqueda nos sorprendió que no existiera ninguna aplicación que realizara esta tarea, dado el gran potencial social que implica conocer a alguien que conoce a alguien que te puede presentar a tu futuro jefe. Sin embargo, los profesores de la asignatura nos recomendaron cambiar de idea ya que no se ajusta a los objetivos de la misma. De este modo se optó por realizar la idea que da título a esta memoria.

Finalmente la idea que se implementó es un vitaminado buscador de perfiles de Facebook que permite realizar diferentes tipos de consultas y que no requiere conexión a internet. Además los resultados se geolocalizan de forma visual en un mapa interactivo.

Hoy en día no resulta raro encontrar personas que tienen Facebook principalmente para formarse una primera impresión de la gente. Se conoce o se escucha hablar sobre alguien y antes de darle al cerebro tiempo a decidir, se busca su perfil en internet y se forma una impresión bastante acertada sobre dicha persona. El problema es que esto, que para algunos es una necesidad, necesita conexión a internet. De este modo el buscador aquí desarrollado simplificará este problema de raíz al funcionar offline.

Además permite realizar búsquedas complejas presentando los resultados de forma visual con la opción de geolocalizar dichos usuarios.

## 2. Desarrollo

Para su desarrollo será necesario obtener información de usuarios de Facebook (crawler/API), almacenarla, indexarla y realizar posteriores búsquedas sobre los datos. Estos apartados pueden considerarse hasta cierto punto independientes, por lo que se pudieron desarrollar en paralelo de una forma cómoda.

### 2.1. Obtención de los datos

Facebook presenta una cómoda API para obtener datos almacenados en sus servidores. El problema es la protección de los mismos, ya que para poder acceder a los amigos de un usuario se necesita que ese usuario esté registrado en nuestra aplicación. Dada la inviabilidad de que todos los usuarios de los que se quiere acceder a los datos estén registrados en nuestra aplicación (tanto para la idea original como para la final), se ha optado por realizarlo de la manera más complicada y menos legal. Crawllear las páginas de Facebook y obtener la información de los usuarios parseando la misma.

Para ello se usará un perfil de Facebook de prueba, con el cual se podrá acceder a todos los datos públicos de los usuarios exportando las cookies de su sesión y enviandolas en las peticiones realizadas.

Se intentó utilizar Apache Nutch pero las pruebas fueron infructuosas debido a que no soporta cookies de la misma forma en la que se pretendía. Aún así, el afán de utilizar un crawler ya implementado llevó a realizar intensivas pruebas con Scrappy, crawler que tampoco se consiguió adaptar a la finalidad deseada para que el uso de cookies, a pesar de su increíble facilidad de uso.

El equipo de desarrollo consideró más fructífero desarrollar su propio crawler en python basándose en el comando de wget que acepta el envío de cookies.

El proceso completo que se ha realizado para el crawling de los datos se ha desarrollado en 3 fases:

- **Creación de un perfil de facebook y extracción de las cookies.**

Desde la página principal de facebook, se han ingresado datos ficticios para realizar las pruebas junto con un correo electrónico temporal. Una vez realizado el registro, facebook envía un mail a la dirección indicada con un enlace para activar la cuenta. Tras clicar en dicho enlace se nos redirige a la página principal de nuestro usuario. En este punto necesitamos extraer las cookies que facebook envía al navegador. En nuestro caso hemos utilizado Firefox por lo que se ha instalado la extensión "Export cookies" la cual nos facilita el guardado de las cookies en un archivo de texto.

.facebook.com	TRUE	/	FALSE	1507728334.619856	datr	dbIbVpQZPw_xDri5g6Uxwbo0
.developers.facebook.com	TRUE	/	FALSE	1511550726	_ga	GA1.3.1400043036.1448478514
.facebook.com	TRUE	/	FALSE	1449851035.293271	locale	en_US
.facebook.com	TRUE	/	TRUE	0	c_user	100010853312759
.facebook.com	TRUE	/	FALSE	1457054949.449091	fr	0YgpUmIDzLHRWYFes.AWWGR_GjRHRcdzmYIzev4Ea5E2c.BWX3i0.En.AAA.0.AWWKmTaa
.facebook.com	TRUE	/	TRUE	0	xs	176%3AHwx3G7HxBZ7fjw%3A2%3A1449278940%3A-1
.facebook.com	TRUE	/	FALSE	0	cs	2
.facebook.com	TRUE	/	TRUE	0	s	Aa78pqIFlqj0EfSu.BWYj3d
.facebook.com	TRUE	/	TRUE	1512350949.449739	lu	Sgj281GESicvVeV0zyzoMbZg
.facebook.com	TRUE	/	FALSE	0	p	-2
.facebook.com	TRUE	/	TRUE	0	presence	EDvF3EtimeF1449279272EuserFA21B10853312759A2EstateFDutF14492792724EatF1449279272468Et
.facebook.com	TRUE	/	FALSE	0	act	1449279273357%2F0

- **Construir el comando 'wget'.**

Las trabas que hemos tenido para poder acceder a los datos públicos de un perfil de facebook pasaban por lanzar una petición que diese a entender que se estaba realizando por un usuario manualmente con lo que, además de las cookies se ha tenido que indicar el UserAgent de un navegador conocido. Como la realización se ha hecho con Firefox el comando final es el que sigue:

```
wget -q --output-document=outFile
"https://www.facebook.com/test.user" --load-cookies cookies.txt --
user-agent="Mozilla/5.0 (Windows NT 6.2; rv:10.0) Gecko/20100101
Firefox/33.0"
```

- **Desarrollo del crawler.**

El crawler desarrollado se ha implementado en python. El proceso que se basa en obtener datos de usuarios en base a las relaciones entre los mismos.

Para empezar se parte de un usuario semilla del cual se descargan datos de su biografía (extensión .biography) y de sus amigos (extensión .friends). El nombre del archivo siempre está formado por el nick del usuario y la correspondiente extensión. Una vez se obtienen los datos del usuario semilla, se hace un parsing de los amigos, se generan las urls que habrá que descargarse y, el usuario ya obtenido pasa a una lista que mantendrá un control de los usuarios ya crawleados para evitar obtener varias veces un mismo usuario, con su consecuente pérdida de tiempo.

Además, se controla que los archivos descargados no contienen un mensaje de error por un bloqueo de la cuenta por parte de facebook. En este caso se cancela el crawling de datos de forma automática.

Como es de esperar, la descarga de los datos se realiza mediante el comando 'wget' antes mencionado tanto para la biografía como para los amigos.

Finalmente se han añadido algunos parámetros configurables como el usuario semilla, el número de peticiones conjuntas, los segundos de retraso entre conjuntos de peticiones o el nivel de profundidad en las relaciones.

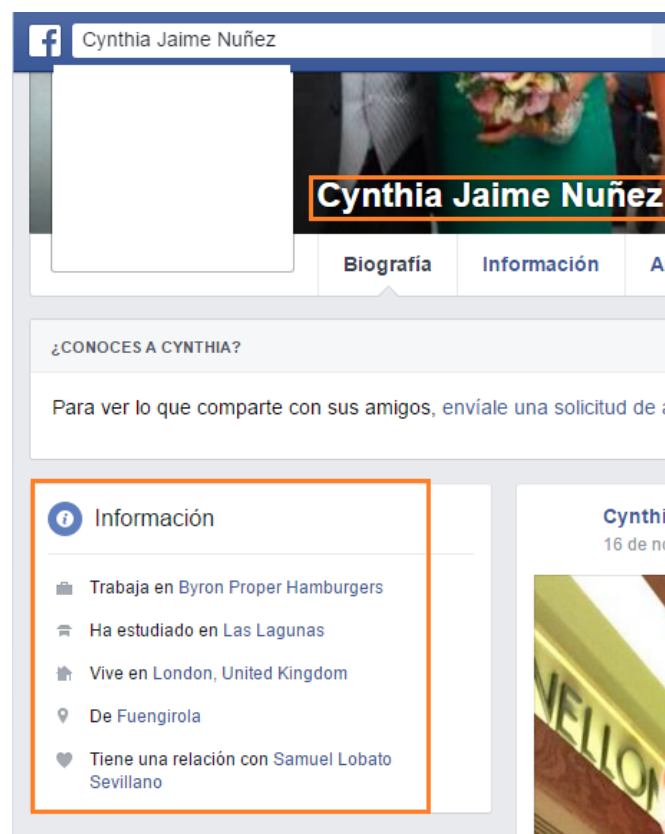
Para empezar a crawlear datos se han usado varios usuarios semilla por selección aleatoria. Se han hecho pruebas inicialmente sin mucho control y no generaban ningún problema de limitaciones en la cuenta por parte de facebook. Los parámetros constaban de niveles de profundidad de un máximo de 3 y ningún tipo de retraso entre peticiones, incluso llegando a lanzarse en muchas de las pruebas de forma paralela. Aunque bien es cierto que aumentando el nivel

de profundidad a 4 provocaba la restricción de acceso a nuevos perfiles desde la cuenta.

Tras realizar este tipo de implementación nos hemos dado cuenta de que no se estaban cumpliendo ciertos requisitos de los crawlers, más concretamente el de politeness por lo que, se han aplicado posteriormente modificaciones para poder tratar de ser más respetuosos con el sitio web. Se han aplicado cambios también con el número de peticiones y el retraso entre las mismas donde, para mantener un bajo ancho de banda se ha configurado una petición cada 10 segundos evitando así consumir un alto ancho de banda con Facebook. La elección de este tiempo concreto se ha basado en que no se especifica en el fichero *robots.txt* ningún *crawl-delay* recomendado, así que se ha usado el sugerido por *Cho et al.* Esta decisión en nuestro caso práctico concreto conlleva un proceso lento pero que asegura que los datos se obtienen correctamente. A pesar de esto, finalmente hemos deducido que Facebook mantiene un control del número de peticiones lanzadas en un período de tiempo o de la actividad de la cuenta para realizar un análisis y determinar si el control de la cuenta lo está realizando un humano o una máquina.

## 2.2. Parseo y almacenamiento de los datos

Una vez obtenidos todas las páginas html de los perfiles de Facebook deseados, es necesario obtener los datos interesantes de las mismas. Parsear un .html no es una tarea fácil ,pero se consiguió mediante el uso de eficientes expresiones regulares.



Debido a que un usuario puede elegir qué características suyas son públicas y cuáles no, además de poder optar por dejarlas en blanco, la extracción de cada una de estas deberá ser independiente para poder abarcar cualquier combinación de las mismas.

Los datos extraídos de cada perfil son almacenados en un fichero .txt con el identificador del usuario como nombre y con un diccionario de las características extraídas, como se puede observar en la siguiente imagen:

```
1 Name: Cynthia Jaime Nuñez
2 Friends: 191
3 Lives in: London, United Kingdom
4 Works at: Byron Proper Hamburgers
5 From: Fuengirola
6 In a relationship with: Samuel Lobato Sevillano
7 Went to: Las Lagunas
8 Friends: ['miguelangel.galannunez', 'davinia.jaimenunez', 'vicky.calventemartin', 'paula.galannunez', 'xPinkYasLadyboss', 'lydia.gomezpacheco', 'samuel.lobatosevillano', 'franciscojavier.gomez.94', 'osunagalvan', 'gere.judit.3', 'albamaria.gomezquero', 'noelia.membrivesgalvan', 'ana.tellez.313', 'domigv', 'maria.nunezgalvan', 'myles.davies.505', 'pablo.deborbon', 'paloma.ramirezmerino.1', 'gloria.gonzalezporras', 'maria.martinruiz.9']
```



## 2.3. Indexado de los datos

Tras parsear los datos en HTML y generar los archivos con los datos necesarios se procede a la indexación de los mismos. En nuestro caso todos los campos son almacenados ya que tiene sentido que, al tratarse de datos de índole pública, puedan ser campos indexados y almacenados para su posterior consulta y recuperación. El único campo que podría ser indexado es el identificador del usuario, pero se decidió almacenarlo para ofrecer una búsqueda más completa de los datos.

Internamente se extraen los datos de los archivos .txt con la clase DataExtractor que lee dichos archivos y crea una representación interna del usuario con los objetos UserInfo para un tratamiento más cómodo de los datos. Un detalle de implementación importante es que, este objeto UserInfo posee un método que getDocument que devuelve un objeto Document de Lucene para insertarlo en el índice. Además, existen dos constructores donde el primero recibe todos los datos del usuario y el segundo un objeto de tipo Document. De esta forma podemos reutilizar esta representación para tratar a los usuarios en toda la aplicación realizada.

```

public class UserInfo {

    private String nick;
    private String name;
    private String studied;
    private String studiedAt;
    private Integer friendsTotal;
    private String bornOn;
    private String livesIn;
    private String worksAt;
    private String from;
    private String marriedTo;
    private String relationshipWith;
    private String followedBy;
    private String wentTo;
    private String goesTo;
    private String profileImgUrl;
    private String[] friends;

    public UserInfo(String nick, String name, String studied, String studiedAt, int friendsTotal) {}

    public UserInfo(Document d) {}

    public Document getDocument() {
        Document doc = new Document();
        doc.add(new TextField(IndexConstants.NICK, nick, Field.Store.YES));
        if (name != null)
            doc.add(new TextField(IndexConstants.NAME, name, Field.Store.YES));
        if (studied != null)
            doc.add(new TextField(IndexConstants.STUDY, studied, Field.Store.YES));
        if (studiedAt != null)
            doc.add(new TextField(IndexConstants.STUDY_AT, studiedAt, Field.Store.YES));
        if (friendsTotal != null)
    }

```

Una vez se extraen los datos de los archivos y tenemos una lista de objetos UserInfo, creamos un analizador estándar de Lucene, se crea el índice y el writer del mismo con lo que procedemos a iterar sobre los usuarios y añadirlos al índice. Finalmente se devuelve un valor entero con el total de usuarios añadidos al índice.

```

public class FbIndexer {

    public static int runIndexer(OpenMode indexMode, String indexName, String dataFolder) throws
    // Get all users from files
    List<UserInfo> users = DataExtractor.extract(dataFolder);
    System.out.println("-----");
    System.out.println("\t Extracted info for " + users.size() + " users");
    System.out.println("-----");

    // Create a standard analyzer
    Analyzer analyzer = new StandardAnalyzer(Version.LUCENE_48);
    // Create a disk index
    Directory directory = FSDirectory.open(new File(indexName));
    // Configure index
    IndexWriterConfig config = new IndexWriterConfig(Version.LUCENE_48, analyzer);
    config.setOpenMode(indexMode);
    // Create index writer
    IndexWriter iwriter = new IndexWriter(directory, config);

    // Get a document for each user and add to index
    for (UserInfo userInfo : users) {
        System.out.println("Adding document of user with nick " + userInfo.getNick());
        iwriter.addDocument(userInfo.getDocument());
    }
    System.out.println("-----");
    System.out.println("\t Added " + users.size() + " documents");
    System.out.println("-----");

    // Close index writer
    iwriter.close();
    // Close index directory
    directory.close();

    return users.size();
}

```



## 2.4. Consultas sobre los datos

Se ha diseñado el proceso de búsqueda para que sea totalmente configurable desde la interfaz, tanto el tipo de búsqueda, los máximos resultados a mostrar, el índice y los campos sobre los que se realiza.

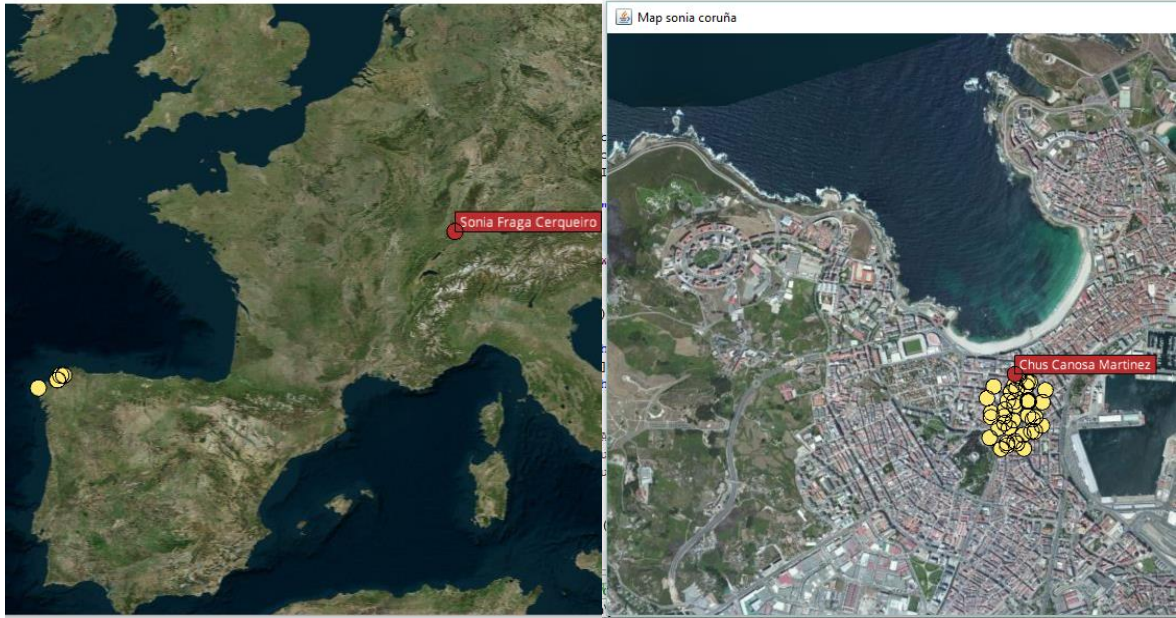
Se han definido 3 tipos de búsqueda, para todos ellos se utiliza un `DirectoryReader` y un `IndexSearcher`:

- Para el tipo de consultas booleanas se ha utilizado una `BooleanQuery`, construída parseando el texto con las palabras clave “-and”, “-not”, “-can” que pueden ir seguidas de tantos términos como se quiera. Además, al tender las búsquedas a mostrar pocos resultados, si no se encuentra ninguno, se relanza automáticamente la misma consulta pero esta vez siendo además una `FuzzyQuery`.
- La consulta por defecto se realiza sobre los campos especificados en el menú de configuración de búsqueda, utilizando un `MultiFieldQueryParser` y un `StandardAnalyzer`.
- Además se realizan `PrefixQueries`, que se lanzan automáticamente a medida que se va escribiendo en el campo de búsqueda para realizar la funcionalidad de recomendación del autocompletado del texto en base a los campos de búsqueda seleccionados en las opciones de configuración.

Además, las dos primeras consultas tienen en cuenta el campo de localización del usuario, transformándolo mediante un proceso de forward geocoding para obtener las coordenadas del lugar. Estos datos se guardan en un fichero xml que es el que se utilizará para mostrar los resultados en un mapa interactivo donde aparece marcada la localización de los usuarios que fueron devueltos por la consulta. La librería utilizada fue `Unfolding Maps`, que permite multitud de opciones para la realización de mapas offline en un applet.

Un ejemplo lo podemos ver si, por ejemplo realizamos una `MultiFieldQuery` buscando “Sonia Coruña”, nos devolverá todas las Sonias que viven en Coruña, todas las personas que se llamen Sonia, y finalmente todas las personas que viven en Coruña (`maxHits` = 50).

ruña



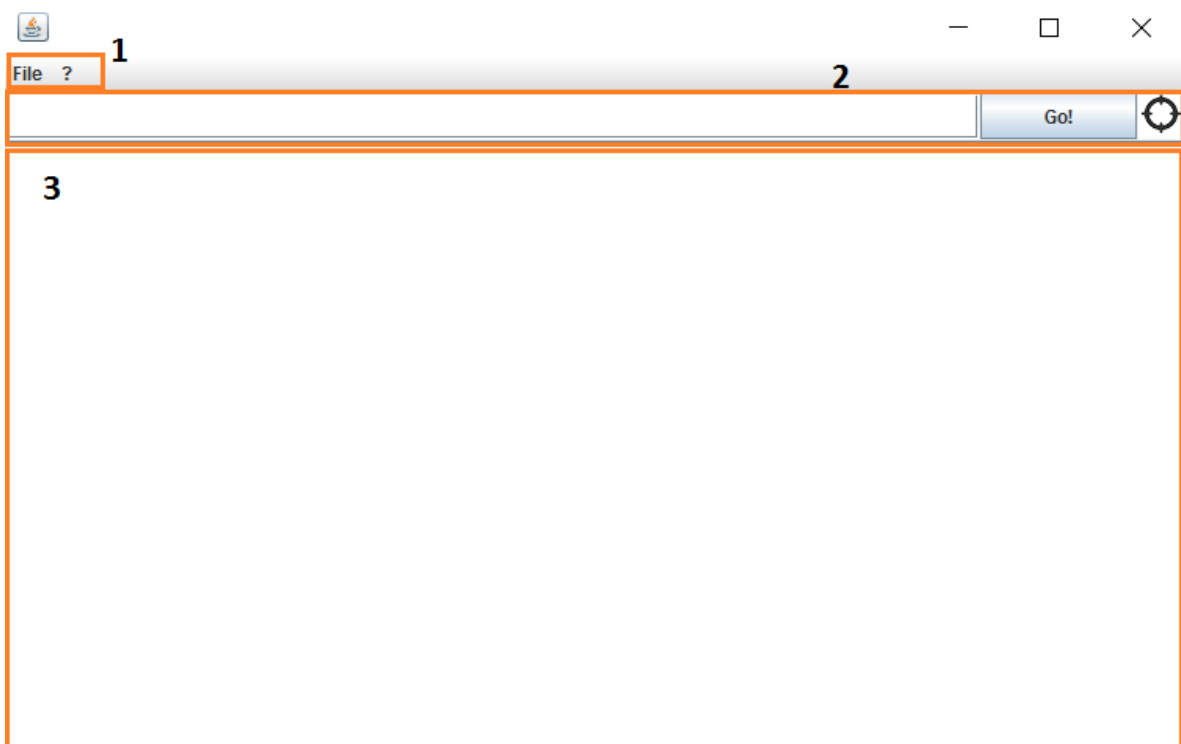
### 3. Ejecución

Para ejecutar la aplicación desarrollada basta con lanzar el jar adjunto con el siguiente comando:

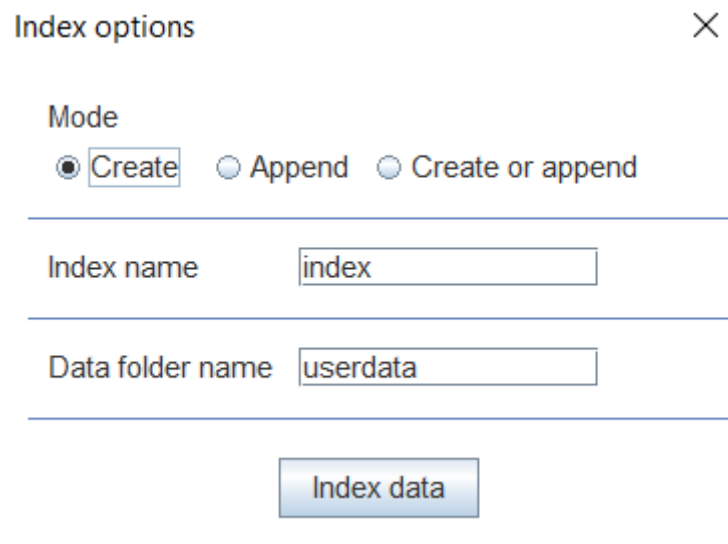
```
java -jar fb-offline-searcher.jar
```

La vista inicial es la que se presenta a continuación, donde la interfaz se divide en tres zonas:

1. Menú.
  - a. Apartado 'File' a través del cual se accede a las opciones de indexado y búsqueda.
  - b. Apartado '?' donde se muestra información de la aplicación.
2. Búsqueda
  - a. Cuadro de búsqueda con la función de autocompletado.
  - b. Botón para lanzar la consulta al índice.
  - c. Botón de geolocalización de los usuarios resultantes de la ejecución de la consulta.
3. Panel de resultados (SERP)
  - a. Panel donde se mostrarán los resultados obtenidos de la consulta lanzada sobre el índice.



Sumergiéndonos más en las opciones disponibles de la aplicación veremos la pantalla referente a los parámetros que podemos variar para realizar la indexación de los datos:

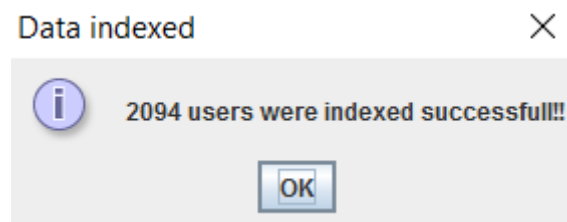


The 'Index options' dialog box features a title bar with a close button (X). Below the title bar, the 'Mode' section contains three radio buttons: 'Create' (selected), 'Append', and 'Create or append'. Below this, there are two text input fields: 'Index name' with the value 'index' and 'Data folder name' with the value 'userdata'. At the bottom of the dialog is a button labeled 'Index data'.

En esta pantalla se puede seleccionar el modo del índice por lo que se creará uno nuevo, se añadirán los resultados a un índice ya existente o dejar que se decida dinámicamente tratando de añadir los resultados y, en caso de no existir el índice, creándolo.

Además, se puede indicar el nombre del índice y la carpeta donde están presentes los archivos .txt de los usuarios.

Tras realizar la acción del botón 'Index data' se indexarán dichos usuarios y, finalmente se mostrará un mensaje informativo con el número de usuarios añadidos. En caso de algún error en los parámetros también se indicará mediante un mensaje de error.



Por otro lado tenemos las opciones de búsqueda, con un cuadro de diálogo similar al de indexación y con las opciones a continuación:

Search options ×

Maximum results

---

Active search fields

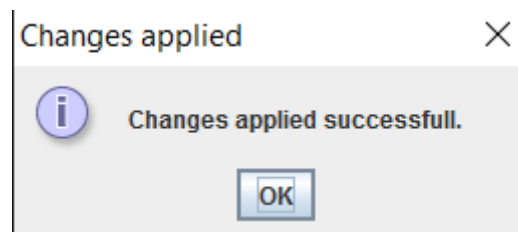
☒ nick      ☒ name      ☒ studied  
☒ studiedAt    ☒ totalFriends    ☒ live  
☒ from      ☒ wentTo      ☒ worksAt

(You can use this names in boolean queries)

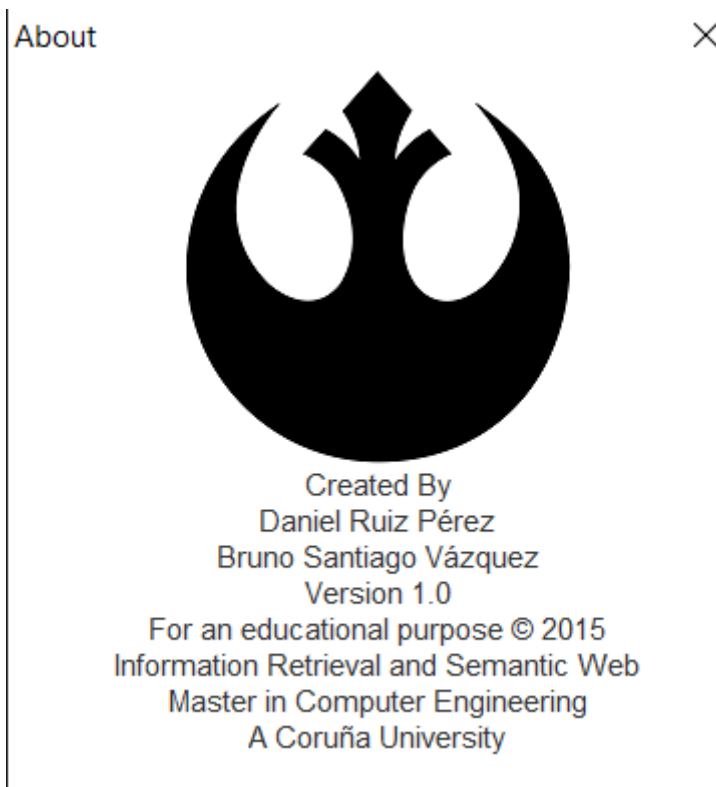
---

Apply changes

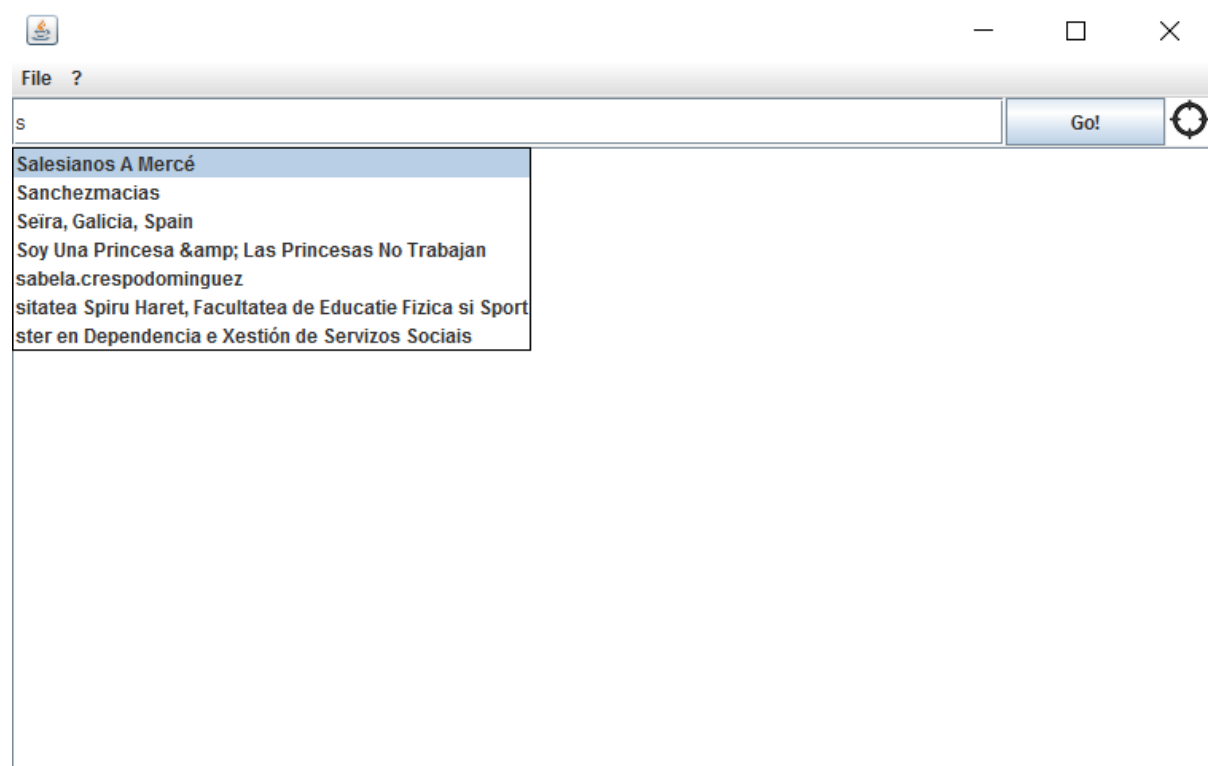
En esta nueva pantalla se pueden indicar el número máximo de resultados que se mostrarán en el panel de resultados y los campos por los que se aplicará la búsqueda, marcando o desmarcando los campos deseados. Una vez hayamos configurado el número de resultados y los campos de búsqueda (que aplicarán también en el autocompletado del cuadro de búsqueda) se mostrará un mensaje indicando que los parámetros se han guardado correctamente o un mensaje de error si alguna opción es inválida.



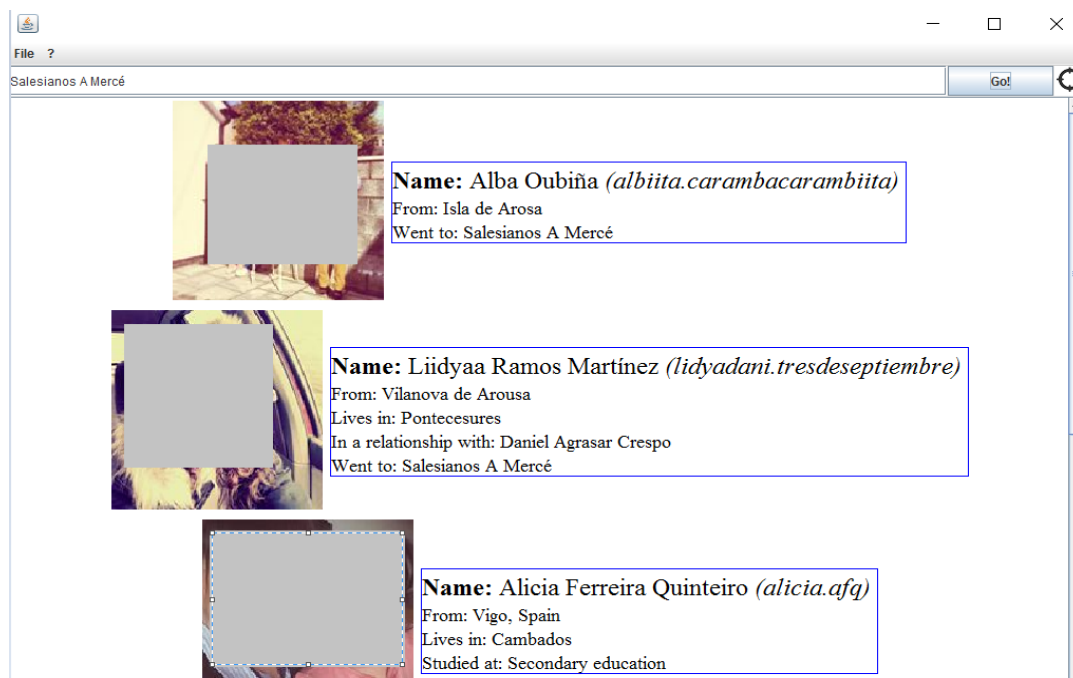
En la pantalla de información de la aplicación se muestran datos de los desarrolladores, la versión de la aplicación y el propósito del software desarrollado, entre otros.



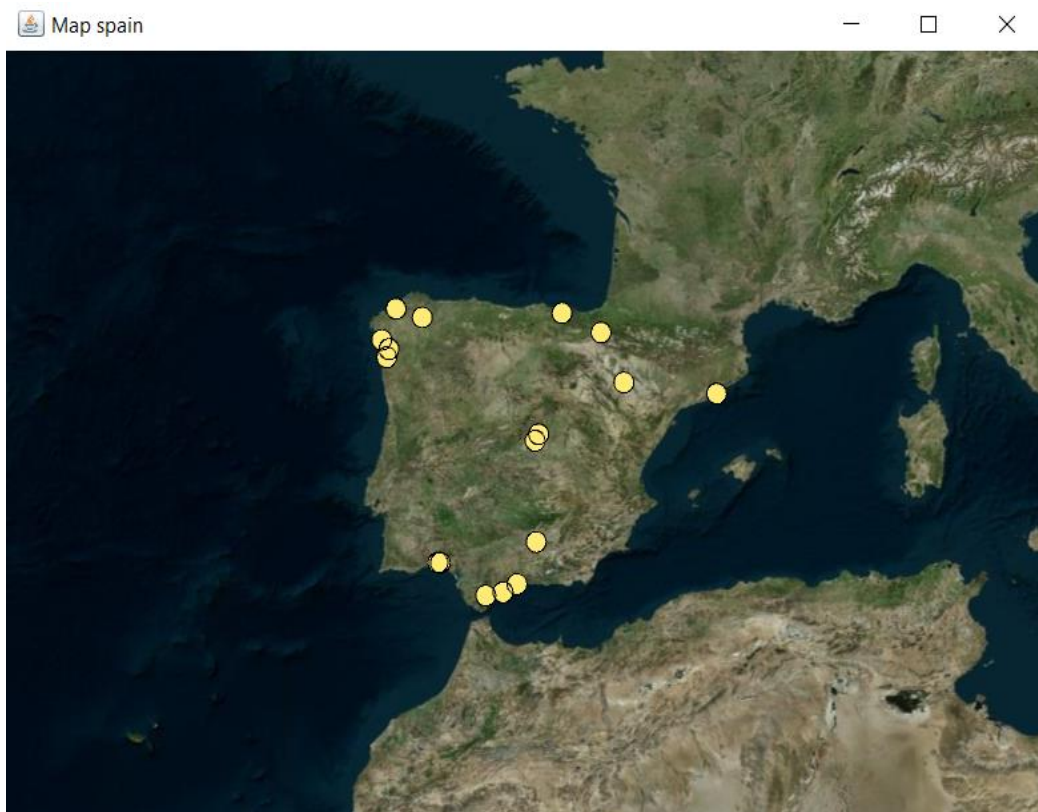
La función de autocompletado, como ya se ha mencionado previamente, aplica sobre los campos sobre los que se realizará la búsqueda y de forma dinámica, con lo que, mientras escribimos nuestra consulta veremos posibles opciones de autocompletado que podremos seleccionar para añadir al cuadro de búsqueda.



Una vez decidida la consulta, pulsamos el botón de búsqueda (Go!). Los resultados obtenidos se mostrarán en el cuadro de resultados.



Tras realizarse la consulta y mostrar los resultados se pueden visualizar en un mapa pulsando el botón de geolocalización. Esta funcionalidad se puede comprobar fácilmente modificando las opciones de búsqueda, cambiando el número máximo de resultados a 100 y realizando una consulta genérica (p.ej. Spain). Luego pulsamos el botón de geolocalización para ver el mapa.



## 4. Conclusiones y desarrollos futuros

A pesar de que la idea inicial que se tenía era perfectamente legal y legítima al acceder a los datos a través de la API, somos conscientes de que la orientación final que se le ha dado no lo es.

Sin embargo, dado que el objetivo de la presente práctica es plenamente docente, sin ningún fin comercial y con un alcance muy reducido se ha considerado legítimo continuar con su desarrollo. El desarrollo de la misma ha supuesto un reto sobre todo al respecto de comprobar si se podía “hackear” Facebook de una forma sencilla y ha resultado que sí. Además no han eliminado al usuario que lanzaba las peticiones si no que se le ha limitado el acceso a los perfiles durante 3 semanas a modo de aviso, por lo que Facebook tampoco considera delictiva la actividad que éste llevó a cabo.

A pesar de todo, la herramienta aquí desarrollada podría utilizarse con interesantes fines no comerciales.

Dada la impermeabilidad con la que facebook trata sus datos, es muy difícil realizar estudios sociológicos sobre los mismos. Nuestra herramienta soluciona este problema debido a que obtiene todos los datos públicos de cualquier perfil automáticamente. Esto podría utilizarse con fines educativos e investigadores. Conocer la forma en la que se relaciona la gente en función de sus estudios, edad, localidad... Podría ser una enorme fuente de conocimiento y de interesantes estudios que ayudarían a comprender la forma en la que nos relacionamos.

Todo esto podría potenciarse aumentando las características de la herramienta. Actualmente no se descargan todas las fotos de los usuarios por problemas de espacio, pero podrían obtenerse perfectamente las más relevantes. Además, el profundizar en la forma en la que se almacenan los datos de las amistades podría ser útil. Actualmente se almacenan como una simple lista, pero podría generarse una red de nodos conectados entre sí, teniendo cada nodo los atributos de cada persona y las relaciones serían las de amistad.

Además en relación a esto podría implementarse la idea inicial que se tenía de la herramienta que, dados dos perfiles cualesquiera de facebook encontrase un camino de amigos que los enlazara. Sin embargo, debería tenerse un porcentaje muy alto de Facebook crawlado para que esto fuera posible. Aun así, podría hacerse una herramienta más específica que solo enlazara usuarios de una zona, por ejemplo tener descargados todos los usuarios de Galicia no sería tan descabellado y ahí sí que se podría calcular el camino mínimo de un usuario a otro.

Además, la modularidad del diseño permite extender el crawler a casi cualquier otra red social de forma similar reutilizando la mayoría de los componentes. Simplemente habría que implementar un parser para ese contexto específico.