UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UNA EXTENSIÓN CHROME PARA ANÁLISIS DE REDES SOCIALES

Laura Solange De Abreu Gil
Tutor: Álvaro Ortigosa Juárez

Julio 2020

DISEÑO E IMPLEMENTACIÓN DE UNA EXTENSIÓN CHROME PARA ANÁLISIS DE REDES SOCIALES

AUTOR: Laura Solange De Abreu Gil
TUTOR: Álvaro Ortigosa Juárez

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
julio de 2020

Resumen

Las redes sociales a lo largo de los últimos años se han ido convirtiendo cada vez más importantes. Twitter, LinkedIn y Facebook son de las redes sociales más influyentes actualmente, ya que desde ellas se pueden compartir opiniones, conocer gente afín, impartir entretenimiento, debates, darse a conocer en el ámbito laboral (LinkedIn) y otros contenidos de forma pública.

La información que se comparte en estas redes sociales, así como las relaciones que se establecen, son una fuente de información muy valiosa para diversos investigadores, tanto en el ámbito social, en el comercial, e incluso en el policial. Un típico problema de estos analistas es que muchas veces no tienen los conocimientos informáticos para hacer extracciones de información para su posterior análisis, o si pueden hacerlo les lleva mucho tiempo.

Por este motivo, este trabajo tiene como objetivo hacer una herramienta práctica para obtener información pública contenida en los perfiles de las redes sociales. Para obtener esta información se han usado herramientas que realizan una técnica llamada *scraping*. El *scraping* se trata de un procedimiento que extrae la información de un sitio web accediendo como si fuese un navegador, transformando el código HTML a un formato de almacenamiento práctico para su estudio.

La herramienta que se ha desarrollado es una extensión de Chrome que utiliza programas de *scraping* para obtener datos de perfiles de las redes sociales de Twitter, LinkedIn y Facebook. Se ha pensado de esta forma adaptarse lo más posible al flujo de trabajo habitual de los analistas, que normalmente desempeñan sus tareas usando precisamente este navegador. Las herramientas de *scraping* están desarrolladas sobre servidores específicos en la nube. Esto facilita que el usuario a la hora de instalarse la extensión no tenga que instalar también todo el programa de *scraping*.

El objetivo final es usar los datos obtenidos a partir del *scraping* para ser usados en herramientas más avanzadas de análisis de redes sociales (SNA). Los SNA tiene como objetivo investigar estructuras sociales mediante el uso de redes y teoría de grafos.

Palabras clave

Twitter, LinkedIn, Facebook, *scraping*, *scraper*.

Abstract (English)

Social networks over the years have become increasingly important. Twitter, LinkedIn and Facebook are the most influential social networks currently, since they can share opinions, meet like-minded people, provide entertainment, debates, make themselves known in the workplace (LinkedIn) and other content in a public way.

The information that is shared on these social networks, as well as the relationships that are established, are a very valuable source of information for investigators, for social, commercial, and even police fields. A typical problem for these analysts is that they often do not have the computer skills to extract information to analyse, or if they can do so, it takes a long time.

For this reason, this work has the objective to make a practical tool to obtain public information contained in social media profiles. To obtain this information, has been used a technique called scraping. Scraping is a procedure that extracts information from a website by accessing it as if it were a browser, transforming the HTML code into a practical storage format for its study.

The tool that has been developed is a Chrome extension that uses scraping programs to obtain profile data from the social networks of Twitter, LinkedIn and Facebook. It has been thought in this way to adapt as much as possible to the usual workflow of analysts, who normally make their tasks using this browser. The scraping tools are developed on specific servers in the cloud. This makes that the user do not have to install the scraping program when installing the extension.

The final objective is using the data obtained from scraping to be used in more advanced social network analysis (SNA) tools. The SNA is the process of investigating social structures through the use of networks and graph theory.

Keywords

Twitter, LinkedIn, Facebook, scraping, scraper.

Agradecimientos

A mis padres, a mi hermano, a mi pareja y toda mi familia, que gracias a ellos soy quien soy y a los que le debo expresar mi mayor agradecimiento por apoyarme a lo largo de esta etapa académica. También deseo agradecer a mis compañeros por estar ahí luchando codo con codo para salir adelante, de los cuales algunos se han convertido en buenos amigos que deseo conservar siempre. Y por último, pero no menos importante, al profesor tutor de este TFG, por haber sido mi guía e inspiración para lograr la meta final de mi carrera profesional.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

El gran uso de las redes sociales ha obtenido mucha importancia a nivel social por lo que crea un gran interés en el análisis de datos en ocasiones para negocios, marketing, servicio al cliente, gestión de reputación y otros. La cantidad de información que comprende es enorme por lo que necesita ser automatizada para simplificar la tarea a los usuarios.

En el caso de este trabajo, el tutor del mismo actuó como usuario, estableciendo los requisitos que debía cumplir el sistema desarrollado. En particular, solicitó el desarrollo de una herramienta práctica para obtener información detalla de perfiles de distintas redes sociales desde una extensión de Chrome. Al usar una extensión de Chrome, se consigue que sea una implementación práctica ya que se encuentra directamente en el navegador y es sencilla de usar.

El análisis de redes sociales es la ciencia de la extracción de datos de estas redes, para después usarlos para dar un seguimiento y un estudio del mismo. Hay muchos tipos de análisis que podrían ser interesantes realizar a partir de estos datos. Este TFG persigue facilitar estos datos. Una vez conseguidos los datos, se pueden realizar análisis de estructuras sociales por medio de redes y teoría de grafos. Ejemplos podrían ser redes de amigos o de seguidores, redes de conocimiento, redes laborales, etc. Otro posible análisis es el respecto a un tema, podría realizarse por ejemplo a un tema comentado en Twitter.

Concluyendo, por los motivos descritos anteriormente, se ha decidido llevar a cabo una extensión de Chrome la cual utiliza herramientas de *Web Scraping* para obtener los datos de perfiles de redes sociales.

1.2 Objetivos

El objetivo es crear una extensión (o también llamado plugin) de Chrome capaz de realizar *scraping* sobre un perfil de las redes sociales de Twitter, LinkedIn y Facebook. Estas redes sociales fueron elegidas como parte de los requisitos impuestos por el cliente.

Se pretende:

- Realizar una extensión de Chrome que solicite peticiones a un servidor para que este le devuelva los datos del perfil de la red social que se precise. Una vez tenga esta información, que la descargue en el navegador en formato CSV.

- Realizar un servidor capaz de atender las peticiones provenientes del plugin y retornar la información que le solicita de un perfil.

- Basarse en herramientas de código abierto existentes, que lleven a cabo la tarea de *scraping* para obtener la información deseada de un perfil, para Twitter, LinkedIn y Facebook y con ello, usarlas sobre el servidor comentado en el punto anterior.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1**: es esta introducción.

- **Capítulo 2:** se realizará una exposición del estado del arte, con el objetivo de dar a conocer los conceptos básicos del proyecto como también las herramientas y librerías que se han estudiado y usado.
- **Capítulo 3:** se explicará en detalle el diseño inicial y final del proyecto
- **Capítulo 4:** se encuentra el desarrollo del trabajo y la programación tanto de los servidores como de la extensión de Chrome.
- **Capítulo 5:** se mostrará una serie de pruebas que demuestren el correcto funcionamiento de la herramienta desarrollada en este TFG.
- **Capítulo 6:** se incluyen las conclusiones del proyecto y posibles ideas de trabajo a futuro.

2 Estado del arte

2.1 Web Scraping

2.1.1 ¿Qué es?

Se trata de una técnica que consiste en la extracción de información de interés en páginas web, para ser manipulada o analizada posteriormente. Los programas que utilizan esta técnica simulan la navegación usando el protocolo HTTP o introduciendo un navegador en una aplicación.

El *Web Scraping* se relaciona con la indexación de la información mediante un robot, técnica utilizada por la mayoría los buscadores como por ejemplo Google, que indexa las páginas web de toda Internet para dar con el resultado de la búsqueda realizada. Nuestro objetivo utiliza el *Web Scraping* para la transformación de datos en formato HTML a datos que podrían ser almacenados en hojas de cálculo, base de datos u otro método de almacenamiento.

Los usos que se le puede dar al *Web Scraping* son diversos: extraer datos y analizarlos, añadirlos a base de datos, realizar migraciones webs, reunir y dar información dispersa de distintas webs. En aplicaciones prácticas puede ser utilizado para marketing de contenidos, ganar visibilidad en redes sociales, motorización de precios de la competencia, etc. [1] [2] [3]

2.1.2 Problemas de Web *Scraping*

Como se indica en la referencia [1] con respecto al ámbito legal y posibles problemas del *Web Scraping* podemos considerar los siguiente:

“Cosas a tener en cuenta antes de empezar un proyecto de *Web Scraping* son: la legalidad vigente en el país de origen, las condiciones legales del sitio web, si se podría llegar a violar los derechos de autor, de propiedad intelectual o de uso de marca registrada, si el usar los datos podría considerarse como competencia desleal (imitación, publicidad comparativa, denigración de la reputación u otras lesiones de derechos, violación de secretos, ...), derecho ‘sui generis‘, etc.

Si no se va a hacer un uso público de los datos scrapeados, es decir, es solo para ‘consumo propio’, scrapear un sitio web será tan legal como pueda serlo acceder a sus datos con un navegador web.”

Otro problema que puede conllevar el uso de *Web Scraping*, es que precise de un inicio de sesión o cookies de sesión para acceder. Y, por último, los bloqueos de IP y/o resolución de un CAPTCHA. Esto puede ocurrir como consecuencia de no acceder a la web de forma manual, solicitar diversas páginas por minuto, usar un navegador no identificado, entre otros motivos.

2.1.3 Herramientas útiles para Web *Scraping*

A continuación, se lista una serie de herramientas que son frecuentemente utilizadas por aplicaciones que realizan *Web Scraping*:

- Beautiful Soup

Se trata de una biblioteca para extraer datos de archivos HTML y XML para analizarlos. Crea un árbol con todos los elementos del documento y con el ello, se puede extraer la información útil para el *Web Scraping*. [4]

- Puppeteer

Es una biblioteca de nodos que proporciona una API de alto nivel para controlar Chrome a través del protocolo DevTools. Entre las posibles cosas que se pueden hacer con esta biblioteca es realizar *scraping* de páginas web. También se puede conseguir capturas de pantalla y archivos PDF de páginas, automatizar envío de formularios, test en UI, etc. [5]

- Selenium

Es un entorno de pruebas de software para aplicaciones fundamentadas en la web. Permite grabar, editar y depurar los casos de prueba que posteriormente pueden ser ejecutados de manera automática. Incluye facilidades para pruebas en diversos lenguajes como Java, C#, Groovy, PHP, Ruby, Python y Perl. [6]

- ChromeDriver:

Es una herramienta para realizar pruebas automáticas en aplicaciones web en diversos navegadores. Se trata de un ejecutable que Selenium usa para controlar Chrome. Permite navegar a páginas web, proporcionar input a los usuarios, ejecuciones de JavaScript, etc. [7]

- Scrapy:

Es un Framework web programado en Python que se usa para rastrear una web y extraer sus datos eficazmente. [8]

2.2 Interfaz de programación de aplicaciones (API)

Una API consiste en un grupo de procedimientos y protocolos de comunicación que permiten obtener datos de una aplicación. Muchas plataformas de red social ofrecen una API para poder, a través de programación, descargar cierta información disponible para sus usuarios.

2.2.1 Web *Scraping* vs API

Las APIs tienen diversas diferencias con el *Web Scraping*, una de ellas es que las APIs dan acceso directo a los datos que deseas, es una herramienta que ofrece directamente la aplicación en cuestión. La información que te ofrece la API puede estar muy restringida y/o también ser costosa. Puede ocurrir que, por ejemplo, desees extraer información de Amazon, pero éste no proporciona API, entonces el *Web Scraping* es la alternativa para poder acceder a estos datos, siempre que estén disponibles en la web. [9]

2.3 Herramientas para extracción de datos en Redes Sociales.

A continuación, se va a listar una serie de herramientas existentes en Internet que pueden servir para realizar la extracción de datos de las redes sociales. Dicha extracción puede ser realizada ya sea por *Web Scraping* o a partir de las APIs proporcionadas por la red social.

2.3.1 Twitter:

- **Twint.**

Descripción: Herramienta de *scraping* avanzada implementada en Python que permite obtener Tweets e información relacionada a un perfil, sin usar la API de Twitter ni autenticación.
Esta información la puede dar en diferentes formatos, tanto para JSON, CSV, como Pandas. [10]
Herramientas de interés: Beautifulsoup.

- **Twitter*scraper*.**

Descripción: Scraper implementado en Python que usa la API de Twitter para obtener Tweets e información relacionada. [11]
Herramientas de interés: API Twitter, Beautifulsoup.

- **Twitter-*scraper*.**

Descripción: Scraper implementado en Python. Es capaz de obtener Tweets e información del perfil. [12]

- **Twitter-php-*scraper***.

Descripción: Scraper implementado en PHP, obtiene Tweets y también información del perfil. [13]

2.3.2 LinkedIn:

- **Scrapedin**

Descripción: Scraper de perfiles de LinkedIn desarrollado en JavaScript y Node.js. Extrae el perfil personal, experiencias, educación, recomendaciones, perfiles relacionados, experiencias de voluntariado, habilidades y logros de un usuario. Necesita iniciar sesión ya sea por correo y contraseña o por cookies de sesión. [14]
Herramientas de interés: Puppeteer
.

- **Scrape-LinkedIn-selenium**

Descripción: Scraper que extrae todos los detalles de los perfiles públicos de LinkedIn y los convierte a formato JSON. También puede realizar *scraping* sobre compañías. Necesita cookies de sesión para iniciar sesión. [15]
Herramientas de interés: Beautifulsoup, Selenium.

- **Scrape-LinkedIn**

Descripción: Scraper de perfiles de LinkedIn en Python, que puede ser usado para transformar el HTML de LinkedIn a una estructura JSON. [16]

- **LinkedIn-profile-*scraper***

Descripción: Scraper de perfiles de LinkedIn. Puede devolver la información en formato JSON. Los parámetros que es capaz de extraer son: perfil personal, experiencias, educación, experiencias de voluntariado y habilidades. Necesita cookies de sesión para poder iniciar sesión y realizar el *scraping*. [17]
Herramientas de interés: Puppeteer.

- **LinkedIn-*scraper***

Descripción: Realiza *scraping* de perfiles de LinkedIn, además de dar la posibilidad de realizar un análisis de los resultados. Necesita iniciar sesión por cookies o por correo y contraseña. [18]
Herramientas de interés: Beautifulsoup.

- **Python-LinkedIn**

Descripción: Interfaz de Python para la API de LinkedIn. La herramienta proporciona una interfaz para las API REST de perfiles, grupos, empresas, trabajos, búsquedas y shares en LinkedIn. [19]
Herramientas de interés: API LinkedIn.

2.3.3 Facebook:

- **Ultimate-Facebook-Scraper**

Descripción: Se trata de una herramienta capaz de realizar *scraping* de casi todo lo relacionado a un perfil de Facebook, como las fotos, videos, lista de amigos, posts, etc. [20]
Herramientas de interés: Selenium, ChromeDriver.

- **Facebook-friends-map**

Descripción: Crea un mapa de los amigos que posee un perfil en Facebook. Incluye también un listado de los amigos en formato JSON junto con su información personal y localización. [21]
Herramientas de interés: Selenium, ChromeDriver.

- **Data-Mining-on-Social-Media**

Descripción: Extrae los posts públicos de usuarios en Facebook y los guarda en una base de datos de PostgreSQL. [22]

- **Facebook-miner**

Descripción: Extrae posts y comentarios de una página usando la API Graph de Facebook en Python. [23]
Herramientas de interés: API Facebook.

- **Fbcrawl**

Descripción: Realiza *scraping* sobre posts y comentarios de una página usando la herramienta Scrapy. [24]
Herramientas de interés: Scrapy.

- **Facebook-post-*scraper***

Descripción: Realiza *scraping* sobre los posts públicos de páginas de Facebook sin usar la API. [25]
Herramientas de interés: Selenium, Beautifulsoup.

- **Facebook-*scraper***

Descripción: Realiza *scraping* sobre los posts públicos de páginas y grupos de Facebook sin usar la API. [26]
Herramientas de interés: Beautifulsoup.

- **Facebook-multi-*scraper***

Descripción: Realiza *scraping* múltiple sobre diferentes páginas en masa para conseguir sus posts y videos. [27]

- **Fb_*scraper***

Descripción: Realiza *scraping* sobre los posts de páginas y grupos de Facebook sin usar la API. [28]

- **Facebook-Page-Crawler**

Descripción: Extrae información sobre los post, comentarios y reacciones de páginas usando la API de Facebook. [29]
Herramientas de interés: API Facebook.

2.3.4 Herramientas seleccionadas para el proyecto

A continuación, se va a explicar para cada una de las redes sociales cuales han sido las herramientas de *scraping* elegidas para el proyecto:

- Twitter.

Para Twitter se ha decidido usar "Twint" [10] porque es una herramienta completa, que incluye diferentes funcionalidades útiles y permite configurarlas, por ejemplo, para realizar *scraping* sobre los tweets de un perfil permite indicar límite de tweets y entre que fechas se desean. Esto último, es muy útil ya que permite tener un mayor control del *scraping*. Y, por otro lado, también ha sido elegida porque es una herramienta que funciona y que está mantenida activamente por un grupo grande de desarrolladores. Los datos de interés para realizar un análisis de redes sociales que se pueden extraer con esta herramienta han sido los tweets, seguidores, personas a las que sigue y tweets a los que le ha dado "Like".

- LinkedIn.

Para LinkedIn ha sido un poco más complicado conseguir una herramienta a la altura de "Twint", que fuese igual de completa y que funcionara. Se ha escogido la herramienta "Scrapedin" [14] que ofrece campos interesantes para el posible análisis del que se ha hablado, por ejemplo, los estudios cursados, la experiencia laboral, la gente relacionada y por último, las recomendaciones recibidas y dadas. Todos estos campos serán explicados en detalle a lo largo de la memoria.

- Facebook.

Facebook a diferencia de las otras dos redes sociales tiene una Política de Términos y Condiciones de Uso más restrictiva para la realización del *Web Scraping*. Esto conlleva a una menor cantidad de herramientas que realicen esta tarea, lo cual limita la elección de un *scraper* útil. Finalmente se decidió usar el *scraper* "Ultimate-Facebook-Scraper" [20], consiguiendo un listado de amigos de Facebook de un usuario. Este *scraper* ha sido la mejor opción a elegir, el resto que han sido contemplados o no daban la información útil para poder realizar un análisis de redes sociales o se encontraban sin funcionar y sin soporte.

3 Diseño

3.1 Esquema funcional

Este es el esquema funcional del proyecto, donde indica qué componentes hay y cómo interaccionan entre ellos.

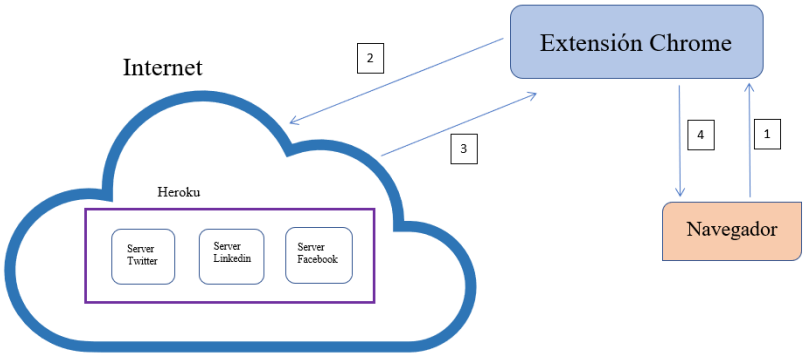


Figura 1. Diseño inicial del proyecto.

A continuación, se va a explicar brevemente el flujo de control y datos del programa indicado en el esquema:

1. **Indicar perfil y red social**: En primer lugar, el usuario accede con su navegador al perfil de una red social (Twitter, LinkedIn o Facebook). Desde ahí la extensión está preparada para identificar qué red social es y extraer el nombre de usuario para su *scraping*.
2. **Solicitar *scraping***: Una vez la extensión tiene el nombre de usuario, solicita realizar el *scraping* de éste al servidor de la red social correspondiente en Heroku.
3. **Datos extraídos**: El servidor retorna a la extensión el resultado del *scraping*.
4. **Descarga**: La extensión recibe este resultado y lo descarga en el navegador, donde el usuario puede acceder a ellos directamente.

El esquema consta de 3 elementos:

- **Heroku**: Ha sido la plataforma utilizada para tener servidores en remoto, encargados de cumplir con las solicitudes provenientes del plugin. Sobre él se encuentran los servidores *scraping* de Twitter, LinkedIn y Facebook.
- **Navegador Chrome**: Es la plataforma encargada de acceder a Internet e indicar el perfil sobre el que se desea realizar *scraping* a la extensión Chrome desde la URL.
- **Extensión de Chrome**: Es la encargada de mostrar al cliente una interfaz de usuario por la que puede solicitar la realización de *scraping* de las redes sociales de Twitter, LinkedIn o Facebook. Una vez que obtiene esos datos a través de los servidores, la descarga en ficheros en el navegador.

Una vez comenzada la fase de implementación, se comprobó que el *scraper* disponible para Facebook no podía ser instalado en el servidor, si no que debía ejecutar en el ordenador cliente (el localhost de la extensión Chrome). Esto se debe a que no se puede implementar un servidor remoto para Facebook en Heroku debido a políticas de seguridad,

esto se detallará en el capítulo de desarrollo. El resultado final es que el diseño debió ser modificado, siendo la versión final la que se detalla en la figura 2.

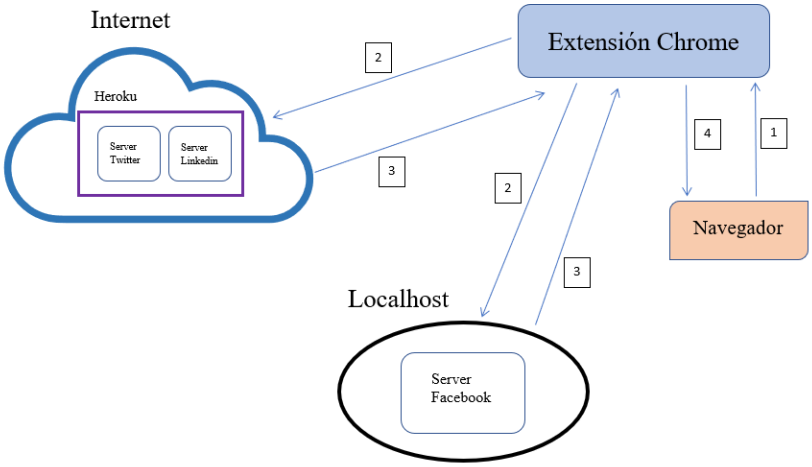


Figura 2. Diseño final del proyecto.

4 Desarrollo

En este apartado se va a hablar sobre el desarrollo que ha llevado a obtener esta extensión Chrome funcional. Primero se realizará una breve introducción de cómo se ha decidido desarrollar los servidores *scraping*. Y luego, se explicará la programación tanto de los servidores como de la propia extensión Chrome en sí.

4.1 Introducción a los servidores scraping

Los servidores son los encargados de realizar la tarea de *scraping* bajo las solicitudes provenientes de la extensión. La extensión solicita por mecanismos de GET y POST tanto el servicio de realizar el *scraping* de la red social como la obtención de la información de dicho *scraping*.
Dos de los servidores se ha desarrollado en la plataforma de Heroku para las redes sociales de Twitter y LinkedIn, mientras que Facebook ha sido implementado para funcionar en localhost.

- Servidores de Heroku:
  Heroku es una plataforma que ofrece servicio de computación en la nube para diferentes lenguajes. Ha sido utilizado para cumplir la función de servidor remoto que se encargue de realizar el *scraping* de la red social bajo la petición del plugin.

  Dado que los servidores de *scraping* de Twitter y LinkedIn están programados en diferentes lenguajes, se decidió realizar un servidor para cada uno. El motivo por el que los servidores estén en diferentes lenguajes se debe a que se usa el mismo lenguaje de programación que emplea el *scraper* escogido.

  El hecho de que sean servidores remotos permite que el usuario no tenga que instalar las herramientas necesarias para realizar el *scraping*. Solo con descargar la extensión, se podría usar directamente el *scraping* sobre Twitter y LinkedIn.

- Localhost:
  Al probar el servidor de *scraping* de Facebook en Heroku se llegó a la conclusión de que no podía ser incorporado a la nube como los demás. El motivo es que al intentar llevar el servidor a Heroku y realizar login en Facebook desde un lugar remoto que no es tu zona habitual, Facebook enseguida lo detecta, y por políticas de seguridad puede bloquear la cuenta. De hecho, ocurrió cuando se usó una cuenta de prueba. Por ello, quedó descartado usar el servidor de Facebook en remoto. Si el usuario está interesado en realizar *scraping* de Facebook debe instalar lo necesario para correr el servidor en local.

Por último, comentar que cada uno de los servidores tienen implementado una especie de caché para guardar todos los *scraping*s que se realicen sobre un perfil. Se guarda todos los *scraping*s que se han realizado en una carpeta llamada “stored” o “data” de forma organizada y a su vez en subcarpetas por nombres y fechas. El objetivo de esto es que más adelante se podría implementar la opción de descargar ficheros antiguos de *scraping* de un usuario en concreto. Para Twitter y Facebook siempre se crea los ficheros de nuevo con su nombre y fecha, sin embargo, LinkedIn, se ha pensado que, si tiene una antigüedad menor

una semana, no se realice el *scraping* nuevamente. Esto último, referente a LinkedIn se decidió así, porque al tratarse del *scraping* de un "currículum" no es algo que se suela cambiar demasiado, y se evita el tiempo de espera de volver a realizar el *scraping*. Estos ficheros guardados tienen extensión JSON.

4.2 Desarrollo de los servidores scraping

4.2.1 Servidor *Scraping* Twitter

El servidor de Twitter utiliza la herramienta de *scraping* Twint [10]. Se trata de una herramienta desarrollada en Python, por lo que se decidió desarrollar el servidor sobre Flask ya que permite crear aplicaciones web rápidamente y de manera sencilla.

Gracias a la herramienta de Twint, el servidor de Twitter permite atender las solicitudes de *scraping* sobre un perfil. A continuación, se va a explicar en detalle la programación de cada uno de los métodos que se encarga de realizar cada *scraping* en el servidor y de retornar el fichero JSON que el corresponde:

4.2.1.1 Obtención de tweets.

Para conseguir los tweets de un usuario se ha desarrollado la siguiente función de tipo GET de HTTP.

```
@app.route('/twitter/get_tweets_json/<string:username>/<int:limit>/<string:since>/<string:until>')
def get_tweets_json(username, limit, since, until):
    now, nombreCarpeta = crearCarpeta()
    counter = 0
    while(counter < 3):
        c = twint.Config()
        c.Username = username
        # Fijar parametros delimitadores
        c.Limit = limit
        if since != "null":
            c.Since = since
        if until != "null":
            c.Until = until
        c.Custom["tweet"] = ["date", "time", "timezone", "username", "tweet"]
        c.Output = nombreCarpeta + "/get_tweets_json" + username + now.strftime("_%d-%m-%Y_%H:%M:%S") + ".json"
        c.Store_json = True
        twint.run.Search(c)
        counter += 1
        try:
            data = [json.loads(line) for line in open(c.Output, 'r')]
        except FileNotFoundError:
            print("No existe el archivo el archivo " + c.Output)
        else:
            return jsonify(data)
    if counter == 3:
        return "Error_twint"
```

Figura 3. Código para obtener los tweets en servidor scraping Twitter.

En la ruta podemos observar que se indica el "username" que sería el usuario sobre el que realizar este *scraping*. Por otro lado, tenemos también "limit" donde se puede indicar el límite de tweets que se quieren descargar y, por último, "since" y "until" que son las fechas de "desde" y "hasta" que queremos que los tweets tengan.

La función lo que hace es ejecutar la funcionalidad de Twint de "twint.run.Search" que se encarga de buscar tweets. Para ello, se ajusta la configuración de la búsqueda en base a los parámetros de la ruta, luego se indica que campos del tweet nos interesan, en nuestro caso nos hemos quedado con "date", "time", "timezone", "username" y "tweet" (donde este último es el tweet en sí), también se asigna un nombre al fichero donde se guardarán los

tweets y en que formato los queremos. En nuestro caso es JSON. Una vez que ese fichero JSON esté almacenado, se lee y se retorna.

4.2.1.2 Obtención de seguidores.

La función de los seguidores es la siguiente.

```
@app.route('/twitter/get_followers_json/<string:username>/<int:limit>')
def get_followers_json(username, limit):
    now, nombreCarpeta = crearCarpeta()
    counter = 0
    while(counter < 3):
        c = twint.Config()
        c.Username = username
        if limit != 0:
            c.Limit = limit
        c.Output = nombreCarpeta + "/get_followers_json" + username + now.strftime("_%d-%m-%Y_%H:%M:%S") + ".json"
        c.Store_json = True
        twint.run.Followers(c)
        counter += 1
        try:
            data = [json.loads(line) for line in open(c.Output, 'r')]
        except FileNotFoundError:
            print("No existe el archivo el archivo " + c.Output)
        else:
            return jsonify(data)
    if counter == 3:
        return "Error_twint"
```

Figura 4. Código para obtener los seguidores en el servidor scraping Twitter.

Se trata de otro método GET, el cual se le tiene que indicar el usuario sobre el que realizar el *scraping* y el límite de seguidores. Se decidió indicar un límite de seguidores para evitar largos periodos de espera innecesarios si el cliente no le interesa tener todos los seguidores, o por simplemente para agilizar las pruebas. En esta ocasión se llama a otra funcionalidad diferente de Twint, la “twint.run.Followers”, para obtener los seguidores de dicho usuario.

4.2.1.3 Obtener personas a las que sigue.

Esta es la función que se encarga de obtener las personas a las que sigue:

```
@app.route('/twitter/get_following_json/<string:username>/<int:limit>')
def get_following_json(username, limit):
    now, nombreCarpeta = crearCarpeta()
    counter = 0
    while(counter < 3):
        c = twint.Config()
        c.Username = username
        if limit != 0:
            c.Limit = limit
        c.Output = nombreCarpeta + "/get_following_json" + username + now.strftime("_%d-%m-%Y_%H:%M:%S") + ".json"
        c.Store_json = True
        twint.run.Following(c)
        counter += 1
        try:
            data = [json.loads(line) for line in open(c.Output, 'r')]
        except FileNotFoundError:
            print("No existe el archivo el archivo " + c.Output)
        else:
            return jsonify(data)
    if counter == 3:
        return "Error_twint"
```

Figura 5. Código obtención de personas a las que sigue en servidor scraping Twitter.

Sigue la misma idea que la función anterior, pero cambiando la funcionalidad a “twint.run.Following”.

4.2.1.4 Obtención de información personal.

Esta función obtiene la información personal del usuario, como su biografía, número de seguidores y a los que sigue (following), localización, nombre y número de tweets.

```python
@app.route('/twitter/get_user_info_json/<string:username>')
def get_user_info_json(username):
    now, nombreCarpeta = crearCarpeta()
    counter = 0
    while(counter < 3):
        c = twint.Config()
        c.Username = username
        c.Custom["user"] = ["id", "username", "bio", "followers", "following", "location", "name", "tweets"]
        c.Output = nombreCarpeta + "/get_user_info_json" + username + now.strftime("_%d-%m-%Y_%H:%M:%S") + ".json"
        c.Store_json = True
        twint.run.Lookup(c)
        counter += 1
        try:
            data = [json.loads(line) for line in open(c.Output, 'r')]
        except FileNotFoundError:
            print("No existe el archivo el archivo " + c.Output)
        else:
            return jsonify(data)

    if counter == 3:
        return "Error_twint"
```

Figura 6. Código que obtiene la información personal en servidor scraping Twitter.

En la ruta solo es necesario indicar el perfil. Y llamando a la funcionalidad "twint.run.Lookup" se obtiene la información personal del usuario.

4.2.1.5 Obtención de tweets a los que le ha dado "Like".

```python
@app.route('/twitter/get_tweets_fav_json/<string:username>/<int:limit>')
def get_tweets_fav_json(username, limit):
    now, nombreCarpeta = crearCarpeta()
    counter = 0
    while(counter < 3):
        c = twint.Config()
        c.Username = username
        c.Limit = limit
        c.Custom["tweet"] = ["date", "time", "timezone", "username", "tweet"]
        c.Output = nombreCarpeta + "/get_tweets_fav_" + username + now.strftime("_%d-%m-%Y_%H:%M:%S") + ".json"
        c.Store_json = True
        twint.run.Favorites(c)
        counter += 1
        try:
            data = [json.loads(line) for line in open(c.Output, 'r')]
        except FileNotFoundError:
            print("No existe el archivo el archivo " + c.Output)
        else:
            return jsonify(data)

    if counter == 3:
        return "Error_twint"
```

Figura 7. Código obtienes los tweets favoritos en el servidor scraping Twitter.

La implementación es bastante parecida a la que obtenía los tweets del usuario, a diferencia de que solo se ha configurado para que se le pueda limitar el número de tweets a recolectar. Esta función está implementada y funciona perfectamente. Sin embargo, debido a unos cambios que ha hecho la plataforma Twitter el día 1 de junio, se ha visto afectado al correcto funcionamiento de la funcionalidad de "twint.run.Favorites". Es un problema sobre el que están trabajando y es cuestión de tiempo de que logren arreglarlo ya que como se ha comentado, es una herramienta que tiene soporte activamente. A pesar de que no funcione por motivos externos, la implementación de esta función es correcta y en cuanto este arreglada la parte de Twint referente a esto se podrá usar este código.

4.2.2 Servidor *scraping* de LinkedIn

El servidor de LinkedIn se hizo con la herramienta de *scraping* llamada Scrapedin [14] hecha sobre Node.js, por lo cual el servidor web se ha desarrollado sobre este mismo entorno.

Para conseguir este servidor Node.js, primero se ha creado el package.json, el cual se encarga de controlar los paquetes que instalamos para el proyecto. Ese fichero es el siguiente:

```
{
  "name": "server_node_scrap",
  "version": "1.0.0",
  "description": "Proyecto scrap",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Laura De Abreu",
  "license": "La mia",
  "dependencies": {
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "scrapedin": "https://github.com/linkedtales/scrapedin",
    "xmlhttprequest": "^1.8.0"
  }
}
```

Figura 8. Package.json del servidor scraping de LinkedIn.

Lo importante de este fichero es el apartado de "dependencies", se puede observar que está incluido el github de la herramienta scrapedin. Y también que está utilizando el framework de "express", el cual es la base para poder crear la aplicación del servidor.

Una vez creado ese package.json, creamos el fichero de app.js que va a tener el código del servidor. A continuación, se explica cada uno de los métodos que forman el app.js.

4.2.2.1 Inicio de sesión

El servidor *scraping* de LinkedIn cuenta con una función POST que realiza el inicio de sesión:

```
app.post('/login',(req,res) => {

    var dict = {
      "email": req.body.email,
      "pass": req.body.pass
    }
    var dictstring = JSON.stringify(dict);
    fs.writeFileSync('login.json', dictstring);


});
```

Figura 9. Gestor del login en el servidor scraping LinkedIn.

Lo que hace, es recibir los parámetros POST del body de la petición HTTP y lo escribe sobre un fichero de credenciales llamado "login.json". Las credenciales son el email y la contraseña. Posteriormente, justo antes de solicitar la realización del *scraping* mediante la herramienta Scrapedin se toman estos valores del fichero "login.json" para iniciar sesión.

4.2.2.2 Funcionalidad encargada de realizar el scraping.

Este servidor *scraping* funciona de manera diferente que el de Twitter. Twint tiene una función para cada categoría del perfil de un usuario, por ejemplo, tiene implementada una función que realiza el *scraping* de tweets, otra de los seguidores, etc. Sin embargo, Scrapedin está programado para que te realice todo el *scraping* completo y te devuelve un objeto con toda la información, que es el siguiente:

```
{
  profile: {
    name, headline, location, summary, connections, followers
  },
  positions:[
    { title, company, description, date1, date2,
      roles: [{ title, description, date1, date2 }]
    }
  ],
  educations: [
    { title, degree, date1, date2 }
  ],
  skills: [
    { title, count }
  ],
  recommendations: [
    { user, text }
  ],
  recommendationsCount: {
    received, given
  },
  recommendationsReceived: [
    { user, text }
  ],
  recommendationsGiven: [
    { user, text }
  ],
  accomplishments: [
   { count, title, items }
  ],
  volunteerExperience: {
    title, experience, location, description, date1, date2
  },
  peopleAlsoViewed: [
    { user, text }
  ]
}
```

Figura 10. Objeto resultante del scraping de la herramienta Scrapedin.

Entonces, se decidió realizar este método GET que se encarga de hacer el *scraping* y a su vez separar cada uno de estos campos en ficheros JSON. Dado que se hace referencia frecuentemente a este método, vamos a ponerle el nombre "general" en base a la ruta que usa.

```
app.get('/linkedin/general/:name', (req, res) => {
  var usuario = req.params.name;
  var url = 'https://www.linkedin.com/in/' + usuario + '/';

  if (checkLastScrap(usuario)){
    console.log('El ultimo scrap esta obsoleto o nuevo usuario');
    var carpeta = './data/' + usuario + '/' + current_date_string ;
    console.log("carpeta "+ carpeta)

    realizarScrap(url, res).then(function(profile) {
      fs.mkdirSync(carpeta,{recursive: true});
      //Profile
      jsonInfoUsuario(usuario, profile.profile, carpeta)
      //Educations
      jsonEducations(usuario, profile.educations,carpeta);
      //Positions (Experiencia laboral)
      jsonPositions(usuario, profile.positions, carpeta)
      //Gente relacionada
      jsonPeopleAlsoViewed(usuario, profile.peopleAlsoViewed, carpeta)
      //Recomendaciones
      jsonRecommendations(usuario, profile.recommendations, carpeta)
      res.end("Scrap correcto")

    }).catch(function (reason) {
      console.log(reason); // Error!
      if (reason.message.includes("login")) {
        res.end("Login error")
      }else{
        res.end("Scrap erroneo")
      }
    });


  }
  else{
    console.log('Scrap correcto existente');
    res.end("Scrap correcto existente");
  }
```

Figura 11. Código encargado de gestionar el scraping en el servidor scraping de LinkedIn.

Primero se puede ver que por la ruta se ha podido obtener el nombre del usuario al que se le va a realizar el *scraping*. Seguidamente vemos una función llamada "checkLastScrap" que verifica si ya ha pasado una semana desde la última vez que se le realizó un *scraping* sobre este perfil. Solo volverá a realizarse el *scraping* en caso de que haya pasado más de una semana desde el último *scraping* realizado. La imagen siguiente es el código de dicha función:

```
//Comprueba que haya pasado al menos una semana del ultimo scrap sobre ese usuario
function checkLastScrap(usuario){

  //Comprobar si no existe la carpeta para ese usuario
  //En caso de que no exista, se devuelve true para que cree la carpeta
  var path = './data/' + usuario;
  try {
    if(fs.accessSync(path)) {
      console.log("Existe la carpeta");
    }
  } catch (e) {
    return true;
  }

  //Calculamos la fecha de la siguiente semana, a partir de la ultima fecha scrap
  var date_last_scrap = getLastScrap(path);
  var nextWeek = new Date(date_last_scrap.getTime() + 7 * 24 * 60 * 60 * 1000);

  console.log(nextWeek);
  console.log(current_date);

  return current_date > nextWeek;

}
```

Figura 12. Código de la función CheckLastScrap.

Volvemos al método principal el comienzo, el llamado "general". Una vez verificado de que para ese usuario no hay caché o su caché es más antigua de lo acordada (7 días), se procede a realizar el *scraping*. Para ello, se creó una función llamada "realizarScrap" la cual devuelve el objeto con toda la información del *scraping*.

```
async function realizarScrap(url) {
    console.log('before start');

    let rawdata = fs.readFileSync('login.json');
    let aux = JSON.parse(rawdata);
    console.log(aux);
    var options = {
      email: aux.email,
      password: aux.pass
    }
    console.log(options);

    const profileScraper = await scrapedin(options);
    const profile = await profileScraper(url)
    console.log(profile);

    console.log('after start');
    return profile;

}
```

Figura 13. Código de la función RealizarScrap.

En la función "realizarScrap" primero realiza el login en base a los datos que se encuentran en el fichero "login.json" comentado anteriormente. Seguidamente, llama a la funcionalidad de "Scrapedin" para que lleve a cabo el *scraping*. Una vez finalizado el *scraping*, retorna el objeto con la información.

Volviendo a la Figura 11, se toma el objeto resultante "profile" de la función "realizarScrap", y es procesado para que cada una de las secciones que lo componen generen un fichero JSON. Para ello, hay varias funciones para que creen cada JSON. A continuación, se va a exponer el código de cada una de estas funciones:

- Función "jsonInfoUsuario", crea el JSON para la información personal del usuario.

```
function jsonInfoUsuario(usuario, profile, carpeta){

  var dictstring = JSON.stringify([profile], ['name', 'headline', 'location'])
  var ficheroJSON = "get_info_usuario_" + usuario + "_" + current_date_string;

  fs.writeFile(carpeta + "/" + ficheroJSON +".json", dictstring, function(err, result) {
    if(err) console.log('error', err);
  });
}
```

Figura 14. Código que genera el JSON de la información de usuario.

- Función “jsonEducations”, crea el JSON con la educación.

```
function jsonEducations(usuario, educations, carpeta){

  var campos = ['title', 'degree', 'date1', 'date2']
  var dictstring = convertirAJSONfy(educations, campos);
  var ficheroJSON = "get_educations_" + usuario + "_" + current_date_string;
  var adrr = carpeta + "/" + ficheroJSON +".json";

  fs.writeFile(adrr, dictstring, function(err, result) {
    if(err) console.log('error', err);
  });

}
```

Figura 15. Código que genera el JSON de la educación del usuario.

- Función “jsonPositions”, crea el JSON con los diferentes trabajos ejercidos por usuario.

```
function jsonPositions(usuario, positions, carpeta){

  var campos = ['title', 'companyName', 'date1', 'date2'];
  var dictstring = convertirAJSONfy(positions, campos);
  var ficheroJSON = "get_positions_" + usuario + "_" + current_date_string;
  var adrr = carpeta + "/" + ficheroJSON +".json";

  fs.writeFile(adrr, dictstring, function(err, result) {
    if(err) console.log('error', err);
  });

}
```

Figura 16. Código que genera el JSON de la experiencia laboral del usuario.

- Función “jsonPeopleAlsoViewed”, genera un JSON con la información de personas que están relacionadas con ese perfil.

```
function jsonPeopleAlsoViewed(usuario, peopleAlsoViewed, carpeta){
  var campos = ['user', 'text'];
  var dictstring = convertirAJSONfy(peopleAlsoViewed, campos);
  var ficheroJSON = "get_peopleAlsoViewed_" + usuario + "_" + current_date_string;
  var adrr = carpeta + "/" + ficheroJSON +".json";

  fs.writeFile(adrr, dictstring, function(err, result) {
    if(err) console.log('error', err);
  });
}
```

Figura 17. Código que genera el JSON de las personas relacionadas con el usuario

- Función “jsonRecommendations, genera dos JSON, uno con la información de las recomendaciones que ha dado el usuario, y otro JSON con la información de las recomendaciones recibidas para usuario.

```
function jsonRecommendations(usuario, recommendations, carpeta){

  var campos = ['user', 'text'];
  var dictstring_given = convertirAJSONfy(recommendations.given, campos);
  var ficheroJSON = "get_recommendations_given_" + usuario + "_" + current_date_string;
  var adrr = carpeta + "/" + ficheroJSON +".json";

  fs.writeFile(adrr, dictstring_given, function(err, result) {
    if(err) console.log('error', err);
  });

  var dictstring_received =  convertirAJSONfy(recommendations.received, campos)
  var ficheroJSON = "get_recommendations_received_" + usuario + "_" + current_date_string;
  var adrr = carpeta + "/" + ficheroJSON +".json";

  fs.writeFile(adrr, dictstring_received, function(err, result) {
    if(err) console.log('error', err);
  });

}
```

Figura 18. Código que genera los JSON de las recomendaciones.

Y lo último a comentar sobre la función “general”, es su control de errores. Tiene en cuenta si ha sido incorrecto el inicio de sesión y también si ha habido algún problema al realizar *scraping*, en cualquier caso, devuelve un mensaje indicando el problema para notificar a la extensión. También retorna si se ha realizado correctamente el *scraping* si es el caso.

4.2.2.3 Funciones que retornan el resultado del scraping

A continuación, vamos a hablar de las funciones encargadas de retornar estos JSON creados que se acaban de explicar. Para cada JSON hay un método que maneja la solicitud de tipo GET que retorna dicho fichero. La extensión Chrome será el haga cada una de las solicitudes para reunir todos los ficheros. Las funciones son las siguientes:

- Para la información del usuario

```
app.get('/linkedin/info_usuario/:name', (req, res) => {
  var usuario = req.params.name;
  var ficheroJSON = "get_info_usuario_" + usuario + "_" + getLastScrap_string('./data/' + usuario);
  returnJSON(usuario, ficheroJSON, res);

});
```

Figura 19. Método encargado de devolver el JSON de la información personal del usuario.

- Para la educación:

```
app.get('/linkedin/educations/:name', (req, res) => {

  var usuario = req.params.name;
  var ficheroJSON = "get_educations_" + usuario + "_" + getLastScrap_string('./data/' + usuario);
  returnJSON(usuario, ficheroJSON, res);

})
```

Figura 20. Método encargado de devolver el JSON de la educación del usuario.

- Para la experiencia laboral:

```
app.get('/linkedin/positions/:name', (req, res) => {

  var usuario = req.params.name;
  var ficheroJSON = "get_positions_" + usuario + "_" + getLastScrap_string('./data/' + usuario);
  returnJSON(usuario, ficheroJSON, res);

})
```

Figura 21. Método encargado de devolver el JSON de la experiencia laboral del usuario.

- Para la gente relacionada:

```
app.get('/linkedin/peopleAlsoViewed/:name', (req, res) => {

  var usuario = req.params.name;
  var ficheroJSON = "get_peopleAlsoViewed_" + usuario + "_" + getLastScrap_string('./data/' + usuario);
  returnJSON(usuario, ficheroJSON, res);

})
```

Figura 22. Método encargado de devolver el JSON de la gente relacionada con el usuario.

- Para las recomendaciones dadas:

```
app.get('/linkedin/recommendations_given/:name', (req, res) => {

  var usuario = req.params.name;
  var ficheroJSON = "get_recommendations_given_" + usuario + "_" + getLastScrap_string('./data/' + usuario);
  returnJSON(usuario, ficheroJSON, res);

})
```

Figura 23. Método encargado de devolver el JSON de las recomendaciones dadas por el usuario.

- Para las recomendaciones recibidas:

```
app.get('/linkedin/recommendations_received/:name', (req, res) => {

  var usuario = req.params.name;
  var ficheroJSON = "get_recommendations_received_" + usuario + "_" + getLastScrap_string('./data/' + usuario);
  returnJSON(usuario, ficheroJSON, res);

})
```

Figura 24. Método encargado de devolver el JSON de las recomendaciones recibidas del usuario.

En todas estas funciones se llama a la función "returnJSON" que se encarga de tomar JSON desde la carpeta donde está almacenado y retornarlo:

```
function returnJSON(usuario, ficheroJSON, res){
  var carpeta = './data/' + usuario + "/" + getLastScrap_string('./data/' + usuario) ;
  var adrr = carpeta + '/' + ficheroJSON +'.json'
  console.log(adrr)

  fs.readFile(adrr, 'utf8', (err, jsonString) => {
    if (err) {
        console.log("Error reading file from disk:", err)
        return
    }
    try {
        const data = JSON.parse(jsonString)
        res.json(data)

  } catch(err) {
        console.log('Error parsing JSON string:', err)
    }
  })

}
```

Figura 25. Función encargada de devolver el JSON.

4.2.3 Servidor *Scraping* Facebook

La herramienta de *scraping* que utiliza es la llamada "Ultimate-Facebook-Scraper" [20], la cual está programada en Python por lo que al igual que Twitter, se ha desarrollado el servidor sobre el framework Flask.

Como se ha comentado anteriormente en la memoria, Facebook tiene una Política de Términos y Condiciones de Uso más restrictiva. Esto implica que las herramientas de *scraping* que hay en la web o bien no son completas o por otro lado no funcionan correctamente en su totalidad porque requieren un mantenimiento constante. Por ello, en base a lo que se ha podido conseguir, se ha decidido que el servidor de Facebook obtenga únicamente los amigos de un usuario, ya que vemos que tiene interés para poder realizar un estudio de análisis de la red de amigos. El resto de los campos que ofrece "Ultimate-Facebook-Scraper", como las fotos, videos, información personal, no ha resultado de interés para añadirla a la implementación del servidor.

Para poder utilizar esta implementación, el usuario debe instalar en su ordenador lo necesario para que el servidor de *scraping* de Facebook funcione. Para ello, se ha creado una guía de instalación para facilitar esta labor al usuario, que se encuentra en el Anexo A.

Para realizar el *scraping* de un perfil como bien indica en su manual se debe indicar las credenciales en un fichero llamado credentials.yaml, para ello, se ha realizado en el servidor una función de login que gestiona este fichero:

```python
@app.route('/facebook/login', methods = ['POST'])
def login():

    email = request.form['email']
    password = request.form['pass']


    with open("credentials.yaml", "r") as ymlfile:
        cfg = yaml.safe_load(stream=ymlfile)
    if email == cfg['email'] and password == cfg['pass']:
        print ("Same login user")
        return "sameLoginUser"

    else:
        dict = {"email": email,
        "pass": password }

        with open('credentials.yaml', 'w') as file:
            yaml.dump(dict, file)
        return "difLoginUser"
```

Figura 26. Método encargado de realizar login en Facebook.

Esta función se encarga de cambiar las credenciales en caso de que sea necesario, a partir de una petición tipo POST. Las credenciales constan del email y la contraseña de Facebook.

4.2.3.1 Scraping de los amigos

La función encargada de realizar el *scraping* es la siguiente:

```python
@app.route('/facebook/get_friends/<string:username>')
def get_friends(username):
    now = datetime.now()

    input_url_user = "https://www.facebook.com/" + username

    file = open("input.txt","w")
    file.write(input_url_user)
    file.close()
    os.system('python scraper/scraper.py --total_scrolls 1')

    nombreCarpeta = "stored/" + username
    try:
        os.makedirs(nombreCarpeta)
    except:
        print("DY")
    ruta_json = nombreCarpeta + "/" + now.strftime("%d-%m-%Y_%H:%M:%S") +'_friends.json'
    text_to_json(username, ruta_json)

    with open(ruta_json) as f:
        data = json.load(f)
        return jsonify(data)
```

Figura 27. Método encargado de realizar scraping y devolver los amigos del usuario en JSON.

Esta función atiende una solicitud de tipo GET, la cual realiza el *scraping* de un perfil y finalmente devuelve un listado de los amigos. La función “text_to_json” transforma el formato .txt que te devuelve el *scraper* a un formato JSON:

```python
def text_to_json(username, ruta_json):
    filename = 'data/' + username + '/All Friends.txt'
    print("USERNAME: " + username )

    location_data = []
    with open(filename) as f:
        for line in f:
            line = line.split(",")
            location_data.append({"url": line[0], "username": line[1]})

    out_file = open(ruta_json, "w")
    json.dump(location_data, out_file)
```

Figura 28. Código de la función Text_to_json.

4.3 Desarrollo de la extensión de Chrome

Una extensión de Chrome es una aplicación que aportan funcionalidad extra al navegador. Nuestra extensión es la encargada de mostrar al cliente una interfaz de usuario por la que puede solicitar la realización de *scraping* a las redes sociales de Twitter, LinkedIn o Facebook. Una vez recoge los datos necesarios (por ejemplo, de que red social se trata, el nombre de perfil, límite de tweets, límite de seguidores, etc.) solicita la realización de *scraping* mediante peticiones GET y POST de HTTP a los servidores que se han explicado anteriormente. Una vez obtiene los ficheros JSON con los resultados del servidor, toma cada uno de ellos y les cambia el formato a CSV, ya que es un estándar para que los usuarios puedan analizar los datos. Una vez se obtiene el fichero CSV, indica al navegador la descargarlos.

Tras explicar cómo funciona nuestra extensión, vamos a ver que ficheros la componen:

4.3.1 Manifest.json.

Es el archivo que contiene la información que define la extensión. La imagen a continuación es nuestro manifiesto:

```
{
  "manifest_version":2,
  "name": "Scrapping Redes Sociales",
  "description" : "Obtener la informacion de un usuario de una red social",
  "version": "1.0",

  "browser_action": {
    "default_icon": "images/hello_extensions.png",
    "default_title": "Scrapping"
  },

  "background": {
    "scripts": ["eventPage.js"],
    "persistent": false
  },

  "permissions":[
    "storage",
    "tabs"]
```

Figura 29. Manifiesto de la extensión de Chrome.

Podemos ver que cuenta con una sección llamada “browser_action” en ella solamente indicamos la imagen que queremos que tenga extensión a la derecha de la barra de herramientas y su título. Sin embargo, en este campo, podríamos indicar por ejemplo el “popup.html” para que fuese nuestra interfaz por defecto, pero nuestra aplicación es algo más compleja para tener únicamente un HTML.

Nuestra extensión Chrome cuenta con varios HTMLs según la red social. El objetivo era tener un HTML para Twitter, otro para LinkedIn y otro para Facebook. Cada HTML debe aparecer con la red social que le corresponda cuando se haga clic sobre el icono de la extensión, para ello se ha llevado a cabo un sistema de eventos en segundo plano que identifica la ruta y le asigna el HTML que le corresponde. Esto está implementado en el fichero “eventPage.js”, indicado en el manifiesto en la sección de “background” que como su nombre indica, controla las acciones en segundo plano de la extensión.

Y, por último, desde el manifiesto podemos ver la sección de “permissions” la cual otorga los permisos necesarios para usar ciertas funcionalidades. Se ha otorgado permisos a la funcionalidad “storage” que permite que se almacenen y recuperen los datos de los usuarios. Y también se le ha dado permisos a “tabs” que sirve para interactuar con el sistema de pestañas del navegador y con ello, crearlas, modificarlas y reorganizarlas.

Ambas funcionalidades se verán usadas a lo largo del código y se entenderá mejor el desempeño que han tenido.

4.3.2 HTMLs

- twitter.html. Es la interfaz de usuario para Twitter, que realiza un formulario el máximo número de tweets, fecha inicio y fin y máximo número de seguidores y seguidos. Y, por último, incluye el botón "Enviar formulario" que realiza el *scraping* en base a los datos introducidos.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Scraper Twitter</title>
    <script src="jquery-3.2.1.min.js"></script>
    <script src="scrapTwitter.js"></script>
    <script src="JsonToCsv.js"></script>


  </head>

  <body>
    <h1>Scraper Twitter</h1>
    <h3>Numero maximo de tweets (incr 20):</h3>
    <input type="number" id="numTweets">
    <h3>Fecha inicio:</h3>
    <input type="date" id="diaInicio">
    <h3>Fecha fin:</h3>
    <input type="date" id="diaFinal">
    <h3>Numero maximo de followers (incr 20):</h3>
    <input type="number" id="numFollows">
    <h3>Numero maximo de following (incr 20):</h3>
    <input type="number" id="numFollowing">

    <input type="submit" id=enviar value="Enviar formulario">


  </body>
</html>
```

Figura 30. Twitter.html

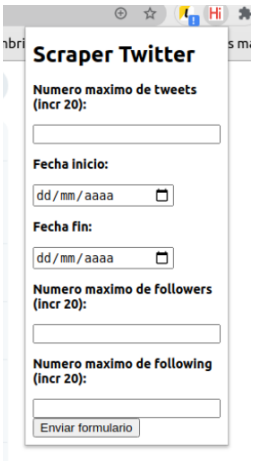


Figura 31. Interfaz twitter.html

- LinkedIn.html. Es la interfaz de usuario para LinkedIn, cuenta con la introducción de las credenciales para iniciar sesión, y el botón "Realizar scrap" para realizar el *scraping*.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Scraper Linkedin</title>
    <script src="jquery-3.2.1.min.js"></script>
    <script src="scrapLinkedin.js"></script>
    <script src="JsonToCsv.js"></script>


  </head>

  <body>
    <h1>Scraper Linkedin</h1>
    <h3>Correo:</h3>
    <input type="email" id="email">
    <h3>Contrase&ntilde;a:</h3>
    <input type="password" id="pass">
    <input type="submit" id=enviar value="Realizar scrap">


  </body>
</html>
```

Figura 32. LinkedIn.html

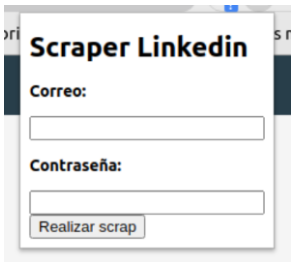


Figura 33. Interfaz LinkedIn.html

- LinkedIn_logged.html. Es la interfaz de usuario para LinkedIn cuando se ha guardado un inicio de sesión. Se puede volver a iniciar sesión con una cuenta diferente haciendo clic sobre “Cerrar Sesión”, llevándote a “LinkedIn.html”.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Scraper Linkedin</title>
    <script src="jquery-3.2.1.min.js"></script>
    <script src="scrapLinkedin.js"></script>
    <script src="JsonToCsv.js"></script>
  </head>

  <body>
    <h1>Scraper Linkedin</h1>
    <input type="submit" id=enviar value="Realizar scrap">
    <a href="linkedin.html"><input type="button" value="Cerrar Sesion"></a>
  </body>
</html>
```

Figura 34. LinkedIn_logged.html

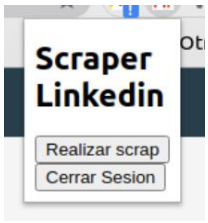


Figura 35. Interfaz LinkedIn_logged.html

- facebook.html. Es la interfaz de usuario para Facebook, cuenta con la introducción de las credenciales para iniciar sesión, y el botón “Realizar scrap” para realizar el *scraping*.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Scraper Facebook</title>
    <script src="jquery-3.2.1.min.js"></script>
    <script src="scrapFacebook.js"></script>
    <script src="JsonToCsv.js"></script>

  </head>

  <body>
    <h1>Scraper amigos Facebook</h1>
    <h3>Correo:</h3>
    <input type="email" id="email">
    <h3>Contrase&ntilde;a:</h3>
    <input type="password" id="pass">
    <input type="submit" id=enviar value="Realizar scrap">


  </body>
</html>
```

Figura 36. Facebook.html

Figura 37. Interfaz Facebook.html

- facebook_logged.html. Es la interfaz de usuario para Facebook cuando se ha guardado un inicio de sesión. Se puede volver a iniciar sesión con una cuenta diferente haciendo clic sobre “Cerrar Sesion”, llevándote a “facebook.html”.

```
<!DOCTYPE html>
<html>
  <head>
    <title> Scraper Facebook</title>
    <script src="jquery-3.2.1.min.js"></script>
    <script src="scrapFacebook.js"></script>
    <script src="JsonToCsv.js"></script>
  </head>

  <body>
    <h1>Scraper amigos Facebook</h1>
    <input type="submit" id=enviar value="Realizar scrap">
    <a href="facebook.html"><input type="button" value="Cerrar Sesion"></a>
  </body>
</html>
```

Figura 38. Facebook_logged.html

Figura 39. Interfaz Facebook_logged.html

4.3.3 ScrapTwitter.js.

Realiza la gestión toma de datos de la interfaz de usuario (HTML) y también la petición de *scraping* del servidor *scraping* de Twitter.

La función se ejecuta siempre que ocurra el evento “.click” sobre el botón con id “#enviar” procedente del twitter.html. A continuación, toma la URL de la pestaña actual del navegador con la funcionalidad “tabs” que hemos comentado en el manifest.json. En la

URL de la pestaña queda indicado el nombre de usuario del perfil, por lo que desde ahí tomamos el usuario para realizar su *scraping*.

```
$(function(){

    $('#enviar').click(function(){
        chrome.tabs.query({'active': true, 'windowId': chrome.windows.WINDOW_ID_CURRENT}, function(tabs){
            url = tabs[0].url;
            usuario = url.substring(20);
            console.log(usuario);
```

Figura 40. Código inicial ScrapTwitter.js

Seguidamente, hay un pequeño control de errores, en caso de lo que se haya referenciado en la variable "usuario" no se trate de un perfil en sí, sino de otra ruta propia de la red social. En caso de que no sea un perfil, se indicará que realice el *scraping* sobre una pestaña de un perfil de Twitter. Controles similares, se realizan también tanto para LinkedIn como para Facebook.

```
// Descarta url que no son un perfil
if (usuario.includes("home") || usuario.includes("explore") || usuario.includes("notifications") || usuario.includes("messages") || usuario.includes("i/")
 || usuario.includes("search") || usuario.includes("settings") || usuario.includes("compose") || usuario.includes("hashtag")){
    console.log("No es un perfil de twitter");
    alert("Seleccione un perfil de Twitter para realizar scraping");
    return
}

// Extraer usuario si esta anidado
if (usuario.includes("/")){
    division = usuario.split("/");
    usuario = division[0];
    console.log("Usuario extraido: " + usuario);
}
```

Figura 41. Código ScrapTwitter.js, control de errores para obtener correctamente el nombre de usuario.

Después se procesan todos los datos del formulario del HTML.

```
 if ($('#numTweets').val())
     numTweets = $('#numTweets').val();
 else
     numTweets = 20

 if ($('#diaInicio').val())
     diaInicio = $('#diaInicio').val();
 else
     diaInicio = "null";

 if ($('#diaFinal').val())
     diaFinal = $('#diaFinal').val();
 else
     diaFinal = "null";


 if ($('#numTweetsFavs').val())
     numTweetsFavs = $('#numTweetsFavs').val();
 else
     numTweetsFavs = 30;

 if ($('#numFollows').val())
     numFollows = $('#numFollows').val();
 else
     numFollows = 0;

 if ($('#numFollowing').val())
     numFollowing = $('#numFollowing').val();
 else
     numFollowing = 0;
```

Figura 42. Procesado de los campos del formulario de Twitter.

Todos estos campos son opcionales, en caso de que no se indique ninguno se le asigna un valor por defecto. Para el número máximo de tweets se le asigna por defecto 20 tweets, para los días de inicio y fin se quedan con un "null", el número máximo de tweets favoritos con 30 tweets, el número de seguidores y siguiendo a 0 (está configurado el servidor de *scraping* de Twitter para que si recibe el valor de 0 devuelva todos los perfiles).

A continuación, se ve todas las peticiones al servidor de *scraping* de Twitter:

```
// Fecha actual
var today = new Date();
var date = today.getDate()+'-'+(today.getMonth()+1)+'-'+ today.getFullYear();


//Tweets
//var url = 'http://127.0.0.1:5000/twitter/get_tweets_json/'+ usuario+ "/" + numTweets + "/" + diaInicio + "/" + diaFinal;
var url = 'https://servidorscrap.herokuapp.com/twitter/get_tweets_json/'+ usuario+ "/" + numTweets + "/" + diaInicio + "/" + diaFinal;
var csvFileName = 'tweets_' + usuario + "_" + date;
downloadURLfromJSOntoCSV(url, csvFileName, headers_tweets, "Twitter");

//Tweets favoritos
// url = 'http://127.0.0.1:5000/twitter/get_tweets_fav_json/'+ usuario + "/" + numTweetsFavs;
// var url = 'https://servidorscrap.herokuapp.com/twitter/get_tweets_fav_json/'+ usuario + "/" + numTweetsFavs;
// csvFileName = 'tweets_favoritos_' + usuario + "_" + date;
// downloadURLfromJSOntoCSV(url, csvFileName, headers_tweets, "Twitter");

//Followers
//url = 'http://127.0.0.1:5000/twitter/get_followers_json/'+ usuario + "/" + numFollows;
url = 'https://servidorscrap.herokuapp.com/twitter/get_followers_json/'+ usuario + "/" + numFollows;
csvFileName = 'followers_' + usuario + "_" + date;
downloadURLfromJSOntoCSV(url, csvFileName, headers_follows, "Twitter");

//Following
//url = 'http://127.0.0.1:5000/twitter/get_following_json/'+ usuario + "/" + numFollowing;
url = 'https://servidorscrap.herokuapp.com/twitter/get_following_json/'+ usuario + "/" + numFollowing;
csvFileName = 'following_' + usuario + "_" + date;
downloadURLfromJSOntoCSV(url, csvFileName, headers_follows, "Twitter");

//Info de usuario
//url = 'http://127.0.0.1:5000/twitter/get_user_info_json/'+ usuario;
url = 'https://servidorscrap.herokuapp.com/twitter/get_user_info_json/'+ usuario;
csvFileName = 'info_usuario_' + usuario + "_" + date;
downloadURLfromJSOntoCSV(url, csvFileName, info_usuario, "Twitter");
```

Figura 43. Peticiones de la extensión al servidor scraping de Twitter

Se puede observar que la URL usada para las peticiones utiliza la dirección de su servidor de *scraping* de Twitter en Heroku (https://servidorscrap.herokuapp.com). Estas peticiones son atendidas por los métodos referenciados en las ilustraciones de la 3 a la 7.

Para cada solicitud se llama a la función “downloadURLfromJSOntoCSV”. Esta función es llamada por todas las distintas peticiones de *scraping* que retornen un fichero JSON de las diferentes redes sociales de la extensión. Se encarga de realizar la petición correspondiente al servidor, tomar el fichero JSON devuelto por el servidor, transformarlo a formato CSV y, por último, descargarlo en el navegador. Toda esta implementación está hecha en el fichero “JsonToCsv.js”.

4.3.4 JsonToCsv.js.

```
function downloadURLfromJSOntoCSV(url, exportName, headers, redSocial){
    $.getJSON(url, function(result){
        console.log(result);
    })
    .done(function(result) {
        exportCSVFile(headers,result,exportName)
    })
    .fail(function(jqXHR, textStatus, errorThrown) {
        console.log("Fallo interno de " + redSocial + ",al intentar leer json");
        alert("Fallo interno de " + redSocial + ",al intentar leer json");
    })
}
```

Figura 44. Función “downloadURLJSONtoCSV” de JsonToCsv.js

Esta función realiza una petición HTTP de tipo GET al servidor. A continuación, desde la función “exportCSVFile” se transforma a formato CSV y se descarga en el navegador.

4.3.5 ScrapLinkedIn.js.

Encargado de gestionar las peticiones al servidor de *scraping* de LinkedIn. A continuación, se muestra como se ha hecho la solicitud POST para realizar el login en el servidor:

```
//var url = 'http://localhost:3000/login';
var url = 'https://serverscrap-node.herokuapp.com/login';
$.post( url, { email: x.usuarioLk, pass: x.passLk })
 .done(function( data ) {
     alert( "Data Loaded: " + data );
 });
```

Figura 45. Solicitud de login en LinkedIn desde la extensión.

En la Figura 46 que aparece debajo, se ve todas las solicitudes al servidor de *scraping* de LinkedIn de Heroku. Se comienza por la petición que realiza el *scraping* con la ruta "/LinkedIn/general/<usuario>" (Figura 11). En caso de que se haya realizado correctamente, solicita todos y cada uno de los ficheros JSON que constituyen el *scraping* para LinkedIn. Los métodos encargados de atender estas solicitudes son los mostrados en las ilustraciones 19 a la 24.

```
//var url = 'http://localhost:3000/linkedin/general/' + usuario;
var url = 'https://serverscrap-node.herokuapp.com/linkedin/general/' + usuario;
$.get(url, function(result){
    console.log(result);

})
.done(function(result) {
    if (result == "Login error"){
        alert("Fallo al iniciar sesion, vuelva a intentarlo");
    }else if (result == "Scrap erroneo"){
        alert("No es posible realizar scrap sobre este usuario, pruebe con otro")
    }else{
        //Perfil
        //var url = 'http://localhost:3000/linkedin/info_usuario/' + usuario;
        var url = 'https://serverscrap-node.herokuapp.com/linkedin/info_usuario/' + usuario;
        var csvFileName = 'info_usuario_' + usuario + "_" + date;
        downloadURLfromJSOntoCSV(url, csvFileName, headers_profile);

        //Educacion
        //var url = 'http://localhost:3000/linkedin/educations/' + usuario;
        var url = 'https://serverscrap-node.herokuapp.com/linkedin/educations/' + usuario;
        var csvFileName = 'educations_' + usuario + "_" + date;
        downloadURLfromJSOntoCSV(url, csvFileName, headers_educations);

        //Experiencia laboral
        //var url = 'http://localhost:3000/linkedin/positions/' + usuario;
        var url = 'https://serverscrap-node.herokuapp.com/linkedin/positions/' + usuario;
        var csvFileName = 'positions_' + usuario + "_" + date;
        downloadURLfromJSOntoCSV(url, csvFileName, headers_positions);

        //Gente relacionada
        //var url = 'http://localhost:3000/linkedin/peopleAlsoViewed/' + usuario;
        var url = 'https://serverscrap-node.herokuapp.com/linkedin/peopleAlsoViewed/' + usuario;
        var csvFileName = 'peopleAlsoViewed_' + usuario + "_" + date;
        downloadURLfromJSOntoCSV(url, csvFileName, headers_peopleAlsoViewed);

        //Recomendaciones recividas
        //var url = 'http://localhost:3000/linkedin/recommendations_received/' + usuario;
        var url = 'https://serverscrap-node.herokuapp.com/linkedin/recommendations_received/' + usuario;
        var csvFileName = 'recommendations_received_' + usuario + "_" + date;
        downloadURLfromJSOntoCSV(url, csvFileName, headers_recommendations);

        //Recomendaciones dadas
        //var url = 'http://localhost:3000/linkedin/recommendations_given/' + usuario;
        var url = 'https://serverscrap-node.herokuapp.com/linkedin/recommendations_given/' + usuario;
        var csvFileName = 'recommendations_given_' + usuario + "_" + date;
        downloadURLfromJSOntoCSV(url, csvFileName, headers_recommendations);
    }

})

.fail(function(jqXHR, textStatus, errorThrown) {

    console.log("Fallo interno de Scrapedin, vuelva a probar mas tarde");
    alert("Fallo interno de Scrapedin, vuelva a probar mas tarde");
})
```

Figura 46. Código de la extensión que realiza las solicitudes al servidor de scraping de LinkedIn

Se puede observar que la URL usada para cada una de las solicitudes es sobre el servidor de *scraping* de LinkedIn en Heroku (https://servidorscrap-node.herokuapp.com).

4.3.6 ScrapFacebook.js.

Encargado de gestionar las peticiones al servidor de *scraping* de Facebook. La solicitud de login es igual que como se hizo con LinkedIn, a diferencia que la URL es localhost.

```
var url = 'http://localhost:5000/facebook/login';
$.post( url, { email: x.usuarioFb, pass: x.passFb })
.done(function( data ) {
    alert( "Data Loaded: " + data );
});
```

Figura 47. Solicitud de login de Facebook en la extensión.

Y, por último, se solicita al servidor local de *scraping* de Facebook, el *scraping* de los amigos de un perfil. El método que recibe esta petición está referenciado en la Figura 27.

```
var url = 'http://127.0.0.1:5000/facebook/get_friends/' + usuario;
var csvFileName = 'amigos_' + usuario + "_" + date;
downloadURLfromJSOntoCSV(url, csvFileName, headers_friends);
```

Figura 48. Solicitud en la extensión al servidor de scraping de Facebook

4.3.7 EvenPage.js.

EvenPage.js es el fichero que trabaja siempre en segundo plano. A partir de eventos por acciones en las pestañas (actualización, creación o cambio de pestaña), asigna el HTML correspondiente.

```
//Cuando actualiza la pagina o al crear una nueva ventana
chrome.tabs.onUpdated.addListener(function(tabId, changeInfo, tab) {
    asignarHTMLs(tab.url, tabId);
});

//Cuando se cambia de pestaña
chrome.tabs.onActivated.addListener(function(activeInfo) {

    chrome.tabs.get(activeInfo.tabId, function(tab){
       asignarHTMLs(tab.url, tab.id);
    });
});
```

Figura 49. Listeners de EvenPage.js

Lo que aparece en la Figura de arriba son listeners que se activan dependiendo del estado de la pestaña actual. Si se crea una pestaña nueva o se actualiza, el listener llamado “onUpdated” es activado, en cambio, si se cambia de pestaña se activa el “onActivated”. Se puede observar que se está aplicando la funcionalidad de “tabs” comentada en el apartado de manifest.json, que era la encargada de gestionar las pestañas. En ambos casos, nos interesa que se asigne el HTML que le corresponde según de la URL que se indique en la pestaña, por ello, se creó la siguiente función:

```
function asignarHTMLs(url, id){
    console.log("url " + url);
    console.log("id " + id);

    if ((url).includes("https://twitter.com/")) {
        chrome.browserAction.setPopup({
            tabId: id,
            popup: 'twitter.html'
        });
    }else if ((url).includes("https://www.linkedin.com/")){
        chrome.storage.sync.get('usuarioLk', function(x){
            console.log("usuarioLk: "+ x.usuarioLk)
            if(x.usuarioLk != undefined){
                chrome.browserAction.setPopup({
                    tabId: id,
                    popup: 'linkedin_logged.html'
                });
            }else{
                chrome.browserAction.setPopup({
                    tabId: id,
                    popup: 'linkedin.html'
                });
            }
        });

    }else if ((url).includes("https://www.facebook.com/")){
        chrome.storage.sync.get('usuarioFb', function(x){
            console.log("usuarioFb: "+ x.usuarioFb)
            if(x.usuarioFb != undefined){
                chrome.browserAction.setPopup({
                    tabId: id,
                    popup: 'facebook_logged.html'
                });
            }else{
                chrome.browserAction.setPopup({
                    tabId: id,
                    popup: 'facebook.html'
                });
            }
        });


    }else{
        console.log("Error identificar red social")
    }
}
```

Figura 50. Función asignarHTMLs de EvenPage.js

Básicamente, en función de la variable “url” se compara con las rutas de cada red social (Twitter, LinkedIn o Facebook) hasta coincidir. Una vez se sepa cuál es la red social, se le asigna el HTML junto con la id de la pestaña. De esta forma se logra vincular la pestaña con el HTML que debe mostrar en cada caso.

Para la implementación de guardado de sesión, se usó la funcionalidad de “storage” para almacenar en una variable las credenciales introducidas por el usuario. En el siguiente apartado se explica el detalle de como se ha implementado el guardado de sesión.

4.3.8 Desarrollo del guardado de sesión para LinkedIn y Facebook.

Tanto LinkedIn como Facebook precisan de un inicio de sesión para poder realizar *scraping*. Para evitar estar introduciendo las credenciales cada vez que se quiera realizar *scraping*, se ha programado un sistema que guarda la sesión.

El usuario la primera vez introducirá las credenciales por la interfaz, y después de que haya dado a realizar *scraping*, quedará guardada su sesión en unas variables de “storage”. El lugar donde se inicializa o sobrescribe estas variables (credenciales) es en los ficheros de scrapLinkedIn.js y scrapFacebook.js. Las variables son “usuarioLk” y “passLk” para LinkedIn y por otro lado “usuarioFb” y “passFb” para Facebook.

Volviendo al fichero de eventPage.js, en la Figura 51 se puede ver que en base a “usuarioLk” para LinkedIn y “usuarioFb” para Facebook se decide ir a un HTML u otro. Esto se debe a que uno de los HTMLs es la interfaz con los campos para introducir las credenciales (“LinkedIn.html” o “facebook.html”) y otro es para cuando la sesión está iniciada (“LinkedIn_logged.html” o “facebook_logged.html”).

5 Integración, pruebas y resultados

En este capítulo se explicarán las pruebas realizadas para comprobar el correcto funcionamiento tanto de los servidores de *scraping* como la propia extensión. Al final, se comentará el tiempo de ejecución de cada uno de los *scraping*s.

5.1 Prueba sobre Twitter.

Desde un perfil de Twitter se hace clic sobre el icono de nuestra extensión. Se despliega la interfaz de Twitter y rellenamos los campos con los valores que aparecen la imagen:

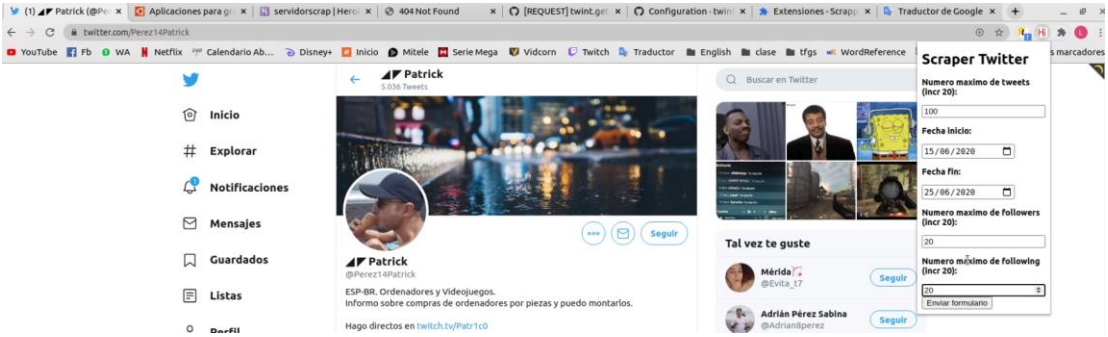


Figura 51. Ejemplo perfil de Twitter

Le damos a enviar formulario y descarga los siguientes ficheros CSV en el navegador:

- Información de usuario.

	A	B	C	D	E	F	G	H
1	Bio	Followers	Following	Id	Location	Name	Tweets	Username
2	ESP-BR. Ordenadores y Videojuegos. Informo sobre compras de ordenadores por piezas y puedo montarlos. Hago directos en http://twitch.tv/Patr1c0	140	215	851761662	PC	Patrick	5036	Perez14Patrick
3								

Figura 52. Ejemplo CSV de la información personal de un usuario de Twitter

- Tweets del usuario. Devuelve los tweets desde los más recientes a los más antiguos. Se puede observar en la imagen, que la obtención de estos tweets se rige fielmente en base a un máximo de 20 tweets y el rango de fechas indicado en el formulario. Parte desde el día 25/06/2020 hasta llegar a los 20 tweets sin sobrepasar la fecha mínima 15/06/2020.

	A	B	C	D
1	Date	Time	Timezone	Tweet
2	2020-06-25	03:52:26	200	Gracias por volver a mostrar en mi TL este ser infeliz y sin humanidad que saca lo más oscuro de mi interior xd
3	2020-06-24	19:27:39	200	@Juliii_9 mejor que la horchata o no jajajajsjhdjajdhf https://twitter.com/soysubnormla666/status/1275724449346617346 …
4	2020-06-24	17:15:40	200	[ESP-PT] Train: day 2 - Warzone https://twitch.tv/patr1c0
5	2020-06-24	16:18:39	200	Otro día en la oficina con estos chalados..
6	2020-06-24	15:11:23	200	No puedo imaginar algo más volátil que unir esas rebanadas xd
7	2020-06-24	05:08:55	200	Minha mão grudada na sua e não grudada no celularSei que é tenso de acreditar sei que ninguém mais dá valorMas a gente é de um tempo on
8	2020-06-23	19:05:02	200	[ESP-PT] Train: day 1 - Warzone y quizás league después https://twitch.tv/patr1c0
9	2020-06-23	18:55:38	200	#RTXOn @FullDros @alerivero15 @opelu8 https://twitter.com/NVIDIAGeForce/status/1275451433324691457 …
10	2020-06-23	02:17:40	200	Pagodinho ❤️
11	2020-06-22	16:48:19	200	pic.twitter.com/5Q7xc7aZik
12	2020-06-22	16:42:34	200	Las deluxe son para consumir 1 vez cada 8 meses y si es de la bandeja de un acompañante mejor jajaja
13	2020-06-22	16:35:58	200	Veo que no es el solo el clima el que se vuelve loco en este planeta..
14	2020-06-22	05:36:17	200	Somente apenas.
15	2020-06-21	04:51:39	200	Bate um papo
16	2020-06-20	21:50:30	200	Están locos la cabeza...
17	2020-06-20	03:13:19	200	Esperaba uno más complejo la verdad
18	2020-06-20	03:00:46	200	Send nudos
19	2020-06-19	21:38:04	200	pic.twitter.com/aJX5e7Dtjd
20	2020-06-19	20:50:11	200	Relajadita
21	2020-06-19	20:10:14	200	[ESP-PT] Cod Warzone - A darleee! https://twitch.tv/patr1c0

Figura 53. Ejemplo CSV de los tweets de un usuario de Twitter

- Seguidores del usuario. Devuelve exactamente los 20 seguidores que se han indicado en el formulario.

	A
1	Username
2	ludloes
3	n_rguez
4	brianseedorff
5	luciiaa771
6	paulareyes68
7	mcdoweladam
8	aaronrodri0
9	thecristosalva1
10	mudetraa
11	saulocabrera3
12	juliii_9
13	fabian_oran
14	goranmaric10
15	daviddesousaro
16	pol__98
17	fatosdotrouxa
18	0406z
19	jaqueli21710137
20	tutto_futbol
21	gohas_2501

Figura 54. Ejemplo CSV de los seguidores de un usuario de Twitter

- Personas que siguen al usuario. Devuelve exactamente los 20 usuarios que se han indicado en el formulario.

	A
1	Username
2	excel
3	vitality_apex
4	quersusfr
5	laucondeull
6	n_rguez
7	nanaotafrija
8	atviassist
9	brianseedorff
10	valorantes
11	kngvito
12	teamvitality
13	getvideobot
14	blizzardcseu_es
15	corsair
16	activision
17	infinityward
18	callofduty
19	frojo__
20	hpespana
21	sacriel42

Figura 55. Ejemplo CSV de las personas que sigue un usuario de Twitter

5.2 Prueba sobre LinkedIn.

Desde un perfil de LinkedIn se hace clic sobre el icono de nuestra extensión. Se despliega la interfaz de LinkedIn, iniciamos sesión y pulsamos en “Realizar scrap”:

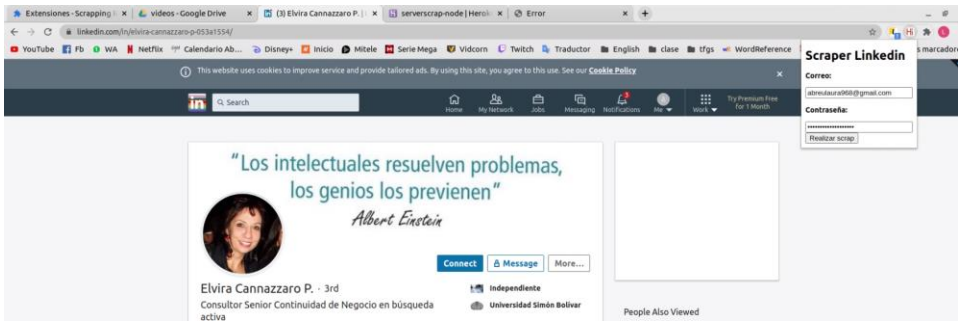


Figura 56. Ejemplo perfil de LinkedIn

A continuación, se realizar el *scraping* y se descargan en formato CSV los siguientes ficheros:

- Información personal.

	A	B	C
1	name	headline	location
2	Elvira Cannazzaro P.	Consultor Senior Continuidad de Negocio en búsqueda activa	Colombia

Figura 57. Ejemplo CSV de la información personal de un usuario LinkedIn

- Estudios. En este ejemplo, el usuario solo tiene un estudio. Los campos de fechas aparecen en blanco porque no están rellenados en el perfil de esta persona.

	A	B	C	D
1	title	degree	date1	date2
2	Universidad Simón Bolívar	Ingeniería en Computación		

Figura 58. Ejemplo CSV de los estudios de un usuario LinkedIn

- Personas relacionadas.

	A	B
1	user	text
2	https://www.linkedin.com/in/zully-escalona/	Consultor en Continuidad de Negocios Seguridad de la Información y Procesos de Gestión TI para iteam Group Corp.
3	https://www.linkedin.com/in/diana-garcia-cifuentes/	PMP® Consultor, Auditor, Continuidad de Negocios y Gestión Integral de Riesgo, Desastres, Riesgo Operativo, Scrum, ITIL
4	https://www.linkedin.com/in/ana-carolina-parra-nieto-2433a036/	Consultor en ManpowerGroup
5	https://www.linkedin.com/in/gustavo-adolfo-jimenez-vera/	I feel love for my career and big responsibility for my family and my work I student of engineer of system.
6	https://www.linkedin.com/in/jaime-adolfo-133091a/	Analista de Desarrollo en Sura
7	https://www.linkedin.com/in/luis-jos%C3%A9-sanguino-garc%C3%ADa-754090104/	Ingeniero mecánico Especialista en Gerencia de Proyectos
8	https://www.linkedin.com/in/norma-bucio-419218178/	Consultor profesional Sale U
9	https://www.linkedin.com/in/ibiceliszambrano/	Consultor de procesos de TI en Xelere
10	https://www.linkedin.com/in/alvaro-quispe-005517122/	Especialista de Networking y Seguridad
11	https://www.linkedin.com/in/cristinafritz/	Senior Manager en Deloitte - SAP CX Leader
12		
13		

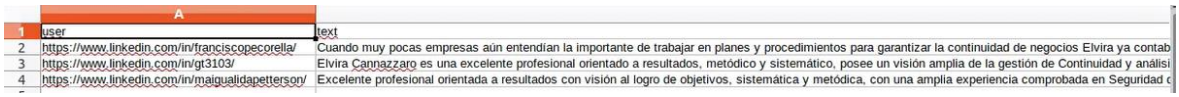
Figura 59. Ejemplo CSV de las personas relacionadas con un usuario LinkedIn

- Experiencia laboral.

	A	B	C	D
1	title	companyName	date1	date2
2	Consultor Senior Continuidad de Negocio	Independiente Freelance	Mar 2020 – Present	4 mos
3	Consultor Senior Continuidad del Negocio Gestión de Riesgo, Auditor, Seguridad de la Información	iteam Group Corp	Mar 2019 – Present	1 yr 4 mos
4	Gerente Proyectos Especiales	AMAGI Venezuela	Jan 2019 – Feb 2019	2 mos
5	Asesor senior Continuidad del Negocio (ISO22301) Seguridad TI, Proyectos, , Riesgos, ITIL, COBIT.	Asesorias y consultorias privadas	Sep 2017 – Jan 2019	1 yr 5 mos
6	Gerente de Continuidad del Negocio	Consorcio Credicard C.A	Oct 2015 – Apr 2017	1 yr 7 mos
7	Consultor de Continuidad del Negocio Riesgos, Seguridad Integral	Movilnet	Jun 2003 – Oct 2015	12 yrs 5 mos
8	Consultor Continuidad del Negocio Seguridad de la Información. Soporte TI	PDVSA Petroleos de Venezuela S.A.	May 1989 – Apr 2003	14 yrs

Figura 60. Ejemplo CSV de la experiencia laboral de un usuario LinkedIn

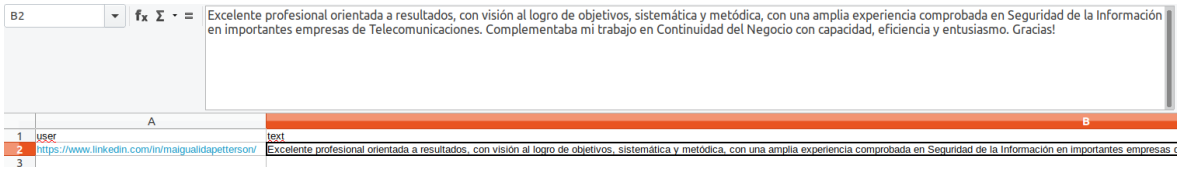
- Recomendaciones recibidas.



	A	
1	user	text
2	https://www.linkedin.com/in/franciscopecorella/	Cuando muy pocas empresas aún entendían la importante de trabajar en planes y procedimientos para garantizar la continuidad de negocios Elvira ya contab
3	https://www.linkedin.com/in/gt3103/	Elvira Cannazzaro es una excelente profesional orientado a resultados, metódico y sistemático, posee un visión amplia de la gestión de Continuidad y análisi
4	https://www.linkedin.com/in/maigualidapetterson/	Excelente profesional orientada a resultados con visión al logro de objetivos, sistemática y metódica, con una amplia experiencia comprobada en Seguridad

Figura 61. Ejemplo CSV de las recomendaciones recibidas a un usuario LinkedIn

- Recomendaciones dadas.



B2: Excelente profesional orientada a resultados, con visión al logro de objetivos, sistemática y metódica, con una amplia experiencia comprobada en Seguridad de la Información en importantes empresas de Telecomunicaciones. Complementaba mi trabajo en Continuidad del Negocio con capacidad, eficiencia y entusiasmo. Gracias!

	A	B
1	user	text
2	https://www.linkedin.com/in/maigualidapetterson/	Excelente profesional orientada a resultados, con visión al logro de objetivos, sistemática y metódica, con una amplia experiencia comprobada en Seguridad de la Información en importantes empresas
3		

Figura 62. Ejemplo CSV de las recomendaciones dadas por el usuario LinkedIn

5.3 Prueba sobre Facebook.

Desde un perfil de Facebook se hace clic sobre el icono de nuestra extensión. Se despliega la interfaz de Facebook, iniciamos sesión y pulsamos en “Realizar scrap”:

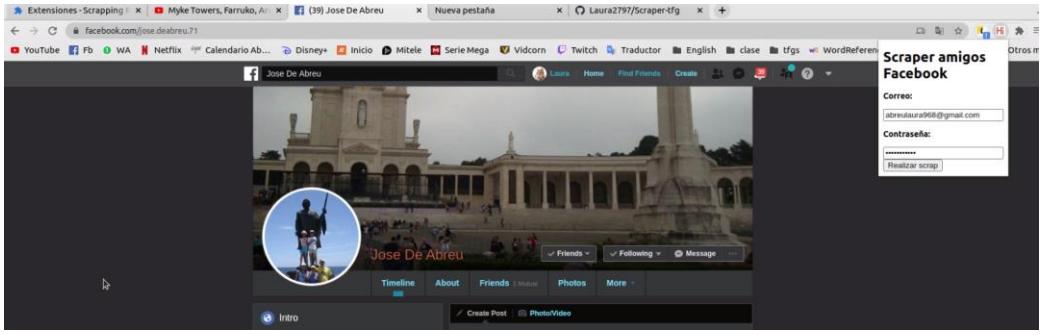


Figura 63. Ejemplo perfil de Facebook

Como se ha comentado a lo largo de la memoria, la implementación *scraping* de Facebook ofrece los amigos de un perfil. El fichero CSV obtenido se encuentra en la Figura de abajo. (En la imagen solo se muestra una parte del listado, ya que cuenta con 166 amigos).

	A	B
1	url	username
2	https://.facebook.com/laura.deabreugil	Laura De Abreu Gil
3	https://.facebook.com/marisol.d.d.3	Marisol De Abreu D
4	https://.facebook.com/gdabreu	Gabriela De Abreu
5	https://.facebook.com/gabi.espinozalopez	Gabriela Espinoza López
6	https://.facebook.com/maigua.landa	Maigua Landa
7	https://.facebook.com/855110220	Pierre Torbay
8	https://.facebook.com/josefrancisco.castellanodelgado	Jose Francisco Castellano Delgado
9	https://.facebook.com/manuel.f.diaz.733	Manuel Fernando Delgado Diaz
10	https://.facebook.com/sebastianneitor	Sebastian Tovar Rodriguez
11	https://.facebook.com/clement.teixeira.73	Clement Teixeira
12	https://.facebook.com/liliabeatriz.garcia	Lilia Beatriz Garcia
13	https://.facebook.com/AAA29	Andrea Acuña De Abreu
14	https://.facebook.com/733890762	Carla Rodrigues
15	https://.facebook.com/elfogue	Johnny Jose
16	https://.facebook.com/samuel.deabreu.9	Samuel De Abreu
17	https://.facebook.com/marbelly.gil	Marbelly Gil
18	https://.facebook.com/felipe.acunasantaella	Felipe Acuña Santaella
19	https://.facebook.com/100029347673028	Angela Fernandes
20	https://.facebook.com/enmanuel.deabreugil	Enmanuel De Abreu Gil
21	https://.facebook.com/delvi.sosa.9	Delvi Sosa
22	https://.facebook.com/eliana.medina.393	Eliana Rodriguez De Medina
23	https://.facebook.com/correo.de.adrian	Adrian Rodriguez
24	https://.facebook.com/elisabeth.fuentesgonzalez	Elisabeth Fuentes Gonzalez
25	https://.facebook.com/ViktorRiveraSanchez	Viktor Rivera Sanchez
26	https://.facebook.com/1146088220	Ana Sanchez
27	https://.facebook.com/Seaandsun.Belt	Marisol Correia
28	https://.facebook.com/esperanzacandelaria.diazreyes	Esperanza Candelaria Diaz Reyes
29	https://.facebook.com/yenny.diaz.712161	Yenny Diaz

Figura 64. Ejemplo CSV de los amigos del usuario de Facebook

5.4 Duración del scraping.

En la siguiente tabla que se indica una aproximación de la duración de cada una de los *scraping*s de prueba de apartado anterior. Por supuesto estos tiempos se deben tomar como una referencia aproximada, porque no dependen sólo del rendimiento del ordenador cliente sino principalmente del tiempo necesario para transferir la información a través de Internet. De todas formas, se listan para dar una idea general del tiempo que puede tardar, el cual es bastante alto.

	Twitter	**LinkedIn**	**Facebook**
Duración (segundos)	11	14	30

Tabla 1. Duración de los scrapings de cada red social de prueba

A continuación, se muestra el tiempo aproximado que supone ciertas tareas de *scraping* para Twitter en nuestro proyecto:

- En función del número de tweets.

Número de tweets	**100**	**1000**
Duración	27 segundos	4.5 minutos

Tabla 2. Duración del scraping en función del número de tweets en Twitter

- En función del número de seguidores (válido también para las personas a las que sigue).

Número de seguidores	**100**	**1000**
Duración	17 segundos	2.8 minutos

Tabla 3. Duración del scraping en función del número de seguidores en Twitter

Por último, se ha hecho lo mismo para el *scraping* de los amigos de Facebook.

Número de amigos	**100**	**1000**
Duración	18 segundos	3 minutos

Tabla 4. Duración del scraping en función del número de amigos en Facebook

6 Conclusiones y trabajo futuro

6.1 Conclusiones

En este trabajo se ha implementado una extensión Chrome capaz de realizar *scraping* sobre ciertas funcionalidades de las redes sociales de Twitter, LinkedIn y Facebook. Con el objetivo de que con estos datos obtenidos se puedan usar en herramientas más avanzadas para posibilitar análisis de redes sociales (SNA).

Al principio la tarea fue buscar *scrapers* en la web, que cumplieran ciertos requisitos:

- Que fuese un *scraper* con un lenguaje preferiblemente en Python para que fuese fácil implementar en un servidor Flask.
- Que incluyera *scraping* que funcionalidades interesantes para un posible análisis de redes sociales.
- Y también que se encontrara funcionando y estuviese mantenido activamente. Muchos de los *scrapers* que se contemplaron no estaban mantenidos y, o no funcionaban o no funcionaban en su totalidad.

Se encontró un buen en *scraper* para Twitter (Twint) sin mucha dificultad, pero para LinkedIn la tarea fue más complicada. No se encontraba un *scraper* de LinkedIn en Python que funcionara y que proporcionada un *scraping* interesante. Al final se tomó el *scraper* de Scrapedin para LinkedIn desarrollado en Node.js, lo que conllevaba a crear un nuevo servidor en ese mismo entorno. Scrapedin no contaba con algunas funcionalidades interesantes que nos hubiesen gustado, pero nos pareció que incluía lo suficiente para poder realizar un SNA.

Con Facebook, como se ha comentado antes, su Política de Términos y Condiciones de Uso es restrictiva. Por lo que, el número de *scrapers* que ofrecían este servicio era reducido. Se contempló realizar *scraping* no de perfiles si no de grupos y páginas de Facebook, buscando un *scraper* más completo para nuestro trabajo. Pero, o no funcionaban correctamente, o el procedimiento para utilizar esas herramientas requería de un permiso del uso de la API de Facebook donde tampoco tenía mucha libertad para obtener datos interesantes.
Cuando se tomó la decisión de usar el "Ultimate-Facebook-Scraper" este contaba con varias funcionalidades, pero las que nos resultaban interesantes eran el listado de amigos y los posts. Actualmente, el funcionamiento de *scraping* de posts en esta herramienta no funciona correctamente por lo que nos quedamos únicamente con el listado de amigos para poder ofrecer la posibilidad de realizar un SNA sobre eso. Una vez creado el servidor de *scraping* con esta herramienta en Python y subirla a Heroku, nos dimos cuenta de que más políticas, en este caso de seguridad, de Facebook restringía nuestra intención de uso remoto. Por lo que finalmente, nuestro servidor de Facebook quedó en local.

Finalmente, podemos decir que este TFG cuenta con una extensión que funciona y es útil para ser usada para el análisis de redes sociales. Es práctica ya que al tratarse de una extensión de Chrome se encuentra en directamente en el navegador.

6.2 Trabajo futuro

A continuación, se va a listar una serie de ideas para poder realizarse sobre este trabajo en el futuro:

- Por supuesto, el siguiente paso sería usar el resultado del *scraping* sobre herramientas de análisis de redes sociales, y llevar a cabo estudios al respecto.
- Mejorar el *scraping* de Facebook para que abarque más funcionalidades, no sólo amigos. Se podría plantear desarrollar un nuevo *scraper* desde cero, ajustándose a las necesidades dada la escasez de un *scraper* completo. Se podría incluir por ejemplo las publicaciones y comentarios de su muro, donde se viese reflejado quien lo ha escrito.
- Se podría mejorar la interfaz del usuario para que fuese más atractiva.
- Sería interesante incluir en el *scraping* de LinkedIn funcionalidades como el listado de amigos y también las publicaciones y comentarios de su muro.
- Para Twitter, se podría además de poder obtener los tweets, conseguir sus comentarios. También, adquirir los retweets del usuario, incluso conseguir aquellos usuarios que le han dado retweet a su tweet.

Referencias

[1] Feliciano Borrego, "Alternativas para realizar *Web Scraping*" , Actualizado el 9/8/2018, Accedido por última vez el 15/6/2020, http://felicianoborrego.com/alternativas-para-realizar-web-*scraping*/

[2] Marq Martí, "Qué es *Web scraping*? Introducción y herramientas", Fecha publicación el 8/4/2016, Accedido por última vez el 15/06/2020, https://sitelabs.es/web-*scraping*-introduccion-y-herramientas/

[3] Matthew A.Russell, "Mining the Social Web: Data mining Facebook, Twitter, LinkedIn, Google+, Github, and more (2nd Edition)", October 2013.

[4] Leonard Richardson, "Beautiful Soup", Actualizado el 17/05/2020, Accedido por última vez el 16/6/2020, https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[5] Puppeteer, Accedido por última vez el 16/06/2020, https://pptr.dev/

[6] Selenium, "The Selenium Browser Automation Project", Accedido por última vez el 16/06/2020, https://www.selenium.dev/documentation/en/

[7] "ChromeDriver – WebDriver for Chrome", Chromium and WebDriver teams, Accedido por última vez el 16/06/2020, https://chromedriver.chromium.org/

[8] "Scrapy 2.2 documentation", Accedido por última vez el 16/06/2020, https://docs.scrapy.org/en/latest/

[9] Martin Pérez, "*Web Scraping* vs API: What's the Difference", Actualizado el 09/03/2020, Accedido por última vez el 16/06/2020, https://www.parsehub.com/blog/web-*scraping*-vs-api/

[10] Francesco Poldi and Cody Zacharias, "Twintproyect/Twint", Accedido por última vez el 26/06/2020, https://github.com/twintproject/twint

[11] Ahmet Taspinar, "Twitter*scraper*", Accedido por última vez el 15/04/2020, https://github.com/taspinar/twitter*scraper*

[12] Buğra İşgüzar, "Twitter-*scraper*", Accedido por última vez el 16/04/2020, https://github.com/bisguzar/twitter-*scraper*

[13] Gustavo Bissolli, "Twitter-php-*scraper*", Accedido por última vez el 16/04/2020, https://github.com/bissolli/twitter-php-*scraper*

[14] Leodardi Wagner, "Scrapedin", Accedido por última vez el 20/05/2020, https://github.com/linkedtales/scrapedin

[15] Austin O'Boyle, "Scraper-LindeIn-selenium", Accedido por última vez el 01/05/2020, https://github.com/austinoboyle/scrape-linkedin-selenium

[16] Eric Fourrier, “scrape-linkedin”, Accedido por última vez el 02/05/2020, https://github.com/ericfourrier/scrape-linkedin

[17] Jordy van den Aardweg, “linkedin-profile-*scraper*”, Accedido por última vez el 15/05/2020, https://github.com/jvandenaardweg/linkedin-profile-*scraper*

[18] Hakim Khalafi, “linkedin-*scraper*”, Accedido por última vez el 02/05/2020, https://github.com/hakimkhalafi/linkedin-*scraper*

[19] Ozgur, “python-linkedin”, Accedido por última vez el 03/05/2020, https://github.com/ozgur/python-linkedin

[20] Haris Muneer, “Ultimate-Facebook-Scraper”, Accedido por última vez el 27/06/2020, https://github.com/harismuneer/Ultimate-Facebook-Scraper

[21] Joe Contini, “facebook-friends-map”, Accedido por última vez el 20/06/2020, https://github.com/jcontini/facebook-friends-map

[22] Xuebin Wei, “Data-Mining-on-Social-Media”, Accedido por última vez el 05/06/2020, https://github.com/xbwei/Data-Mining-on-Social-Media

[23] Numsap Srisakolkit, “facebook-miner”, Accedido por última vez el 05/06/2020, https://github.com/nsrisakolkit/facebook-miner

[24] Rugantio Costa, “fbcrawl”, Accedido por última vez el 06/06/2020, https://github.com/rugantio/fbcrawl

[25] Steven Xia, “facebook-post-*scraper*”, Accedido por última vez el 06/06/2020, https://github.com/brutalsavage/facebook-post-*scraper*

[26] Kevin Zúñiga, “facebook-*scraper*”, Accedido por última vez el 05/06/2020, https://github.com/kevinzg/facebook-*scraper*

[27] Jordan Ryda, “facebook-multi-*scraper*”, Accedido por última vez el 07/06/2020, https://github.com/jpryda/facebook-multi-*scraper*

[28] Isaacmg, “fb_*scraper*”, Accedido por última vez el 07/06/2020, https://github.com/isaacmg/fb_*scraper*

[29] Jacob Chen, “Facebook-Page-Crawler”, Accedido por última vez el 06/06/2020, https://github.com/chenjr0719/Facebook-Page-Crawler

Glosario

- API Application Programming Interface
- SNA Social Network Analysis
- URL Uniform Resource Locator

Anexos

A. Manual de instalación del servidor scraping de Facebook

En este apartado se va a explicar cómo instalar el servidor de *scraping* de Facebook, para facilitarle la tarea al usuario de incorporar la funcionalidad de *scraping* de esta red social. Para ello, se tiene que seguir los siguientes pasos:

1. Ir al GitHub del *scraper* "Ultimate-Facebook-Scraper", cuyo enlace se encuentra en la referencia [20] y clonar el repositorio.
2. Una vez se tiene el repositorio en el ordenador, seguir los pasos de instalación que indica el *scraper*.
3. Copiar el código del servidor (app.py) que se encuentra en el Anexo B e incorporarlo en la misma carpeta que se encuentra el repositorio del *scraper*.
4. Y por último instalar los siguientes programas:
    a. Python 3
    b. Flask
    c. Flask_cors

Ya tendríamos lo necesario para poder ejecutar esto en nuestro ordenador en local. Abrimos una terminal y escribimos el siguiente comando para arrancar el servidor:

```
$ python app.py
```

A partir de este momento, el servidor está en espera de peticiones.

B. Código del servidor scraping de Facebook.

```
#!flask/bin/python
from flask import Flask, jsonify, request
import os
import json
from flask_cors import CORS
from os import remove
import os.path as path
import yaml
from datetime import datetime



app = Flask(__name__)
CORS(app)

#App Ultimate-facebook-scraper
@app.route('/facebook/login', methods = ['POST'])
def login():

    email = request.form['email']
    password = request.form['pass']


    with open("credentials.yaml", "r") as ymlfile:
        cfg = yaml.safe_load(stream=ymlfile)
    if email == cfg['email'] and password == cfg['pass']:
        print ("Same login user")
        return "sameLoginUser"

    else:
        dict = {"email": email,
        "pass": password }

        with open('credentials.yaml', 'w') as file:
            yaml.dump(dict, file)
        return "difLoginUser"


@app.route('/facebook/get_friends/<string:username>')
def get_friends(username):
    now = datetime.now()

    input_url_user = "https://www.facebook.com/" + username

    file = open("input.txt","w")
    file.write(input_url_user)
    file.close()
    os.system('python scraper/scraper.py --total_scrolls 1')

    nombreCarpeta = "stored/" + username
    try:
        os.makedirs(nombreCarpeta)
    except:
        print("DY")
    ruta_json = nombreCarpeta + "/" + now.strftime("%d-%m-%Y_%H:%M:%S")
+'_friends.json'
```

```
    text_to_json(username, ruta_json)

    with open(ruta_json) as f:
        data = json.load(f)
        return jsonify(data)



def text_to_json(username, ruta_json):
    filename = 'data/' + username + '/All Friends.txt'
    print("USERNAME: " + username )

    location_data = []
    with open(filename) as f:
        for line in f:
            line = line.split(",")
            location_data.append({"url": line[0], "username": line[1]})

    out_file = open(ruta_json, "w")
    json.dump(location_data, out_file)


if __name__ == '__main__':
    app.run(debug=True)
```