UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Diseño e implementación de una biblioteca Javascript para la visualización, manipulación y análisis de grafos

Autor: Andrés Barbero Valentín
Tutor: Álvaro Ortigosa Juárez

junio 2021

Andrés Barbero Valentín
Diseño e implementación de una biblioteca Javascript para la visualización, manipulación y análisis de grafos

Andrés Barbero Valentín

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Resumen

Los avances en las Redes Sociales han ido aumentando de manera exponencial en los últimos años en términos como las funciones o servicios disponibles para los usuarios, la trascendencia de las publicaciones ahora es muy alta en comparación con sus inicios o incluso ahora hay un gran número de redes sociales distintas, y de la misma forma, cada vez hay más usuarios que interactúan a través de una o más de ellas.

Todo ello ha traído de la mano un incremento importante en la gran cantidad de datos que ahora se manejan, por lo que es muy útil tener herramientas disponibles que muestren estos datos con visualizaciones atractivas, sencillas, manejables y representativas para poder realizar un análisis sobre todos esos datos que actualmente se manejan.

Para ello, este proyecto se ha centrado en crear una herramienta que se encarga de obtener datos de la red social Twitter y transformar los datos obtenidos para crear una visualización en forma de grafo que ayude a poder realizar un análisis sobre esos datos.

Los datos han sido obtenidos de la red social Twitter mediante la utilización de la biblioteca *Twython*, que hace de envoltorio a la API de Twitter para simplificar el proceso de autenticación y obtención de los datos.

En cuanto a la transformación de datos, consiste en la creación de un frontal en el que se muestran los datos obtenidos en una visualización en forma de grafo en donde se ven los distintos nodos y enlaces correspondientes a los usuarios y las relaciones que existen entre ellos para las distintas funcionalidades implementadas. Además, el usuario tiene acceso a diferente información asociada al grafo y le da la posibilidad de poder realizar transformaciones para cambiar su apariencia, todo ello mediante la interacción del usuario con esta visualización en forma de grafo.

Palabras clave

nodos, enlaces, mouseover, mouseout, mouseclick, tweet, retweet, like, hashtag, seguidores

Abstract

Advances in Social Networks have been increasing exponentially in recent years in terms such as functions or services available to users, the importance of publications is now very high compared to its beginnings or even now there are a large number of different social networks, and in the same way, more and more users interact through one or more of them.

All this has led to a significant increase in the large amount of data that is now handled, so it is very useful to have tools available that show this data with attractive, simple, manageable and representative visualizations to be able to carry out an analysis on those data that are currently being handled.

For this, this project has focused on creating a tool that is responsible for obtaining data from the social network Twitter and transforming the data obtained to create a visualization in the form of a graph that helps to perform an analysis on that data.

The data has been obtained from the social network Twitter by using the Twython library, which wraps the Twitter API to simplify the authentication and data collection process.

Regarding the transformation of data, it consists of the creation of a front in which the data obtained is shown in a visualization in the form of a graph where the different nodes and links corresponding to the users and the relationships that exist between them for the different functionalities implemented. In addition, the user has access to different information associated with the graph and gives them the possibility of being able to carry out transformations to change its appearance, all of this through the interaction of the user with this visualization in the form of a graph.

Keywords

nodes, edges, mouseover, mouseout, mouseclick, tweet, retweet, like, hashtag, followers

Índice

Listas

Lista de figuras

1

Introducción

Las Redes Sociales surgieron a finales de la década de 1990 o incluso principios de los años 2000, permitiéndonos así tener una herramienta en la que poder crearnos un perfil y agregar a otros participantes. Así es como surgió en un primer momento.

Sin embargo, eran pocas las personas que imaginaban que las Redes Sociales tendrían el impacto que tienen hoy en día. El deseo de interactuar con otras personas desde cualquier lugar del mundo provocó que, ya no solo personas, sino empresas u organizaciones estén muy involucradas en este tipo de herramientas.

En el caso de empresas u organizaciones, han visto la oportunidad que brinda una Red Social de comunicarse con su público o clientes de una manera mas efectiva y rápida. Pudiendo compartir la visión de la empresa, poder saber más sobre sus clientes o tener la posibilidad de vender sus productos y servicios, entre otras muchas cosas, mediante este tipo de herramientas.

Dado el abanico de nuevas opciones que brinda una Red Social, uno puede pensar que sería muy útil poder realizar un análisis a una red social. El análisis de una red social consiste en modelar y realizar un estudio sobre fenómenos sociales enfatizando en el estudio de las relaciones entre actores, ya sean relaciones entre países, empresas, organizaciones, personas, etc. En estos análisis se investiga y se da mucha importancia a las relaciones, enlaces, contactos, la coherencia y cohesión, la centralidad y la composición de los nodos, en resumidas cuentas, las interrelaciones existentes entre entidades que interactúan en una red.

Hoy en día, cada vez existen más herramientas para hacer este tipo de análisis sobre una Red Social pero no existen alternativas sencillas para complementar los análisis con visualizaciones escalables y manipulables de las redes analizadas.

Por tanto, este proyecto está centrado en la implementación de una biblioteca JavaScript que represente datos obtenidos de una Red Social visualizando dichos datos en forma de grafo para enriquecer los análisis de Redes Sociales que existen actualmente.

Para conseguir el objetivo, lo primero es obtener datos de la Red Social en cuestión. Para ello, se ha utilizado la librería *Twython* (escrita en Python), que hace de envoltorio a la API de Twitter para así

simplificar el proceso de autenticación y obtención de datos.

Una vez obtenidos los datos, se realizan tratamientos a dichos datos mediante JavaScript para generar los nodos y enlaces que compondrán un grafo final para cada funcionalidad implementada.

Este grafo se realiza con la biblioteca JavaScript *D3.js*, que nos permite trabajar con una gran cantidad de datos y nos brinda dinamismo e interacción con las visualizaciones.

El frontal implementado para mostrar este grafo se ha realizado con Bootstrap [1] y con el apoyo de la biblioteca JavaScript mencionada anteriormente D3.js.

En este frontal, además de mostrar el grafo final, se permitirá ir navegando por las distintas funcionalidades implementadas, contiene una leyenda del grafo (donde se muestran los nodos que lo componen, con su nombre de usuario y color asignado a cada nodo), así como, un panel inferior en donde se puede cambiar la apariencia de los nodos y enlaces del grafo, tanto su color como su tamaño. También podemos elegir cambiar solo la apariencia a un nodo o enlace.

Por último, para ayudar a realizar un mejor análisis, la visualización en forma de grafo permitirá mostrar la información asociada a cualquier nodo del grafo.

1.1. Objetivos

En función de lo hablado anteriormente, se van a definir los objetivos de este trabajo. Entre otras cosas, esta definición de objetivos permitirá extraer conclusiones al final y poder ver si se ha conseguido lo que se quería inicialmente.

Estos objetivos son los siguientes:

- **o.1**: obtención de datos de una red social, Twitter en este caso, mediante la librería Twython para su posterior tratamiento.
- **o.2**: tratamiento de los datos obtenidos para la generación de visualización en forma de grafo usando JavaScript.
- **o.3**: generación del frontal (*front end*) en el que se incluya la visualización en forma de grafo, añadiendo más elementos que enriquecen dicho frontal para que permitan o ayuden a realizar un análisis sobre la red social de la que se recogen los datos.
- **o.4**: Entender mejor el funcionamiento de las redes sociales en cuanto a las distintas relaciones que pueden existir entre los usuarios.

1.2. Estructura

A continuación, se va a explicar la organización llevada a cabo en esta memoria de TFG.

La sección 2 se centra en poner en contexto las tecnologías existentes en el mercado o el entorno

tecnológico por el que se ha desarrollado este proyecto. En donde se realiza un análisis de las opciones que existen actualmente y los motivos de la elección de alguna de ellas.

En cuanto a la sección 3, en los apartados 3.1 y 3.2 se detallan los diferentes requisitos funcionales y no funcionales del sistema. En función de eso, la arquitectura elegida para el desarrollo e implementación del mismo. Después, en el apartado 3.3, se muestra el diagrama de arquitectura simple y se explica las comunicaciones que se realizan entre los distintos componentes del mismo. El capitulo termina con el apartado 3.4 en el que se cuenta la organización del código, es decir, los diferentes ficheros por los que está compuesto el proyecto.

El capítulo 4 se centra en explicar el proceso de implementación de las funcionalidades del proyecto. En los apartados 4.1 y 4.2 se habla de los componentes Servidor y Analyzer. En el apartado 4.3 se explican las opciones a las que tiene acceso el usuario en cada una de las funcionalidades implementadas. En cuanto al apartado 4.4, se explica cómo se han obtenido los datos de la red social Twitter. Por último, en el apartado 4.5, se muestran y se explican algunos aspectos de código relevantes.

En lo referente al capítulo 5, este representa una batería de las pruebas realizadas, adjuntando evidencias, para comprobar el correcto funcionamiento de todas la funcionalidades implementadas en el proyecto.

Por último, se termina con un capitulo dedicado a las conclusiones en el que se hace una reflexión sobre el trabajo que se ha realizado explicando aspectos como el aprendizaje obtenido, posibles mejoras futuras, que es lo que se podía haber hecho mejor y que es lo que faltó hacer.

2

Tecnología existente

El estudio se ha centrado en las bibliotecas JavaScript que permitieran tratar con gran cantidad de datos y doten de la posibilidad de creación de visualizaciones que puedan representar los datos obtenidos, ya que es mejor apoyarse en componentes ya existentes y probados para no tener que reproducir funcionalidad ya implementada.

Entre todas las opciones que hay, se ha profundizado sobre las siguientes:

- **d3.js**
  Según podemos ver en los libros [2] y [3], esta biblioteca usa HTML, SVG y CSS. Tiene una buena combinación de componentes de visualización con un enfoque "data-driven" en la manipulación del DOM. Además, proporciona facilidad de depurar con el inspector de elementos del navegador. Permite usar gran cantidad de datos y comportamiento dinámico de visualizaciones. Y, por último, da la posibilidad de trabajar con muchos tipos de gráficos.
- **Raphael.js**
  Es una biblioteca que simplifica la manipulación de grafos vectoriales. Se pueden crear charts a medida o widgets que recorten o roten imágenes. Utiliza SVG, W3C Recommendation y VML. Esto implica que cada gráfico que creas es al mismo tiempo un objeto DOM que permite añadirle manejadores de eventos para su manipulación mediante JavaScript. Lo podemos ver con más detalle en [4]
- **JsPlumb.js**
  Ayuda a desarrollar aplicaciones donde la conectividad visual de elementos sea un punto clave. Añade una capa de funcionalidades adicional como layouts, zooms, etc. La funcionalidad de la misma y cómo se usa lo podemos ver en su documentación [5]
- **Cytoscape.js**
  De acuerdo al articulo [6] está muy optimizada para el tratamiento de grafos, incluyendo algoritmos de teoría de grafos y redes para poder analizar y visualizar grafos complejos. Además de tener exportación de datos a ficheros JSON.
- **vis.js**
  Permite realizar distintos tipos de gráficos, es fácil de usar y ofrece la posibilidad de realizar visualizaciones de grandes conjuntos de datos dinámicos que se pueden manipular de forma interactiva (recogido en su documentación [7]), por ejemplo: redes, líneas temporales, gráficas 2D y gráficas 3D.
- **Charts.js**
  Según podemos ver en el libro [8] es simple, potente y flexible para desarrollar componentes de visualización de datos. Aporta buen rendimiento de representación en distintos navegadores web y en distintas resoluciones mediante el uso de canvas de HTML5. Utiliza 8 tipos de gráficos, animados y personalizables.

De entre estas opciones, elegí utilizar d3.js por las siguientes razones:

- Utiliza de SVG (Scalable Vector Graphics), el cual es un formato de imagen de vectores por lo que la imagen es descrita mediante una composición de formas geométricas en vez de definirse dicha imagen como un conjunto de píxeles. Esto hace que el buscador dibuje en coordenadas específicas. Otro aspecto a resaltar es que estas imágenes tienen la capacidad de escalarse de acuerdo a las proporciones de la ventana del buscador sin perder nitidez ni distorsionarse, es decir, se adaptan a sus dimensiones.
- Es una librería que está implementada para manipular documentos con base en datos.
- Es una librería especializada en la visualización de datos en páginas web.
- Es una librería basada en los estándares web.
- Es una librería que permite manipular el DOM.

Por todas estas razones esta librería es ideal para conseguir obtener el objetivo de este proyecto.

Por otro lado, también el estudio se centró en buscar las herramientas, actualmente existentes, que ofrecen visualizaciones para realizar análisis de redes sociales para ver si se podía adoptar alguna idea. Entre ellas se destacan las siguientes:

- **Gephi**
  Herramienta *open-source* que permite visualizar datos en forma de grafo para su posterior análisis. Debido al uso de un motor renderizado en 3D te da la posibilidad de mostrar gráficos en tiempo real, así como, analizar, filtrar, manipular e incluso la exportación de diferentes tipos de visualizaciones o gráficos. Es una herramienta muy potente. [9]
- **Commetrix**
  Herramienta de análisis de datos de red dinámicos. Combina visualización de gráficos dinámicos, análisis de redes sociales, teoría gestáltica y minería de datos con algoritmos de búsqueda y filtrado. [10]
- **Guess**
  Herramienta para el análisis y visualización de datos en forma de grafo. Tiene un sistema de base de datos que te ofrece la posibilidad de incluir atributos más complejos de los habituales (continuos, categóricos y binarios) en los nodos y enlaces. Admite datos dinámicos y te permite un control completo sobre los nodos y enlaces del grafo. [11]

De entre estas opciones se ha elegido profundizar más sobre Gephi ya que algunas herramientas que te ofrece este software se asemejan a los objetivos iniciales del trabajo y, por tanto, serviría de ayuda o referencia a la hora de implementarlas para la interfaz de la aplicación web.

Algunas ideas que se adoptaron para el proyecto son las siguientes:

- Dar la posibilidad al usuario de interactuar con la visualización en forma de grafo.
- Aportar al usuario la posibilidad de poder desplazar los nodos de la visualización en forma de grafo por todo el espacio dedicado al mismo.
- Permitir al usuario poder cambiar el tamaño y color de los nodos y aristas de la visualización en forma de grafo.
- Aportar al usuario un pequeño apartado a modo informativo sobre el contenido de la visualización en forma de grafo en cuanto a número de nodos y aristas existentes en el mismo.

3

Análisis y Diseño

En este apartado se va a describir los distintos requisitos funcionales y no funcionales que tiene que cumplir nuestro sistema. En función de eso, la elección de la arquitectura con una explicación de las diferentes comunicaciones que se realizan entre el cliente-servidor y la biblioteca JavaScript implementada para llegar al resultado final. Por último, se contará cómo se organiza el código implementado para el proyecto.

3.1. Requisitos funcionales

En primer lugar, se va a describir los requisitos funcionales del sistema.

Los requisitos funcionales que han guiado el diseño de la aplicación son los siguientes:

- **RF1:** El sistema analizará los retweets de un usuario que realiza retweets a otros usuarios para tranformarlos visualmente en un grafo mostrando como nodos los usuarios y como enlaces los retweets que ha habido entre esos usuarios.
- **RF2:** El sistema analizará los retweets de un usuario que realiza retweet a otros usuarios para transformarlos visualmente en un grafo mostrando como nodos los tweets y usuarios. Como enlaces, se mostrará a que usuarios pertenecen esos tweets y los retweets que ha habido entre los usuarios.
- **RF3:** El sistema analizará los likes que ha realizado un usuario a otros usuarios para transformarlos visualmente en un grafo mostrando como nodos los usuarios y como enlaces los likes que ha habido entre esos usuarios.
- **RF4:** El sistema analizará los likes que ha realizado un usuario a otros usuarios para transformarlos visualmente en un grafo mostrando como nodos los tweets y usuarios. Como enlaces, se mostrará a que usuarios pertenecen esos tweets y los likes que ha habido entre los usuarios.
- **RF5:** El sistema analizará los seguidores de un usuario para transformarlo visualmente en un grafo mostrando los seguidores de ese usuario.
- **RF6:** El sistema analizará los retweets que han realizado distintos usuarios sobre los tweets que tienen un hashtag concreto mostrando como nodos los usuarios y tweets que contienen ese hashtag. Como enlaces se mostrarán que usuarios han realizado retweet a ese tweet que contiene ese hashtag.
- **RF7:** El sistema ofrece la posibilidad de cambiar el color de los nodos de la visualización en forma de grafo.
- **RF8:** El sistema ofrece la posibilidad de cambiar el tamaño del radio de los nodos de la visualización en forma de grafo.
- **RF9:** El sistema ofrece la posibilidad de cambiar el color de un nodo seleccionado dentro de la visualización en

forma de grafo.

- **RF10:** El sistema ofrece la posibilidad de cambiar el tamaño del radio de un nodo seleccionado dentro de la visualización en forma de grafo.
- **RF11:** El sistema ofrece la posibilidad de cambiar el color de los enlaces de la visualización en forma de grafo.
- **RF12:** El sistema ofrece la posibilidad de cambiar el grosor de los enlaces de la visualización en forma de grafo.
- **RF13:** El sistema ofrece la posibilidad de cambiar el color de un enlace seleccionado dentro de la visualización en forma de grafo.
- **RF14:** El sistema ofrece la posibilidad de cambiar el grosor de un enlace seleccionado dentro de la visualización en forma de grafo.
- **RF15:** El sistema mostrará la información asociada a un nodo cuando nos posicionemos con el ratón encima de dicho nodo dentro de la visualización en forma de grafo.
- **RF16:** El sistema resaltará los nodos y enlaces que están conectados cambiando su color por defecto cuando seleccionemos un nodo dentro de la visualización en forma de grafo.
- **RF17:** El sistema dará la posibilidad de cambiar la posición de un nodo dentro de la visualización en forma de grafo.

3.2. Requisitos no funcionales

Por otro lado, los requisitos no funcionales serían los siguientes:

- **RNF1:** El tiempo de carga no debe superar los 3 segundos de respuesta.
- **RNF2:** El sistema presentará una interfaz de usuario intuitiva y sencilla para facilitar el manejo y uso a los usuarios del sistema.
- **RNF3:** El sistema estará alojado en un servidor eficiente que pueda dar respuesta a una gran concurrencia de peticiones de distintos usuarios en ciertos periodos de tiempo.
- **RNF4:** El sistema debe asegurar que determinados datos estén protegidos del acceso no autorizado.
- **RNF5:** El sistema debe funcionar en todos los navegadores web modernos.
- **RNF6:** La aplicación debe adaptarse a diferentes resoluciones de pantalla.

En función de estos requisitos definimos la arquitectura que va a tener nuestro sistema, en este caso debe ser una arquitectura cliente-servidor [12] ya que es un modelo de aplicación distribuida en donde las tareas se diversifican repartiéndose entre los componentes del sistema.

3.3. Comunicación componentes sistema

Estos componentes tienen asignados unos roles, es decir, se encargan de diferentes tareas. El Cliente tiene el rol de visualizar el contenido generado por el Analyzer o la página web devuelta por el Servidor. El Analyzer se encarga de realizar el tratamiento de los datos obtenidos del servidor para la creación de la visualización en forma de grafo y las diferentes acciones que se brindan al usuario. Y, por último, el Servidor se encarga de atender las peticiones recibidas devolviendo páginas web o

ficheros de datos en formato json.

En el diagrama 3.1 podemos observar las comunicaciones entre Cliente, Analyzer y Servidor.

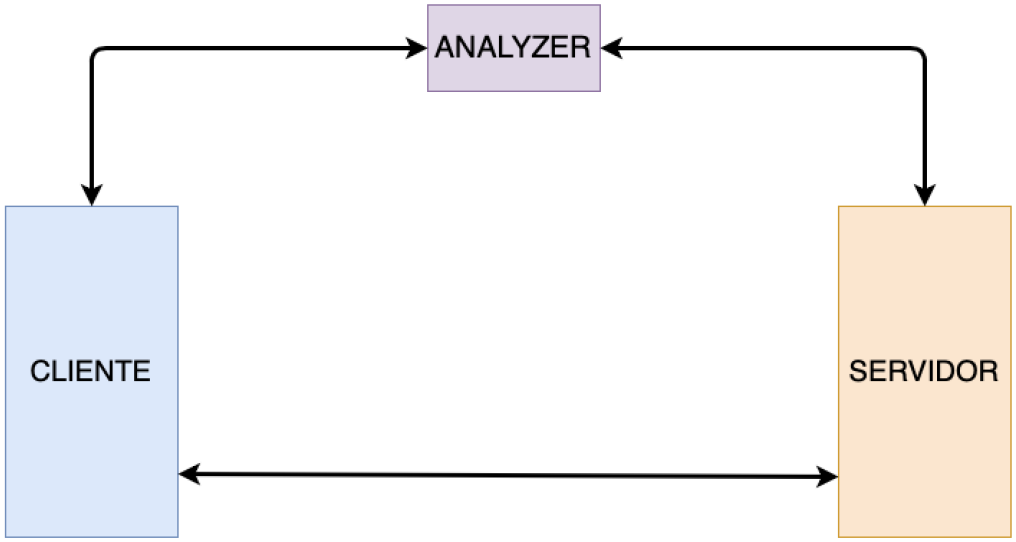

Figura 3.1: Diagrama de arquitectura

Una vez desplegada la aplicación web, el cliente (navegador) muestra la página inicial de la aplicación (index.html) en donde se encuentra el acceso, mediante botones, a las distintas funcionalidades implementadas.

Tras la elección de una de ellas, el cliente envía una petición al servidor correspondiente a dicha funcionalidad seleccionada para que el servidor le devuelva la página web relacionada con esa funcionalidad.

Durante la carga del template, el cliente hace una llamada a la función JavaScript correspondiente la cual, en primer lugar, realiza una petición al servidor para obtener los datos relacionados para esa funcionalidad.

El servidor se encarga de recoger la petición y devolver los datos al Analyzer en formato json.

Tras la recepción de dichos datos, mediante JavaScript, el Analyzer realiza el tratamiento de esos datos para la creación de los nodos y enlaces del grafo (con sus tamaños y colores), los manejadores de los distintos eventos que posee el grafo, leyenda del grafo, información del número de nodos y enlaces del grafo y la herramienta para poder cambiar el aspecto de nodos y/o enlaces.

Una vez realizadas todas estas operaciones, el cliente visualiza todo el contenido generado.

3.4. Organización del código

En lo referente a la organización del código, éste está compuesto por: servidor, templates, ficheros JavaScript, ficheros de datos json, fichero css y script de obtención y mapeo de datos Twitter.

- **dummyServer.py**
  Script que implementa el despliegue del servidor y el tratamiento de las peticiones recibidas por el cliente (envío de templates y datos). Se ha llamado *dummy* debido a que las tareas que realiza son sencillas, tal y como se acaba de comentar.
- **Templates**
  Ficheros html donde se definen todos los componentes necesarios a mostrar por el cliente.
- **Ficheros JavaScript**
  Ficheros donde se realiza la petición al servidor de obtención de datos y generación de todo lo relacionado con la interacción y visualización de cada funcionalidad del proyecto.
- **Ficheros datos Json**
  Ficheros de datos obtenidos de Twitter.
- **css.css**
  Fichero que dota de estilo a los distintos templates del proyecto.
- **twitter_auth.py**
  Script encargado de realizar la conexión con la API de Twitter para obtener los datos y realizar el mapeo correspondiente para adaptarlo a la estructura necesaria para cada funcionalidad del proyecto.

4

Implementación

En este apartado se detalla el proceso de implementación de las funcionalidades del proyecto y cuáles han sido las tecnologías utilizadas en el proceso: lenguajes, librerías y herramientas de programación utilizadas.

Finalmente, se explicarán los procesos de obtención de datos y algunos aspectos de código relevantes.

Como se ha comentado anteriormente en el diagrama de arquitectura 3.1, el proyecto se ha basado en la implementación de 3 componentes principales: Servidor, Analyzer y Cliente (navegador web).

4.1. Servidor

En cuanto al **Servidor**, éste ha sido llamado *dummyServer* por el hecho de que sus funciones son la de atender peticiones sencillas que llegan por parte del cliente, estas peticiones sencillas hacen referencia a devolver un template o devolver un fichero de datos en formato JSON al cliente. Este servidor se ha creado para poder realizar las pruebas pertinentes y así comprobar el correcto funcionamiento del desarrollo implementado pero el objetivo final del trabajo es que el Analyzer se pueda integrar en otros servidores más sofisticados. Ha sido implementado con el framework Flask [13], el cuál está escrito en Python.

4.2. Analyzer

El componente **Analyzer** es el encargado de realizar una petición al servidor para obtener los datos necesarios para su tratamiento y transformación en la visualización en forma de grafo y generación de todos los demás componentes que se ven en el frontal.

Una vez obtenidos los datos, lo primero que se hace es crear el contenedor SVG en donde se incluirá la visualización en forma de grafo.

Una vez hecho esto, se pasa al proceso de preparación de los datos asociados a los nodos y enlaces del grafo, cuya información viene recogida del fichero de datos JSON obtenido inicialmente.

Después, generamos el apartado que se encarga de mostrar la información del grafo en forma de número de nodos y número de enlaces.

Ahora viene la parte de simulación de fuerzas que llevará nuestro grafo y la relación y representación de nodos y enlaces, las cuales se explicarán con más detalle más adelante.

Tras esto, se genera lo que se llama la leyenda del grafo que crea un círculo por cada nodo del grafo con su color que ha sido asociado en su creación y el nombre de usuario al que pertenece. Ahora viene la generación del apartado dentro del frontal en el que se permite cambiar la apariencia de los nodos y aristas del grafo. Para ello, creamos un elemento de tipo *select* en el que con el evento *on* se maneja la entrada recibida por el usuario para poder actualizar el nuevo color que tendrá los nodos o aristas del grafo. En cuanto al nuevo tamaño del radio de los nodos o el tamaño del grosor de las aristas, el elemento que creamos es de tipo *input* pero siguiendo la misma estrategia de usar el evento *on* para actualizar esos nuevos tamaños acorde a la entrada introducida por el usuario.

A continuación, se va a explicar los distintos manejadores implementados para dotar de interacción a los nodos y enlaces por parte del usuario.

En primer lugar, el manejador encargado de cambiar la apariencia tanto del color como del grosor de un enlace al pinchar sobre él (evento *click*). Para ello, lo primero ha sido asociar el evento *click* en la creación e inserción de cada enlace al contenedor SVG mencionado anteriormente. Este manejador añade un nuevo *div* al cuerpo del template, en donde añadimos un elemento de tipo *select* para que el usuario pueda seleccionar el nuevo color y se actualiza mediante el evento *change* en el que dentro recogemos el color seleccionado y se lo asignamos a ese enlace cambiando el valor de su propiedad *stroke*. En cuanto al grosor de dicho enlace, se añade el elemento *input* para que el usuario pueda introducir el nuevo grosor y se actualiza mediante el evento *change* en el que en su interior se recoge el valor elegido por el usuario y se cambia el valor del atributo *stroke-width* del enlace.

En segundo lugar, el manejador que se encarga de mostrar la información asociada a cada nodo al pasar el cursor por encima de él (evento *mouseover*). Aquí lo primero es recoger la información asociada al nodo seleccionado para, posteriormente, mostrarla en un elemento de tipo *div*, que se representa mediante el color que tiene asociado ese nodo, en el que se van añadiendo elementos de tipo *p* por cada registro de información que queremos mostrar de dicho nodo. Finalmente, usando esa información, nos recorremos el grafo para cambiar el color de ese nodo, sus enlaces y los nodos con los que está asociado para diferenciarlos visualmente de los demás nodos y enlaces del grafo haciendo así que el usuario, de un vistazo, sepa que conjunto de nodos y enlaces se están analizando dentro de todo el grafo. Estos cambios se realizan modificando los valores de los atributos *stroke* y *stroke-opacity* para los enlaces. Para los nodos los atributos que cambiamos su valor serían *fill* y *fill-opacity*.

En tercer lugar, el manejador que se encarga de eliminar la información asociada a cada nodo al pasar el cursor por encima de él. Esto se activa al quitar el cursor fuera de ese nodo (evento *mouseout*). En este caso, lo único que hay que hacer es revertir lo que ha generado el evento *mouseover*. Para ello, se vuelve a asignar el color que tenían inicialmente los nodos y enlaces cambiando los valores de los mismos atributos mencionados al final del párrafo anterior.

El último manejador es el encargado de cambiar la apariencia tanto del color como del tamaño del radio de un nodo al pinchar sobre él (evento *click*). Lo primero ha sido asociar el evento *click* en la creación e inserción de cada nodo al contenedor SVG. El manejador de este evento añade un nuevo elemento de tipo *div* al cuerpo del template, donde se añade un elemento de tipo *select* para que el usuario pueda seleccionar el nuevo color y se actualiza mediante el evento *change* en el que dentro se obtiene el color seleccionado y se asigna a ese nodo cambiando el valor de su propiedad *fill*. Para cambiar el radio de dicho nodo, se ha añadido un elemento de tipo *input* para que el usuario pueda introducir el nuevo tamaño de radio y se actualiza mediante el evento *change* en el que en su interior se recoge el valor elegido por el usuario y se cambia el valor del atributo *r* del nodo.

Para realizar la implementación de todo esto se ha usado el lenguaje de programación JavaScript y la librería D3.js escrita en el mismo lenguaje de programación.

Ahora se va a explicar en que consiste cada funcionalidad implementada en este proyecto:

- **Usuario retweetea a otro usuario:** Esta funcionalidad representa en forma de grafo los usuarios a los que ha retweeteado el usuario elegido. A su vez, realiza la misma operación para los usuarios a los que el usuario elegido ha retweeteado algún tweet y los muestra en dicho grafo.
- **Usuario retweetea un tweet escrito por un usuario:** Esta funcionalidad representa en forma de grafo los usuarios a los que ha retweeteado el usuario elegido mostrando también los nodos tweet sobre los que los usuarios han realizado un retweet. A su vez, realiza la misma operación para los usuarios a los que el usuario elegido ha retweeteado algún tweet y los muestra en dicho grafo.
- **Usuario le da like a otro usuario:** Esta funcionalidad representa en forma de grafo los usuarios a los que ha dado “like” el usuario elegido. A su vez, realiza la misma operación para los usuarios a los que el usuario elegido ha dado “like” algún tweet y los muestra en dicho grafo.
- **Usuario le da like a un tweet de un usuario:** Esta funcionalidad representa en forma de grafo los usuarios a los que ha dado “like” el usuario elegido mostrando a su vez los nodos de los tweet a los que los usuarios han dado “like”. A su vez, realiza la misma operación para los usuarios a los que el usuario elegido ha dado “like” y los muestra en dicho grafo.
- **Seguidores de un usuario:** Esta funcionalidad representa en forma de grafo los seguidores que tiene un usuario en particular. A su vez, realiza la misma operación para los usuarios que son seguidores del usuario en particular.
- **Retweets de un hashtag concreto:** Esta funcionalidad representa en forma de grafo los retweets que han realizado distintos usuarios para los tweets en los que se ha usado un hashtag en concreto.

4.3. Funcionalidades comunes

En todas y cada una de las funcionalidades mencionadas anteriormente se tiene acceso a lo siguiente:

- **Información del grafo**
  Este apartado nos muestra un recuento del número de nodos y enlaces existentes en el grafo.
- **Leyenda del grafo**
  Muestra una leyenda de los nodos existentes del grafo con los nombres de los usuarios y su color asignado dentro del grafo.
- **Cambiar apariencia de los nodos**
  Funcionalidad que nos permite cambiar la apariencia de los nodos del grafo, tanto su color como su radio. Al confirmar los cambios, estos se verán reflejados en la leyenda mencionada anteriormente, adaptándose así a su nueva apariencia.
- **Cambiar apariencia de los enlaces**
  Funcionalidad que nos permite cambiar la apariencia de los enlaces del grafo, tanto su color como su grosor.
- **Cambiar apariencia de un nodo**
  Funcionalidad que nos permite cambiar la apariencia de un nodo en concreto, tanto su color como su radio. Para usar esta funcionalidad es necesario pinchar (evento “onclick”) sobre el nodo que queremos cambiar su apariencia. Al confirmar los cambios, estos se verán reflejados en la leyenda mencionada anteriormente, adaptándose así a su nueva apariencia.
- **Cambiar apariencia de un enlace**
  Funcionalidad que nos permite cambiar la apariencia de un enlace en concreto, tanto su color como su grosor. Esta funcionalidad se activa pinchando (evento “onclick”) sobre el enlace en cuestión que queremos cambiar su apariencia.
- **Evento "mouseover"**
  Al pasar el cursor por encima de un nodo del grafo, este se “activa” cambiando su color inicial por otro color por defecto, así como, los nodos con los que se relaciona. A su vez, debajo del grafo, se muestra información asociada a este nodo. Esta información varía en función de la funcionalidad que estemos consultando.
- **Evento "mouseout"**
  Al quitar el cursor sobre un nodo al que previamente pusimos el cursor encima de él, este se “desactiva” cambiando el color por defecto y volviendo así a su color inicial, así como, los nodos que estuvieran relacionados mediante enlaces con este.

4.4. Obtención Datos

En este apartado se va a comentar como ha sido el proceso de obtención de los datos de Twitter.

En primer lugar, ha sido necesario obtener las credenciales de acceso a la API, que posteriormente han permitido implementar una pequeña funcionalidad que permite obtener datos relativos a la red social Twitter.

Para obtener las credenciales, es necesario acceder al centro de desarrolladores de Twitter y registrar una aplicación. Una vez registrada, tendremos acceso al *Consumer Key* y al *Consumer Secret*.

Para la implementación de la funcionalidad de adquisición de datos se ha decidido utilizar *Twython* [14], una librería de Python que hace envoltorio a la api de Twitter, simplificando el proceso de autenticación y obtención de los datos.

Da la posibilidad de autenticarse tanto utilizando *OAuth1* como *OAuth2*, el primero de ellos permite escribir y leer datos y el segundo únicamente leer, en este caso, se ha utilizado la segunda, ya que el único objetivo era obtener datos para posteriormente pintarlos utilizando la librería de JavaScript implementada.

Para la autenticación únicamente se han realizado tres llamadas a funciones 4.1:

```
twitter = Twython(APP_KEY, APP_SECRET, oauth_version=2)
ACCESS_TOKEN = twitter.obtain_access_token()

twitter = Twython(APP_KEY, access_token=ACCESS_TOKEN)
```

Figura 4.1: Autenticación API Twitter

Con la primera de ellas se crea un objeto de la clase Twython pasándole las claves provistas por Twitter y la versión de autenticación, en la segunda obtenemos el token y en la última utilizando la key y el token se finaliza el proceso.

Una vez finalizado el proceso anterior tenemos un objeto de la clase Twython desde el que podemos llamar a las distintas funciones que nos provee la librería. En el caso de este trabajo hemos utilizado las siguientes:

- **get_followers_list(...)**
  Permite obtener la lista de seguidores del usuario indicado en el parámetro *screen_name*.
- **get_user_timeline(...)**
  Esta función sirve para obtener los tweet y retweets del usuario que se le pasa como parámetro.
- **get_favorites(...)**
  Con esta llamada obtenemos los likes del usuario especificado.
- **search(...)**
  Permite obtener una colección de tweets que corresponda con una expresión enviada como párametro. En este caso, ha sido utilizada para obtener la colección de tweets de un hashtag enviado como parámetro.

4.5. Aspectos de Código Relevantes

A continuación, vamos a mostrar y explicar algunos aspectos relevantes del código implementado:

4.5.1. Relación entre datos y su representación

En la imagen 4.2 mostramos cómo establecemos la relación entre los datos y su representación.

```
var edge = svg.append("g")
    .attr("stroke", "#999")
    .attr("stroke-opacity", 0.6)
    .selectAll("line")
    .data(edges)
    .join("line")
    .attr("id", function(d, i) { return 'edge' + i })
    .on("click", handleMouseClickEdge);
```

Figura 4.2: Relación datos-representación enlaces

En primer lugar, seleccionamos el svg definido (mediante la variable svg) a la que añadiremos el elemento *g* que sirve para definir un contenedor en el cual agrupar objetos y que permite que las transformaciones aplicadas a este elemento son realizadas sobre los hijos de dicho elemento. En este caso, los enlaces del grafo.

En segundo lugar, con el método *selectAll("line")*, accedemos al contenedor para seleccionar todos los elementos de tipo line (en este momento ninguno porque todavía no se han creado).

En tercer lugar, con el método *data(edges)*, realizamos la asociación de cada objeto contenido en el array *edges* con la selección realizada (inicialmente vacía).

Por último, con el método *join("line")*, lo que conseguimos es que para cada objeto "libre", añadimos un elemento de tipo *line*. Además, la añadimos con un color y opacidad determinadas. Así como, le asignamos un id a la arista creada.

En cuanto a los nodos 4.3, seguimos el mismo patrón recientemente explicado:

La única diferencia con respecto a las aristas, es que a los nodos le tenemos que añadir el atributo *r* que representa el tamaño del radio de dicho nodo.

Cabe destacar que las llamadas al método *on* en ambos casos es para que la simulación de fuerzas, que explicaremos a continuación, sirva para que dicha simulación pueda escuchar los eventos que pasas por argumentos a dicho método.

```
var node = svg.append("g")
    .attr("stroke", "#fff")
    .attr("stroke-width", 1.5)
    .selectAll("circle")
    .data(nodes)
    .join("circle")
    .attr("id", function(d, i) { return 'circle' + i })
    .attr("r", function(d) { return d.r })
    .attr("fill", randomColor)
    .on("mouseover", handleMouseOver)
    .on("mouseout", handleMouseOut)
    .on("click", handleMouseClick)
    .call(drag(simulation));
```

Figura 4.3: Relación datos-representación nodos

4.5.2. Simulación de fuerzas

En esta imagen 4.4 mostramos la creación de la simulación de fuerzas que se va a aplicar sobre el grafo creado.

```
var simulation = d3.forceSimulation(nodes)
    .force("link", d3.forceLink(edges).distance(200).id(function(d) { return d.user }))
    .force("charge", d3.forceManyBody())
    .force('collision', d3.forceCollide().radius(function(d) {
        return d.r
    }))
    .force("center", d3.forceCenter(w / 2, h / 2));

simulation.on("tick", () => {
    edge
        .attr("x1", d => d.source.x)
        .attr("y1", d => d.source.y)
        .attr("x2", d => d.target.x)
        .attr("y2", d => d.target.y);

    node
        .attr("cx", d => d.x)
        .attr("cy", d => d.y);
});
```

Figura 4.4: Simulación de fuerzas

En primer lugar, con el método *forceSimulation(nodes)*, creamos una nueva simulación que se va a aplicar sobre *nodes* (nodos del grafo).

En segundo lugar, las llamadas a los métodos *force(...)*, sirven para especificar las fuerzas de la simulación creada.

- **link**
  Establecemos una fuerza sobre los enlaces creados, además de realizar la asignación enlace-nodo mediante un id pasado como argumento al método *id* con la distancia especificada en el método *distance*.
- **charge**
  Sirve para aplicar la fuerza en muchos cuerpos, *forceManyBody()*, en este caso se aplica de igual manera entre todos los nodos.
- **collision**
  Evita que los nodos se superpongan creando una fuerza de colisión circular, *forceCollide*, con el radio especificado, *radius*, como argumento de entrada a dicha función (tamaño de radio de cada nodo).
- **center**
  Crea una fuerza de centrado haciendo así que la posición media de todos los nodos esté basada en una posición dada, *forceCenter*, en este caso basada en la posición central del tamaño del svg creado anteriormente. Ayudando así a que los nodos se mantengan en el centro de la ventana gráfica.

Por último, a pesar de que la simulación se inicia automáticamente, para escuchar eventos *tick* se realiza la llamada al método *on*. Esto sirve para poder arrastrar los nodos y enlaces por todo el svg, que la simulación recalcule las nuevas posiciones de los nodos y enlaces, quedándose así registradas las nuevas coordenadas del lugar donde sueltas el cursor.

5

Prueba y resultados

En este apartado se va a mostrar con diferentes ejemplos las distintas funcionalidades implementadas para este proyecto.

En primer lugar, la pantalla inicial 5.1 es la siguiente:

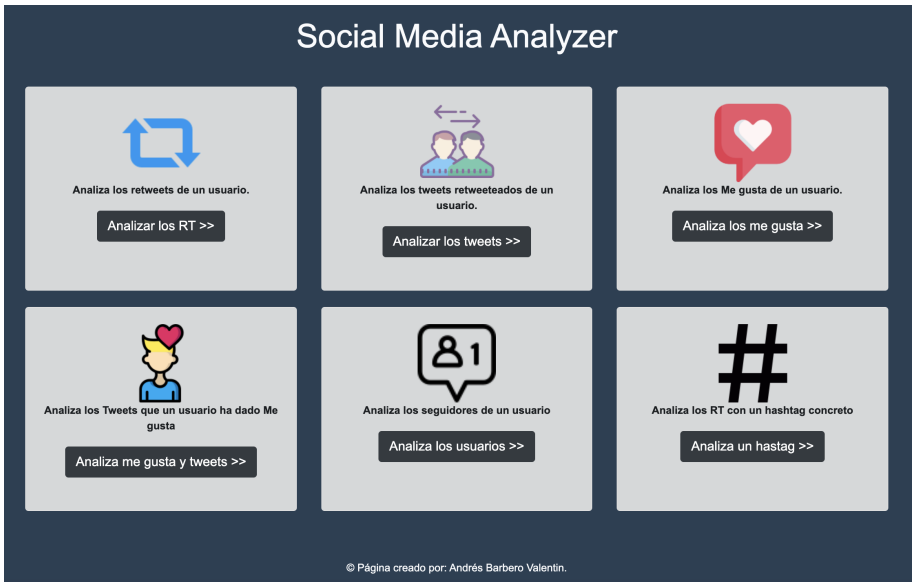

Figura 5.1: Página inicial

En donde nos encontramos con el acceso a todas las funcionalidades implementadas por lo que bastaría con pinchar en el botón asociado a cada funcionalidad según nos interese para poder entrar a ella.

Pinchando en el primer botón accedemos a la funcionalidad *Usuario retweetea a otro usuario* como vemos en 5.2

Donde nos muestra un grafo de los usuarios a los que el usuario andresbv5 ha realizado algún retweet. Los nodos representan usuarios y los enlaces representan que ha habido un retweet por parte de esos usuarios que mantienen esa relación.

Empezando por la parte de arriba, tenemos un pequeño panel horizontal en el que podemos ir

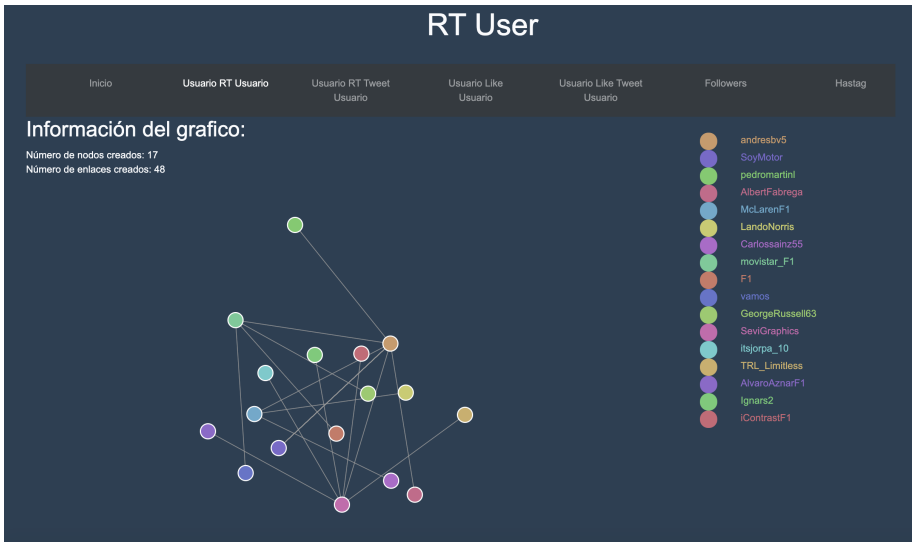

Figura 5.2: Usuario retweetea a otro usuario

navegando entre las distintas funcionalidades sin tener que retroceder al inicio.

Un poco mas abajo a la izquierda, podemos ver la información del gráfico que consta del número de nodos y del número de enlaces que tiene el grafo que se está mostrando.

A la derecha del grafo, nos encontramos con la leyenda del mismo en el que se muestra la información de los usuarios que están involucrados, mostrando su nombre de usuario y su color asociado.

Si movemos el cursor sobre uno de los nodos del grafo, *andresbv5* en este caso, vemos el funcionamiento del evento *mouseover* 5.3 implementado:

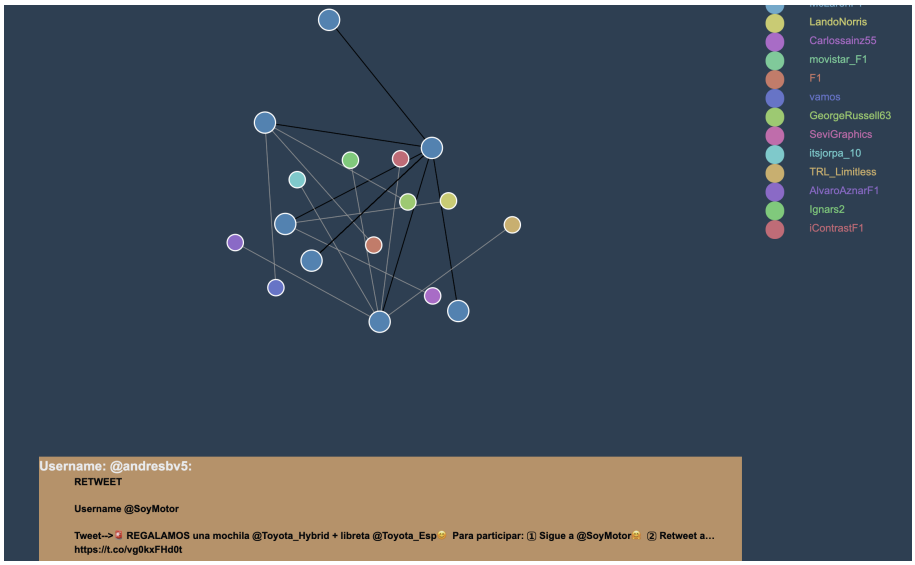

Figura 5.3: Evento MouseOver - Usuario retweetea a otro usuario

Como podemos observar, el nodo sobre el que nos hemos situado encima con el cursor y sus

enlaces asociados cambian de color para resaltar el propio nodo seleccionado y sus conexiones, diferenciándolos así de los demás nodos del grafo.

Al trasladar el cursor fuera del nodo, entra en escena el manejador del evento *mouseout* que vuelve a dejar en el estado original el grafo, es decir, el nodo seleccionado y sus enlaces y nodos asociados con los colores que tenían inicialmente.

Justo debajo podemos ver como se despliega la información asociada al nodo seleccionado, que mas detalladamente podemos ver en 5.4 y 5.5:

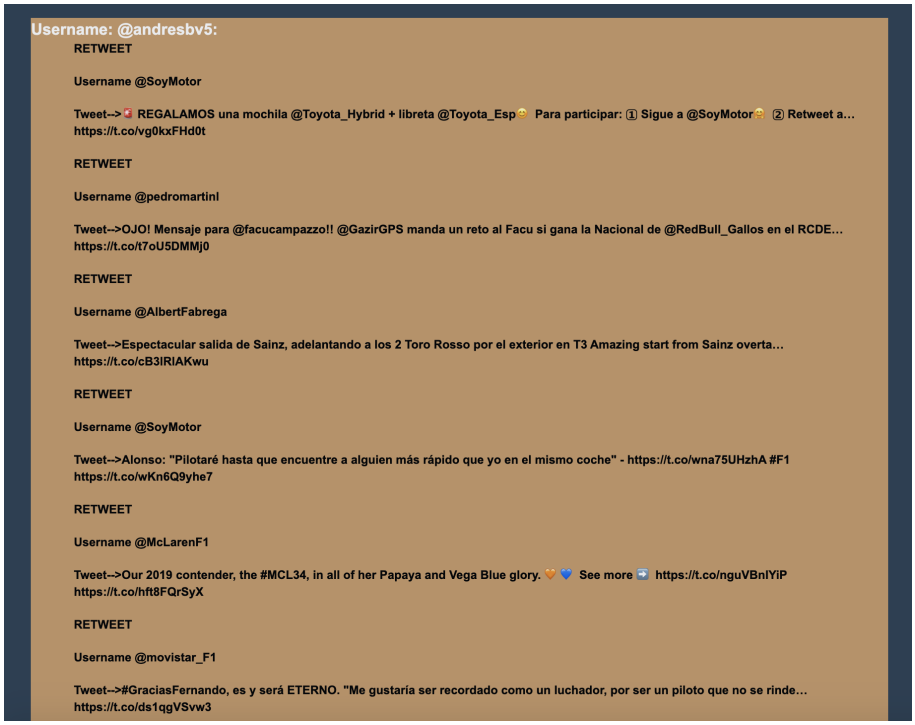

Figura 5.4: Información Nodo - Usuario retweetea a otro usuario

Donde podemos apreciar los retweets realizados por el usuario *andresbv5*, a que usuario le hizo el retweet y el contenido del tweet retweeteado. En cualquier momento que queramos dejar de visualizar esta información, pinchamos en el botón *Cerrar*.

En cuanto a la funcionalidad de *cambio de apariencia de nodos y aristas* 5.6, esta se sitúa justo debajo del grafo:

Donde podremos elegir cambiar el color de todos los nodos y/o todas las aristas según nos convenga, así como, el tamaño de los radios de los nodos y/o el grosor de las aristas.

Cambiando los nodos al color *red* obtenemos 5.7:

Como podemos observar se han cambiado de color todos los nodos del grafo al color seleccionado, por lo que también su leyenda se actualiza con el mismo color (situada arriba a la derecha).

Figura 5.5: Información Nodo 1 - Usuario retweetea a otro usuario

Figura 5.6: Cambio apariencia nodos y enlaces - Usuario retweetea a otro usuario

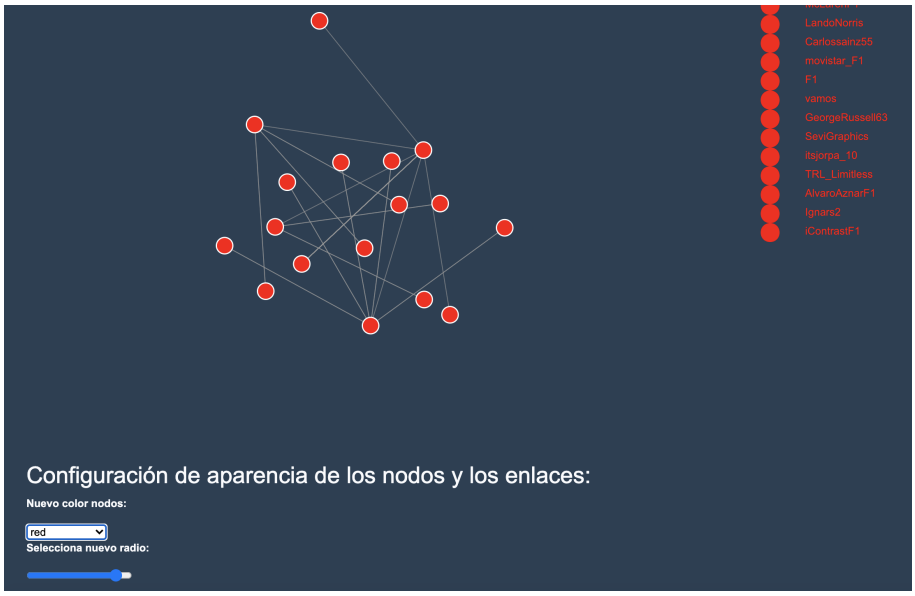

Figura 5.7: Cambio apariencia nodos - Usuario retweetea a otro usuario

Modificando el radio de los nodos ilustrado en 5.8

El resultado es la modificación del radio de los nodos, en este caso a un tamaño más pequeño que el inicial.

En cuanto al cambio de apariencia relacionada con los enlaces, podemos cambiar el color de las mismas, así como, el grosor.

Cambiando su color a *yellow* obtenemos el siguiente resultado 5.9

Y vemos como se aplica ese cambio en el nodo del grafo transformando en todas las aristas al nuevo color seleccionado.

Ahora procedemos a modificar el grosor de las mismas y obtenemos 5.10

Como podemos ver se han cambiado los grosores de todas las aristas al nuevo grosor seleccionado.

Pero quizás solo queremos aplicar estos cambios de apariencia a un nodo en concreto, para ello también tenemos dicha posibilidad pinchando sobre el nodo o la arista sobre el cuál queremos realizar el cambio de apariencia.

Vamos a realizar primero el cambio de apariencia para un nodo, pinchando sobre un nodo (evento *mouseclick*) nos aparece el siguiente panel 5.11:

En donde nos indica el nombre del usuario asociado a ese nodo sobre el que queremos cambiar su apariencia, al igual que antes, tenemos la opción de cambiar su color y su tamaño. Realizamos el cambio que queremos, pinchamos en el botón *Confirmar cambios* y obtenemos 5.12:

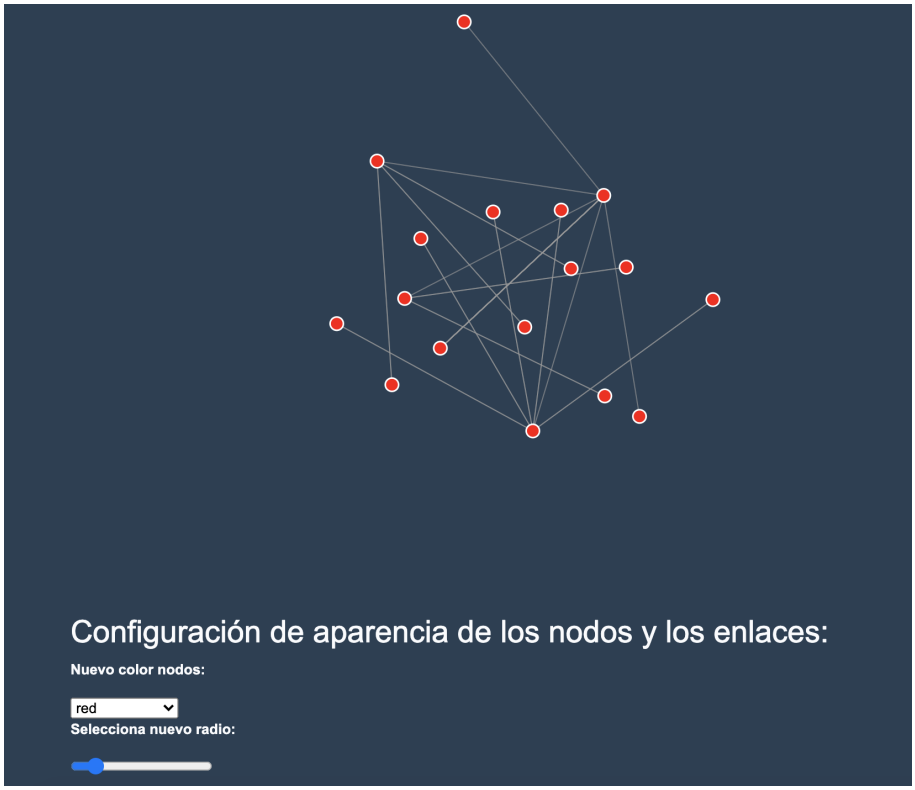

Figura 5.8: Cambio apariencia radio nodos - Usuario retweetea a otro usuario

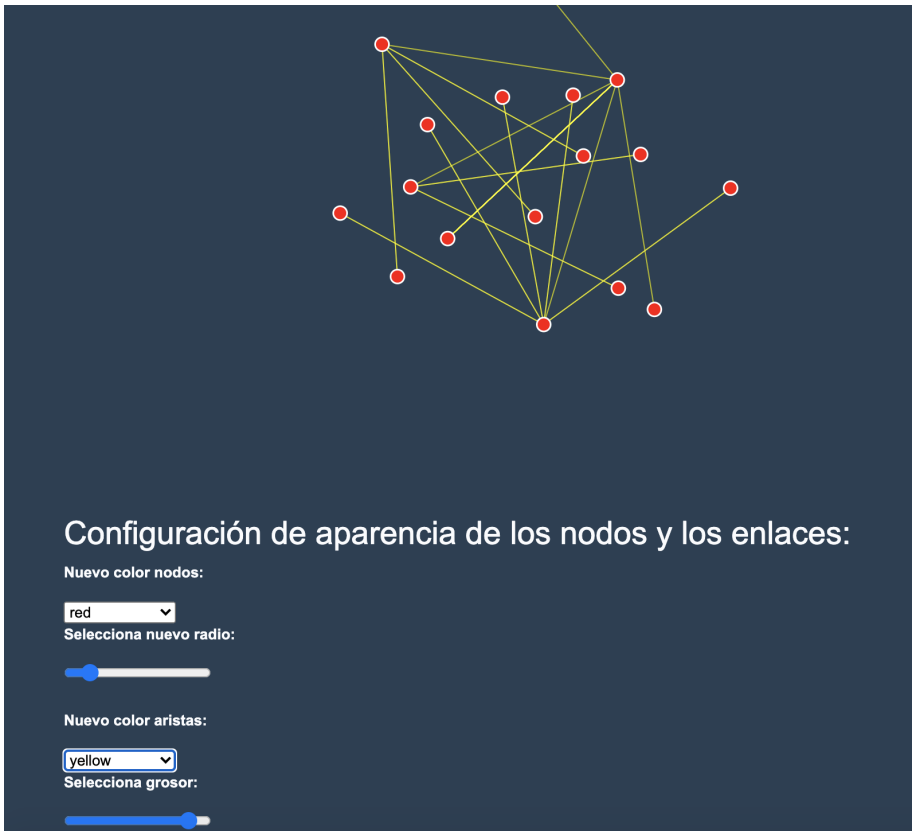

Figura 5.9: Cambio apariencia enlaces - Usuario retweetea a otro usuario

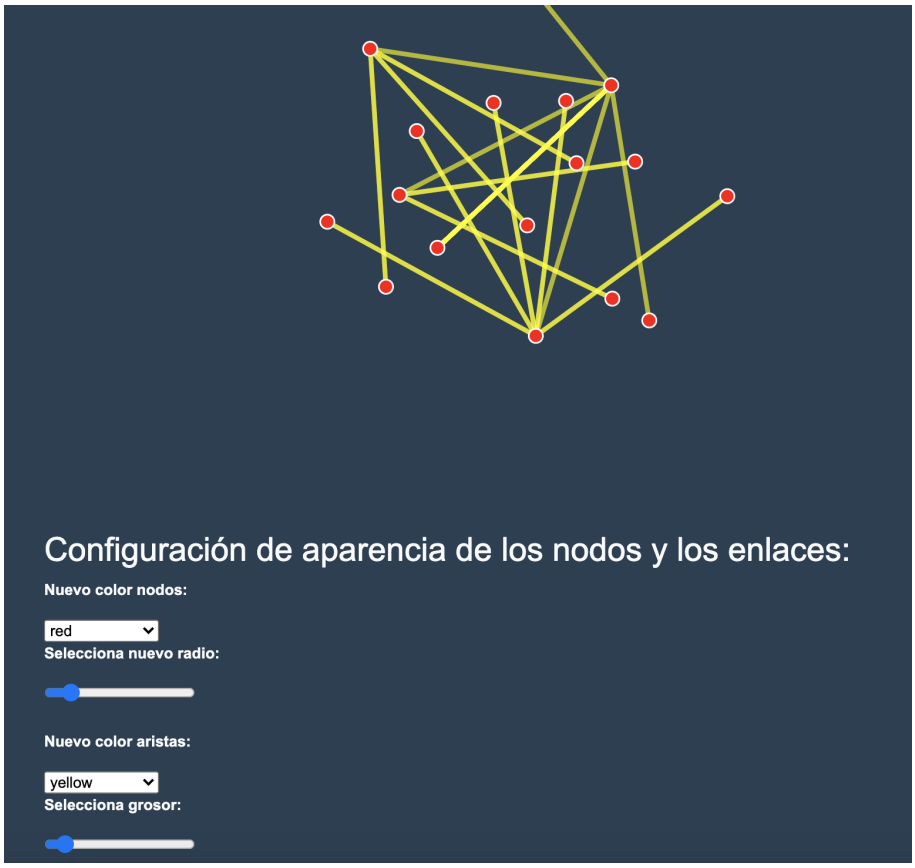

Figura 5.10: Cambio apariencia grosor enlaces - Usuario retweetea a otro usuario

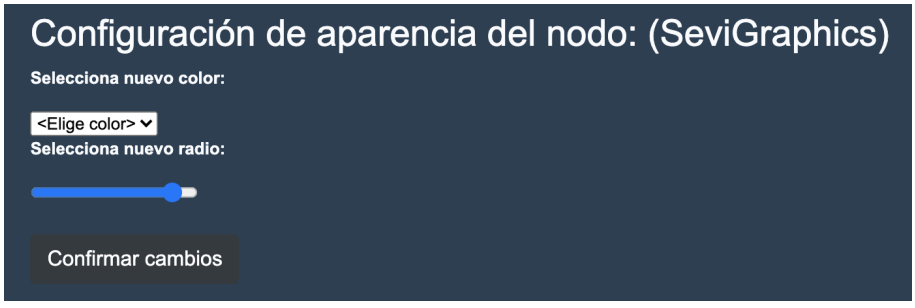

Figura 5.11: Panel cambio apariencia nodo - Usuario retweetea a otro usuario

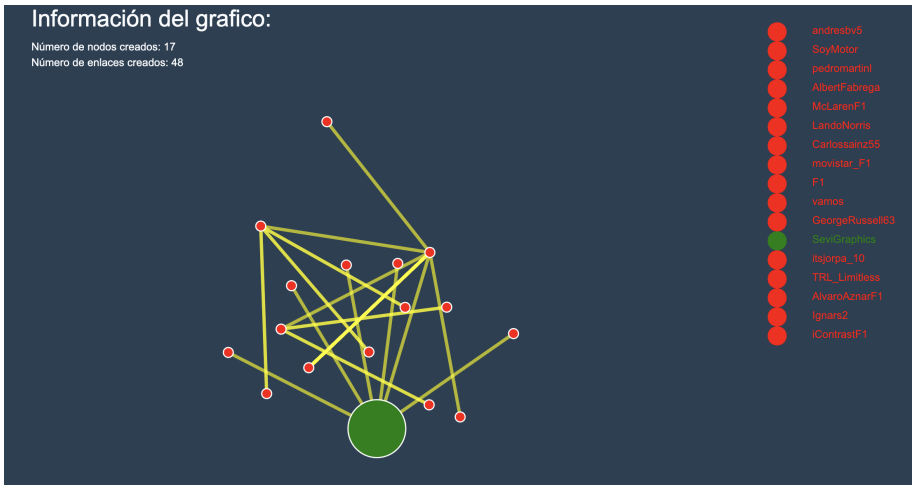

Figura 5.12: Grafo después cambiar apariencia nodo - Usuario retweetea a otro usuario

Cómo podemos observar, el nodo ha cambiado al color y tamaño seleccionados y, por tanto, también se produce actualización en la leyenda del grafo.

Ahora vamos a modificar la apariencia de una arista en concreto, para ello pinchamos sobre la arista (evento *mouseclick*) que queremos cambiar su apariencia y volvemos a obtener el mismo panel anterior pero relacionado con la apariencia de dicha arista 5.13:

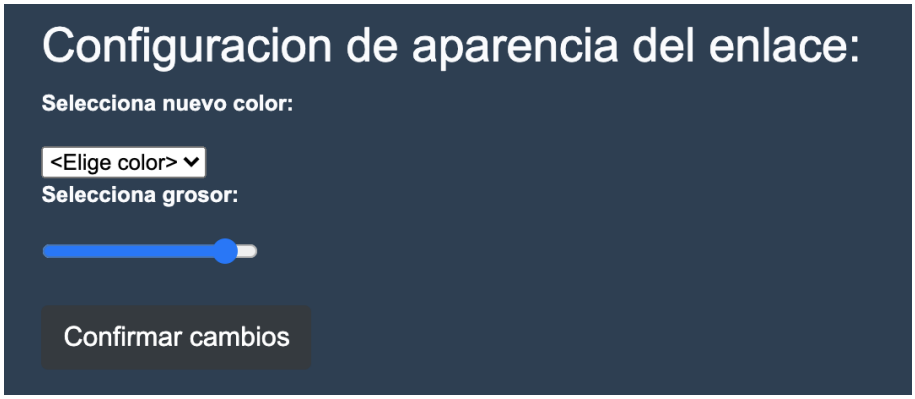

Figura 5.13: Panel cambio apariencia enlace - Usuario retweetea a otro usuario

Elegimos el color y grosor que queramos, le damos al botón *Confirmar cambios* y obtenemos lo siguiente 5.14:

Vemos que los cambios se han aplicado correctamente a nuestras preferencias seleccionadas.

Ahora vamos a mostrar pruebas y resultados de las demás funcionalidades implementadas centrándonos, sobre todo, en las diferencias o aspectos nuevos con respecto a lo explicado en el ejemplo anterior.

Accediendo a la segunda funcionalidad, *Usuario retweetea un tweet escrito por un usuario*, pinchando en el segundo botón de la página inicial obtenemos el siguiente grafo 5.15:

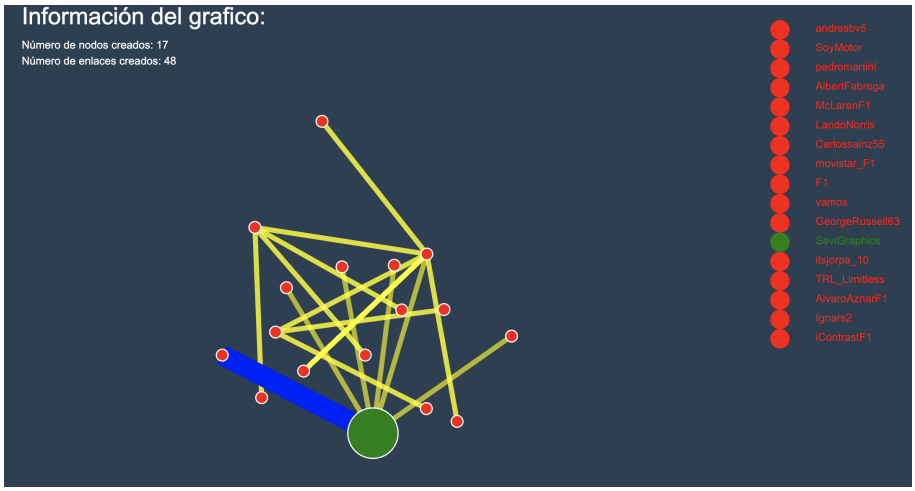

Figura 5.14: Grafo después cambiar apariencia enlace - Usuario retweetea a otro usuario

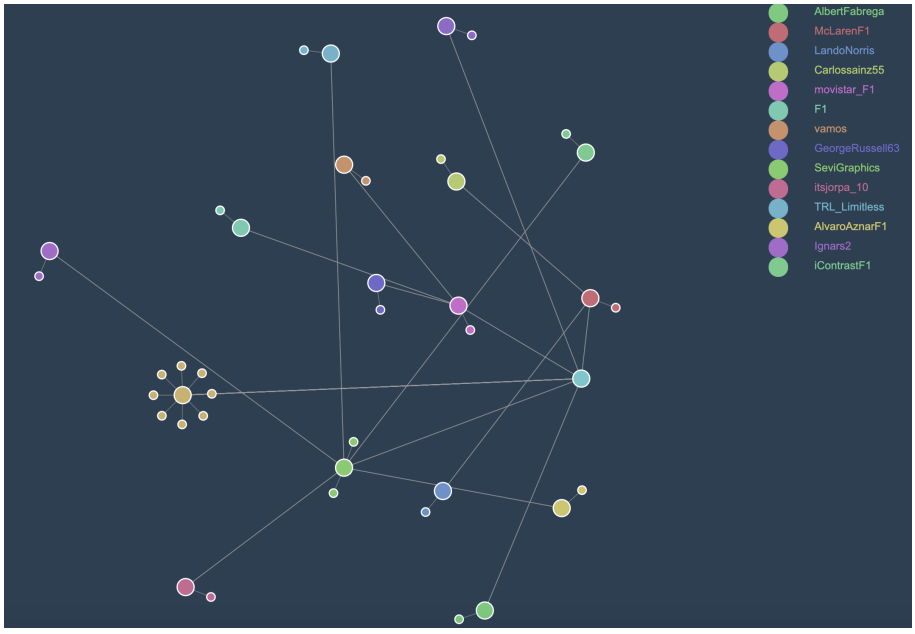

Figura 5.15: Usuario retweetea un tweet escrito por un usuario

En este caso, aparte de mostrar como nodos los nodos usuario, mostramos también como nodos los tweets involucrados. Como podemos comprobar, estos *nodos tweet* van asociados al usuario que escribió dicho tweet mediante el mismo color que tiene asociado el usuario en cuestión y con un tamaño inferior a los *nodos usuario*.

Al igual que antes, a la derecha del grafo tenemos la leyenda del mismo en el que nos muestra los nombres de los distintos usuarios con sus respectivos colores asociados.

Llevando el cursor sobre otro usuario distinto al anterior, *SoyMotor* en este caso, obtenemos la siguiente información asociada 5.16:

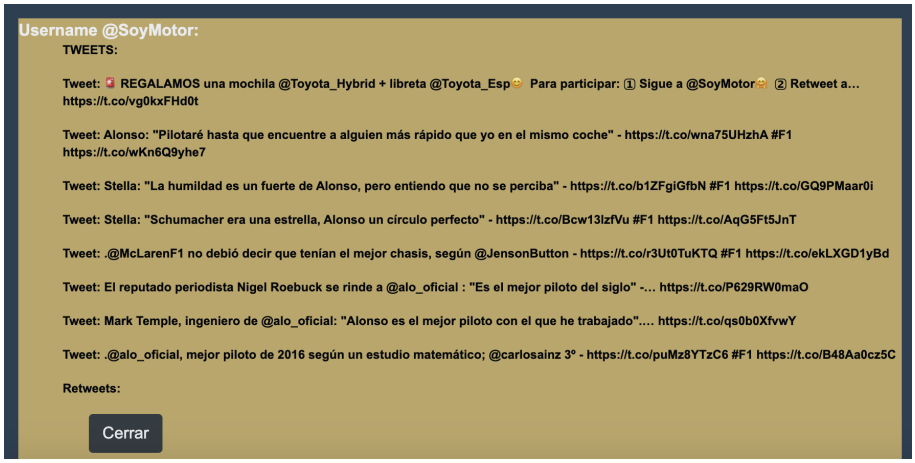

Figura 5.16: Información Nodo - Usuario retweetea un tweet escrito por un usuario

Vemos el usuario seleccionado, los tweets asociados a dicho usuario y si ha realizado algún retweet a algún usuario del grafo, en este caso no ha realizado ningún retweet.

Ahora seleccionando algún tweet de este usuario, obtenemos la siguiente información 5.17:

En primer lugar, vemos la selección que hemos hecho mediante el cambio de su color asociado mostrándonos visualmente así el usuario al que está asociado dicho tweet.

En cuanto a la información mostrada, podemos ver de quién es el tweet, así como, el contenido del mismo y que usuarios del grafo han retweeteado dicho tweet, en este caso, el usuario *andresbv5*.

Realizamos algunos cambios de apariencia en los nodos y enlaces del grafo para demostrar el correcto funcionamiento obteniendo lo siguiente 5.18:

Como vemos se han aplicado correctamente todos los cambios de apariencia que hemos seleccionado, a destacar que al cambiar el color de un *nodo tweet* también el cambio se propaga al usuario propietario de ese tweet dando coherencia a los cambios aplicados, así como, en la leyenda del grafo.

Accediendo a la tercera funcionalidad, *Usuario le da like a otro usuario*, pinchando en el tercer botón de la página inicial, obtenemos el siguiente grafo 5.19:

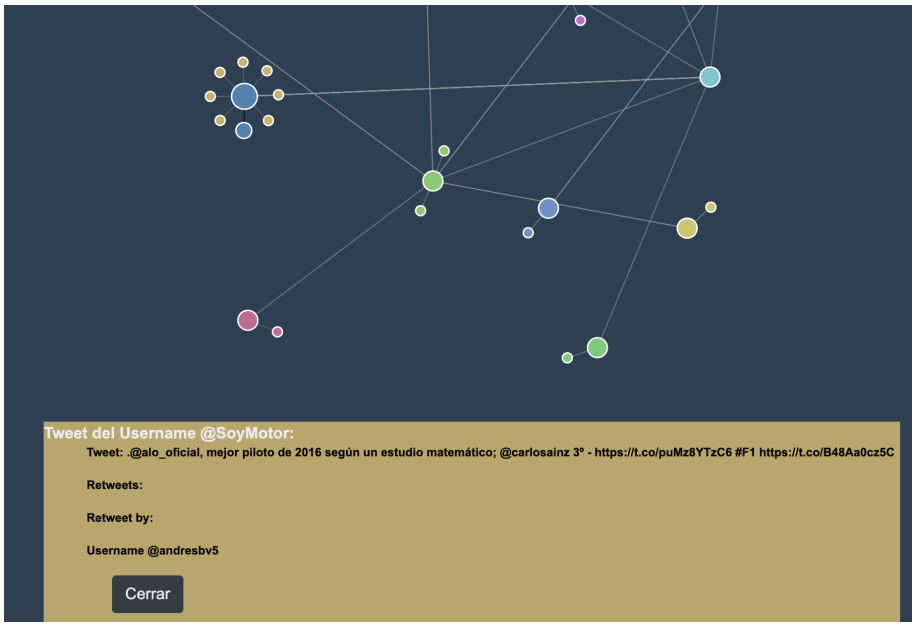

Figura 5.17: Información Nodo Tweet - Usuario retweetea un tweet escrito por un usuario

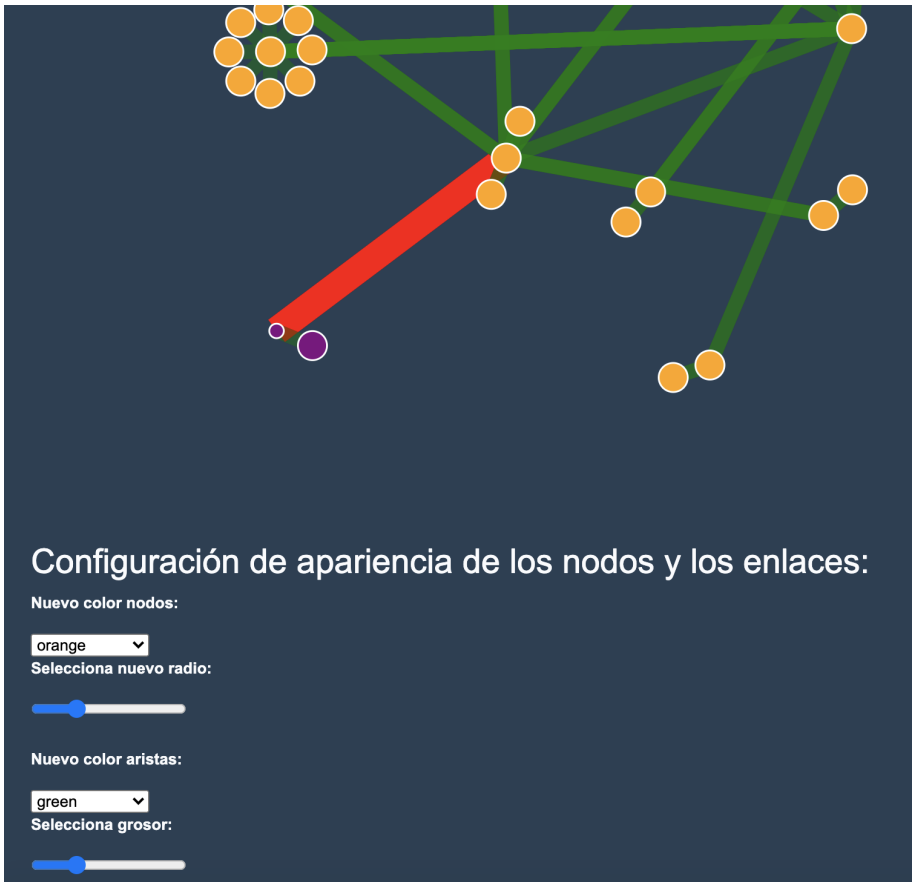

Figura 5.18: Grafo después cambiar apariencia - Usuario retweetea un tweet escrito por un usuario

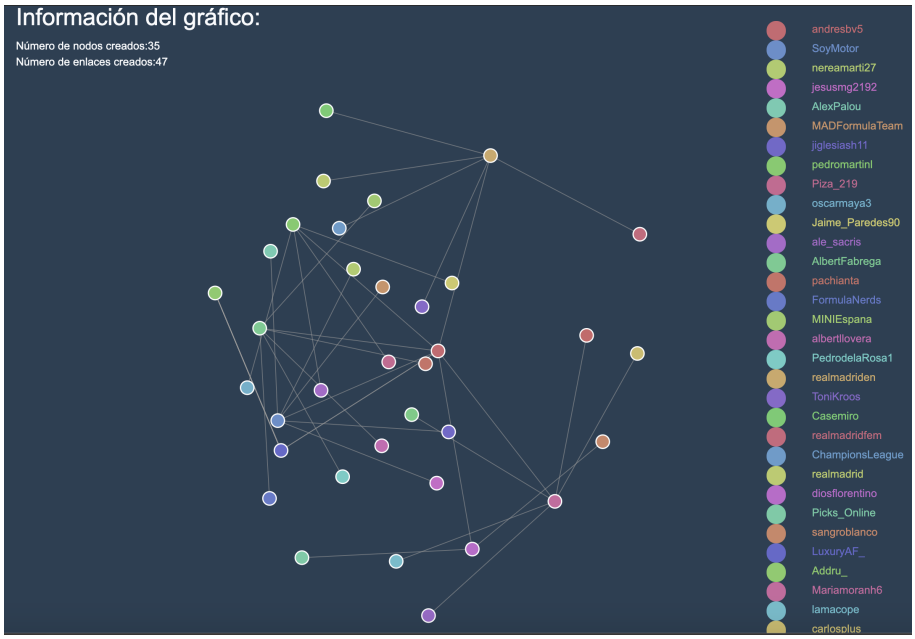

Figura 5.19: Usuario le da like a otro usuario

Para esta funcionalidad se muestra un grafo en el que vemos los usuarios a los que el usuario *andresbv5* ha dado like a algún tweet de dichos usuarios.

A la derecha del grafo, se muestra la leyenda del mismo indicando los usuarios involucrados en el mismo con su nombre de usuario y su color asociados.

En lo referente a la información asociada a cada nodo, llevamos el cursor sobre el nodo (evento *mouseover*) que nos interesa y la información que obtenemos viene representada en 5.20 y 5.21:

Donde vemos el usuario del nodo que hemos seleccionado para ver su información asociada. En este caso, muestra el usuario seleccionado (*andresbv5*) mostrando también los tweets a los que ha dado like, así como, el propietario de ese tweet y el texto del mismo.

Realizamos algunos cambios de apariencia en los nodos y enlaces del grafo para demostrar el correcto funcionamiento obteniendo lo siguiente 5.22:

Nuevamente vemos que todos los cambios de apariencia que queríamos se han aplicado correctamente.

Accediendo a la cuarta funcionalidad, *Usuario le da like a un tweet de un usuario*, pinchando en el cuarto botón de la página inicial, obtenemos el siguiente grafo 5.23:

En esta visualización, aparte de mostrar como nodos los nodos usuario, visualizamos también como nodos los tweets asociados. Como podemos apreciar, estos *nodos tweet* van ligados al usuario que escribió dicho tweet mediante el mismo color que tiene asignado el usuario en cuestión y con un tamaño inferior a los *nodos usuario*.

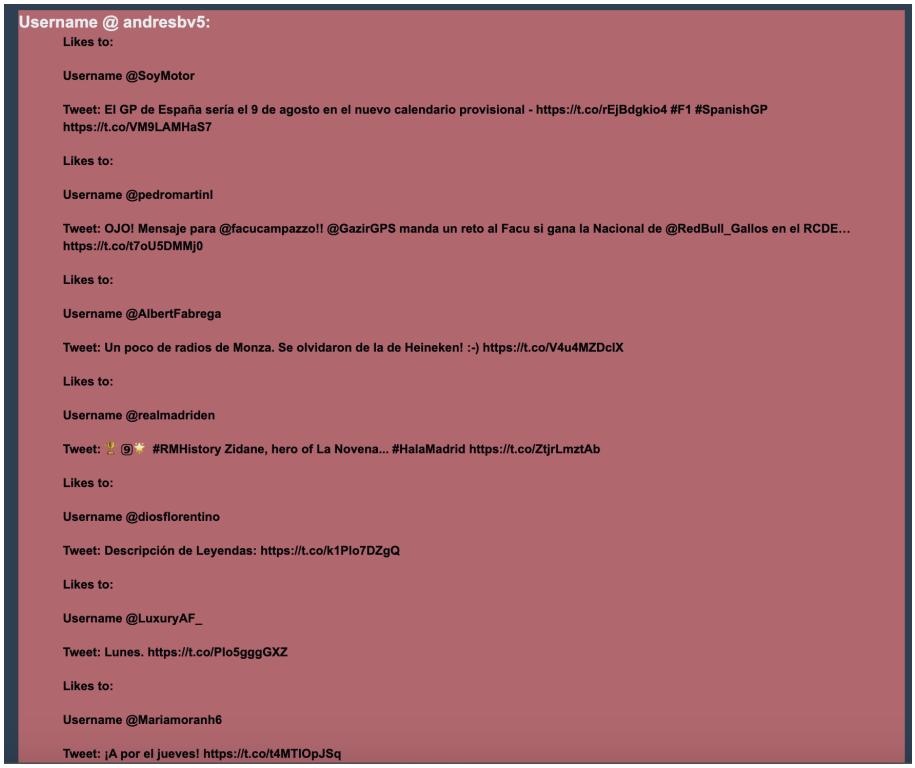

Figura 5.20: Información Nodo - Usuario le da like a otro usuario

Figura 5.21: Información Nodo 1 - Usuario le da like a otro usuario

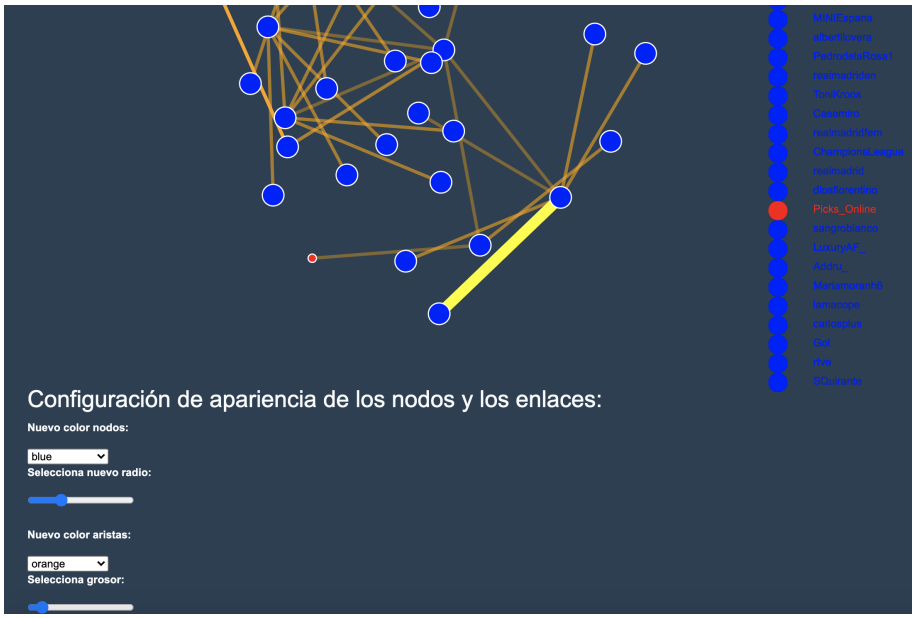

Figura 5.22: Grafo después cambiar apariencia - Usuario le da like a otro usuario

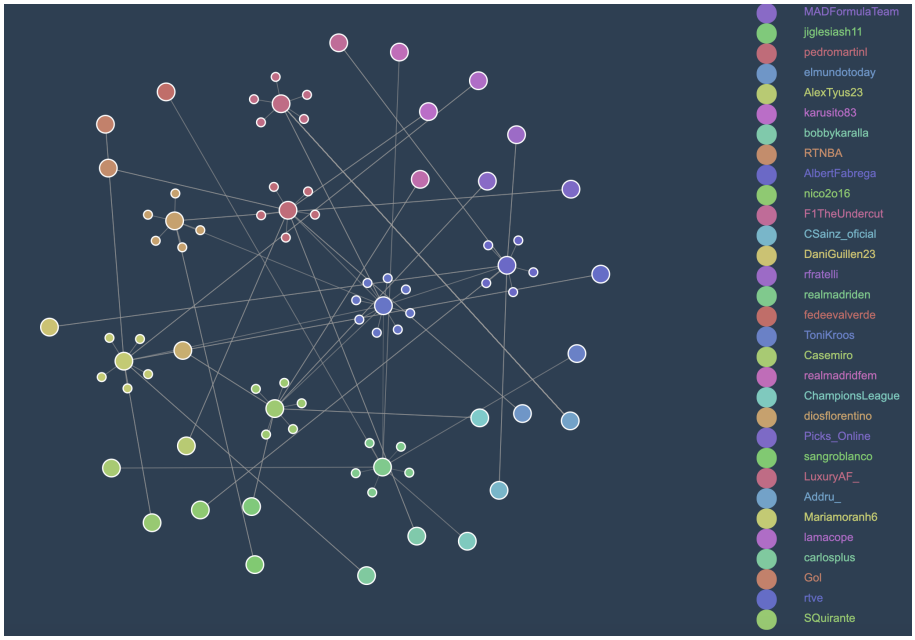

Figura 5.23: Usuario le da like a un tweet de un usuario

Como en funcionalidades anteriores, a la derecha del grafo tenemos la leyenda del mismo en el que nos muestra los nombres de los distintos usuarios con sus respectivos colores asociados.

Llevando el cursor sobre otro usuario distinto al anterior, *realmadriden* en este caso, obtenemos la siguiente información asociada viendolo en 5.24 y 5.25:

Figura 5.24: Información Nodo - Usuario le da like a un tweet de un usuario

Figura 5.25: Información Nodo 1 - Usuario le da like a un tweet de un usuario

Desgranando lo que estamos visualizando, podemos ver los tweets realizados por este usuario, así como, a que usuarios ha dado like indicando el nombre del usuario y el texto del tweet al que dio like.

Si seleccionamos algún tweet de este usuario, obtenemos la siguiente información 5.26:

Vemos como se “sombrean” el nodo tweet y el nodo usuario asociado pero también de quién es dicho tweet, su contenido y los likes que ha recibido, en este caso, del usuario *Casemiro*.

Realizamos algunos cambios de apariencia en los nodos y enlaces del grafo para demostrar el correcto funcionamiento obteniendo lo siguiente 5.27:

De nuevo vemos que todos los cambios de apariencia que queríamos se han aplicado correctamente.

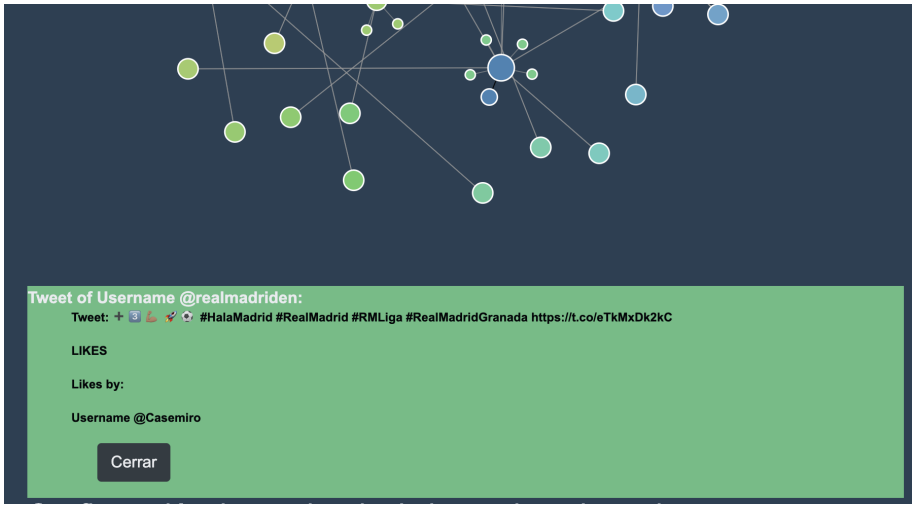

Figura 5.26: Información Nodo Tweet - Usuario le da like a un tweet de un usuario

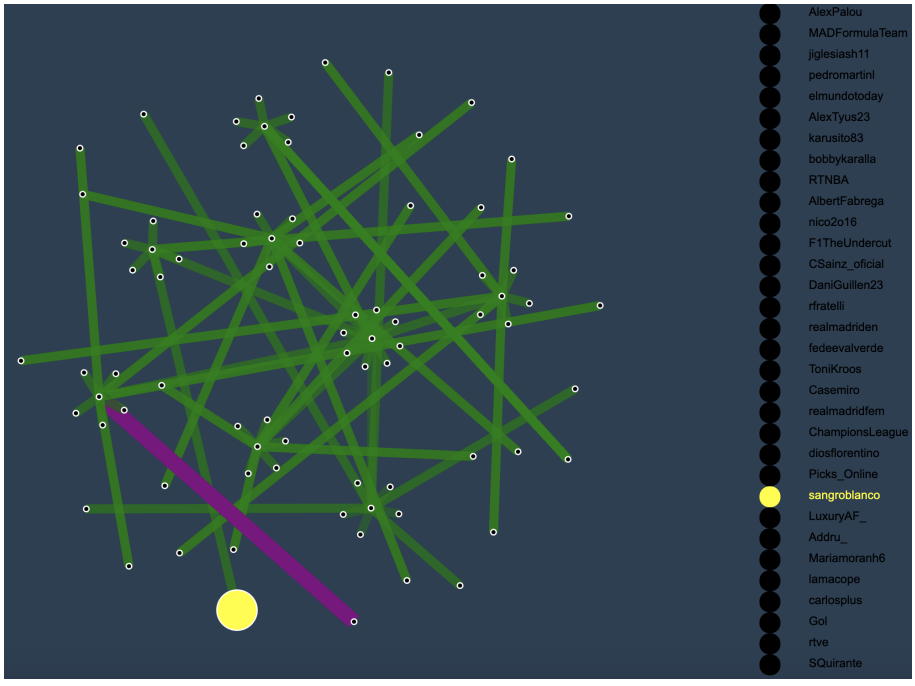

Figura 5.27: Grafo después cambiar apariencia - Usuario le da like a un tweet de un usuario

Accediendo a la quinta funcionalidad, *Seguidores de un usuario*, pinchando en el quinto botón de la página inicial, obtenemos el siguiente grafo 5.28:

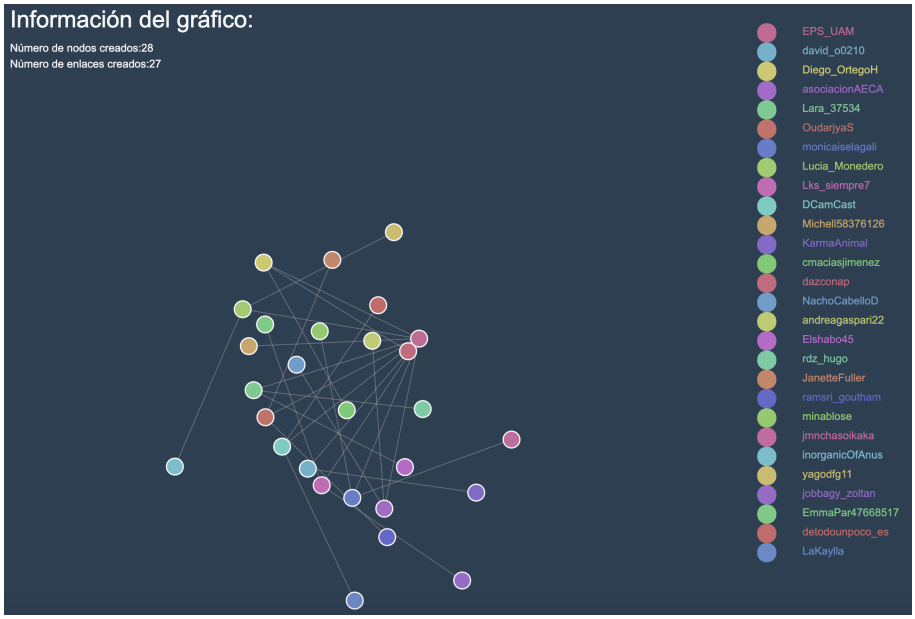

Figura 5.28: Seguidores de un usuario

Aquí podemos ver los seguidores del usuario *EPS_UAM* y los seguidores que tienen esos seguidores del usuario principal.

Como en todas las funcionalidades, a la derecha del grafo tenemos la leyenda del mismo en el que nos muestra los nombres de los distintos usuarios involucrados con sus respectivos colores.

Llevando el cursor sobre el usuario *EPS_UAM*, obtenemos la siguiente información asociada 5.29:

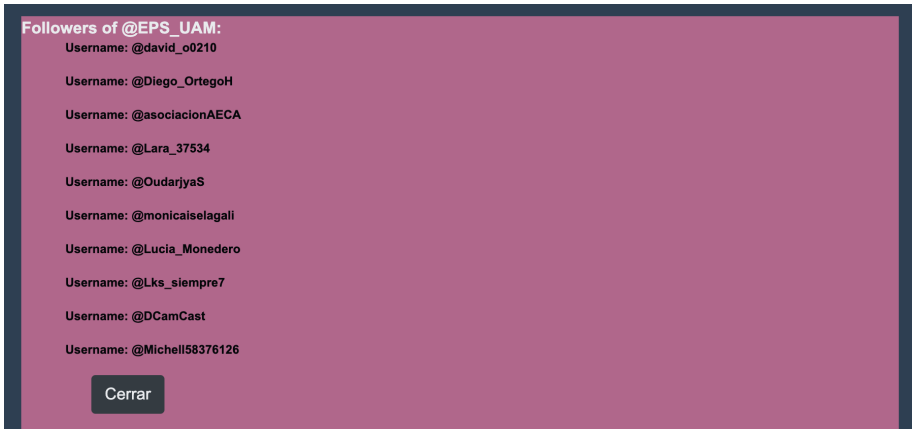

Figura 5.29: Información Nodo - Seguidores de un usuario

Donde podemos ver la lista de los seguidores del usuario mencionado mostrando el nombre de usuario de estos.

Hacemos algunos cambios de apariencia en los nodos y enlaces del grafo para demostrar el co-

rrecto funcionamiento obteniendo lo siguiente 5.30:

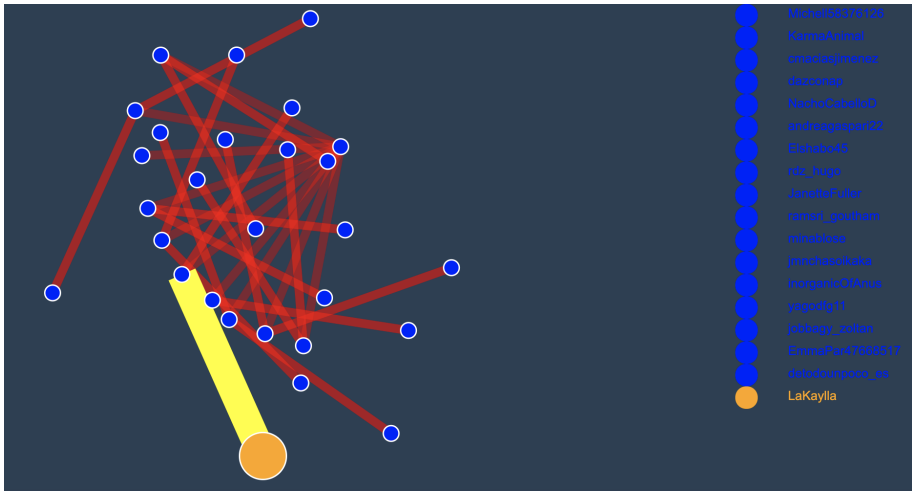

Figura 5.30: Grafo después cambiar apariencia - Seguidores de un usuario

Accediendo a la sexta funcionalidad, *Retweets de un hashtag concreto*, pinchando en el sexto botón de la página inicial, obtenemos el siguiente grafo 5.31:

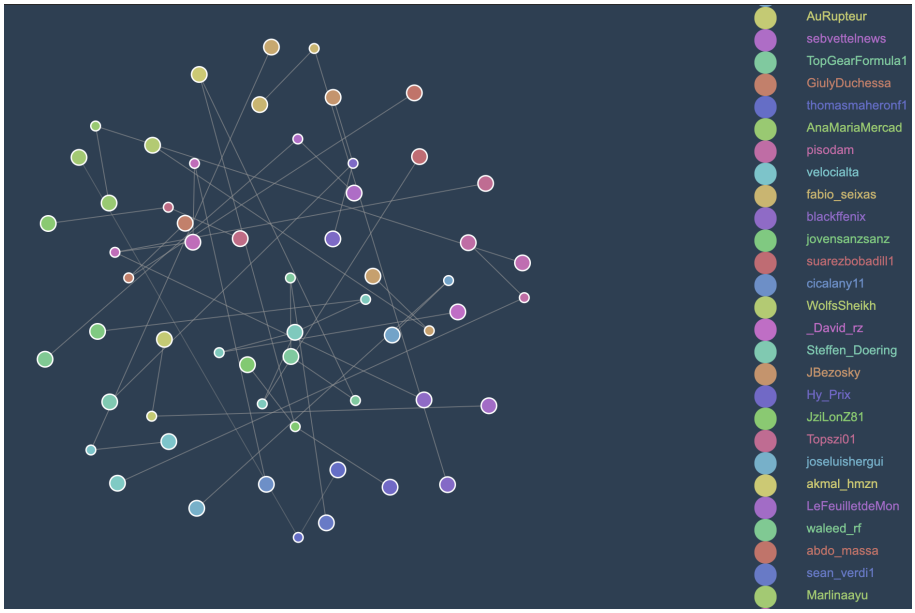

Figura 5.31: Retweets de un hashtag concreto

El hashtag analizado en este caso es *#F1*, aquí se muestran nodos de los usuarios que han realizado algún retweet a un tweet con dicho hashtag y los nodos tweet que llevan asociado como contenido el hashtag *#F1*.

Si seleccionamos cualquier nodo tweet, la información que se muestra es la siguiente 5.32:

Donde vemos el hashtag que se está analizando, el contenido del tweet donde podemos comprobar que se encuentra dicho hashtag, los retweets que se han realizado sobre ese tweet y quién lo ha

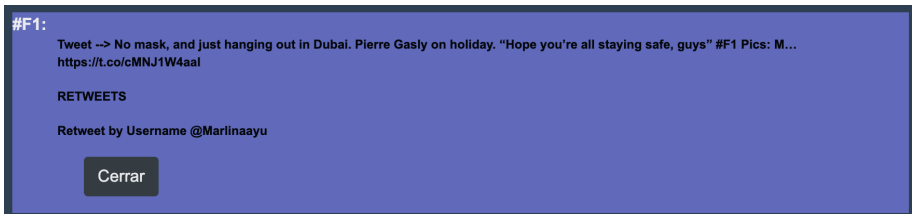

Figura 5.32: Información Nodo - Retweets de un hashtag concreto

realizado.

Por último, volvemos a realizar cambios de apariencias para demostrar que en esta funcionalidad también se pueden aplicar dichos cambios si asi lo queremos, quedándonos de la siguiente forma 5.33:

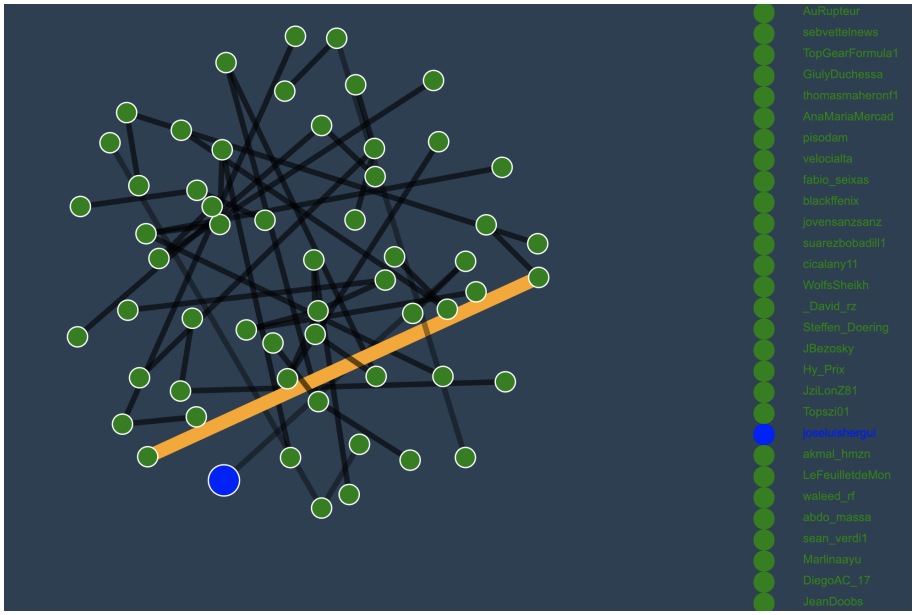

Figura 5.33: Grafo después cambiar apariencia - Retweets de un hashtag concreto

Los cambios se vuelven a aplicar correctamente a nuestra elección tomada.

Con esto damos por concluido este apartado de “Pruebas y Resultados”.

6

Conclusiones

En este último apartado se va a hacer una reflexión sobre el trabajo que se ha realizado para intentar explicar aspectos como el aprendizaje, posibles mejoras futuras, que me hubiera gustado haber hecho mejor o que me faltó hacer.

En primer lugar, respecto al aprendizaje que me llevo de realizar este trabajo, he de decir que me ha supuesto aprender a manejar nuevas tecnologías o profundizar sobre otras, las cuáles únicamente tenía ciertas nociones básicas sobre ellas y que ahora conozco más en profundidad gracias a este trabajo. A destacar diría el manejo de la biblioteca escrita en JavaScript D3.js la cual desconocía y que para manejar datos y realizar operaciones para así quitar carga a la parte del servidor es muy potente y de una manera más sencilla que si se realizara el mismo trabajo con otras herramientas. Esta biblioteca te facilita la implementación en términos de visualización pero también tiene una curva de aprendizaje muy elevada por lo que uno de los mayores esfuerzos realizados en el proyecto ha sido entender y utilizar apropiadamente dicha biblioteca.

En segundo lugar, tratando de posibles mejoras, estaría el hecho de poder abarcar más variantes de las funcionalidades implementadas para así poder realizar más tipos de análisis sobre los datos y obtener otro tipo de conclusiones gracias a ello.

En tercer lugar, en lo referente a qué me falto hacer y que se podría hacer mejor, diría que dar la posibilidad al usuario a poder introducir exactamente los datos que quiere recoger de Twitter para así dotar a la herramienta de más poder y cubrir un abanico más amplio de opciones teniendo a su disposición diferentes datos o que se pudieran recoger datos de más de una red social y entrelazarlos para enriquecer y potenciar el análisis. Partiendo de la base de que la herramienta creada se centra en la parte de visualización, la idea es que se pueda combinar con servidores más potentes y complejos ya que el servidor implementado sólo tiene como objetivo demostrar la validez de la implementación propuesta.

Considero que se podría haber hecho mejor la parte de mostrar la información de cada nodo para hacerlo visualmente más atractivo o dotar de más amplia gama de colores y diferentes texturas a los nodos y enlaces.

La mayoría de los requisitos no funcionales han sido cubiertos salvo algunos que no han podido ser probados debido a que no se ha tenido la ocasión de desplegar la herramienta en un entorno productivo ya que las pruebas realizadas se han hecho en un entorno local. El requisito no funcional más crítico sería que el servidor pueda dar respuesta a una gran concurrencia de peticiones de distintos usuarios en ciertos periodos de tiempo.

Para concluir, en general, lo desarrollado en este proyecto es una buena herramienta para permitir realizar un buen análisis sobre los datos ya que visualmente se ven muy bien, es muy intuitivo y la información que se muestra en cada funcionalidad es la más relevante.

Bibliografía

[1] P. Hong, *Practical web design: Learn the fundamentals of web design with HTML5, CSS3, bootstrap, jQuery, and vue. js*. Packt Publishing Ltd, 2018.

[2] N. Q. Zhu, *Data visualization with D3. js cookbook*. Packt Publishing Ltd, 2013.

[3] E. Meeks, *D3. js in Action*. Manning Shelter Island, NY, 2015.

[4] "An intro to rapahel js." http://raphaeljs.com. Fecha de último acceso: 12-09-2020.

[5] "jsplumb toolkit." https://docs.jsplumbtoolkit.com/toolkit/current/index.html. Fecha de último acceso: 12-09-2020.

[6] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader, "Cytoscape. js: a graph theory library for visualisation and analysis," *Bioinformatics*, vol. 32, no. 2, pp. 309–311, 2016.

[7] "Visjs documentation." https://visjs.github.io/vis-network/docs/network/. Fecha de último acceso: 12-09-2020.

[8] H. Da Rocha, *Learn Chart. js: Create interactive visualizations for the Web with Chart. js 2*. Packt Publishing Ltd, 2019.

[9] K. Cherven, *Mastering Gephi network visualization*. Packt Publishing Ltd, 2015.

[10] B. Meyer and S. Spiekermann, "skillmap: dynamic visualization of shared organizational context," *Institute of Information Systems, Humboldt University Berlin,(Feb. 20, 2006)*, vol. 13, 2006.

[11] E. Adar, "Guess: a language and interface for graph exploration," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pp. 791–800, 2006.

[12] E. Marini, "El modelo cliente/servidor," *Recuperado el*, vol. 5, 2012.

[13] M. Grinberg, *Flask web development: developing web applications with python*. ."O'Reilly Media, Inc.", 2018.

[14] "Twython 3.8.0 documentation." https://twython.readthedocs.io/en/latest/. Fecha de último acceso: 18-03-2021.