Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Componentes de Visualización Analítica para Grafos.

Autor: Jaime Cabeza Expolio
Tutores: Pablo Toharia Rabasco y Jorge Acosta Hernández

Madrid, Junio 2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Título: Componentes de Visualización Analítica para Grafos.

Junio 2024

Autor: Jaime Cabeza Expolio
Tutor: Pablo Toharia Rabasco y Jorge Acosta Hernández
Departamento de Arquitectura y Tecnología de sistemas informáticos
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Resumen

Actualmente la visualización de grafos es una herramienta esencial en diversos campos, permitiendo analizar grandes conjuntos de datos mediante la representación de nodos y enlaces para identificar patrones y/o relaciones entre ellos. Este trabajo tiene como objetivo diseñar y desarrollar un componente de visualización de grafos que sea aplicable a distintas áreas como las redes sociales, las ciencias de la información o la neurociencia. Para ello se investigarán las tecnologías más avanzadas en el ámbito de la representación de conjuntos de datos (D3.js, G6 o JS InfoVis), para que nos faciliten la creación de este componente y su integración en los navegadores web.

El trabajo abarca el estudio de metodologías de desarrollo de componentes de visualización asi como la selección de la tecnología que se usara para su implementación. Así mismo, también se buscara enfocarse en el diseño, implementación y presentación de dicho componente. Se investigarán las mejores prácticas y técnicas actuales, se evaluarán las ventajas y desventajas de las mismas y se desarrollará un componente que garantice su funcionalidad y usabilidad. El resultado sera una herramienta web práctica y accesible que mejore la comprensión y análisis de grafos, beneficiando así a investigadores o analistas que trabajarán con datos complejos.

Abstract

Nowadays, graph visualisation is an essential tool in several fields, allowing to analyse large datasets by representing nodes and links to identify patterns and relationships between them. This work aims to design and develop a graph visualisation component that is applicable to different areas such as social networks, information sciences or neuroscience. To this end, the most advanced technologies in the field of data set representation (D3.js, G6 or JS InfoVis) will be investigated to facilitate the creation of this component and its integration into web browsers.

The work includes the study of development methodologies of visualisation components as well as the selection of the technology to be used for its implementation. It will also seek to focus on the design, implementation and presentation of such a component. Current best practices and techniques will be investigated, their advantages and disadvantages will be evaluated and a component will be developed to ensure functionality and usability. The result will be a practical and accessible web tool that improves the understanding and analysis of graphs, thus benefiting researchers or analysts working with complex data.

Tabla de contenidos

Capítulo 1

Introducción

La teoría de visualización de datos y en concreto la de visualización de grafos es una herramienta esencial en diversos campos. Un ejemplo de ello, es en el análisis de redes de coautoría científica en los que se han descubierto patrones de colaboración entre científicos y se ha identificado a los investigadores más influyentes [1].

Gracias a la visualización de grafos se pueden analizar grandes conjuntos de datos mediante la representación de nodos y enlaces para buscar relaciones entre ellos. Este trabajo busca diseñar y desarrollar un componente de visualización de grafos para que se puedan aplicar a distintas áreas como pueden ser las redes sociales, la biología, las ciencias de la información o la neurociencia.

1.1. Motivación

La motivación para la realización de este trabajo reside en ofrecer una alternativa a los actuales componentes de representación de grafos, que a menudo son complejos y presentan interfaces de usuario poco amigables e intuitivas. Para lograr este objetivo, se emplearán las tecnologías más modernas de desarrollo de aplicaciones web junto con las bibliotecas de visualización de datos más avanzadas.

1.2. Objetivos

El objetivo principal, es crear un componente de visualización interactiva que facilite el análisis de grafos en navegadores web. Los objetivos especificados para este trabajo son:

1. **Estudiar la metodología de desarrollo de componentes de visualización:** Se investigarán cuales son las mejores prácticas y técnicas actuales para la visualización de grafos.
2. **Selección de las tecnologías que se usaran para el desarrollo de este trabajo:** Una vez estudiadas las ventajas y desventajas de cada tecnología,

se procederá a seleccionar cuales de ellas se usarán para desarrollar el componente de visualización.

3. **Diseño del componente:** Se conceptual izará y planificará el diseño del componente para cumplir con el objetivo marcado.
4. **Implementación del componente:** Se desarrollara el componente utilizando las herramientas seleccionadas, garantizando su funcionalidad y estabilidad.

Capítulo 2

Estado del arte

En el campo de la visualización de grafos existen un gran número de herramientas que vamos a analizar para proporcionar una base sólida para el desarrollo del proyecto y situar el trabajo en un contexto acorde a las tendencias y avances actuales.

2.1. Graphviz

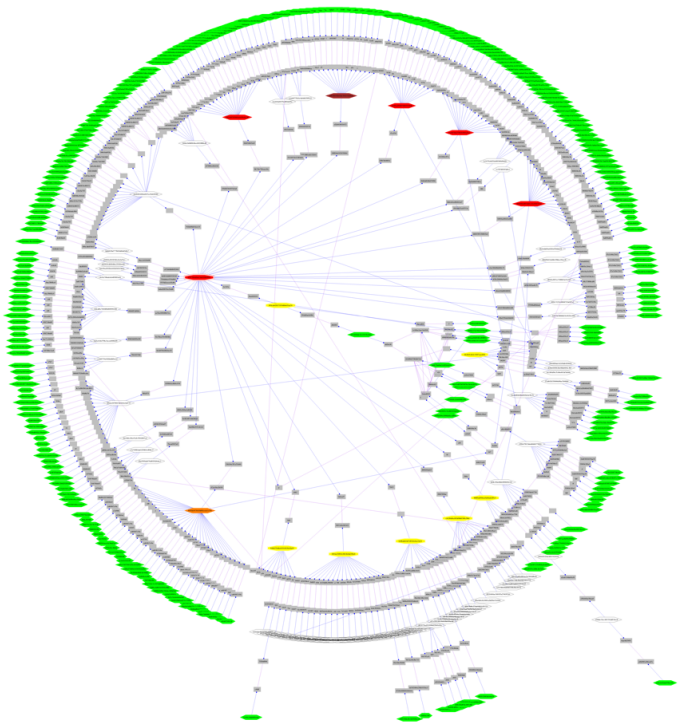

Figura 2.1: Ejemplo de grafo creado con Graphviz [2]

Graphviz o Graph Visualization Software, es un proyecto de código abierto que se enfoca en loa creación de gráficos y diagramas estructurados. Fue desarrollado

por AT&T Labs Research y destaca por su capacidad de generación automática de layouts de grafos y sus distintos formatos de salida como PNG, PDF y SVG [3]. Esta herramienta es ampliamente utilizada en diversas áreas incluyendo la informática, la bioinformática, la ingeniería o la organización empresarial.

Una de las características principales de esta herramienta, es el lenguaje de definición de gráficos DOT, el cual define una sintaxis especifica, simple y declarativa, que permite a los usuarios definir nodos y aristas de forma textual. Un archivo DOT define todo lo necesario para poder representar un gráfico.

Graphviz puede representar tanto grafos dirigidos como no dirigidos y grafos jerárquicos, los cuales son útiles para diagramas jerárquicos como arboles y grafos planos. Además la generación de estos grafos puede guardarse en distintos formatos como PNG, PDF, SVG y PostScript, haciéndolo extremadamente versátil para diferentes usos y plataformas.

La herramienta se puede integrar con varios lenguajes de programación, ofreciendo interfaces para Python, Perl, Ruby y otros lenguajes.

Sus aplicaciones van dirigidas a la creación de diagramas de flujo, arboles de decisión en el caso de la inteligencia artificial y a las redes de dependencias de proyectos para gestionar tareas.

Aunque Graphviz es altamente eficiente y versátil, esta herramienta cuenta con algunas limitaciones, los gráficos que se crean no son interactivos y no ofrecen una personalizaron dinámica. Por otra hay una falta de una interfaz gráfica, la cual puede generar una desventaja importante dependiendo del contexto y las necesidades del usuario.

2.2. Gephi

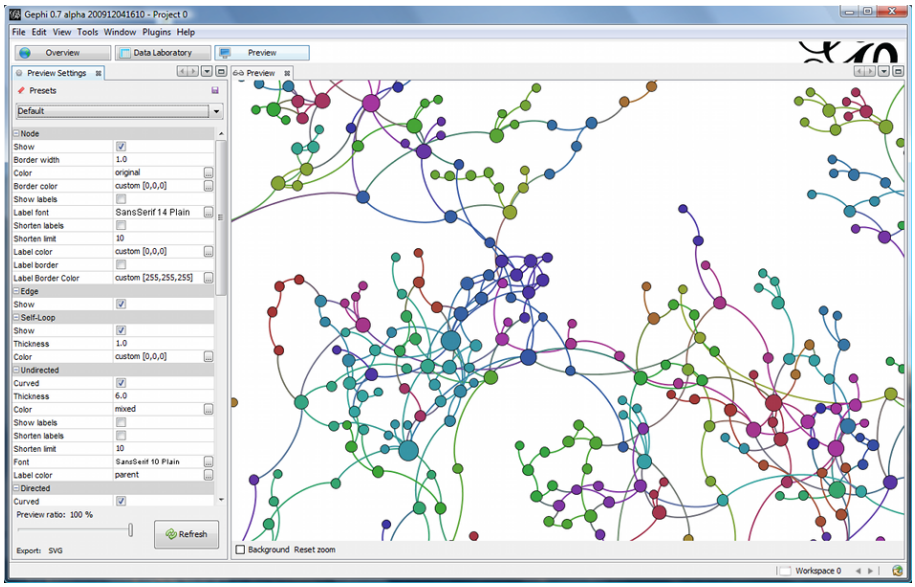

Figura 2.2: Captura del programa de Gephi [4]

Gephi es una plataforma de software de código abierto diseñada para la visualización y análisis interactivo de redes y grafos. Esta herramienta ofrece una interfaz que permite a los usuarios manejar grandes conjuntos de datos y descubrir patrones[5].Su amplio uso viene de investigadores de diversas disciplinas y analistas de datos.

La herramienta cuenta con una interfaz de usuario intuitiva, que facilita la interacción con los datos. Un ejemplo de esto es que se permite arrastrar y soltar nodos, así como, aplicar filtros y manipular visualizaciones en tiempo real.

Para organizar los nodos existen diversos algoritmos que implementa la herramienta como ForceAtlas, Fruchterman-Reingold, Yifan Hu, y otros. Además Gephi cuenta con herramientas para el análisis de métricas en redes como la centralidad, modularidad, diámetro, densidad y otros.

Además, se pueden importar datos en diversos formatos, como CSV, GEXF o GraphML; y manipular la edición directa de atributos de nodos y aristas, lo que proporciona un gran control sobre los datos.

Gephi es altamente usado en el análisis y visualización de redes sociales, en área de la bioinformática para visualizar la interacción proteína-proteína, en la investigación académica para las colaboraciones entre autores y en la gestión de infraestructuras como el análisis de redes de transporte.

Aunque Gephi es fácil de usar e interactiva, su rendimiento merma cuando se manejan redes extremadamente grandes que cuentan con millones de nodos y aristas, lo que puede llevar a tiempos largos de procesamiento y una experiencia menos fluida. Siguiendo con lo mencionado anteriormente, para obtener un rendimiento óptimo, se requiere una máquina con altas especificaciones en memoria RAM y CPU.

2.3. Cytoscape

Cytoscape también es una plataforma de software de código abierto que se utiliza principalmente para la visualización y análisis de redes complejas, ya que es especialmente popular en ese área por su capacidad de integrar y visualizar datos biológicos complejos y su extensibilidad mediante plugins y aplicaciones [7].

La herramienta esta preparada para interactuar con los datos de red de forma visual y para facilitar la importación y manipulación de datos, así como la personalización de las visualizaciones en diferentes formatos mediante la aplicación de algoritmos de disposición para organizar nodos y aristas.

Dentro de Cytoscape, se pueden hacer análisis complejos como la detección de comunidades o nodos centrales, el cálculo de métricas de red (centralidad, conectividad, etc.) y la visualización de redes multinivel.

Adicionalmente, Cytoscape es especialmente poderosa en el campo de la biología, ya que permite integrar y visualizar datos de omics, y superponer datos

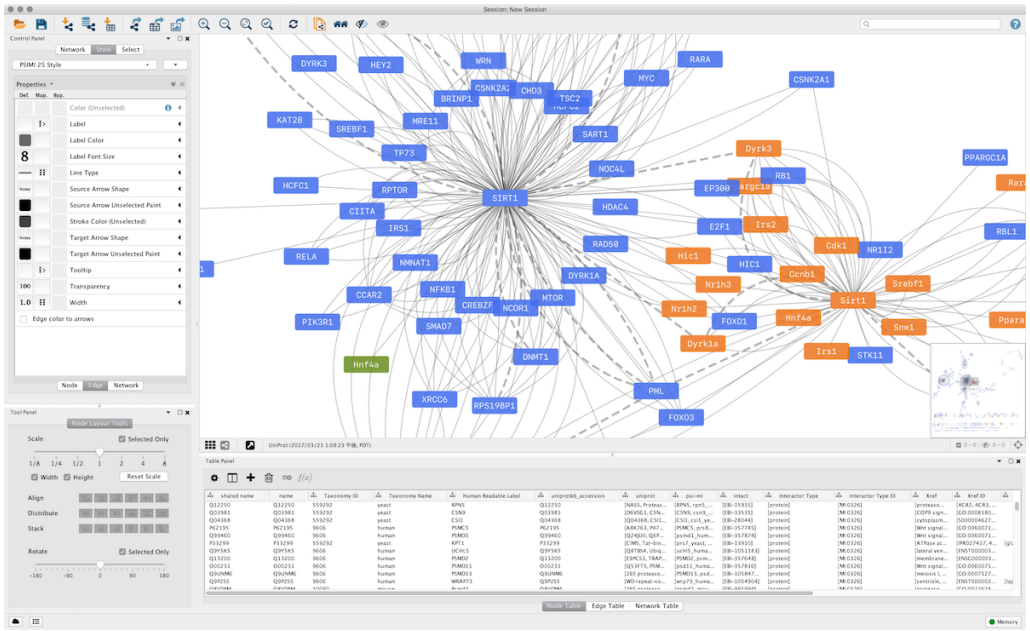

Figura 2.3: Captura del programa de Cytoscape [6]

experimentales sobre redes biológicas para identificar patrones y relaciones significativas.

En cuanto a sus limitaciones, Cytoscape se ve afectada por los requisitos de hardware para alcanzar rendimientos normales y su curva de aprendizaje es más prolongada debido a la cantidad de funcionalidades avanzadas con las que cuenta, ocasionando largos tiempos de adaptación a nuevos usuarios con poca experiencia en el análisis de redes.

2.4. Graphext

Graphext a diferencia de otras plataformas, no es de código abierto y fue fundada en 2015 por Victoriano Izquierdo y Miguel Cantón en España. Su objetivo era crear una herramienta que combinara simplicidad con uso de capacidades analíticas avanzadas, permitiendo a los usuarios transformar datos crudos en conocimientos prácticos, sin necesidad de un conocimiento extenso en programación.

De todas las herramientas analizadas, Graphext es la que ofrece la mejor interfaz gráfica, es muy fácil de usar y permite a los usuarios interactuar con sus datos visual e interactivamente. Dando lugar a una mejor importación, manipulación, análisis y visualización de los datos.

Esta plataforma soporta datos de muchos origenes, incluyendo: CSV, Excel, bases de datos SQL, APIs y servicios en la nube como Google Sheets.Por otra parte, Graphext proporciona herramientas de análisis exploratorio para identificar patrones, tendencias y anomalías, incluyendo la capacidad de realizar análisis

Figura 2.4: Ejemplo de analisis hecho con Graphex [8]

estadísticos tanto básicos como avanzados, generar gráficas y otras visualizaciones interactivas.

Otra de las características con las que cuenta, es la posibilidad de aplicar técnicas de machine learning para entrenar y evaluar modelos predictivos. Se soportan algoritmos de aprendizaje supervisado y no supervisado, lo que facilita tareas de clasificación, regresión, clustering y análisis de series temporales.

Por último, existe la posibilidad de automatizar tareas repetitivas mediante la creación de pipelines reproducibles, lo que asegura que los análisis puedan ser replicados y actualizados con facilidad.

Dado todas las funcionalidades que ofrece, podemos asumir que sera necesaria de una máquina potente para la ejecución de modelos complejos, así mismo observamos que la curva de aprendizaje aumenta su dificultad en las funcionalidades más avanzadas. Otra de las cosas a tener en cuenta es que al no ser una plataforma de código abierto, para hacer uso de ella será necesario pagar.

2.5. Conclusiones

El análisis realizado a algunas de las herramientas de visualización de grafos existentes nos dice que cada una cuenta con distintas fortalezas y limitaciones. Graphviz cuenta con su generación automática de grafos, aunque carece de interactividad y una interfaz gráfica amigable. Gephi tiene una interfaz intuitiva y algoritmos avanzados para la disposición de redes pero se ve limitada por su rendimiento con redes muy grandes. Por otra parte Cytoscape permite el uso de datos omics y realizar análisis complejos, pero al igual que Gephi es necesario un hardware potente para tener un rendimiento optimo. Y Graphext ofrece la mejor de las interfaces gráficas y unas capacidades avanzadas para realizar técnicas de machine learning, pero no es un proyecto de código abierto y demanda muchos recursos. Todo esto nos da una perspectiva adicional para saber cual es el contexto actual y poder realizar un trabajo que se posicione en dicho contexto.

Capítulo 3

Análisis previo

3.1. Tipos de datos

Entre nuestros objetivos, poder codificar los datos de las propiedades que puedan contener los nodos es muy importante y para ello, hay que conocer de que tipo pueden ser estas variables y implementar su codificación de diferente manera. Los tipos de variables se pueden resumir en los siguientes:

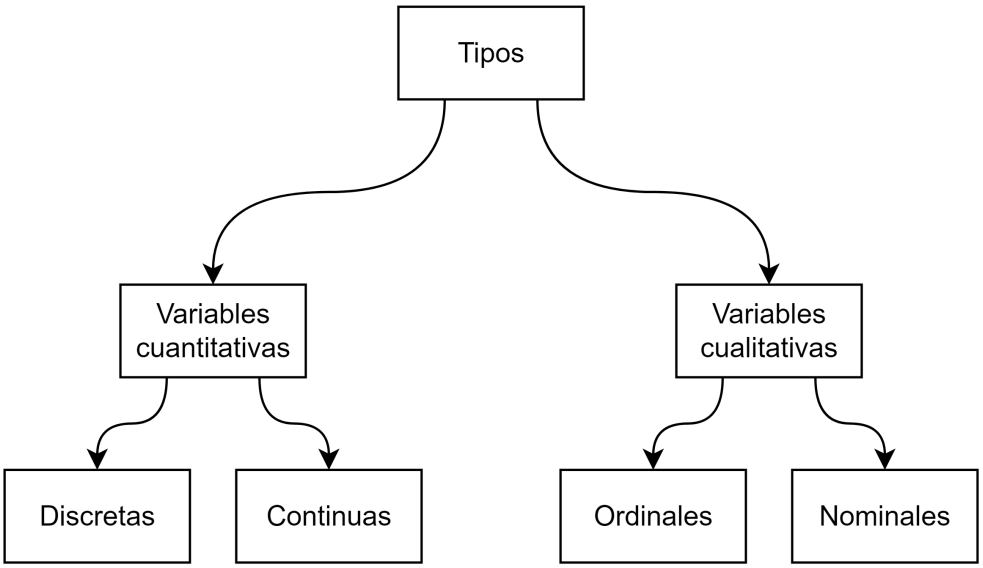

Figura 3.1: Diagrama jerárquico de los tipos de variables

- **Variables cualitativas**
  - Categóricas

    - Ordinales
- **Variables cuantitativas**
    - Discretas
    - Continuas

Hacer esta distinción a la hora de diseñar nuestro componente de visualización va a ser vital para cumplir con los objetivos de este trabajo.

3.1.1. Variables cualitativas

Son variables cualitativas (o categóricas), aquellas que describen cualidades o características y no pueden medirse numéricamente. Este tipo de variable se divide en:

- **Variables categóricas**: Son variables que no tienen un orden intrínseco, es decir no hay relación entre las variables, por ejemplo el estado civil o el departamento de una empresa.
- **Variables Ordinales**: Son variables que cuentan con un orden o relación que permite comparar los valores entre si mismos. Como el nivel de satisfacción o el nivel de un videojuego [9].

3.1.2. Variables cuantitativas

Son variables cuantitativas (o numéricas), aquellas que se pueden medir y permiten realizar operaciones aritméticas. Este tipo de variable se puede dividir en:

- **Variables discretas**: Son un tipo de variable que solo pueden tomar valores entero, como el número de coches o el número de película.
- **Variables continuas**: A diferencia de las discretas este tipo de variables puede tomar cualquier valor dentro de un rango. La altura el peso o la temperatura son algunos ejemplos de estas [9].

3.2. Formatos de representación de grafos

3.2.1. Definición de grafo

Antes de definir que formatos existen para representar un grafo en forma de datos en un fichero, Es importante recordar la definición de un grafo desde el punto de vista matemático y desde el punto de vista de la ciencia de computadores para entender las razones de uso de cada formato.

Desde el punto de vista matemático, un grafo es una colección de nodos o vértices y aristas que conectan pares de nodos y permiten modelizar relaciones entre estos nodos [10].

Cuando aplicamos esta definición en el campo de la ciencia de computadores, obtenemos una estructura de datos que consiste en un número finito de nodos (o

vertices) seguido de un conjunto de aristas que conectan pares de nodos. Algunos ejemplos que en los que se usa esta estructura pueden ser las redes, ya sean sociales o informáticas, representación de mapas y caminos o en bioinformática donde se modelan las relaciones entre proteínas y genes.

3.2.2. Matriz de adyacencia

Una matriz de adyacencia es un vector de dos dimensiones de tamaño $n \times n$ donde n es el numero de nodos del grafo. Los elementos en la fila i y la columna j tiene los siguientes valores:

- $A[i][j] > 0$ Indica la existencia de una arista entre el nodo i y el nodo j.
- $A[i][j] = 0$ Indica la no existencia de una arista.

Esta forma de representar grafos es una de las mas simples y eficientes para grafos densos que permite una complejidad de $O(1)$ cuando se quiere comprobar la existencia de una relación entre dos nodos. Sin embargo cuando trabajamos con grandes grafos que están dispersos (contienen muy pocas aristas) este formato suele ser menos eficiente a la hora de visitar las aristas entre nodos $O(V^2)$ [11].

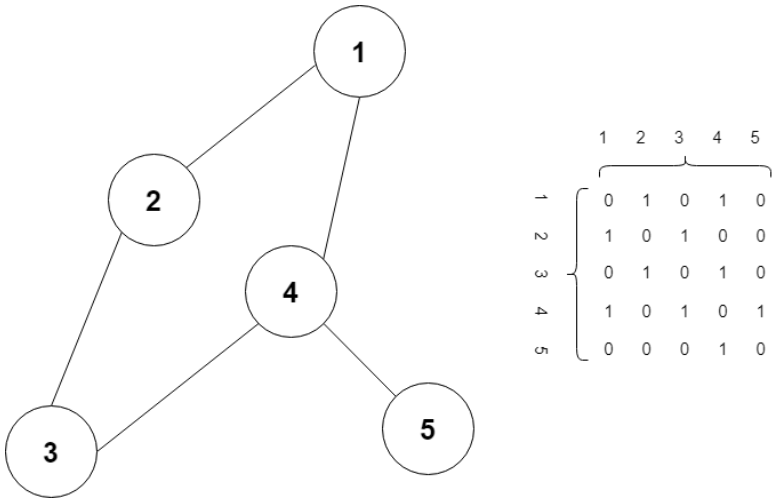

Figura 3.2: Ejemplo de matriz de adyacencia

3.2.3. Lista de adyacencia

La lista de adyacencia representa el grafo mediante un array de listas, donde el índice del array representa el nodo y cada elemento de la lista asociada representa los vértices con los que esta relacionado.

A diferencia de la matriz de adyacencia, este formato permite ser más eficiente a la hora de guardar grafos dispersos, ya que, su complejidad para acceder a todas las aristas es $O(V + E)$ y es más escalable para cuando tratamos con grandes conjuntos de datos. Por otra parte la desventaja es que es algo más complejo de implementar comparado con una matriz de adyacencia por ejemplo [11].

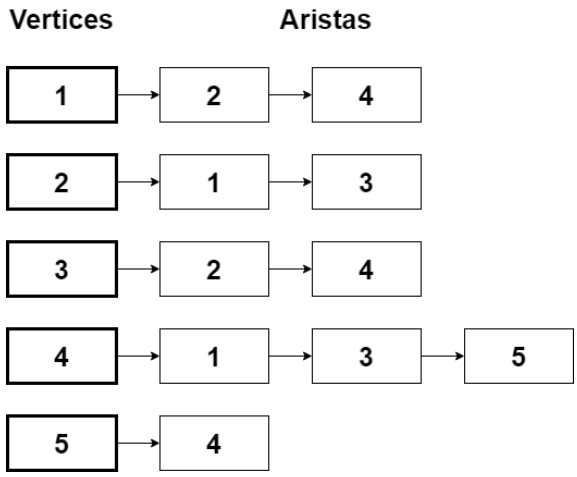

Figura 3.3: Ejemplo de Lista de adyacencia

3.2.4. Lista de aristas

En este formato, el grafo es representado mediante pares de vértices (A, B) donde A es el origen y B el destino, y en caso de ser un grafo con pesos la forma de representarlo sería (A, B, C) donde C indica el peso de esa arista.

Comparándolo con otros formatos de representación, la lista de aristas es un formato muy simple de entender y implementar con complejidad espacial $O(E)$ en el peor de los casos. Esto implica que es muy eficiente para grafos dispersos, pero por el contrario no lo será tanto para los grafos densos [11].

3.2.5. Matriz de incidencia

La matriz de incidencia funciona mediante una matriz de dos dimensiones de tamaño $n \times m$, donde n es el numero de vértices y m es el numero de aristas y $I[i][j]$ indica si el nodo i pertenece a la arista j.

- $I[i][j] = 1$ Indica la incidencia del nodo i sobre la arista j.
- $I[i][j] = 0$ Indica que el nodo i no incide sobre la arista j.

La ventaja mas relevante que propicia esta forma de representar un grafo, es la facilidad de saber si un vértice pertenece a una arista $O(1)$. Sin embargo cuenta con una complejidad espacial alta $O(V \times E)$ [11]

3.2.6. Conclusiones

Analizando los métodos de representación obtenemos como resultado la siguiente tabla que muestra las complejidades al realizar distintas acciones sobre las estructuras resultantes como comprobar, insertar o eliminar aristas:

Algunas de las observaciones que podemos hacer al mirar a esta tabla son que las complejidades de comprobación, inserción y eliminación son de $O(1)$ en los

Operación	**Matriz de Adyacencia**	**Lista de Adyacencia**	**Lista de Aristas**	**Matriz de Incidencia**
Complejidad Espacial	$O(V^2)$	$O(V+E)$	$O(E)$	$O(V \times E)$
Comprobación de Arista	$O(1)$	$O(V)$	$O(E)$	$O(1)$
Inserción de Arista	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Eliminación de Arista	$O(1)$	$O(V)$	$O(E)$	$O(1)$
Recorrido de Aristas	$O(V^2)$	$O(V+E)$	$O(E)$	$O(V \times E)$

Tabla 3.1: Complejidades de Representaciones de Grafos

formatos que son de tipo matriz, aunque a la hora de recorrer las aristas vemos que son mucho menos eficientes. Por otra parte los métodos de lista ofrecen mayor velocidad a la hora de recorrer las aristas con complejidades de $O(V+E)$ y $O(E)$ pero son mas lentos en las otras operaciones.

3.3. Representación visual de grafos

Ahora que hemos visto como podemos representar los datos de un grafo en un fichero y los tipos de variables con los que nos podemos encontrar en los atributos, es importante entender como podemos representar visualmente toda esta información [12].

3.3.1. Marcas y canales

Entender que son las marcas y canales es fundamental para poder representar los datos de una manera coherente y entendible. Incluso codificaciones visuales complejas se pueden dividir en componentes que se pueden analizar en términos de su estructura de marcas y canales.

- Una **marca** es un elemento gráfico primitivo de representación, que se clasifica en función del número de dimensiones que se requiera. Por ejemplo, un punto correspondería a un elemento de cero dimensiones y una linea a un elemento de una dimensión.
- Un **canal** visual es una forma de control sobre la apariencia de las marcas y se puede utilizar para codificar atributos de los datos que se quieran representar. Algunos ejemplos de canales visuales son la posición de la marca, el color, la forma o su tamaño.

Estas marcas y canales se pueden combinar para crear cualquier tipo de representación, por ejemplo, un diagrama de barras usa una linea como marca y su tamaño en el eje y indica el valor de la variable cuantitativa. Por otra parte, el color puede ser usado para clasificar los valores de una variable categórica.

Marcas

Con las marcas somos capaces de ilustrar tanto elementos de una fuente de datos como las relaciones entre ellos. Para mostrar las relaciones, podemos hacerlo

de dos maneras, usando marcas de contención como áreas cuyos limites los fija una linea, o con marcas de conexión, las cuales unen dos o más elementos.

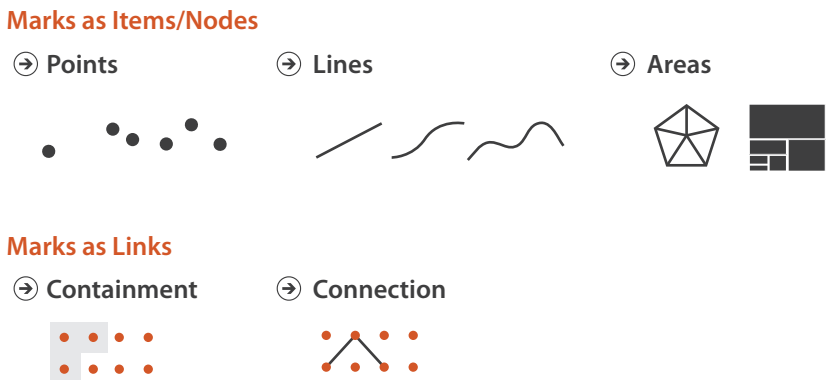

Figura 3.4: Tipos de marcas [13]

Canales

En cuanto a la selección del canal visual para codificar un atributo, hay que identificar primero su tipo, ya que existen canales que se pueden utilizar de diferentes maneras para ilustrarlos. Por ejemplo, una escala de colores puede ser categórica donde cada color es claramente distinguible de los demás, o puede ser secuencial donde se utiliza la saturación o la luminancia para distinguir un orden de magnitud entre dos valores del atributo.

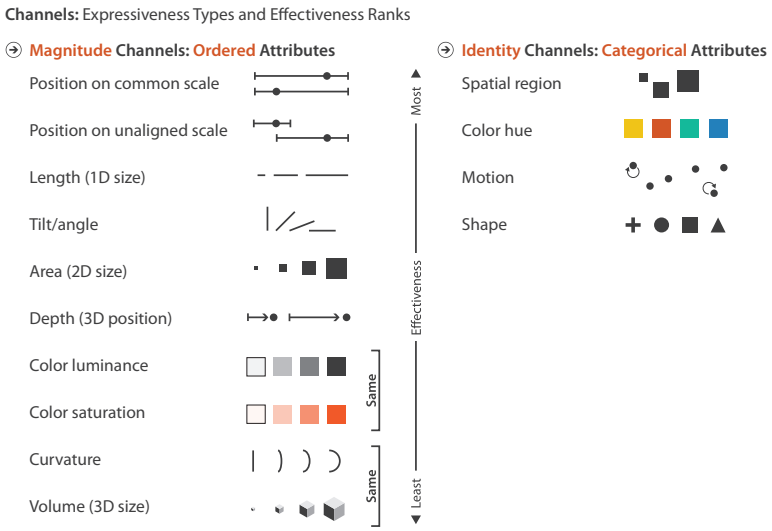

Figura 3.5: Canales visuales [14]

3.3.2. Grafos

Una vez comprendidas las marcas y los canales, cuando se quiere visualizar un grafo, podemos distinguir varias formas de hacerlo:

- **Diagramas nodo-enlace**: Utilizan puntos para los ilustrar los nodos y lineas para las relaciones entre ellos.
- **Matriz de adyacencia**: Se deriva la matriz de datos a un contexto visual, donde la presencia de color marca la relación entre dos nodos.

Aunque podemos representar los grafos como la ilustración de una matriz de adyacencia, las desventajas son claras, los canales visuales no son lo suficientemente intuitivos como para identificar las relaciones o patrones en los nodos, por eso, lo más común es visualizar los grafos como diagramas de nodo-enlace.

Capítulo 4

Desarrollo

4.1. Introducción

Durante este capítulo detallaremos las fases por las que ha pasado el trabajo para la realización de este componente de análisis de datos. Primero hablaremos de la fase de planificación, donde identificamos el alcance que tendría este trabajo, así como los objetivos, la metodología que se seguirá y las funcionalidades con las que contará nuestra aplicación.

En la fase de diseño y arquitectura, se describirán como se ha estructurado la aplicación, su funcionamiento esperado, el diseño de su interfaz de usuario y las tecnologías y herramientas que forman parte en la implementación.

Por último, se darán detalles sobre la implementación y las decisiones tomadas dadas las limitaciones y complicaciones que se han encontrado durante la realización del trabajo.

4.2. Fase de planificación

4.2.1. Alcance y objetivos

Para este trabajo se ha buscado tener un componente inicial que sea capaz de representar grafos usando dos tipos de técnicas diferentes:

- **Simulación de fuerzas (o force-directed layout)**: Es una técnica que posiciona los nodos de un grafo de manera que se minimicen las fuerzas entre ellos emulando un sistema físico donde los nodos se comportan como partículas con fuerzas aplicadas entre ellas.Esto da lugar a una disposición de nodos intuitiva y fácil de entender.

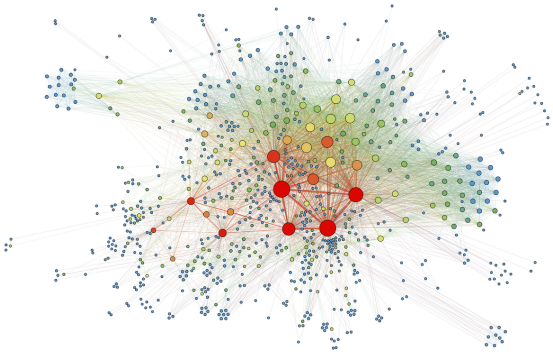

Figura 4.1: Representación de una simulación de fuerzas [15]

- **Layout circular**: Posiciona los nodos alrededor de una circunferencia para destacar la estructura global del grafo y las conexiones entre nodos centrándose más en una visión clara y ordenada de las relaciones que hay entre ellos.

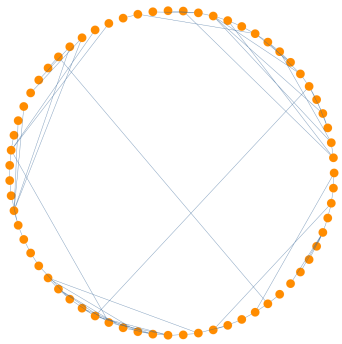

Figura 4.2: Representación de un layout circular [16]

Por otra parte, otros de los objetivos que se han definido, es que sea posible codificar atributos, tanto de los nodos, como de los enlaces para que sea posible visualizarlos de alguna manera cuando estos se generen. En nuestro trabajo se implementará la codificación de atributos con el tamaño y el color de nodo y enlace.

Para codificar dichos atributos habrá que saber de que tipo de variable estamos hablando. La aplicación se encargará de inferir inicialmente el tipo o los tipos de variables en los que puede encajar la codificación y será el usuario final el que decida su tipo además de las escalas de colores en los que se quiera codificar si es que decide codificar un atributo en función del color.

Por ultimo, en cuanto a la entrada de datos se definirán varios formatos de entrada tanto para ficheros de tipo JSON (JavaScript Object Notation) como para ficheros de tipo CSV (Comma Separated Values) y se deberán implementar la funcionalidad necesaria para que sean procesados.

4.2.2. Metodología

Desarrollaremos este trabajo siguiendo la metodología Agile, ya que es la que mejor se ajusta a nuestra situación en comparación como podría ser el desarrollo en cascada o el modelo de SCRUM.

Agile, como indica la palabra, se trata de una metodología de trabajo ágil que centra sus esfuerzos en recoger feedback constantemente de sus clientes, en este caso los tutores, y funciona mediante la creación de sprints de corta duración con tareas definidas. De esta manera tenemos iteraciones sobre el desarrollo después de cada sprint.

Esto a su vez supone un reto debido a la naturaleza de esta metodología, ya que en ocasiones se puede perder el focoy atrasar ciertas tareas, por realizar cambios constantes a la lógica y el funcionamiento de la aplicación.

4.3. Fase de Diseño

4.3.1. Introducción

Como resultado de la planificación y la investigación realizada en el estado del arte, se ha decidido que la aplicación va a ser un orientada a una arquitectura web, utilizando tecnologías para la construcción de paginas web actuales y bibliotecas de visualización de datos, sin olvidarnos de las librerías subyacentes para hacer que el trabajo realizado sea efectivo.

Dado este tipo de aplicación web, sera necesario un servidor que de servicio a los usuarios. Aunque como veremos más adelante se seguirá un esquema de Client-Side rendering, donde el servidor solo se encarga de enviar el HTML junto con los archivos JavaScript necesarios.

Para hacer un flujo de datos eficiente entre los componentes de la página resultante, se investigarán librerías que sean capaces de manejar el estado de la aplicación.

4.3.2. Bibliotecas de representación de grafos

Existen múltiples Bibliotecas en muchos lenguajes de programación que se usan para la representación de grafos. En este caso hemos optado por usar una libre-

ría implementada en JavaScript ya que, queríamos que nuestro componente estuviese basado en una aplicación web.

D3.js

D3.js (Data-Driven Documents) es una biblioteca desarrollada por Mike Bostock muy utilizada para generar componentes de visualización de datos dinámicos y de manera interactiva en navegadores web. Esta biblioteca aprovecha los estándares actuales de HTML, SVG y CSS para proporcionar una herramienta poderosa y flexible.

Entre sus características principales destaca su amplio desarrollo con su gran cantidad de utilidades. Puedes diseñar una gran cantidad de componentes de visualización, satisfaciendo cualquier necesidad de los usuarios. Gracias a su enfoque declarativo, en lugar de especificar como debe ser el resultado final, D3 se encarga de los detalles y el desarrollador solo tiene que especificar los parametros y la entrada de datos.

Por otra parte D3 tiene la capacidad de crear, seleccionar y manipular elementos del DOM (Document Object Model), permitiendole aplicar transformaciones a elementos de manera más eficiente para visualizar un componente [17].

G6.js

G6.js es una reciente biblioteca de JavaScript que ha sido desarrollada por AntV, unidad de investigación de Alibaba Group y de la misma manera que D3, esta diseñada para la visualización y análisis de gráficos y redes.

Esta biblioteca se enfoca en ofrecer representaciones altamente personalízales y con un renderizado eficiente basado en técnicas optimizadas para manejar gráficos con un gran nodo número de nodos y aristas que garantiza un rendimiento fluido [18].

InfoVis Toolkit

Desarrollada por Nicolas Garcia [19], InfoVis Toolkit proporciona herramientas para crear visualizaciones interactivas de datos a través de la web. Esta biblioteca destaca por su habilidad para manejar grandes conjuntos de datos y su flexibilidad en los casos que hay que realizar visualizaciones complejas.

Esta biblioteca es capaz de representar tanto árboles y grafos, como mapas de color, treemaps y gráficos de barras de manera interactiva y con suaves animaciones. Además implementa funcionalidades como la de hacer zoom y desplazarse sobre las visualizaciones.

En cuanto a los tipos de datos que soporta, solo acepta datos en formatos JSON, lo cual es común para la transmisión de datos en aplicaciones web [20].

Conclusiones

Una vez analizadas ambas bibliotecas, llegamos a la conclusión de que son librerías muy completas con un amplio repertorio de utilidades para representar grafos, sin embargo, aunque G6.js es una librería más reciente y moderna que D3.js, D3 cuenta con más desarrollo y comunidad y por tanto hemos optado por usar esta librería para el desarrollo de este componente de visualización de grafos.

4.3.3. Desarrollo web

Dado que el enfoque de este trabajo esta en la visualización de datos y que las herramientas de desarrollo web como HTML, CSS y JavaScript son accesibles y satisfacen esa necesidad, una aplicación web será la dirección que tomemos para implementar nuestro componente. Para lograr esto de manera eficiente y efectiva, necesitaremos elegir un framework que nos permita construir una aplicación robusta, y mantenible. Por eso, evaluaremos tres de los frameworks más populares: React, Vue y Angular.

React

React es una biblioteca de JavaScript que fue desarrollada por Facebook y lanzada en 2013. Su objetivo es la construcción de interfaces de usuario basadas en componentes reutilizables que pueden manejar su propio estado [21]. Entre sus características, destaca el uso del Virtual DOM, que mediante la reducción de la manipulación directa del DOM real, consigue optimizar la actualización de la interfaz de usuario dejando a React que maneje el DOM real. Por otra parte, React promueve un flujo de datos unidireccional, haciendo más entendible la gestión del estado de la aplicación. React también diseño una nueva sintaxis de JavaScript conocida como JSX, que permite escribir HTML dentro de un script, con el objetivo de mejorar la legibilidad del código [22]. Estas características junto con el soporte y la comunidad que tiene React, la hacen una opción poderosa para la realización del trabajo.

Vue

Vue es un framework enfocado en una metodología progresiva para construir interfaces de usuario, fue desarrollado por Evan You y lanzada en 2014 [23]. Vue destaca por su simplicidad y flexibilidad, permitiendo a los desarrolladores integrarlo en sus proyectos existentes para componentes pequeños o utilizarlo como un framework completo para una aplicación grande. Además, Vue cuenta con un sistema de reactividad que actualiza automáticamente el DOM cuando el estado cambia, dando lugar a la construcción de interfaces dinámicas. Por otra parte el ecosistema de Vue cuenta con herramientas como Pinia que se pueden utilizar para el manejo de estado en la aplicación [24]. Todo esto facilita la creación de aplicaciones modulares y escalables. Finalmente, la curva de aprendizaje es una de las cosas que hacen a Vue una opción atractiva para muchos desarrolladores.

Angular

Angular es un framework completo para el desarrollo de aplicaciones web. Fue desarrollado por Google y lanzado oficialmente en 2016 como una alternativa a la existente librería AngularJS [25]. El lenguaje sobre el que esta escrito es TypeScript que aporta ventajas como la tipificación de variables. Angular adopta el patrón de diseño Model-View-Controller (MVC) Y está basado en una arquitectura de componentes de manera similar a React y Vue. Por otra parte, Angular por si solo incluye un sistema de inyección de dependencias que mejora la modularidad y testabilidad del código y además utiliza RxJS para introducir reactividad en sus componentes [26]. Estas características nos demuestran que Angular es ideal para aplicaciones de carácter empresarial y complejas que requieren una estructura robusta y con herramientas avanzadas.

Conclusiones

Analizando cada uno de estos frameworks, vemos que ofrece característica y beneficios únicos que pueden ser decisivos dependiendo de las necesidades de cada usuario. React es interesante por su flexibilidad y eficiente manejo del DOM, Angular por su parte ofrece una estructura robusta y avanzada con el uso de TypeScript y Vue destaca por su suave curva de aprendizaje, su fuerte reactividad y su completo ecosistema, facilitando la construcción de interfaces de usuario dinámicas y escalables.

Finalmente llegamos a la conclusión de que Vue es la mejor opción para el desarrollo de nuestro componente. Su simplicidad y flexibilidad acelerará el proceso de desarrollo y sus herramientas de manejo del estado permitirán desarrollar una aplicación que sea robusta y a la vez mantenible en el futuro.

4.3.4. Node.js

Para dar servicio se utilizará Node.js, un entorno de ejecución de JavaScript construido sobre el motor V8 de Google Chrome por Ryan Dahl en 2009, con el objetivo de construir aplicaciones de red escalables y rápidas [27]. Esta tecnología habilita a desarrolladores a ejecutar códifo de JavaScript en el lado del servidor, cuando antes no era posible.

4.3.5. Vite

Vite es una herramienta de construcción para proyectos web modernos. Del creador del framework de Vue, esta es la mejor opción a la hora de desarrollar un proyecto con dicho framework. Su objetivo principal es proporcionar una experiencia de desarrollo más rápida y ligera mediante usando módulos nativos del navegador para proveer un tiempo de inicio casi instantáneo y permitiendo actualizaciones rápidas y precisas durante la implementación.

Comparándolo con otras herramientas como Webpack o Parcel, Vite demuestra superioridad en su eficiencia gracias a que no necesita realizar empaquetados

completos cuando se realiza algún cambio, y su manejo de las configuraciones amigables y simples [28].

4.3.6. Pinia

Pinia fue creada como alternativa moderna y más intuitiva que Vuex. Esta biblioteca fue creada por Eduardo San Martin Morote, miembro del equipo de desarrollo de Vue.js, y esta diseñada para manejar el estado en Vue de una manera más sencilla y para aprovechar al máximo las características de Vue 3 y su Composition API.

Algunas características claves son el soporte que ofrece para TypeScript, con una integración completa y libre de errores, su total reactividad nativa otorgada por Vue 3, dando lugar que cualquier cambio sobre el estado, se vea reflejado automáticamente en la interfaz de usuario. Por ultimo, esta biblioteca permito creación de multiples almacenes (o stores), que facilitan la modularización de la aplicación.

Para nuestro proyecto se generaran dos almacenes de estado, el cual uno manejara la lógica de la representación del grafo, y el otro se encargará de guardar toda la información relacionada con los subcomponentes dedicados a la entrada de datos en la aplicación.

4.4. Implementación

En esta sección, vamos a abordar la solución implementada para el componente de visualización de grafos. Mostraremos la estructura de archivos y pasaremos a describir como se ha llevado a cabo la implementación, así como las dificultades encontradas durante el proceso.

4.4.1. Estructura del proyecto

La estructura del proyecto está organizada para facilitar el desarrollo, mantenimiento y escalabilidad futuros de la aplicación. A continuación, describiremos cada uno de los directorios y los archivos principales del código fuente.

- **dist/**: Alberga los archivos generados tras el proceso de construcción del proyecto
- **src/**: Contiene todo el código fuente del proyecto.
  - **assets/**: Contiene los ficheros de estilo base que formatean todas las fuentes, colores y tamaños de cada elemento de la página.
  - **components/**: Contiene el código que muestra y da funcionalidad a los subcomponentes de la pagina.
    - **FileWizard/**: Este directorio maneja toda la lógica que se encarga de introducir los datos de entrada, procesarlos y codificarlos para luego ser renderizado.

        - **NavBar.vue**: subcomponente de la aplicación que maneja la barra de navegación de la página.
        - **NotificationComponent.vue**: Componente de visualización, para mostrar notificaciones al usuario.
    - **stores/**: Este directorio guarda los ficheros fuente que almacenan los datos que se utilizan dentro de la aplicación, tanto variables reactivas para mostrar y esconder elementos como los datos procesados del grafo y funciones que realizan acciones sobre ellos.
    - **App.vue**: Componente principal de la aplicación. Es el orquestador del resto de componentes de la página y se encarga de recibir eventos del resto de componentes y ejecutar la función que renderiza el grafo cuando ya se han procesado sus datos.
    - **main.js**: Punto de entrada de la aplicación, se encarga de montar los componentes, los plugins que se hayan decidido usar y los almacenes de estado de Pinia.
    - **circularSimulation.js**: Fichero que guarda la funcionalidad que genera el grafo de tipo circular.
    - **forceSimulation.js**: Fichero que guarda la funcionalidad que genera el grafo de tipo simulación de fuerzas.
    - **notificationPlugin.js**: Es un plugin que habilita una función para todos los componentes de la aplicación, la cual permite mandar las notificaciones y mostrarlas.
    - **utils.js**: Fichero para albergar funciones auxiliares que sean útiles para los componentes.
- **index.html**: Sirve como esqueleto donde se va a montar la aplicación y será el primer archivo que cargará el usuario cuando acceda al sitio web.
- **jsconfig.json**: Fichero de configuración de las opciones del entorno de desarrollo.
- **package.json**: Fichero esencial para node.js donde se definen las dependencias del proyecto, scripts de ejecución y otras configuraciones esenciales.
- **package-lock.json**: Fichero que garantiza la correcta instalación de las versiones de las dependencias externas.
- **vite.config.js**: Almacena la configuración principal de Vite. Define el comportamiento del servidor de desarrollo, como se deberán tratar los módulos, y cómo se debe construir el proyecto para producción.

4.4.2. Almacenes

Como se ha mencionado anteriormente, mediante el uso de Pinia hemos diseñado dos almacenes que manejan el estado de la aplicación:

- **GraphStore.js**: Guarda la información de todos las escalas de colores que se pueden usar para representar la codificación de atributos del grafo a los nodos y enlaces, además de guardar la información del grafo una vez ha sido procesado.
- **FileUploadStore.js**: Almacena todas las variables reactivas usadas para la visualización de elementos dentro del modal que se encarga de procesar los datos. Ofrece funciones tanto como para manipular los datos del grafo como modificar el comportamiento de lo elementos.

Gracias a este esquema podemos mantener separada la lógica de procesamiento de datos con la de su visualización, lo que hace fácil de mantener y escalar para añadir nuevas funcionalidades en el futuro.

4.4.3. App.vue

La pantalla de inicio esta codificada en el componente de App.vue, donde se inicializan tanto el código html, los estilos de CSS de algunos de los elementos de dicho componente y el script de js que inicializa los almacenes en sus valores por defecto y maneja el evento que indica cuando generar el grafo.

En la parte del código html se utilizan directivas de vue como v-on para enlazar eventos como el cierre del modal que procesa los datos de los grafos, a las funciones que tenemos en los almacenes y que modifican las variables reactivas guardan el estado de visibilidad.

4.4.4. NavBar.vue

NavBar.vue es el componente que da lugar a la barra horizontal de navegación de la aplicación. Aquí se encuentra el botón que abrirá el modal, el cual se encarga de procesar los ficheros de los grafos. Cuando el botón es pulsado, se manda una señal a App.vue, el cual se encarga de visualizar el modal mencionado.

4.4.5. NotificationComponent.vue

Este componente guarda el código html y css que visualiza los mensajes de notificaciones, ya sea por que ha ocurrido un error al validar los datos o porque se esta produciendo la generación del grafo. La idea es que el usuario este al tanto en cualquier punto del proceso de si se le ha olvidado rellenar algún campo, lo ha rellenado erróneamente o simplemente que la aplicación le informa de algo.

Se han implementado tres tipos de notificaciones:

- **Error**: Se notifica al usuario cuando ha ocurrido algún error.
- **Info**: Se notifica al usuario información del estado de un proceso.

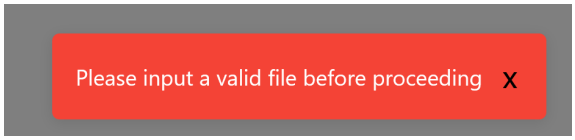

Figura 4.3: Ejemplo de notificación de error

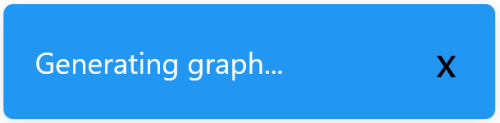

Figura 4.4: Ejemplo de notificación de información

- **Success**: Se notifica al usuario que un proceso a finalizado correctamente.

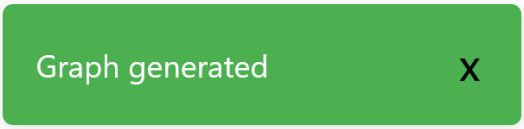

Figura 4.5: Ejemplo de notificación exitosa

4.4.6. FileWizard.vue

FileWizard representa un componente que actúa como contenedor y orquestador de otros componentes que se encargan del proceso de codificación de un grafo. Mediante el uso de variables reactivas es capaz de saber en que paso del proceso se encuentra y con que tipos de datos esta tratando.

De esta manera podemos identificar y validar los datos antes de pasar a la siguiente etapa del proceso. Finalmente, una vez se hayan coleccionado los datos necesarios, emitirá un evento donde se obtendrán todos los datos del almacén y se les trasmitirán al almacén principal, para que el componente principal se encargue de dibujar el grafo.

WizardSetup.vue

Este componente, esta preparado para que el usuario introduzca el tipo de entrada que va a procesarse y como se va a representar el grafo posteriormente.

En cuanto a los formatos de entrada que soportara la aplicación, podemos dividirlo por tipo de fichero y la forma en la que nos llegan los datos en este fichero:

- **JSON**

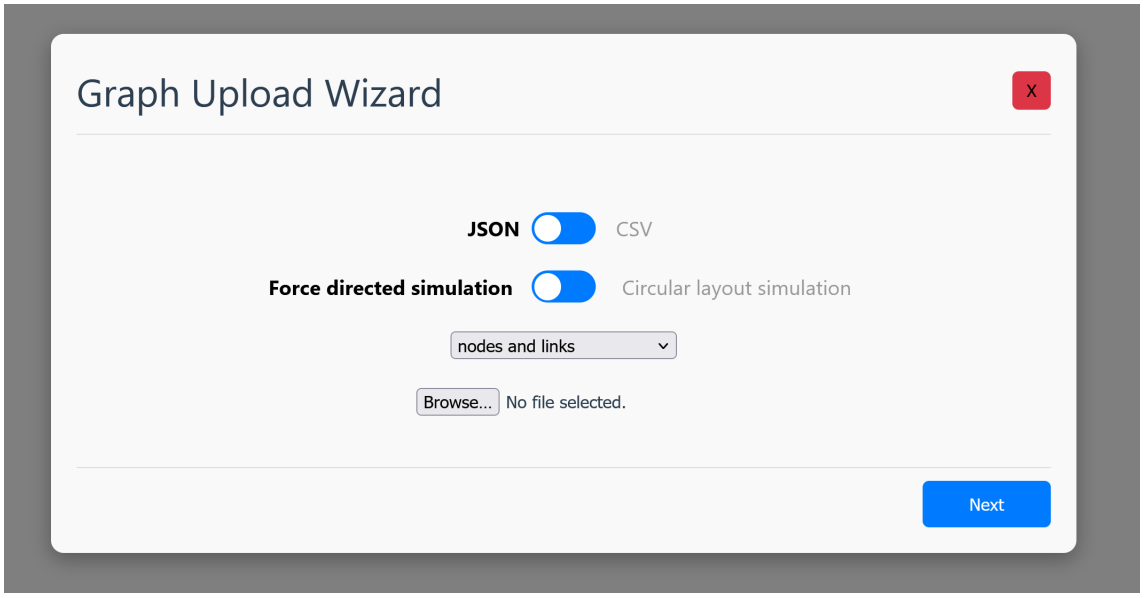

Figura 4.6: Imagén del componente de File Wizard

- **Nodos y enlaces**: El formato más completo que permite codificar tanto atributos a los nodos como a los enlaces.
- **Nodos con lista de adyacencia**: Un formato más compacto donde los nodos son los únicos a los que se le puede codificar con atributos.

- **CSV**
  - **fichero de nodos y enlaces separados**:Aunque necesita de dos fuentes diferentes, permite la misma funcionalidad que el formato JSON.
  - **Matriz de adyacencia**:El formato más simple, no se pueden codifica ni nodos ni atributos.

La idea principal es que la aplicación pudiese aceptar muchos tipos de formatos, pero a medida que se quería incorporar uno nuevo, la complejidad de procesamiento aumentaba mucho el tiempo que había que dedicar a cada tipo, así que, se decidieron que estos serían los formatos que podría recibir la aplicación.

Más adelante podremos ver como las elecciones en este componente pueden afectar a la visibilidad de ciertos elementos en la siguiente fase del procesamiento.

ToggleButtons

Este componente, actúa como un botón de alternancia que permite al usuario elegir entre dos opciones distintas. Su implementación permite reusabilidad en caso de ser necesaria en un futuro.

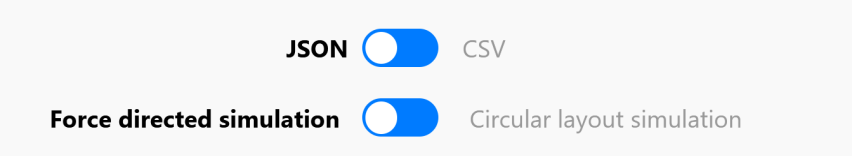

Figura 4.7: Imagen del componente ToggleButton

NodeLinkSetup.vue

En este paso se trata de obtener información relativa a los identificadores de los nodos, y los identificadores asociados en los enlaces para saber cual es el origen y el destino del enlace. Según el formato de datos que se introduzca, se procesarán los datos en consecuencia.

En el formato JSON habrá que identificar que clave guarda la información de los nodos y cual la de los enlaces en el caso de elegir el formato de nodos y enlaces. Esto cambia cuando se selecciona el formato de nodos y lista de adyacencia, ya que los elementos de selección de enlaces dependen de lo que se seleccione en los elementos de selección de los nodos.

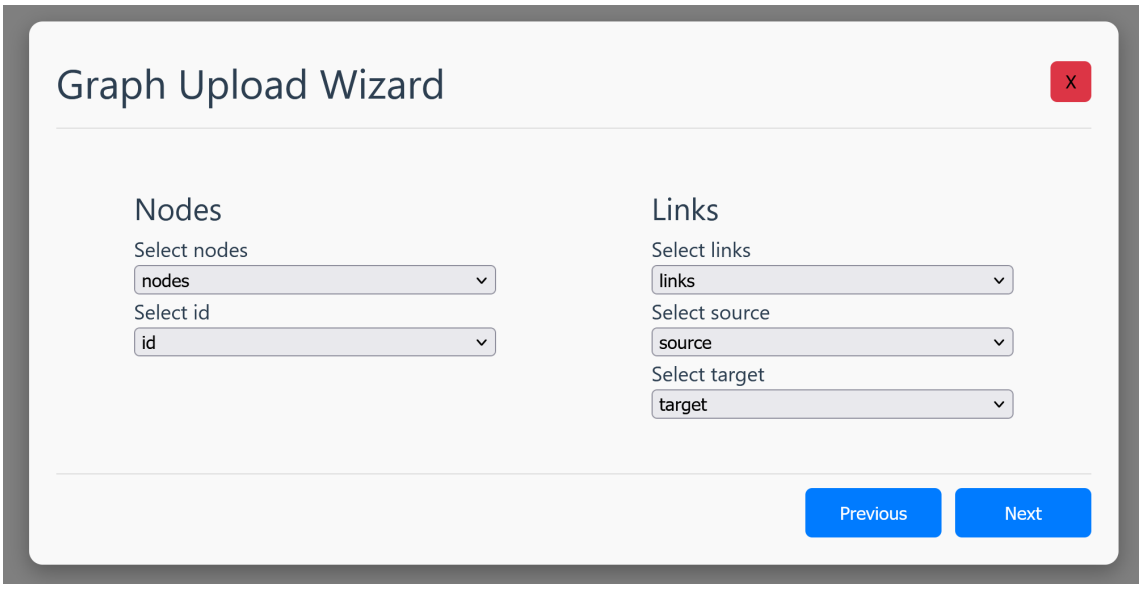

Figura 4.8: Imagen del componente de NodeLinkSetup

En cuanto al formato de CSV, no habrá que elegir que fichero es el de nodos y cual el de enlaces, ya que al momento de subirlos, se identifica cual es cada uno, y el resto de configuraciones funciona igual que el formato gemelo de JSON. Por otra parte en caso de elegir la matriz de adyacencia, habrá que elegir solamente la columna que contiene los datos de los identificadores de los nodos.

AttributeComponent.vue

Este es el ultimo paso para generar el grafo. En este componente se recorren los datos para encontrar aquellos que son atributos de los nodos y de los enlaces. Además se hacen comprobaciones para detectar el tipo de variable en función de la cantidad valores que puede tomar el atributo y su naturaleza. Por ejemplo, si una variable toma valores de texto solo podrá ser de tipo nominal.

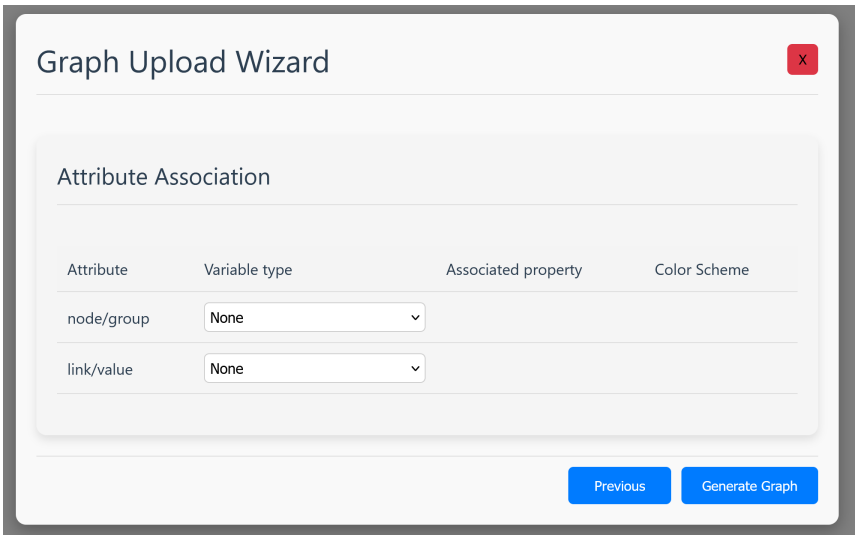

Figura 4.9: Imagen inicial del componente de AttributeComponent

Los elementos de este componente están organizados dentro de una tabla en la que gracias a la reactividad se ha ahorrado mucho trabajo ganando en simplicidad y limpieza del código, habilitando el componente para futuras modificaciones.

4.4.7. Generación de grafos

Cuando llegan los datos de vuelta al componente App.vue, se comienza la generación del grafo apoyándonos en la librería de D3.js, cogiendo el ancho y largo del contenedor que alojará el grafo para que la función sea capaz de renderizar acorde al tamaño que tiene disponible. Además se asegura que antes de que se enseñe el nuevo grafo se haya borrado el contenido del anterior grafo que se haya pintado.

forceSimulation.js

Para crear la simulación de fuerzas se utiliza forceSimulation() una función de D3 que implementa un integrador numérico de velocidad Verlet para simular las fuerzas físicas sobre las partículas, en este caso nodos. Se asume que cada paso de tiempo en la simulación tiene una duración de 1 unidad y que todas las partículas tienen una masa constante.

Dado que estas dos variables son constantes, una fuerza F aplicada a una par-

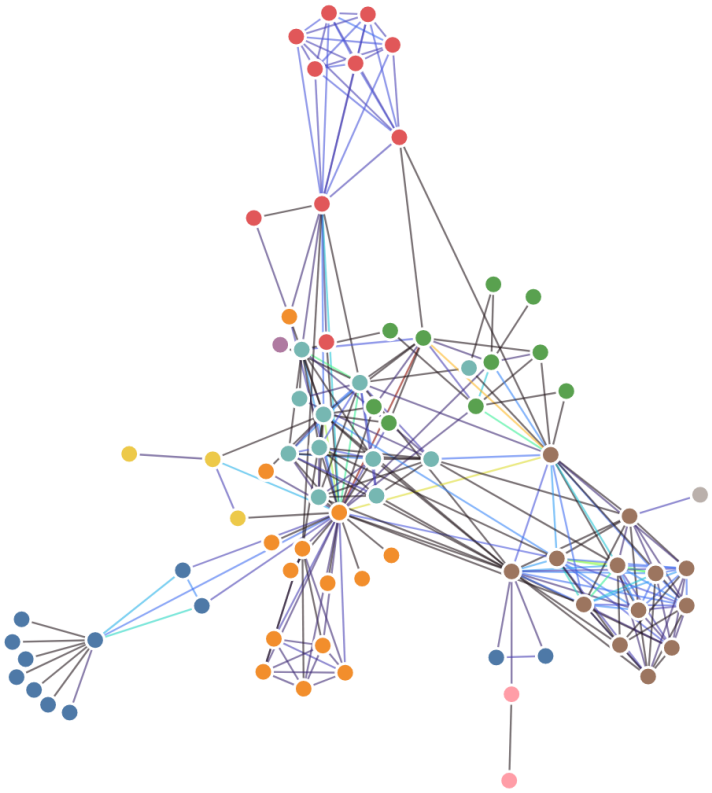

Figura 4.10: Imagen de simulación de fuerzas

tícula se traduce directamente en una aceleración constante a lo largo del intervalo de tiempo. A partir de aquí se le van asignando fuerzas a la simulación utilizando el método especifico simulation.force() [29].

A continuación se define el componente svg que albergará nuestro grafo y le agregamos un soporte para poder hacer zoom sobre el sin necesidad de usar el zoom por defecto del navegador.

Luego se procede a la creación y configuración de nodos y enlaces, donde aplicamos una escala de color o tamaño en función del atributo codificado. En caso de no haber atributos codificados se utilizaran valores por defecto como un color único para cada nodo y un tamaño fijo para los enlaces.

Para este layout, se han especificado funciones que permiten arrastrar nodos y hacer más facil la visualización del grafo.

circularSimulation.js

En cuanto al layout circular, se recorren los nodos, calculando el ángulo de cada uno para asegurar que los nodos se distribuyan uniformemente a lo largo d un

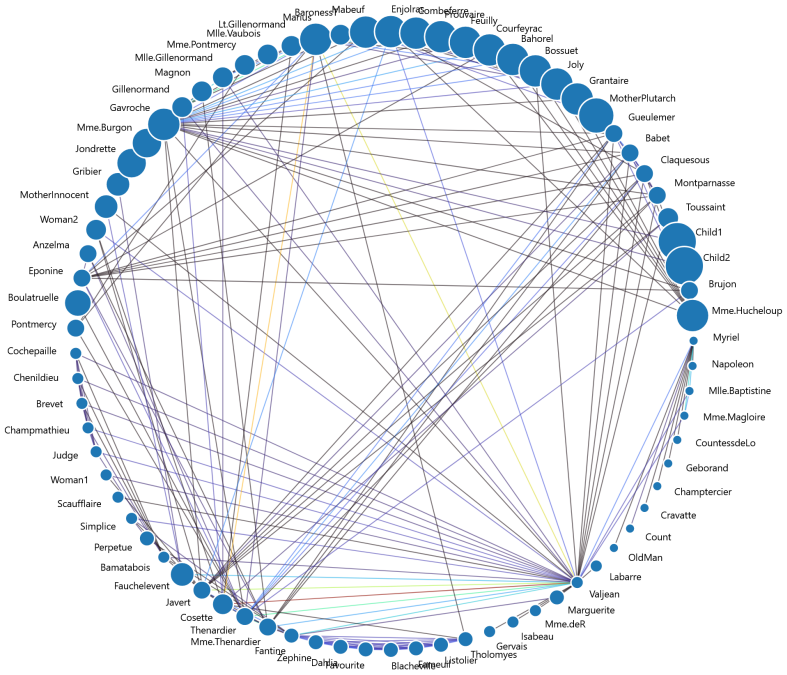

Figura 4.11: Imagen del layout circular

circulo. Después se proceden a calcular las coordenadas del nodo usando la formula $centro+radio*Math.cos(angulo)$ para el eje x y $centro+radio*Math.sin(angulo)$ para el eje y.

Posteriormente se crea el elemento svg y se añaden los nodos y enlaces como la misma funcionalidad explicadas con anterioridad sobre las escalas. Al igual que en el layout de fuerzas se aplica una funcionalidad de zoom sobre el grafo.

Tooltips

Sobre las dos funciones que representan grafos, se ha implementado adicionalmente un tooltip que muestra la información del nodo seleccionado. Dejando un contenedor oculto inicialmente y apareciendo tan pronto como el cursor pasa por encima.

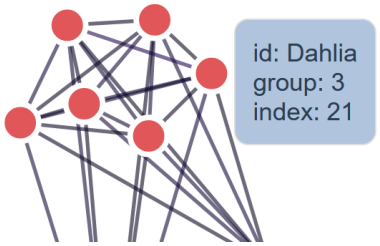

Figura 4.12: Imagen de tooltip de ejemplo

Capítulo 5

Conclusiones y trabajo futuro

Después de haber realizado este trabajo, podemos decir que se han conseguido los objetivos propuestos. Hemos visto cuales son las herramientas y tecnologías involucradas en la visualización de grafos; en base a esa información, hemos diseñado nuestro componente y finalmente hemos logrado implementarlo, garantizando su funcionalidad.

En cuanto a las conclusiones, la realización de este trabajo ha supuesto un considerable aumento de mis conocimientos en los campos de desarrollo web y las tecnologías modernas utilizadas. Además he adquirido habilidades en el uso de una herramienta muy ponente para la visualización de datos como lo es D3.js. Estos dos temas se abordan superficialmente en el grado, por lo que esta experiencia me ha permitido aprender y profundizar en aspectos en los que tenía poca experiencia.

Sin embargo, durante el desarrollo también he encontrado ciertas limitaciones. Aunque he aprendido mucho sobre el desarrollo web, el familiarizarse con el framework de Vue ha ralentizado la planificación propuesta puesto que en la fase de implementación se han encontrado dificultades con el manejo tanto de las variables reactivas como del funcionamiento de las directivas. Por otra parte se hizo complejo la manipulación del DOM con D3.js debido a que requiere un profundo conocimiento sobre el lenguaje de JavaScript.

Dicho esto, el enfoque realizado durante el diseño e implementación de la aplicación, ha supuesto que en el futuro se pueda mantener y añadir nuevas funcionalidades. Algunas características que se pueden realizar a futuro son las siguientes:

- Realización de un menú de navegación que permita filtrar la visualización de los grafos por los valores de los atributos de nodos y enlaces o por las propiedades computadas de estos.
- Agregar más formatos de entrada de datos como listas de aristas o matrices de incidencias, así como, permitir fuentes de ficheros como GraphML.
- Dar soporte a la visualización de grafos multicapa y agrupación de nodos

bajo un mismo atributo para detectar comunidades y relaciones que puedan ser difíciles de ver a simple vista.

Capítulo 6

Análisis de impacto

Tras la investigación realizada y el desarrollo para realizar el trabajo, vamos a ver el impacto potencial en diversos contextos en los pienso que este componente de visualización puede ser beneficioso y que efectos adversos habría asociados al uso de esta tecnología.

En lo personal, el trabajo con lenguajes, herramientas y tecnologías a los que no estoy acostumbrado y no se enseñan en la universidad, ha fomentado un desarrollo de mis habilidades técnicas y mi versatilidad como programador, incrementando a su vez mi capacidad de resolución de problemas.

Por otra parte, la visualización de grafos puede ser de gran ayuda para las empresas, ya que les permitirá comprender mejor sus datos y tomar decisiones basadas en patrones y relaciones detectadas.

Dado que el análisis de redes sociales pasan por la visualización de grafos para encontrar patrones de relaciones, comunidades o personas influyentes, este trabajo puede facilitar la representación visual de todos estos aspectos, mejorando así la comprensión y el análisis de la información contenida.

6.1. Objetivos de desarrollo sostenible

La implementación de un componente de visualización de grafos como este tiene el potencial de generar beneficios significativos en varios contextos. La alineación con los Objetivos de Desarrollo Sostenible refuerza el valor y la relevancia de este trabajo en un contexto global.

- **Trabajo decente y crecimiento económico**: El análisis de la economía de diferentes localidades mediante la visualización de grafos puede ayudar a identificar puntos en los que sea necesario actuar para mejorar la economía.
- **Alianzas para lograr objetivos**: Gracias a la visualización de grafos es posible visualizar las comunidades de personas que trabajan en la consecución de estos objetivos y identificar a las personas influyentes dentro de ellas.

De esta manera podemos conectar estas comunidades y que se beneficien entre ellas.

Bibliografía

[1] B. d. P. F. e. Fonseca, R. B. Sampaio, M. V. d. A. Fonseca y F. Zicker, «Co-authorship network analysis in health research: method and potential use», *Health research policy and systems*, vol. 14, págs. 1-10, 2016.

[2] *A Network Map*. dirección: https://graphviz.org/Gallery/twopi/twopi2.html.

[3] *What is Graphviz?* Dirección: https://graphviz.org.

[4] Dirección: https://gephi.org/features/.

[5] *Features*. dirección: https://gephi.org/features/.

[6] Dirección: https://cytoscape.org/what_is_cytoscape.html.

[7] *Cytoscape main page*. dirección: https://cytoscape.org.

[8] *La imagen que transmite la política española*. dirección: https://www.graphext.com/post/la-imagen-que-transmite-la-politica-espanola.

[9] «Variables y tipos». (), dirección: https://www.jordimas.cat/courses/dataanalysis_es/tema1/analisis_datos_es_variables/.

[10] D. B. West et al., *Introduction to graph theory*. Prentice hall Upper Saddle River, 2001, vol. 2.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, *Introduction to algorithms*. MIT press, 2022.

[12] T. Munzner, *Visualization analysis and design*. CRC press, 2014.

[13] T. Munzner, *Visualization Analysis and Design*. CRC Press, 2014, Figure used under CC-BY-4.0 license. Original figure: "Figure 5.5: Mark Types". dirección: https://www.cs.ubc.ca/~tmm/vadbook/eamonn-figs/fig5.5.pdf.

[14] T. Munzner, *Visualization Analysis and Design*. CRC Press, 2014, Figure used under CC-BY-4.0 license. Original figure: "Figure 5.6: Channels". dirección: https://www.cs.ubc.ca/~tmm/vadbook/eamonn-figs/fig5.6.pdf.

[15] M. Grandjean, «Introduction à la visualisation de données: l'analyse de réseau en histoire», *Geschichte und Informatik*, n.º 18/19, págs. 109-128, 2015. dirección: https://www.martingrandjean.ch/wp-content/uploads/2015/09/Grandjean2015.pdf.

[16] *Circular Graph Layout*. dirección: https://www.yworks.com/pages/circular-graph-layout.

[17] M. Bostock, *What is D3*, 2024. dirección: https://d3js.org/what-is-d3.

[18] AntV, *G6: A Graph Visualization Framework*, 2021. dirección: http://g6-v3-2.antv.vision/en/docs/manual/introduction.

[19] *JavaScript InfoVis Toolkit*. dirección: https://philogb.github.io/jit/index.html.

[20] *Core.js*. dirección: https://philogb.github.io/jit/static/v20/Docs/files/Core/Core-js.html.

[21] F. Hámori, *The History of React.js on a Timeline*, 2022. dirección: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/.

[22] *Describing the UI*. dirección: https://react.dev/learn/describing-the-ui.

[23] E. You, *FIRST WEEK OF LAUNCHING VUE.JS*, 2014. dirección: https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/.

[24] *Introduction*. dirección: https://vuejs.org/guide/introduction.html.

[25] *Angular, version 2: proprioception-reinforcement*, 2016. dirección: https://web.archive.org/web/20170312063434/http://angularjs.blogspot.com/2016/09/angular2-final.html.

[26] *What is Angular?* Dirección: https://v17.angular.io/guide/what-is-angular.

[27] L. Orsini, *What You Need To Know About Node.js*, 2013. dirección: https://readwrite.com/what-you-need-to-know-about-nodejs/.

[28] *Vite Documentation*. dirección: https://vitejs.dev/guide/.

[29] *Force simulations*. dirección: https://d3js.org/d3-force/simulation#simulation_force.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Mon Jun 03 19:21:32 CEST 2024
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)