UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aplicación para el análisis de la red de personajes de Cosmere

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Llobell Alfaro, Alejandro

Tutor/a: Rebollo Pedruelo, Miguel

CURSO ACADÉMICO: 2022/2023

Resumen

Brandon Sanderson se ha proclamado como uno de los escritores más prolíficos de nuestra época. De entre todas sus obras destaca el universo del Cosmere, una serie de libros de fantasía que mezclan la magia con la tecnología haciéndose valer de unas reglas comunes a todo el universo. Las diferentes sagas dentro del Cosmere ocurren en diferentes planetas y sus historias se alargan durante décadas. En este universo tan extenso encontramos personajes que aparecen en distintos libros, algunos de ellos pertenecientes a otros planetas. El análisis consiste en generar un grafo a partir de las relaciones entre personajes. A partir de la red haremos un estudio de esta para descubrir que personajes tienen más influencia en el Cosmere, cómo evolucionan y quienes son los protagonistas de cada libro.

Para realizar el análisis utilizamos un algoritmo de *web scraping* para conseguir los personajes de los libros. Luego realizamos un análisis de reconocimiento de entidades para averiguar las relaciones entre los personajes. Una vez tenemos las relaciones, las mostramos en una página web en forma de grafo. Además, mostramos diferentes métricas como el *pagerank* y el *eigenvector*, las cuales utilizamos más adelante para sacar conclusiones de los libros. Así, hemos podido descubrir un paralelismo en las sagas de *Nacidos de la bruma* y que Kaladin Bendito por la Tormenta es el protagonista del Cosmere.

Palabras clave: análisis de redes, NLP, web scrapping, Angular, Cosmere, Spacy

Resum

Brandon Sanderson s'ha proclamat com un dels escriptors més prolífics de la nostra època. D'entre totes les seves obres destaca l'univers del Cosmere, una sèrie de llibres de fantasia que barregen la màgia amb la tecnologia fent ús d'unes regles comunes a tot l'univers. Les diferents sagas dins del Cosmere ocorren en diferents planetes i les seves històries s'allarguen durant dècades. En aquest univers tan extens, trobem personatges que apareixen en diferents llibres, alguns d'ells pertanyents a altres planetes. L'anàlisi consisteix en generar un graf a partir de les relacions entre personatges. A partir de la xarxa, farem un estudi d'aquesta per descobrir quins personatges tenen més influència en el Cosmere, com evolucionen i qui són els protagonistes de cada llibre.

Per realitzar l'anàlisi, vam utilitzar un algorisme de web scraping per aconseguir els personatges dels llibres. Després, vam realitzar un anàlisi de reconeixement d'entitats per descobrir les relacions entre els personatges. Un cop tenim les relacions, les mostrem en una pàgina web en forma de graf. A més, mostrem diferents mètriques com el Pagerank i l'Eigenvector, les quals utilitzem més endavant per a treure conclusions dels llibres. D'aquesta manera, hem pogut descobrir un paral·lelisme en les sagas de "Nascuts de la boiraï que Kaladin Bendit per la Tempesta és el protagonista del Cosmere.

Paraules clau: anàlisis de xarxes, NLP, web scrapping, Angular, Cosmere, Spacy

Abstract

Brandon Sanderson has proclaimed himself as one of the most prolific writers of our time. Among all his works, the standout is the Cosmere universe, a series of fantasy books that blend magic with technology, relying on common rules throughout the universe. The different sagas within the Cosmere take place on different planets, and their stories span decades. In this extensive universe, we encounter characters that appear in

various books, some of them belonging to other planets. The analysis consists of generating a graph based on the relationships between characters. Through this network, we conduct a study to discover which characters have more influence in the Cosmere, how they evolve, and who are the protagonists of each book.

To perform the analysis, we used a web scraping algorithm to obtain the characters from the books. Then, we conducted an entity recognition analysis to discover the relationships between the characters. Once we have the relationships, we displayed them on a web page in the form of a graph. Additionally, we showed different metrics such as Pagerank and Eigenvector, which we later used to draw conclusions from the books. Thus, we have been able to discover a parallelism in the "Mistborn" sagas and that Kaladin Stormblessed is the protagonist of the Cosmere.

Key words: network analysis, NLP, web scrapping, Angular, Cosmere, Spacy

Índice general

Índice de figuras

Índice de tablas

A todos los
hijos de Honor

CAPÍTULO 1
Introducción

Brandon Sanderson se ha convertido en uno de los autores de fantasía más relevantes de la época contemporánea. El autor estadounidense publicó su primer libro en 2005, Elantris, y desde entonces ha ido publicando un libro tras otro con una regularidad envidiable. Esto sumado a los increíbles mundos de sus historias lo ha llevado a convertirse en un autor de referencia para muchos. De entre todos los mundos que ha escrito destaca el Cosmere, un universo de fantasía caracterizado por tener un sistema de magias fuerte, con unas reglas claras y comunes a todo el universo.

Nosotros recogeremos eso libros y realizaremos un análisis de la red de personajes. Vamos a analizar los libros más importantes del Cosmere para averiguar los personajes que tengan alguna relación y construir un grafo. Luego, utilizaremos técnicas de análisis de redes para sacar conclusiones sobre que personajes conectan los mundos, quienes tienen más peso en la historia y, finalmente, averiguar quién es el protagonista del Cosmere.

1.1 Motivación

Nos parece muy interesante realizar un análisis sobre lenguaje natural. El reto de extraer las relaciones a partir de texto sin estructura utilizando herramientas de análisis del lenguaje natural y de redes. Además, es muy instructivo realizar un proyecto completo, desde el *backend* hasta el *frontend* para acabar publicando una aplicación completa y funcional.

Desde el punto de vista del *fandom*, creemos que nuestro trabajo puede ayudar a desentrañar algunos misterios de los libros y que puede resultar una herramienta muy interesante para los fans más *hardcore*.

1.2 Objetivos

El objetivo es desarrollar una aplicación capaz de analizar un libro y extraer las relaciones de los personajes que aparecen en él. Además, queremos proporcionar una página web dinámica y familiar donde mostrar los resultados obtenidos del análisis de forma que el público general pueda entender los datos.

Para conseguir realizar estos objetivos hemos definido algunos subobjetivos que nos ayudaran a llevar a cabo el proyecto:

O1 : Escribir un algoritmo de *web scraping* para obtener el nombre y los alias de los personajes del Cosmere.

O2 : Realizar un algoritmo de análisis de entidades capaz de, a partir de la lista de personajes y un libro, obtener que personajes están relacionados.

O3 : Construir una página web capaz de mostrar la red de personajes y sus relaciones y algunas métricas de la red.

O4 : Publicar la página web para que todo el mundo pueda consultar y sacar sus propias conclusiones.

1.3 Metodología

La metodología empleada en el desarrollo de este proyecto de software se ha basado en diversas herramientas y enfoques para garantizar una gestión eficiente y una implementación sólida.

Para organizar y capturar las ideas iniciales, se utilizó la aplicación Keep Notes de Google. Esta herramienta permitió registrar y estructurar las ideas del proyecto a medida que surgían. Su familiaridad y uso cotidiano facilitaron la incorporación de nuevas ideas de manera rápida y eficiente. Además, al ser multiplataforma, se pudo acceder a las notas desde diferentes dispositivos, lo que brindó comodidad y flexibilidad durante el proceso de desarrollo.

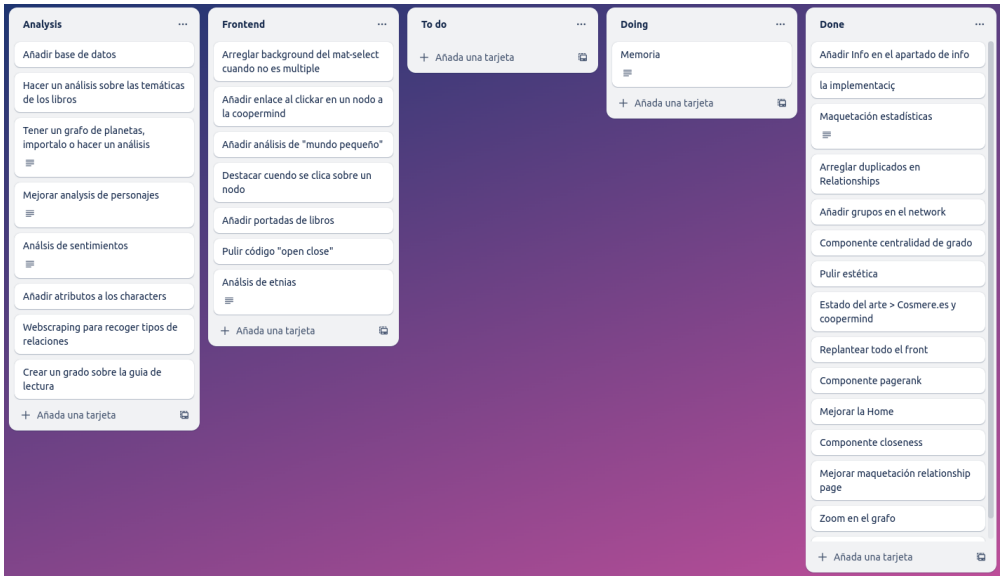

Figura 1.1: Tareas organizadas en tarjetas de Trello

Para la gestión y seguimiento de las tareas, se empleó la plataforma Trello(figura 1.1). Aunque el proyecto se desarrolló de manera individual, la funcionalidad de tableros y tarjetas de Trello permitió organizar las ideas y mantener un seguimiento claro de las tareas realizadas y pendientes. Esta metodología ayudó a estructurar las ideas inicialmente registradas en Keep Notes y a mantener un flujo de trabajo eficiente.

El proyecto comenzó con una reunión con el tutor, donde se estableció la idea principal y se delinearon los enfoques y objetivos del proyecto. Esta reunión proporcionó una base sólida y una dirección clara para iniciar la producción. Dado que el equipo de desarrollo

constaba únicamente de una persona, se tuvo la flexibilidad de adaptar las tareas según las necesidades más apremiantes, alternando entre implementación tanto en el backend como en el frontend, así como tareas de diseño. Esta dinámica flexible y personalizada permitió aumentar la productividad, ya que se pudo enfocar en las tareas que generaban mayor motivación y satisfacción personal.

El éxito en la gestión y desarrollo de este proyecto también se debe a la aplicación de buenas prácticas de programación, siguiendo los principios SOLID. Estos principios, junto con la arquitectura hexagonal y la programación basada en componentes de Angular, permitieron realizar cambios en la implementación de una parte del proyecto sin afectar al resto. Al establecer una base sólida desde el principio y mantenerla inalterada, se logró una mayor estabilidad y mantenibilidad en el código.

1.4 Estructura de la memória

Capítulo 1: Introducción : La introducción comenzará exponiendo el contexto, quien es Brandon Sanderson y que es el Cosmere. También declararemos las motivaciones que nos han llevado a realizar el proyecto, los objetivos y la metodología empleada.

Capítulo 2: Estado del arte : En el estado del arte hablaremos de proyectos similares o relacionados con el proyecto, en que consisten y qué puede aportar nuestro proyecto que no hagan ya estos.

Capítulo 3: Análisis : En el capítulo de análisis describiremos el marco legal de nuestro proyecto y los requisitos funcionales y no funcionales que debe cumplir para que tenga éxito.

Capítulo 4: Diseño : Durante la parte de diseño se expondrá como se va a organizar, así como que necesitamos para comenzar el desarrollo del este.

Capítulo 5: Desarrollo Aquí expondremos los detalles más relevantes de la implementación del código, así como los problemas que han surgido durante el desarrollo.

Capítulo 6: Análisis de la red : Aprovechando la interfaz web de nuestro proyecto realizaremos un análisis del grafo para averiguar quién, es el protagonista actual del Cosmere.

Capítulo 7: Conclusiones : Cerraremos la memoria con las conclusiones donde resumiremos y expondremos las reflexiones finales del proyecto. Además, enunciaremos los trabajos futuros que tenemos planeados para una próxima versión de la aplicación.

CAPÍTULO 2
Estado del arte

2.1 Cosmere y coopermind

Actualmente existen varias páginas web sobre el Cosmere. Primeramente, la página oficial de Brandon Sandeson, brandonsanderson.com(*Wiki oficial del Cosmere y la literatura de Brandon Sanderson*, 2023), donde los lectores pueden mantenerse actualizados sobre las últimas noticias, eventos y lanzamientos relacionados con Sanderson. Además, brandonsanderson.com proporciona acceso a una gran cantidad de recursos exclusivos, como borradores y anotaciones de obras en proceso, entrevistas, artículos y blogs escritos directamente por el autor.

También existe Cosmere.es(*Wiki oficial del Cosmere en español*, 2023), una página web oficial en español sobre el Cosmere. Creada por Ángel Lorenzo y Tamara Tonetti se dedican a traducir todo lo que tenga relación con el Cosmere y Brandon Sanderson. Proporciona recursos como artículos, teorías, mapas y perfiles de personajes, junto con una comunidad activa de entusiastas del Cosmere en castellano.

Por último tenemos la Coppermind(*La Coppermind Wiki - 17th Shard, the Official Brandon Sanderson Fansite*, 2023), una enciclopedia en línea colaborativa y uno de los proyectos más destacados de la comunidad. Es un vasto repositorio de conocimiento sobre el Cosmere y sus obras asociadas. La Coppermind está alimentada por una comunidad de voluntarios que contribuyen con información precisa y completa sobre los mundos, personajes, eventos y conceptos que componen este universo. La página se ha convertido en un recurso invaluable para los lectores que desean profundizar en los detalles y las Conexiones del Cosmere. Permite explorar temas específicos, descubrir curiosidades, resolver dudas y adentrarse en teorías sobre los misterios aún no revelados.

2.2 Network analysis

En el campo de la lingüística computacional y el procesamiento del lenguaje natural, se han llevado a cabo diversos proyectos y estudios para el reconocimiento de entidades en textos y el análisis de los grafos que se generan a partir de las relaciones entre estas entidades.

El reconocimiento de entidades en texto es un proceso que implica identificar y clasificar las menciones de entidades nombradas, como personas, organizaciones, lugares, fechas, cantidades, entre otros, en un texto dado. Esto se logra mediante técnicas de aprendizaje automático y algoritmos que analizan patrones y características lingüísticas para realizar

la extracción de información relevante.

El análisis del grafo resultante puede proporcionar información valiosa sobre la estructura y la semántica del texto. Al aplicar técnicas de procesamiento de grafos, es posible descubrir patrones, identificar comunidades de entidades relacionadas, determinar la importancia de las entidades en función de su grado de conexión, entre otros análisis.

2.2.1. Network of Thrones

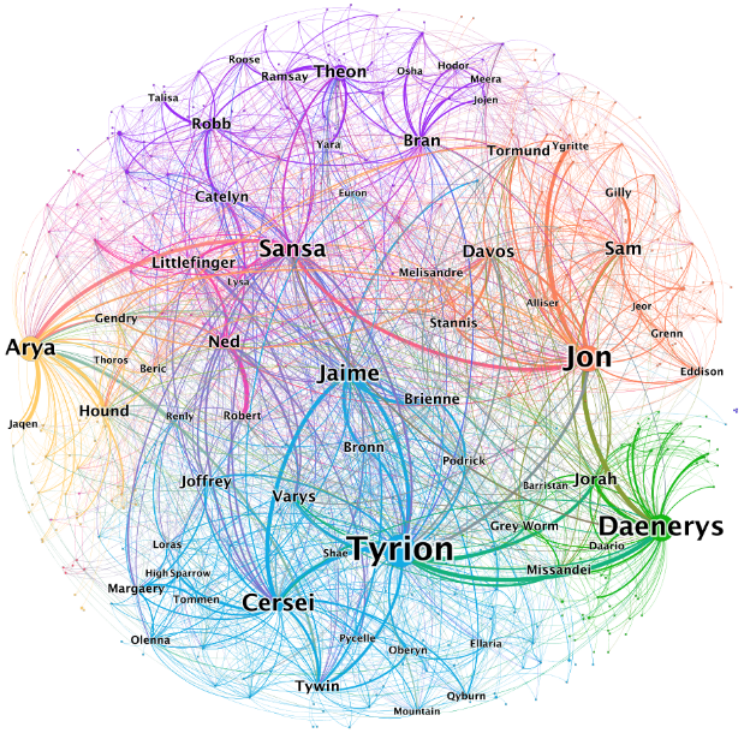

Figura 2.1: Grafo de la red de *Juego de tronos*

Network of Thrones(Beveridge y Shan, 2016) se enfoca en el análisis de redes de la popular saga literaria *Canción de Hielo y Fuego*, escrita por George R.R. Martin. El artículo examina las relaciones entre los personajes y las interacciones sociales que se desarrollan a lo largo de la extensa trama de la saga. El artículo proporciona una visión más profunda de las complejas relaciones que hay en la saga *Canción de Hielo y Fuego*, permitiendo comprender mejor las motivaciones, lealtades y tensiones que impulsan a sus personajes.

Es el artículo que más parecido tiene a nuestro proyecto y, por tanto, el que más valor puede aportarnos. Nos gustaría destacar el análisis de redes ya que su análisis de entidades ha sido adaptado al contexto de los libros de R.R. Martin por lo que no tiene tanto interés para nosotros. Las métricas utilizadas han sido el *pagerank*, el *eigenvector*, el *betweenness*, el grado y el grado ponderado. Además, han usado una combinación de estas métricas para definir lo que han llamado *fractal protagonism* del cual hablaremos más adelante. También es interesante la forma en la que exponen los resultados, poniendo los resultados de cada libro por separado para finalmente realizar el análisis de la saga en su conjunto.

Utilizando técnicas de análisis de entidades, los investigadores construyeron un grafo que representa las Conexiones entre los personajes. Los criterios para decir que dos personajes están relacionados son: dos personajes que hablan entre ellos, dos personajes aparecen juntos en el mismo lugar, o un personaje escucha el nombre de otro personaje. La primera medida es trivial, dos personajes que hablan entre sí han tenido una interacción. También es lógico pensar que si dos personajes se encuentran en un mismo sitio habrán tenido alguna interacción, aunque sea mínima y la tercera la define como una interacción indirecta. Argumentan que de alguna forma debe haber una conexión en la mente de la persona que habla del tercero o del que la escucha para estar hablando de esa tercera persona.

Una de las preguntas que explora este artículo es quien es el protagonista de toda la saga. Como habíamos adelantado, utilizan una medida llamada *fractal protagonism*, una métrica derivada de otras tres: el *pagerank*, el *eigenvector* y el grado. El *fractal protagonism* busca encontrar el personaje que tiene interacciones más importantes(*pagerank*), el que tiene Conexiones importantes(*eigenvector*) y el que tiene más interacciones(grado). Esta decisión se basa en la idea de que un personaje principal, un héroe y un protagonista no tienen necesariamente porque ser la misma cosa. Se explica que un protagonista debe, además de aparecer con frecuencia como lo haría un personaje principal y tener rasgos característicos como un héroe, realizar acciones que avancen la trama y determinar el destino de a quienes rodea.

El artículo también explora temas como la centralidad de los personajes, identificando quiénes son los actores clave dentro de la red, así como las comunidades y grupos de personajes que se forman. Analizan la trama de los libros a través de las relaciones de sus comunidades e intentan explicar la baja o elevada influencia de un nodo según el contexto de los libros.

El proyecto sobre el análisis de los libros de George R. R. Martin no acaba ahí. Los autores han publicado el código fuente por lo que se puede consultar. Este es opensource lo que brinda una herramienta más de análisis de lenguaje natural y análisis de redes.

2.2.2. Joseph Mathia Svoboda personal diaries

Otro ejemplo destacado de proyecto que utiliza el reconocimiento de entidades y el análisis de redes es el trabajo de Sam Fields(Fields, Cole, Oei, y Chen, 2022). Este estudio emplea la biblioteca Spacy, entre otras herramientas, para realizar el análisis de las entidades presentes en los diarios personales de Joseph Mathia Svoboda del siglo XIX en el Imperio Otomano-Irakí. Mediante el reconocimiento de entidades y la construcción de grafos, el proyecto busca distinguir las redes personales de los contextos sociales más amplios, brindando una nueva perspectiva sobre las interacciones humanas y la estructura social en ese período histórico específico.

Method	Agreement type	48			49		
		Prec.	Recall	F	Prec.	Recall	F
SpaCy[a]	Identical	0.69	0.70	0.70	0.74	0.67	0.70
	Overlap	0.86	0.87	0.86	0.91	0.82	0.86
Human-in-the-loop bootstrap	Identical	0.68	0.77	0.72	0.79	0.79	0.79
	Overlap	0.84	0.94	0.89	0.93	0.94	0.93

Figura 2.2: Rendimiento promedio del NER de los diarios personales de Joseph Mathia Svoboda

La metodología empleada para el análisis de entidades es muy interesante. Lo primero que hacen es definir dos tipos de coincidencias: total y solapamiento(figura 2.2). La primera ocurre sí una palabra es exactamente igual al nombre de un personaje y la segunda cuando es parcial o se puede intuir. En el primer análisis se dieron cuenta de que algunas entidades no se estaban identificando correctamente, algunas no se detectaban y otras que sí en realidad no eran personas. Para solucionarlo pusieron a una persona a revisar lo que el análisis automático identificaba para reajustarlo en caso de ser necesario, obteniendo unas ligeras mejoras.

Además, utilizaron *named entity linking*(NEL) para solucionar tres problemas. El primero eran las entidades de gran longitud. Spacy separa los nombres que contengan tres o más *tokens* por lo que, si un personaje aparece con nombres y apellidos, el programa lo separaría en dos partes. El segundo problema son los alias. Alguien se puede referir a una misma persona por diferentes nombres y es importante poder identificarla como el mismo personaje. Por último, está el problema de que dos personajes aparezcan muy juntos de tal forma que hay que indicarle al programa si debe combinarlos en una misma entidad o dejarlos separados.

Este ejemplo ilustra cómo las técnicas de reconocimiento de entidades y análisis de grafos, como las utilizadas en Spacy, pueden ser aplicadas en proyectos de investigación para obtener conocimientos valiosos sobre relaciones y contextos en textos históricos y contemporáneos.

2.2.3. Saltamundos

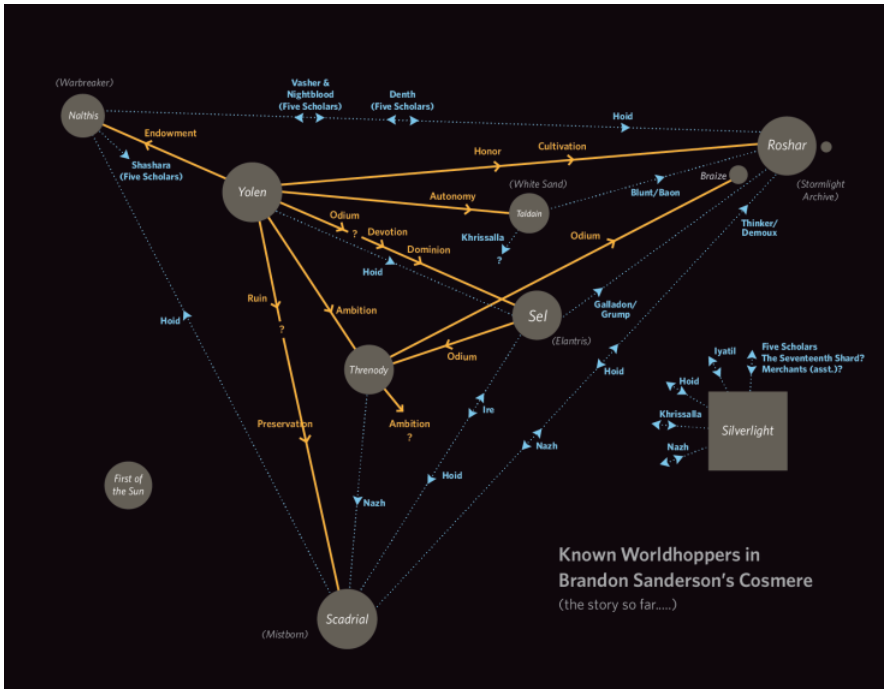

Figura 2.3: Red de los saltadores de mundo

Existe un pequeño artículo escrito por Chris Lough(Lough, 2017) que habla sobre los *saltamundos*, personajes del Cosmere que son capaces de viajar entre los distintos planetas del universo de los libros. En el artículo se estudian las relaciones entre los planetas en función de los personajes que los han visitado construyendo un grafo dirigido. Aunque no hacen un análisis sobre la red si se exponen conclusiones sobre las implicaciones que esto tiene ya que la mayoría de los saltamundos son esquirlas, personas milenarias con los que se puede explicar el nacimiento de la vida en diferentes planetas. Además, la construcción de la red se hizo de forma artesanal, haciendo una investigación sobre los libros, por lo que no fue necesario la creación de un programa capaz de automatizar el proceso.

Aunque la mayor parte del artículo se dedica a realizar preguntas, una de las conclusiones a las que llegan es que Yolen, el planeta originario de las Esquirlas, parece ser el punto de partida. Las dudas surgen después, ¿qué Esquirlas llegaron primero a qué planetas? ¿Todos dejaron el sistema Yolen al mismo tiempo o lo hicieron en etapas?, son algunas de las preguntas que deja este artículo.

CAPÍTULO 3
Análisis

En este capítulo, se realizará un análisis integral que abarcará el marco legal de los libros del Cosmere sujetos a derechos de autor, así como los requisitos funcionales y no funcionales de la aplicación. Además, se presentarán los diagramas de casos de uso que ilustrarán las interacciones entre los usuarios y el sistema. A través de este análisis, se establecerán las bases legales, se definirán las funcionalidades clave y se visualizará el flujo de trabajo de la aplicación, sentando así las bases para su desarrollo y diseño.

3.1 Marco legal y ético

El presente trabajo de investigación se desarrolla dentro de un marco legal y ético que establece los principios y las pautas a seguir en relación con la utilización de información proveniente de obras literarias y recursos en línea. Es esencial comprender y respetar los aspectos legales y éticos relevantes para garantizar la integridad y la validez de este proyecto.

En primer lugar, es importante destacar la cuestión de los derechos de autor y la propiedad intelectual. En este estudio, se analizan los libros del autor Brandon Sanderson, específicamente de su serie conocida como el Cosmere, lo cual implica el uso de textos literarios protegidos por derechos de autor. Como tanto el autor como la editorial *Dragonsteel* provienen de Estados Unidos podemos regirnos por el *fair use*(Office, 2023). Para que nuestro proyecto entre dentro del *fair use* debe cumplir cuatro factores que expondremos ahora.

Factor 1: Carácter y propósito de uso El primer factor establece que las obras con ánimo de lucro no pueden hacer uso del *fair use*. Como nuestro proyecto es un trabajo *opensource* que no busca de ninguna forma sacar provecho monetario cumple con este propósito.

Factor 2: La naturaleza En Estados Unidos es legítimo el uso de obras protegidas si se utilizan para proyectos de investigación o educación. Nuestro proyecto es de investigación y educación por lo que cumple perfectamente este factor.

Factor 3: La cantidad utilizada Es importante que, aunque el proyecto cumpla el resto de los factores, no use una cantidad elevada de la obra. En nuestro caso es algo complicado ya que por una parte utilizamos todo el contenido de los libros para el análisis de entidades. Pero, como la parte expuesta de la aplicación, la página web, solo utiliza los nombres de los personajes y los libros también cumplimos este tercer factor.

Factor 4: Las implicaciones en el mercado Por último, es importante que el proyecto que usa una obra protegida por derechos de autor no tenga un efecto negativo en la misma. Con esto se refiere a no se puede proteger un proyecto por *fair use* si este perjudica a las ventas de la obra. Nuestra aplicación no solo no lo perjudica, sino que además podría servir para ayudar a popularizar la obra.

Por todos estos motivos creemos que el proyecto está recogido dentro del ámbito del *fair use* y es completamente legítimo.

Además, se ha empleado la página web *coopermind.es* como una fuente adicional de información. Esta página web es de acceso público y contiene datos relacionados con los personajes presentes en los libros del Cosmere. Para enriquecer el análisis, se ha realizado el proceso de *web scraping* con el fin de obtener nombres y otros datos de los personajes. Se ha intentado realizar esta actividad el mínimo número de veces con el objetivo de no perjudicar a la página además de leer solo la información que es pública.

3.2 Análisis de requisitos

3.2.1. Requisitos funcionales

Los requisitos funcionales son las características y funcionalidades específicas que nuestro proyecto debe tener para satisfacer las necesidades de los usuarios y cumplir con los objetivos del proyecto. A continuación, se enumeran los requisitos funcionales de nuestro proyecto:

RF1 : Proporcionar una navegación clara y estructurada para ayudar a los usuarios a encontrar la información que necesitan de manera rápida y sencilla.

RF2 : El usuario ha de ser capaz de personalizar su experiencia al aplicar filtros y seleccionar qué información desea visualizar en el grafo. Por ejemplo, el usuario podrá filtrar por libro específico, personajes favoritos o tipos de relaciones.

RF3 : Debe permitir la importación de datos de los libros, incluyendo información sobre los personajes y las relaciones entre ellos.

RF4 : Debe ser capaz de realizar el análisis de las relaciones entre los personajes con base en la información proporcionada en los libros. Debe identificar las Conexiones y la naturaleza de las relaciones, como amistad, parentesco o enemistad.

RF5 : Ha de ser posible generar un grafo visual que represente las relaciones entre los personajes. El grafo debe mostrar de manera clara y comprensible las Conexiones y la estructura de la red de personajes.

RF6 : El sistema debe realizar un análisis del grafo generado, identificando patrones, comunidades o *clusters* de personajes, así como medidas de centralidad o importancia de los nodos en la red.

3.2.2. Requisitos no funcionales

Además de los requisitos funcionales, es importante tener en cuenta los requisitos no funcionales que deben cumplirse para garantizar un producto de alta calidad y una experiencia de usuario satisfactoria:

RNF1 : El sistema debe ser intuitivo y fácil de usar, con una interfaz de usuario clara y bien diseñada que permita a los usuarios navegar y personalizar la experiencia de manera sencilla.

RNF2 : La web debería ser capaz de manejar grandes cantidades de datos y generar el grafo de relaciones de manera eficiente. Debe ofrecer un rendimiento óptimo incluso cuando la base de datos sea extensa.

RNF3 : El proyecto ha de ser escalable, lo que significa que debe poder adaptarse a un aumento en el número de usuarios, así como a una mayor cantidad de libros y personajes sin comprometer su rendimiento.

RNF4 : Se tiene que garantizar la seguridad y la privacidad de los datos de los usuarios y de la información importada de los libros. Debe contar con medidas de seguridad para proteger la integridad de los datos.

RNF5 : La página web debe ser compatible con diferentes plataformas y dispositivos, lo que permitirá a los usuarios acceder y utilizar la aplicación desde una variedad de dispositivos, como computadoras de escritorio, dispositivos móviles o tabletas.

3.3 Diagramas de casos de uso

A continuación, se presentan los diagramas de casos de uso para describir las diferentes funcionalidades y características del sistema. Describen las acciones que realizan los actores involucrados en el sistema y las secuencias de eventos que ocurren en cada acción.

Caso de uso	C1. Ver contenido
Actores	Usuario
Requisitos	RF1, RF3
Precondiciones	El usuario ha entrado en la página web
Secuencia	1. Se cargan los datos 2. Redirige a la pestaña de *Red*
Postcondiciones	El usuario ha visto el contenido deseado

Tabla 3.1: C1. Ver contenido

Caso de uso	C2. Filtrado del contenido
Actores	Usuario
Requisitos	RF2, RF3
Precondiciones	El usuario está en el Home
Secuencia	1. Se cargan los datos 2. Redirige a la pestaña de *Análisis* 3. Se habre el menú de configuración 4. Se selecciona el libro *El imperio final*
Postcondiciones	Se ha filtrado el contenido de la web

Tabla 3.2: C2. Filtrado del contenido

Caso de uso	C3. Generar lista de personajes
Actores	Administrador
Requisitos	RF4, RF5
Precondiciones	Está la consola abierta en el directorio de *scripts*
Secuencia	1. Se ejecuta el comando de web scraping 2. Mueve el fichero generado con los personajes a la base de datos del *frontend*
Postcondiciones	Los personajes de la página web han sido actualizados

Tabla 3.3: C3. Generar lista de personajes

Caso de uso	C4. Preanálisis de un libro
Actores	Administrador
Requisitos	RF5
Precondiciones	Está la consola abierta en el directorio de *scripts*
Secuencia	1. Se ejecuta el comando para realizar el preanálisis del texto
Postcondiciones	Se ha generado un fichero binario con el análisis del texto

Tabla 3.4: C4. Preanálisis de un libro

Caso de uso	C5. análisis de un libro
Actores	Administrador
Requisitos	RF6
Precondiciones	Existe un fichero binario del libro con su preanálisis y está la consola abierta en el directorio de *scripts*
Secuencia	1. Se ejecuta el comando para realizar el análisis del libro 2. Mueve el fichero *json* generado con las relaciones de los personajes a la base de datos del *frontend*
Postcondiciones	Se ha creado un fichero con las relaciones del libro en el *frontend*

Tabla 3.5: C5. análisis de un libro

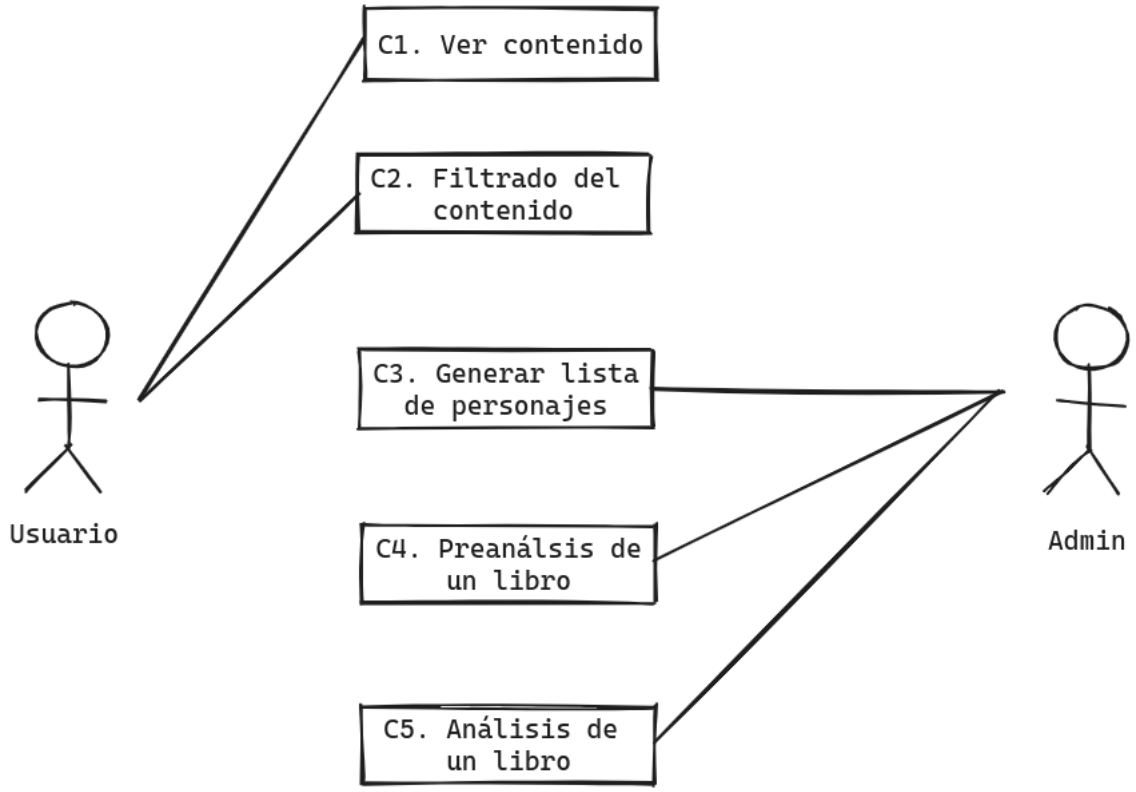

Figura 3.1: Diagrama de casos de uso

CAPÍTULO 4
Diseño

4.1 Arquitectura del sistema

4.1.1. Domain driven design

Domain-driven-design(Evans, 2004) o DDD es un concepto que busca crear una correlación directa entre el mundo que intenta plasmar el proyecto de software y el mundo real. La idea es poner énfasis en que la nomenclatura utilizada es crucial durante el desarrollo de software porque para los seres humanos es mucho más sencillo entender que debe hacer algo si este tiene un reflejo en el mundo real.

4.1.2. Arquitectura hexagonal

La arquitectura hexagonal(*Hexagonal Architecture*, 2023) recoge los principios del *doman-driven-design* y aísla la capa de dominio de los elementos externos de nuestra aplicación con dos nuevas capas, la capa de aplicación y la capa de infraestructura. Esta arquitectura permite cumplir los principios SOLID, los cuales son muy importantes para asegurar el mantenimiento de la aplicación y las buenas prácticas.

Nuestro proyecto se divide en dos aplicaciones, el *backend* y el *frontend*, cada cual con unas necesidades específicas por lo que, a pesar de aplicar las mismas ideas de arquitectura, sus implementaciones varían según las necesidades de cada una. En líneas generales ambas partes se dividirán en dominio, aplicación e infraestructura.

Comenzando por el dominio, en esta capa tendremos las reglas de negocio. Es la capa más próxima al programador y, por tanto, donde más control tenemos. La idea es tener los elementos que conformarán nuestra aplicación y que no sean dependientes de elementos externos como librerías o tecnologías.

La segunda capa, la de aplicación, es donde se realizan las acciones. Actúa como puente entre las otras dos y se encarga de aplicar la lógica de negocio, cogiendo las reglas establecidas en el dominio y llevando a cabo los casos de uso de nuestra aplicación.

Finalmente tenemos la capa de infraestructura, la más externa. Expuesta al exterior, es la capa donde menos control tenemos ya que es la única donde tenemos dependencias externas. Algunas de sus funciones son comunicarse con la base de datos para asegurar la permanencia de los datos y enviar o recibir peticiones del exterior. También es donde se ubicará cualquier elemento que contenga alguna librería de terceros.

4.2 Modelo de persistencia

Para el modelo de persistencia de la aplicación, se ha optado por utilizar ficheros *json* como formato de almacenamiento. Esta elección se basa en permitir realizar peticiones de manera sencilla y rápida a pesar de no contar con una base de datos convencional. Las entidades se guardan en los *assets* de la página web, lo que evita la necesidad de levantar un servidor.

En total, hay cinco entidades, y cada una de ellas se almacena en un fichero *json* como una lista plana de entidades, excepto en el caso de las *relationships*, donde se utiliza un fichero separado por cada uno de los libros. Esta estructura ha sido diseñada para brindar la posibilidad de solicitar un único libro sin tener que pedir todos los libros en cada consulta.

Aunque modificar una fila de la base de datos puede resultar complicado sin una base de datos convencional, este enfoque permite reanalizar el libro afectado evitando tener que realizar el análisis de todos los libros en cada ocasión. Además, al estar escrito en lenguaje natural, permite que, de ser necesario, se hagan modificaciones manualmente mediante reemplazos o escribiendo una fila manualmente.

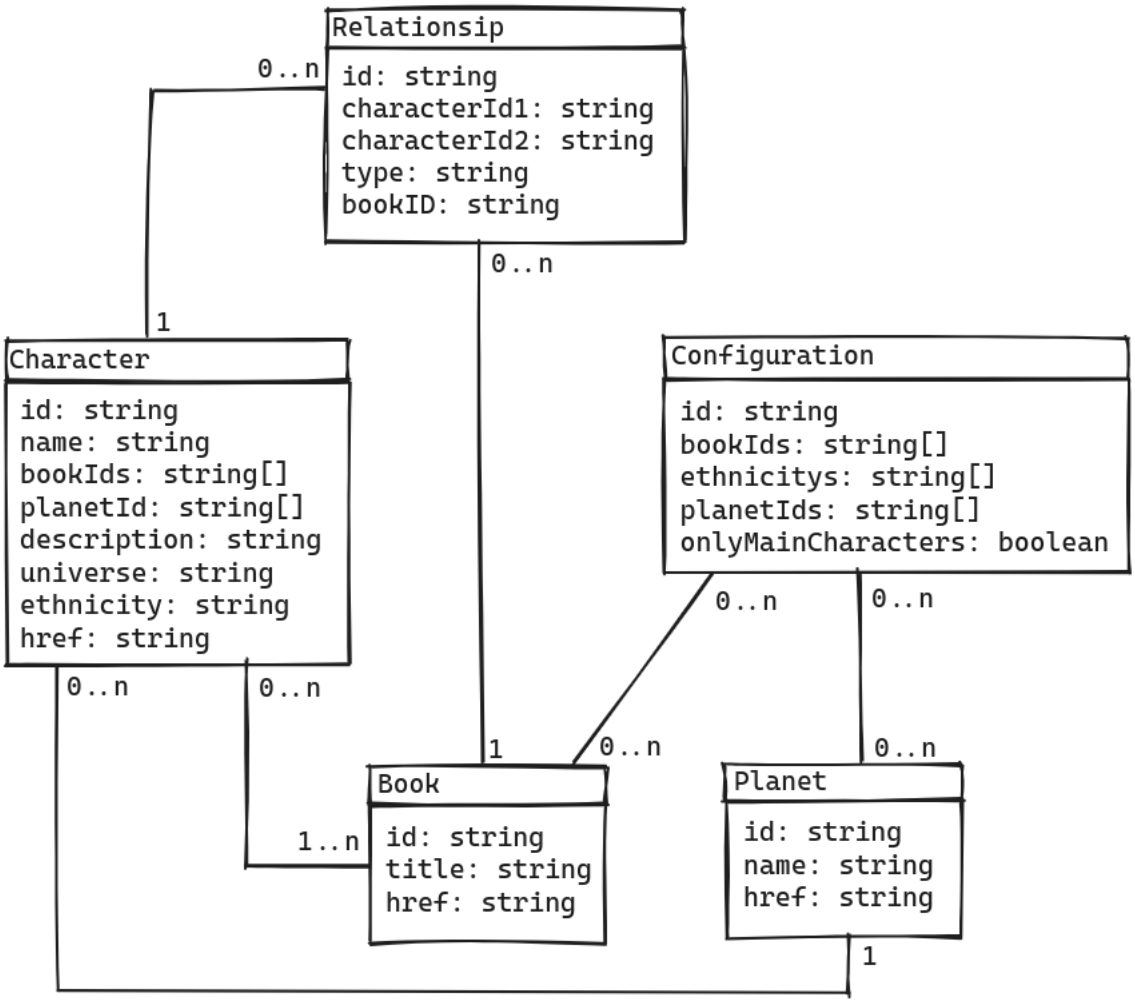

Figura 4.1: UML

Cada una de las entidades tiene un atributo id, un identificador único universal. Esto permite asegurar una forma de buscar una entidad de forma inequívoca además de obligar a los programadores a no utilizar el identificador para otras funciones. Se ha optado por cohesionar todas las entidades evitando así más latencia de la necesaria ya que casi todas las entidades están relacionadas y, de acoplarlas, obligaría a traer toda la base de datos en cada llamada.

La entidad *character* representa a un personaje del Cosmere. Está relacionada con un conjunto de libros en los que aparece y el planeta en el que nació. Las *relationship* son de un libro. Con esto conseguimos poder diferenciar si un personaje aparece en uno o más libros. Una *relationship* modela que dos personajes se han conocido según los criterios ya comentados. Tanto los libros como los planetas contienen un título o un nombre, respectivamente, y un href que referencia a la *coopermind*. Finalmente tenemos la configuración, una entidad que no se guarda actualmente en base de datos pero que sirve para mantener en la *store* que información mostrar al usuario y así poder filtrarla.

4.3 Backend

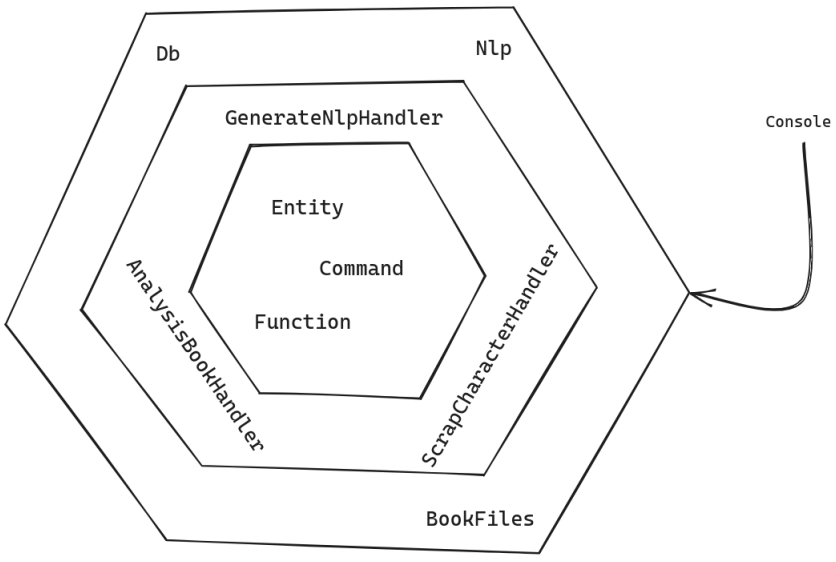

Figura 4.2: Esquema de la arquitectura del backend

Para el *backend* hemos divido el código en tres partes: dominio, aplicación e infraestructura. Cada capa se representa como un nivel dentro de los hexágonos en la figura 4.2.

```python
class Character:
    def __init__(self, command: CreateCharacter):
        self.id = command.id
        self.name = command.name
        self.href = command.href
        self.bookIds = command.bookIds
        self.planet = command.planet
        self.aliases = command.aliases
        self.titles = command.titles
        self.ethnicity = command.ethnicity
        self.universe = command.universe

    def __str__(self):
      return self.name

    def __str__(self):
        string = '{'
        string += '\"id\": \"' + self.id + '\", '
        string += '\"name\": \"' + self.name + '\", '
        string += '\"href\": \"' + self.href + '\", '
        string += '\"bookIds\": '  + listToJsonArray(self.bookIds) + ', '
        string += '\"planet\": \"' + self.planet + '\",'
        string += '\"aliases\": ' + listToJsonArray(self.aliases) + ', '
        string += '\"titles\": ' + listToJsonArray(self.titles) + ', '
        string += '\"universe\": \"' + self.universe + '\",'
        string += '\"ethnicity\": \"' + self.ethnicity + '\"'

        string += '}'
        return string

    def __hash__(self):
      return hash(self.id)

    def __eq__(self, character):
        return character.id == self.id
```

Figura 4.3: Modelo de la entidad *Character* en el *backend*

```python
class CreateCharacter:
    def __init__(self, id: str, name: str, bookIds: list, planet: str, aliases: list, titles: list, ethnicity: str, universe: str):
        self.id = id
        self.name = name
        self.href = "https://coppermind.net/wiki/"+id
        self.bookIds = bookIds
        self.planet = planet
        self.aliases = aliases
        self.titles = titles
        self.ethnicity = ethnicity
        self.universe = universe
```

Figura 4.4: Comando de creación de la entidad *Character*

```python
def write_entities(bookName: str, characters: list, relationships: list):
    print("Writing json files 🚀🚀🚀")
    write_json('../db/characters-'+bookName+'.json', characters)
    write_json('../db/relationships-'+bookName+'.json', relationships)
    print("DONE!")

def write_relationships(relationships: list, bookName: str):
    write_json(relationships, '../../infrastructure/db/relationships-'+bookName+'.json')

def write_characters(characters: list, bookName: str):
    write_json(characters, '../../infrastructure/db/characters-'+bookName+'.json')

def write_json(object_list: list, jsonPath: str,):
    object_str = '['
    for i, e in enumerate(object_list):
        object_str += str(e)
        if i < len(object_list) - 1:
            object_str += ', '
    object_str += ']'
    with open(jsonPath, "w") as json_file:
        json_file.write(str(object_str))

def readJson(route: str):
    f = open(route)
    data = json.load(f)
    f.close()
    return data

def listToJsonArray(array: list):
    res = "["
    for i, e in enumerate(array):
        res += '\"' + e + '\"'
        if i < len(array) - 1:
            res += ", "
    res += "]"
    return res
```

Figura 4.5: Función de utilidad para escribir entidades en formato *json*

En el dominio tenemos las clases y funciones de utilidad. Estas son la clase de *relationship* y la clase de *character*. Además, tenemos una carpeta de comandos haciendo uso del patrón de diseño. Las funciones de utilidad son funciones de escritura que recogen listas de nuestros objetos y las escriben en formato *json*. Para poder hacer esto ha sido necesario crear funciones *str* en cada objeto para tener los objetos en un tipo de datos serializarle. También ha sido necesario proveer a cada clase de una función hash para poder ser guardados en estructuras de datos como los diccionarios.

La capa de aplicación se compone de tres scripts principales que se explicarán más adelante. Para ejecutar cada uno de los ficheros se emplea el terminal. Esto nos provee la capacidad de pasarle argumentos de forma sencilla para, por ejemplo, cambiar que libro se quiere analizar.

Por último, tenemos la infraestructura. Compuesta por tres carpetas es donde se persisten los datos. Aquí se guardan ficheros *txt* de los libros, los objetos *nlp* del preanálisis y los *json* resultantes del procesamiento de los libros. Lo ideal sería haber hecho un controlador en la infraestructura que se comunique con la capa de aplicación, pero por la naturaleza simple del *backend* se decidió que no era necesario y usamos la consola para comunicarnos directamente.

4.4 Frontend

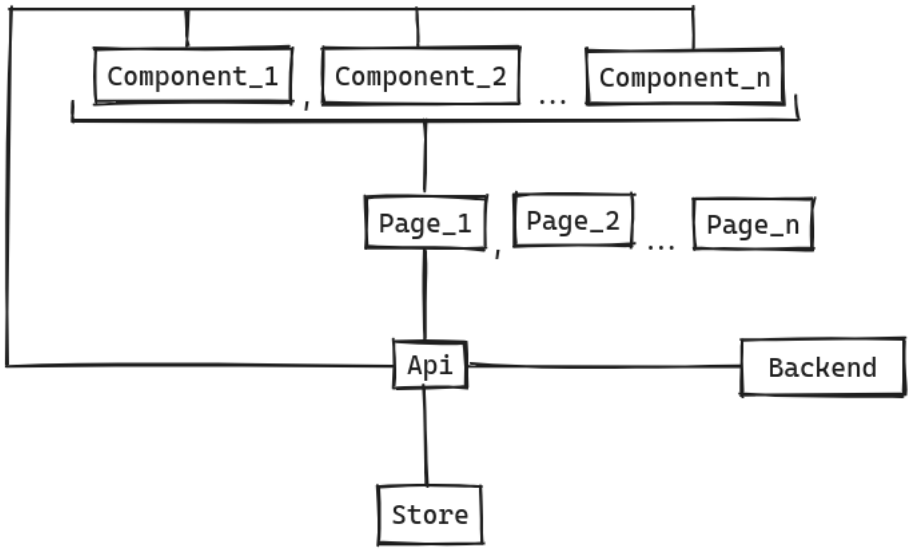

Figura 4.6: Esquema de la arquitectura del frontend

Como se ha comentado anteriormente, en este proyecto se ha aplicado una arquitectura hexagonal por lo que el *frontend* se ha organizado siguiendo esta filosofía. Todo el grueso de la aplicación se encuentra dentro de la carpeta src. En esta carpeta podemos encontrar las carpetas *app*, *assets*, *domain*, *enviroments*, *styles* y *theme*. Las carpetas *enviroments*, *styles* y *theme* contienen ficheros de configuración, del resto hablaremos más adelante.

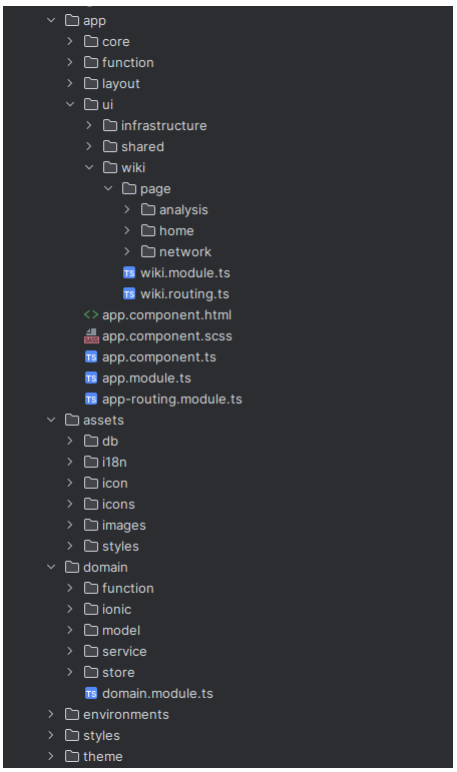

Figura 4.7: Estructura de directorios del *frontend*

En el directorio raíz también tenemos la carpeta *docs* donde se guarda la aplicación una vez construida para luego poder desplegarla. La configuración se realiza desde el archivo *angular.json*, desde donde también se especifica el entorno de producción y la ubicación de la hoja de estilos entre otros.

4.4.1. Assets

El uso principal de la carpeta *assets* es el de guardar la información del análisis de los libros. Su función es actuar como una base de datos donde se guardan los archivos *json* para, posteriormente, hacer peticiones a la misma página web y recuperar los datos. Esto permite guardar la información dentro de la propia web y así evitar tener un servidor lo que aumentaría muchísimo el coste del proyecto tanto por el tiempo de producción como por el coste de mantenimiento una vez terminado. También se guardan *assets* comunes como la imagen principal de la web y los ficheros de traducción.

4.4.2. Domain

Dentro de la carpeta *domain* se encuentran los elementos fundamentales de la aplicación. Aquí se albergan las interfaces que representan las entidades, así como las funciones *helper* que ayudan en la lógica del negocio. Además, la carpeta contiene subcarpetas como *service* donde se encuentran las API y otros servicios relacionados, y la carpeta de *store* para el almacenamiento de datos.

4.4.3. App

La carpeta *app* dentro del proyecto de software juega un papel crucial en la arquitectura en general, adoptando una estructura similar a la carpeta de infraestructura en un proyecto de arquitectura hexagonal. Esta carpeta está dedicada a todo lo relacionado con la página web en sí, utilizando Angular como infraestructura base. Dentro de la carpeta *app*, se encuentran cuatro subcarpetas principales: *core*, *function*, *layout* y *ui*.

La carpeta *core* alberga archivos de configuración y tiene la responsabilidad de levantar y mantener en funcionamiento la aplicación. Su función principal es garantizar el correcto arranque y operación de todos los componentes y servicios necesarios.

La carpeta *function* tiene una finalidad similar a la carpeta *function* del dominio. Aquí se encuentran utilidades proporcionadas en forma de funciones puras, diseñadas para ayudar a los distintos componentes de la aplicación en su lógica y funcionalidades específicas.

En cuanto a la carpeta *layout*, esta contiene el marco o estructura principal de la aplicación. Es aquí donde se encuentra la base que agrupa y organiza todos los demás componentes de la aplicación. Esta separación permite realizar cambios específicos en elementos como la barra de navegación, adaptándola, por ejemplo, para dispositivos móviles, sin necesidad de modificar el resto de los componentes. De esta manera, se sigue el principio de *open-close* de los principios SOLID.

Finalmente, la carpeta *ui* es el núcleo de la aplicación. Aquí se encuentran las páginas y los componentes que forman la página web en sí. Esta carpeta está subdividida en distintos contextos, tales como *infraestructure*, *shared* y *wiki*. La carpeta *wiki* representa la página web propiamente dicha, es decir, lo que se visualiza. Aquí se encuentran cada una

de las páginas que a su vez se componen de componentes tanto de este mismo contexto como del resto. La carpeta *shared* contiene componentes que no están asociados a un contexto específico. Estos componentes de Angular son independientes de la aplicación en sí y podrían utilizarse en otros proyectos sin mayores inconvenientes. Por último, la carpeta *infraestructure* agrupa los componentes que tienen dependencias con librerías externas. Mediante este aislamiento, se garantiza la flexibilidad para cambiar componentes en caso de que una librería deje de ser compatible o de brindar soporte, sustituyéndola por otra sin generar problemas significativos en el resto del proyecto.

4.4.4. La página web

La página web se compone de tres partes fundamentales que brindan una experiencia completa a los usuarios. Estas partes son la *home*, el *network* y el análisis.

En la sección de *home*, los visitantes pueden encontrar una descripción detallada de la aplicación y qué pueden esperar al utilizarla. Aquí se ofrece una visión general de sus funcionalidades principales, destacando los beneficios y características más relevantes.

El *network* es otra sección importante de la página web. En esta parte, los usuarios pueden explorar un grafo interactivo que representa las Conexiones entre los personajes. Este grafo permite visualizar y comprender las relaciones entre los diferentes elementos dentro del contexto del proyecto. Además, los nodos y los arcos del gráfo se colorean aplicando un algoritmo de comunidades.

Por último, está la sección de análisis, donde se presentan diversas gráficas y visualizaciones relacionadas con la red representada en el grafo. Estas gráficas proporcionan información valiosa sobre la estructura y las características de la red, lo que permite a los usuarios profundizar en su comprensión y realizar análisis más detallados.

Además de estas secciones principales, la página web también incluye una pestaña de configuración. En esta sección, los usuarios tienen la posibilidad de personalizar su experiencia al seleccionar qué libros desean ver en la aplicación. Esto brinda flexibilidad y permite a los usuarios adaptar la información y los datos mostrados según sus preferencias y necesidades específicas. La configuración se mantiene entre la red y el análisis.

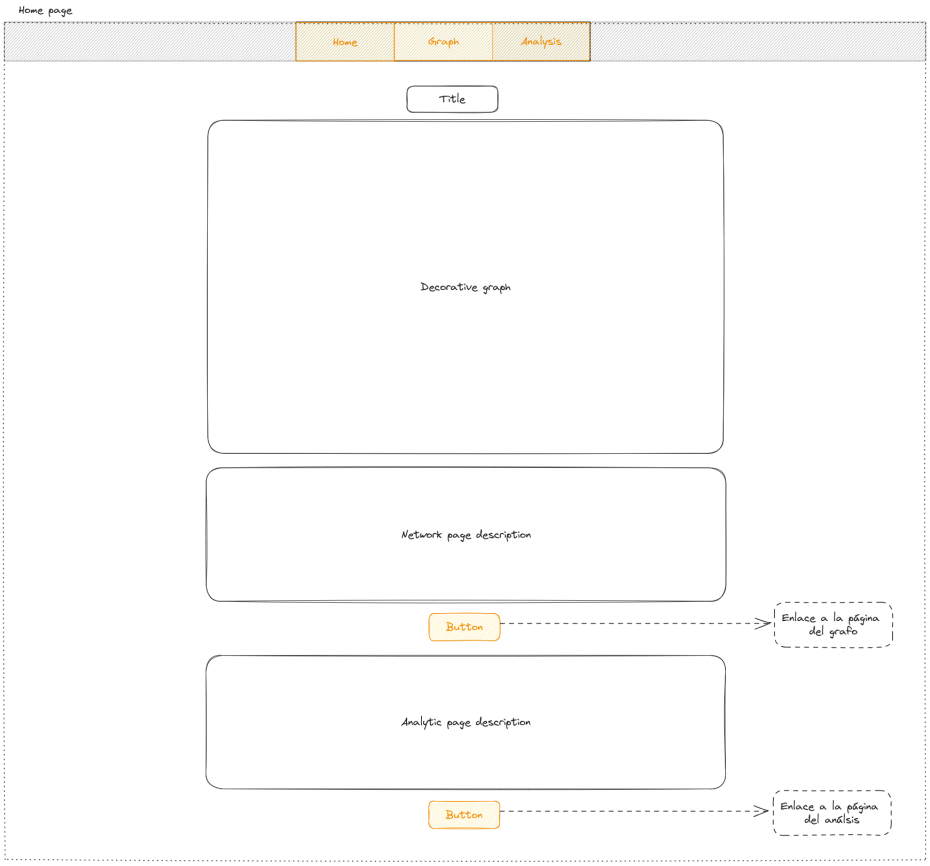

Figura 4.8: Mockup *home page*

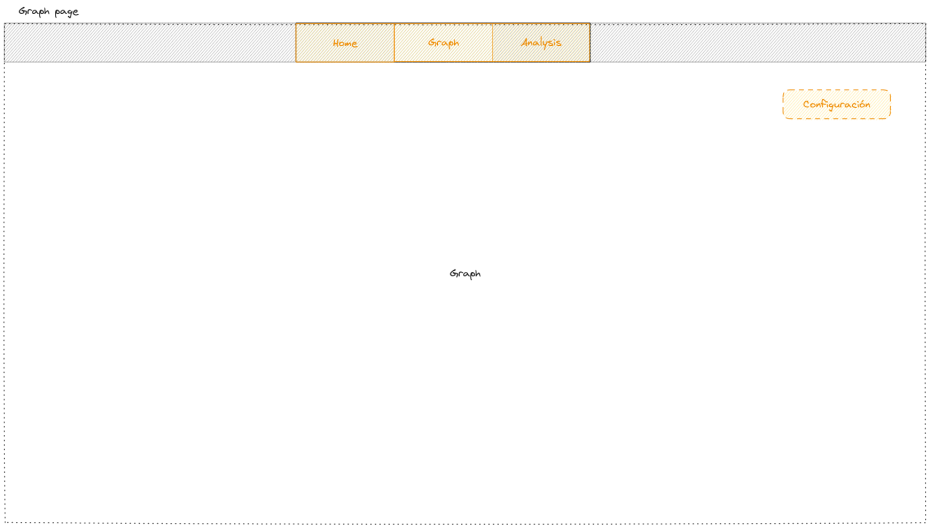

Figura 4.9: Mockup *network page*

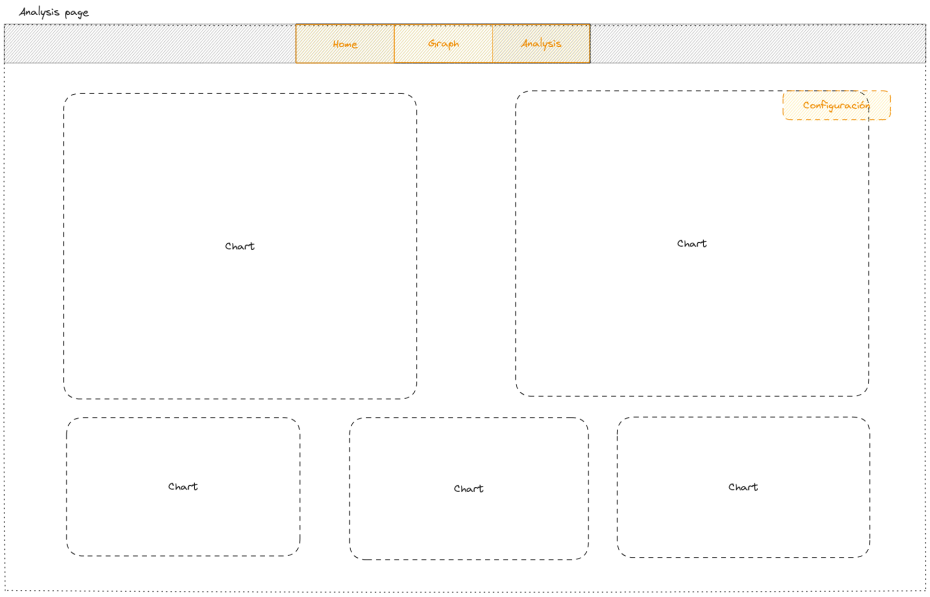

Figura 4.10: Mockup *analysis page*

4.5 Tecnologias empleadas

4.5.1. WebStorm

WebStorm, es un ambiente de desarrollo integrado (IDE) desarrollado por JetBrains, que ofrece una amplia gama de características para mejorar la eficiencia en la programación. Dentro de sus diferentes versiones, WebStorm destaca como una herramienta especialmente diseñada para el desarrollo de aplicaciones web.

Una de las principales características de WebStorm es su avanzado motor de autocompletado inteligente, que nos ha ayudado durante el proceso de desarrollo permitiéndonos escribir código con mayor velocidad y sin errores. Esto se debe a que el IDE ofrece sugerencias de código en tiempo real, lo que hace más fácil el trabajo.

Además, WebStorm cuenta con un indexador, una herramienta que mejora más la velocidad y eficiencia durante el proceso de programación. Esta utilidad es capaz de analizar y comprender la estructura de un proyecto, indexando todos los archivos y dependencias relacionadas. Gracias a esto, WebStorm nos proporciona una navegación rápida por el código, autocompletar inteligente, *refactoring* preciso y sugerencias contextuales. Al tener un índice completo y actualizado, localizar clases, métodos y variables se vuelve rápido y sencillo, lo que agiliza el desarrollo y evita pérdidas de tiempo innecesarias al buscar referencias o realizar cambios en el código.

WebStorm también se integra muy bien con sistemas de control de versiones como Git, lo que nos ha permitido trabajar de manera más eficiente y segura, guardando los cambios en la nube y permitiendo hacer *rollback* cuando fuera necesario.

4.5.2. Angular

Angular(Jain, Bhansali, y Mehta, 2014) es un *framework* de Typescript ampliamente utilizado para construir aplicaciones web. Ofrece una gran cantidad de características que hacen que el desarrollo web sea más rápido y sencillo, especialmente en lo que respecta a la creación de aplicaciones de una sola página (SPA).

Una de las características más destacadas de Angular es su modularidad. Esto nos ha permitido dividir la aplicación en módulos, lo que permite separar diferentes partes de la aplicación en diferentes archivos y carpetas. Esto hace que el código sea más fácil de leer y entender, lo que a su vez reduce la complejidad y aumenta la escalabilidad de la aplicación. Además, los módulos se pueden reutilizar en diferentes partes de la aplicación o en diferentes aplicaciones, lo que ahorra tiempo y esfuerzo.

Otra ventaja importante de Angular es su arquitectura basada en componentes. Los componentes son elementos independientes de la interfaz de usuario que se pueden reutilizar en diferentes partes de la aplicación. Esta característica nos ha permitido construir componentes complejos a partir de componentes más simples y utilizarlos para crear una interfaz de usuario coherente en toda la aplicación. Esto hace que la aplicación sea más fácil de mantener y actualizar, ya que los cambios realizados en un componente se propagan automáticamente a todas las partes de la aplicación donde se utiliza ese componente.

Además, Angular cuenta con una amplia variedad de herramientas y bibliotecas que hacen que el desarrollo web sea más rápido y sencillo. Por ejemplo, la biblioteca RxJS(*RxJS*

Reactive Extensions Library for JavaScript, 2023) facilita la gestión de la asincronía y la programación reactiva, lo que mejora significativamente el rendimiento de la aplicación.

4.5.3. Typescript

TypeScript es un lenguaje de programación de código abierto que fue lanzado por Microsoft en 2012. Fue creado como una alternativa a JavaScript, aunque está basado en él y se compila en JavaScript. TypeScript se diferencia de JavaScript(Flanagan y Ferguson, 1998) por incluir tipado estático, lo que ayuda a evitar errores en tiempo de compilación y mejora la calidad del código.

El objetivo principal de TypeScript es proporcionar una herramienta de desarrollo más robusta. Los creadores de TypeScript notaron que JavaScript, aunque muy flexible y fácil de aprender, no ofrecía suficientes garantías. Por ello, decidieron crear TypeScript para agregar características adicionales que permitan a los desarrolladores escribir código más seguro, más fácil de mantener y más escalable.

TypeScript tiene una estrecha relación con JavaScript, ya que está basado en él y se compila en JavaScript. Esto significa que el código TypeScript se puede ejecutar en cualquier plataforma que soporte JavaScript.

Entre las características de TypeScript destacan el tipado estático, que permite detectar errores en tiempo de compilación en lugar de en tiempo de ejecución. También ofrece una mayor facilidad para *refactoring*, lo que facilita la tarea de cambiar el código de una aplicación sin tener que preocuparse por afectar otros elementos. Además, TypeScript ofrece soporte para programación orientada a objetos, incluyendo clases, interfaces y herencia.

Aunque la caracteristica más importante es que Angular está construido con TypeScript y es el lenguaje recomendado para desarrollar aplicaciones con este *framework*.

4.5.4. Tailwind

Tailwind(*Documentation - Tailwind CSS*, 2023) es un *framework* de CSS que se ha vuelto muy popular entre los desarrolladores de aplicaciones web en los últimos años. Fue creado por Adam Wathan, Steve Schoger, Jonathan Reinink y David Hemphill en 2017, con el objetivo de simplificar el proceso de diseño y desarrollo de interfaces de usuario para la web.

La principal característica de Tailwind es su enfoque en las clases utilitarias, que permiten aplicar estilos de manera rápida y eficiente. Cada clase utilitaria corresponde a un estilo específico, como colores, tipografía, márgenes, entre otros, lo que hace que sea muy fácil aplicar estilos a los elementos de una página web.

Además, Tailwind cuenta con una gran cantidad de clases utilitarias predefinidas, lo que significa que no tenemos que escribir todo el código desde cero. Esto acelera significativamente el proceso de desarrollo, ya que podemos simplemente utilizar las clases existentes para aplicar los estilos deseados.

4.5.5. Git

Git(*Documentation - Git*, s.f.) es un sistema de control de versiones de software de código abierto que fue creado por Linus Torvalds en 2005. A lo largo de los años, Git se ha convertido en uno de los sistemas de control de versiones más populares y utilizados en el mundo del desarrollo de software, y ha sido utilizado en proyectos tan famosos como Linux, Android y Ruby on Rails. La principal característica de Git es su capacidad para controlar y registrar los cambios realizados en el código fuente de un proyecto de software.

Git ofrece una serie de características que lo hacen especialmente útil para el desarrollo de software. Entre ellas se incluyen la capacidad de hacer *rollback* para deshacer cambios, la capacidad de crear ramas de desarrollo independientes para trabajar en diferentes partes del código simultáneamente y la capacidad de realizar *merge* de ramas para fusionar los cambios realizados por diferentes desarrolladores.

Incluso siendo una sola persona Git puede ser de mucha utilidad, ofreciendo la capacidad de mantener un historial completo de los cambios realizados en el código fuente, lo que permite a los desarrolladores ver cómo ha evolucionado el código a lo largo del tiempo y realizar *rollback* en caso de que algo salga mal.

Otra ventaja importante de Git es que permite almacenar sus repositorios en la nube, lo que proporciona una capa adicional de seguridad y redundancia en caso de que se produzcan fallos en el hardware o en el equipo.

4.5.6. Spacy

Spacy(Honnibal y Montani, 2017) es una biblioteca de procesamiento de lenguaje natural (NLP, por sus siglas en inglés) escrita en Python. Se ha convertido en una de las herramientas más populares y poderosas para trabajar con texto en Python debido a su eficiencia, facilidad de uso y versatilidad.

La libreria se ha usado para hacer el procesamiento de los libros. Spacy realiza un análisis capaz de extraer los nombres completos de las entidades por oración, lo que ha simplificado significativamente el procesamiento de los libros.

4.5.7. D3.js

D3.js es una biblioteca de JavaScript ampliamente utilizada para crear visualizaciones de datos interactivas en la web. D3, que significa *Data-Driven Documents* (Documentos basados en datos), permite manipular y presentar datos utilizando estándares web como HTML, SVG y CSS.

En la aplicación se ha utilizado D3.js para generar y mostrar los gráficos necesarios. Estos gráficos, tanto el grafo principal como las diferentes gráficas del análisis de red, dependen de componentes que están construidos utilizando D3.js. Para desarrollar estos componentes, he recurrido ampliamente a los ejemplos proporcionados en la página web oficial de D3.js.

La elección de D3.js se basa en varias razones. En primer lugar, es una biblioteca muy poderosa y completa, con un amplio soporte(Bostock, 2012) y una documentación extensa. Esto ha facilitado el proceso de desarrollo y solución de problemas, ya que se ha

podido aprovechar la gran cantidad de recursos disponibles en la comunidad de usuarios de D3.js.

Además, D3.js ha permitido lograr un alto grado de personalización en la apariencia y funcionalidad de los gráficos. Al tener un control granular sobre cada elemento visual y las interacciones, puedo adaptar completamente la apariencia y el comportamiento de los gráficos según los requisitos específicos de mi aplicación.

Sin embargo, es importante mencionar que el uso de D3.js no es tan simple como utilizar una biblioteca específica de gráficos o de redes. Requiere un nivel de conocimiento y experiencia más profundo en JavaScript y en la propia biblioteca. Aunque esto ha implicado un desafío adicional en el desarrollo, los beneficios de utilizar D3.js han superado ampliamente las dificultades técnicas, permitiéndome crear visualizaciones altamente personalizadas y de calidad en mi aplicación.

4.5.8. Graphology

Esta librería(Plique, 2023) de JavaScript se ha utilizado para realizar los análisis sobre el grafo. Contiene muchos de los algoritmos básicos de análisis de redes como el cálculo del *pagerank* o el análisis de comunidades. La elección de esta librería viene por la necesidad de hacer análisis de redes en el *frontend* ya que la aplicación requería una forma dinámica de hacer el análisis. Si no fuese de esta forma seria imposible hacer un panel de configuración donde poder elegir que libros elegir. Además, es una librería muy sencilla de utilizar por lo que aprender a usarla es relativamente fácil.

CAPÍTULO 5
Desarrollo

5.1 Análisis del Cosmere

5.1.1. Web scraping

El técnica de *web scraping* es una herramienta fundamental para extraer información de manera automatizada de la *coopermind*, utilizando técnicas de web scraping implementadas en Python con la ayuda de la biblioteca Beautiful Soup.

El propósito principal de este código es recopilar datos relevantes sobre los personajes presentes en el sitio web mencionado. Para ello, se realiza un proceso de extracción de información en la sección *Category:Characters*, donde se encuentran listados todos los personajes junto con las URL correspondientes a sus páginas individuales.

Una vez obtenidas las URL, se procede a iterar sobre ellas para realizar el *scraping* de cada página de personaje de manera individual. La página web de *coopermind* destaca por su estructura altamente organizada al tratarse de una wiki. Cada personaje cuenta con una tabla que contiene información valiosa, como su nombre único (también utilizado como identificador), títulos, etnia, planeta de origen, alias y el universo al que pertenecen.

La información adicional recolectada de cada personaje, como los alias, resulta especialmente relevante para llevar a cabo el análisis de los libros. Esto permite filtrar y clasificar los personajes en el *frontend* según su origen (planeta u etnia). Además, el dato del universo es utilizado para descartar aquellos personajes que no pertenecen al universo del Cosmere. Esta distinción resulta útil, ya que en la pantalla inicial de *Category:Characters* también se incluyen personajes de otras sagas o universos que no están relacionados con el Cosmere.

5.1.2. Generación de ficheros NLP

Los ficheros NLP son el resultado del análisis de Spacy sobre un texto. Antes de ejecutar el *script* es necesario dividir los libros en capítulos. Se ha definido que una división de capítulo es una línea que solo contenga números. Es importante realizar este preproceso porque Spacy es incapaz de analizar textos tan largos.

Para la generación del fichero NLP de un libro del Cosmere, lo primero que se hace es leer de la base de datos (el directorio con los archivos *txt*) el libro en cuestión. Este proceso se realiza a través de la especificación de un parámetro que se pasa por consola, correspon-

diente al nombre del libro.

Una vez que el libro se ha cargado, se procede a utilizar el modelo de lenguaje *en_core_web_trf* de Spacy. Este modelo específico se elige debido a que los libros están en inglés, y se espera obtener los resultados más adecuados en este idioma, el original.

```
def separar_texto(texto):
    patron = r'\n\d+\n'
    resultado = re.split(patron, texto)
    return resultado
```

Figura 5.1: Expresión regular línea con solo números

El siguiente paso consiste en dividir el libro en capítulos. Como se ha avanzado antes, los libros contienen líneas con números así que se usa una expresión regular (figura 5.1) para realizar un *split*.Es necesario dividir el libro de entrada para que Spacy sea capaz de realizar el análisis. Además, esta división en capítulos brinda la ventaja de posibilitar análisis específicos por capítulo en el futuro, en caso de ser necesario. Cada capítulo queda como un objeto NLP de Spacy, se almacena en una estructura de datos llamada *DocBin* y se guarda en disco usando la función *to_disk* de Spacy.

El preanálisis de los textos se realiza debido a que es bastante caro computacionalmente por lo que si no se guardase en disco ralentizaría demasiado las iteraciones del análisis de los libros.

Es importante destacar que este script tiene la capacidad de funcionar con cualquier texto que se encuentre dividido en capítulos o que haya sido previamente tratado para ser estructurado de esa manera. Los «capítulos» pueden ser representados mediante líneas individuales que contienen exclusivamente un número, lo cual se utiliza como una referencia para identificar cada uno de los capítulos en el texto. También es necesario contar con una lista con los nombres y alias de los personajes de los cuales se quiere extraer la red.

5.1.3. Análisis de un libro

El análisis de un libro se realiza siguiendo el siguiente procedimiento. En primer lugar, se carga el archivo *DocBin* que contiene una lista de objetos *nlp* representando los capítulos del libro. Se utiliza un parámetro en consola para indicar que libro se quiere analizar. Además, se carga la lista de personajes que ha sido previamente generada mediante *web scraping*.

A partir de esta lista de personajes, se descartan aquellos personajes que no pertenecen al universo del Cosmere. Esta selección se realiza utilizando la información del universo a la que pertenece cada personaje, presente en la lista obtenida del *web scraping*. Idealmente se querría poder descartar personajes que no aparezcan en el libro para evitar que dos personajes con el mismo nombre o apodo se confundan, pero actualmente no se ha automatizado. Para descartar personajes que no pertenezcan al libro en cuestión se debe introducir su id en una lista de personajes a descartar.

Posteriormente, se crea un diccionario invertido donde se establece una relación entre el nombre y los alias de cada personaje con su identificador. Esta estructura de datos permite acceder rápidamente a la información de un personaje en función de su nombre o alias.

A continuación, se procede a iterar cada capítulo del libro y, dentro de cada capítulo, se itera sobre las oraciones. Para esto, se utiliza el atributo *sents* proporcionado por los objetos *nlp*. De cada oración, se iteran las entidades utilizando el atributo *ents* proporcionado por Spacy. El atributo *label* indica si la entidad es de tipo *PERSON* y, si es así, se añade su identificador al conjunto de personajes de la oración usando el diccionario invertido. El conjunto de personajes se reinicia en cada oración a no ser que la oración anterior y la nueva contengan comillas. Esto se hace debido a que se considera que dos oraciones que contengan comillas son partes de un mismo dialogo.

Los conjuntos de personajes se van guardando en una lista a medida que se analiza el libro. Una vez que se ha completado el análisis de todo el libro, se itera la lista de conjuntos para extraer todas las relaciones existentes. En este caso, se considera que dos personajes están relacionados si aparecen en la misma oración, es decir, si están presentes en un mismo conjunto de identificadores. Además, se considera que dos oraciones son la misma si la oración actual y la anterior contienen comillas. Esto se hace para juntar frases de diálogos, mejorando los resultados.

Finalmente, la lista de relaciones obtenida se guarda en fichero en formato *json*. Esto permite un almacenamiento estructurado, rápido para el *frontend* ya que se guarda en texto plano.

5.2 Desarrollo de la página web

Es importante comentar que existe una *api* y una *store* asociada a cada entidad. Además, la implementación de las *api* y las *store* son siempre las mismas por lo que se va a usar la entidad *Relationship* para ejemplificar las implementaciones.

5.2.1. Model

En el desarrollo del proyecto, hemos utilizado interfaces de TypeScript como los modelos de dominio. Estos modelos desempeñan un papel fundamental al proporcionar tipos de datos durante el proceso de programación, para prevenir errores humanos. Gracias a los modelos, podemos asegurarnos de mantener una lógica de negocio coherente que refleje el mundo real. Al imponer límites y restricciones, nos ayudan a escribir un código más limpio y escalable.

```
export interface Relationship extends Entity {

  characterId1: string;

  characterId2: string;

  type: RelationshipType;

  bookId: string;
}
```

Figura 5.2: Modelo de *Relationship*

Un ejemplo de modelo seria el modelo de la clase *Relationship*(figura 5.2). Se declara una interfaz que extiende de otra interfaz llamada *Entity* que tiene un atributo id de

tipo *string*. Luego cada modelo tiene los atributos que necesite y que se han establecido anteriormente.

```
export interface D3Node {
  id: string;

  label: string;

  group: string;

  score: number;
}

5+ usages  Dinacode
export interface D3Link {
  source: string;

  target: string;

  weight: number;

  group: string;
}
```

Figura 5.3: Entidades de D3.js

Además de los modelos de dominio, también hemos implementado modelos de infraestructura. Estos modelos están estrechamente ligados a las bibliotecas y herramientas utilizadas en el proyecto. Su principal función es facilitar la transformación de entidades del dominio a entidades de infraestructura sin necesidad de conocer los detalles internos de los componentes.

```
export const charactersToD3Nodes = (characters: Character[], relationships: Relationship[]): D3Node[] => {
  if (characters.length === 0)
    return [];
  if (relationships.length === 0)
    return [];

  const normalizedScore : Record<string, number>  = characterScoresFromRelationships(characters, relationships);
  return characters.map(character : Character  => ({
    id: character.id,
    label: character.name,
    group: character.planet,
    score: normalizedScore[character.id]
  }));
};
```

Figura 5.4: Función de ayuda *charactersToD3Nodes*

```
export const relationshipsToLinks = (relationships: Relationship[]): D3Link[] => {
  const groupedEdges: Record<string, number> = {};
  for (const relationship : Relationship  of relationships) {
    const key : string  = `${relationship.characterId1}-${relationship.characterId2}`;
    if (groupedEdges[key]) {
      groupedEdges[key] += 1;
    } else {
      groupedEdges[key] = 1;
    }
  }
  let resultEdges: D3Link[] = [];

  for (const key : string  in groupedEdges) {
    const [source : string , target : string ] = key.split( separator: '-');
    const weight : number  = groupedEdges[key];
    const group : "undefined"  = 'undefined';
    resultEdges.push({source, target, weight, group});
  }
  return resultEdges;
};
```

Figura 5.5: Función de ayuda *relationshipsToLinks*

Esto se logra mediante el uso de funciones puras que realizan las conversiones necesarias(figura 5.4)(figura 5.5). Al emplear modelos de infraestructura, evitamos errores comunes, como omitir algún atributo necesario, ya que podemos tener la certeza de que las entidades que llegan a los componentes tendrán todos los atributos requeridos.

5.2.2. Api

Las APIs desempeñan un papel crucial en la aplicación porque son responsables de realizar las solicitudes de lectura de datos, constituyendo la capa más externa del sistema. Ellas son las encargadas de realizar llamadas a la base de datos, sin importar cuál sea su ubicación actual. Incluso si los datos se encuentran actualmente en los *assets*, el diseño de la aplicación está preparado para que en el futuro sea posible cambiar la ubicación de estas solicitudes sin generar problemas significativos.

```
public get<T>(routeName: string): Promise<T> {
  const url : string  = this.router.generate(routeName);
  return new Promise<T>( executor: async (resolve, reject) : Promise<void>  => {
    try {
      const httpResponse : T  = await firstValueFrom(this.http.get<T>(url).pipe(
        catchError( selector: (error: any) => throwError(error))
      ));

      resolve(httpResponse);
    } catch (e) {
      reject(e);
    }
  });
}
```

Figura 5.6: Función *get* de *ApiClient*

Todas las APIs de la aplicación se inyectan el *ApiClient* que tiene una función común a todas, el *get*. Esta función es la encargada de llamar al *endpoint* que se necesite, en nuestro caso, a los *assets*.

Las APIs se encargan de almacenar los objetos resultantes en la *store*, que actúa como

un almacén centralizado de datos en la aplicación. Este enfoque permite una gestión eficiente y coherente de la información, facilitando su acceso y actualización desde diferentes componentes de la aplicación. En cuanto a las funciones principales de las APIs, se pueden distinguir tres categorías.

```
async fetchAllRelationshipByBookId(bookId: string): Promise<void> {
  const httpRelationship : Relationship[]  = await this.api.get('relationships-' + bookId) as Relationship[];
  this.store.saveAllRelationship(httpRelationship);
}
```

Figura 5.7: Función *fetchAllRelationships* de *RelationshipApi*

En primer lugar, las funciones de tipo *fetch*, se utilizan para realizar solicitudes de recuperación de información desde la base de datos. Estas funciones gestionan la comunicación con la base de datos y obtienen los datos necesarios para su posterior procesamiento. En nuestra aplicación hemos llamamos a una función *get*(figura 5.6) que hace una llamada a los *assets*. Esta función es la que usan todas las APIs por lo que cuando sea necesario se puede cambiar el *scope* a un servidor solo cambiando la implementación de esa función.

```
cosmereRelationships(): Observable<Relationship[]> {
  return this.store.relationships$.pipe(map( project: relationships : Relationship[]  =>
    relationships.filter(relationship : Relationship  => cosmereBookIds().includes(relationship.bookId))
  ));
}
```

Figura 5.8: Función *cosmereRelationships* de *RelationshipApi*

Por otro lado, las funciones de tipo *get* devuelven un observable, lo que permite obtener datos de manera asíncrona. Esta característica resulta especialmente útil, ya que permite que los componentes se suscriban a estos observables y se mantengan actualizados en tiempo real, sin tener que esperar o recargar la página. Esto mejora significativamente la experiencia del usuario ya que la información se actualiza de forma automática y dinámica. La función *cosmereRelationships*(figura 5.8) además filtra para devolver solo las relaciones que sean de libros del Cosmere usando un *pipe*.

```
syncAllRelationship(): Relationship[] {
  return this.store.syncState().relationships;
}
```

Figura 5.9: Función *cosmereRelationships* de *RelationshipApi*

Finalmente, en situaciones en las que no es posible trabajar con información asíncrona, se utilizan las funciones de tipo *sync*. Estas funciones devuelven información de la *store* de manera síncrona, es decir, sin esperar a la respuesta de una llamada externa. Sin embargo, una limitación de estas funciones es que no garantizan que la información almacenada en la *store* esté actualizada en todo momento. Por lo tanto, se deben implementar acciones adicionales para garantizar la coherencia y actualización de los datos.

5.2.3. Store

La función de la *store* es tan simple como crucial: guardar los datos. Cada vez que se solicita información a la base de datos, es fundamental actualizar la *store* para mantenerlos accesibles y disponibles. La *store* actúa como un almacén centralizado de datos que está siempre a disposición de la API.

```
export interface RelationshipState {
  relationships: Relationship[];
}
2 usages  Dinacode
const emptyState = (): RelationshipState => ({
  relationships: []
});
2 usages  Dinacode
@Injectable({
  providedIn: 'root',
})
export class RelationshipStore extends ComponentStore<RelationshipState> {

  public readonly relationships$ : Observable<Relationship[]>  = this.select( projector: state : RelationshipState  => state.relationships);

  no usages  Dinacode
  constructor() {
    super(emptyState());
  }

  3 usages  Dinacode
  saveAllRelationship(relationships: Relationship[]): void {
    if (isEqual(this.get().relationships, relationships))
      return;
    this.patchState( partialStateOrUpdaterFn: state : RelationshipState  => ({
      relationships: mergeArrays(state.relationships, relationships)
    }));
  }

  1 usage  Dinacode
  syncState(): RelationshipState {
    return this.get();
  }

  no usages  Dinacode
  reset(): void {
    this.patchState(emptyState());
  }
}
```

Figura 5.10: *RelationshipStore*

La *store* utiliza observables para almacenar la información que le es proporcionada por la API, pero lo hace de manera inteligente. Para lograrlo, se emplean dos funciones puras de utilidad: *mergeArrays* y *pushOrReplace*.

```
export const mergeArrays = <T>(originalArray: T[], arrayToMerge: T[]): T[] => {
  if (!originalArray)
    originalArray = [];

  let newArray : T[]  = [...originalArray];

  arrayToMerge.forEach((element: T) : void  => {
    newArray = pushOrReplace(newArray, element);
  });

  return newArray;
};
```

Figura 5.11: Función *mergeArrays*

La función *mergeArrays*(figura 5.11) tiene la tarea de combinar el contenido actual de la *store* con uno o varios elementos nuevos. Su objetivo principal es evitar que los componentes suscritos a esta información consideren el cambio como una actualización completa. De esta manera, se evitan recargas innecesarias y se mejora el rendimiento de la aplicación.

```
export const pushOrReplace = <T>(array: T[], entry: T): T[] => {
  const genericEntry: any = entry;
  let newArray : T[]  = [...array];

  if (some(newArray, predicate: (node: any) : boolean  => node.id === genericEntry.id))
    newArray = newArray.map((node: any) => (node.id === genericEntry.id) ? entry : node);
  else
    newArray.push(entry);

  return newArray;
};
```

Figura 5.12: Función *pushOrReplace*

Por otro lado, la función *pushOrReplace*(figura 5.12) se encarga de agregar o reemplazar un elemento específico en la *store*. Esta función es especialmente útil cuando se necesita actualizar un dato específico sin afectar el resto de los datos almacenados en la *store*. Proporciona flexibilidad y precisión al gestionar los cambios en los datos almacenados.

En conjunto, estas dos funciones de utilidad contribuyen a optimizar el rendimiento y la eficiencia de la aplicación al actualizar y gestionar los datos almacenados en la *store*. Gracias a ellas, los componentes suscritos a la información en la *store* pueden recibir actualizaciones precisas y relevantes, evitando recargas innecesarias y mejorando la experiencia del usuario.

Además de las funciones *save* también tenemos el *syncState* que se usa en el api para la función de tipo *sync* y el *reset* que devuelve el estado de la *store* al *emptyState*. Aunque el ejemplo mostrado(figura 5.10) sea de la entidad *relationship* todas las *stores* son iguales cambiando el tipo de datos.

5.2.4. Estructura de una página

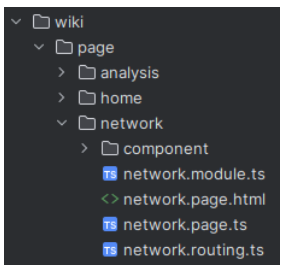

Figura 5.13: Directorios de una página del *frontend*

```
@NgModule({
  declarations: [
    SpoilerAlertComponent
  ],
  imports: [
    PipeModule,
    FuseCardModule,
    MatButtonModule,
    SharedModule,
    CommonModule
  ],
  exports: [
    SpoilerAlertComponent,
  ]
})
export class ModalModule {
}
```

Figura 5.14: Modulo de una modal del *frontend*

Todas las páginas tienen asociadas un módulo, un fichero *routing*, un fichero html y un fichero ts. El módulo se declaran los componentes que se declaran, importan y exportan(figura 5.14) para ese directorio. Es una funcionalidad de Angular y suele ser el IDE el encargado de asegurarse que no exista inversión de dependencias y de declarar los componentes en los módulos correspondientes.

```
export const wikiRouting: Route[] = [
  {
    path: '',
    redirectTo: 'home',
    pathMatch: 'full'
  },
  {
    path: 'home',
    loadChildren: (): any => HomeModule
  },
  {
    path: 'network',
    loadChildren: (): any => NetworkModule
  },
  {
    path: 'analysis',
    loadChildren: (): any => AnalysisModule
  },
];
```

Figura 5.15: *Routing* de la aplicación

```
export const networkRouting: Route[] = [
  {
    path: '',
    component: NetworkPage,
  },
];
```

Figura 5.16: *Routing* de la página de la red

El fichero de *routing* es donde se declaran las direcciones a donde apunta la página. El único fichero de este tipo de interés es el principal de la web que contiene la variable *wikiRouting*(figura 5.15). En él se declaran las rutas de cada una de las páginas: la *home*, la red y el análisis. El resto de *routings* son iguales al de la figura 5.16, apuntan a sus respectivas páginas.

```
<div class="inset-0 flex flex-col min-w-0 w-full overflow-hidden">
  <configuration></configuration>
  <div class="flex flex-row h-full grow justify-center">
    <characters-relationships-graph class="w-full"></characters-relationships-graph>
  </div>
</div>
```

Figura 5.17: HTML de la página de la red

Finalmente tenemos las páginas que las dividimos en su archivo html y su archivo ts. Además, cada página puede contener a su vez una carpeta de componentes propios por si necesita modularizarse. Por ejemplo, la página de la red utiliza un componente llamado *characters-relationships-graph* para la red en sí(figura 5.17). Tanto las páginas como los componentes pueden inyectarse tantas dependencias como quieran por lo que pueden inyectarse las APIs, suscribirse a su contenido y realizar los *fetch* pertinentes(figura 5.18).

```
constructor(private characterApi: CharacterApi, private relationshipApi: RelationshipApi, private formBuilder: FormBuilderService,
            private bookApi: BookApi, private planetApi: PlanetApi, private modal: Modal, private configurationApi: ConfigurationApi) {
  super();
}

no usages  Dinacode
async ngOnInit() : Promise<void>  {
  this.subscribe(this.characterApi.cosmereCharacters(),  next: characters : Character[]  => {
    this.onCharactersChanges(characters);
  });
  this.subscribe(this.relationshipApi.cosmereRelationships(),  next: relationships : Relationship[]  => {
    this.onRelationshipsChanges(relationships);
  });
  this.subscribe(this.configurationApi.configuration(),  next: (configuration : Configuration ) => this.onConfigurationChanges(configuration)

  defer( func: async () : Promise<void>  => {
    await this.characterApi.fetchAllCosmereCharacter();
    await this.relationshipApi.fetchAllCosmereRelationship();
  });
}
```

Figura 5.18: Constructor y *ngOnInit* del componente *CharactersRelationshipsGraphComponent*

La función *ngOnInit* es parte del ciclo de vida de Angular y se ejecuta al iniciar el componente. A veces también se usa *ngOnChanges* ya que se ejecuta también cuando se recarga la web.

5.3 Despliegue de la aplicación

Hemos utilizado el repositorio de Github para mantener los cambios del proyecto y utilizar guardado en la nube. Aprovechando que el proyecto se ubicaba en esta plataforma hemos empleado la sección de *Github pages*(*Websites for you and your projects*, 2023) para desplegar nuestra aplicación. Para hacerlo ha sido necesario configurar Angular para hacer que el *build* de la aplicación se haga en la carpeta *docs*. Una vez tenemos la aplicación compilada, Gitgub da la posibilidad de desplegar una rama que elijas, que en nuestro caso fue master, la rama principal, para desplegar el contenido del directorio *docs*. Así, de forma sencilla, podemos tener la aplicación abierta para todo el público.

5.4 Problemas durante el desarrollo

Durante el desarrollo de este proyecto de software nos encontramos con desafíos, como suele suceder en cualquier proyecto. Uno de los momentos críticos surgió al tener que decidir cómo mostrar la red del Cosmere, el cual es una parte fundamental de la aplicación. Inicialmente, optamos por utilizar Vis.js, una biblioteca de JavaScript que utiliza el elemento Canvas para representar redes gráficas.

La elección de Vis.js se basó en su simplicidad de implementación y en la amplia gama de funcionalidades que ofrecía de forma nativa. Esto nos permitió ahorrar tiempo en el desarrollo inicial. Sin embargo, hubo un aspecto fundamental que no tuvimos en cuenta en ese momento: el rendimiento. El grafo completo del Cosmere consta de alrededor de 2200 arcos y 250 nodos, lo cual no es una red muy grande en términos absolutos. Sin embargo, experimentamos una disminución significativa en el rendimiento de la aplicación al renderizar el grafo utilizando Vis.js.

Además, nos dimos cuenta de que las opciones de personalización que ofrecía Vis.js eran limitadas, lo cual restringía nuestras posibilidades de adaptar visualmente el grafo a nuestras necesidades específicas. Esto nos llevó a replantearnos nuestra elección de biblioteca y finalmente optamos por utilizar D3.js, que utiliza SVG para la renderización de gráficos web.

El cambio a D3.js nos permitió mejorar el rendimiento del grafo, ya que la renderización basada en SVG resultó ser más eficiente en este caso. Además, D3.js ofrece una amplia gama de opciones de personalización, lo cual nos brindó un mayor control sobre la apariencia y la interacción del grafo.

Sin embargo, aunque el rendimiento ha mejorado con el uso de D3.js, somos conscientes de que el tamaño del grafo puede seguir creciendo a medida que se agreguen nuevos libros al Cosmere. Por lo tanto, no descartamos la posibilidad de volver a evaluar nuestras opciones y considerar un cambio a otra biblioteca en el futuro si el tamaño del grafo continúa aumentando y experimentamos limitaciones en el rendimiento.

CAPÍTULO 6
Análisis del Cosmere

En el presente apartado, se llevará a cabo un análisis detallado del Cosmere, un universo literario creado por el renombrado autor Brandon Sanderson. El objetivo principal de este análisis es evaluar diferentes métricas del grafo del Cosmere dentro de una aplicación específica. Estas métricas incluyen el *pagerank*, *eigenvector*, *betweenness* y *closeness*, que se aplicarán a diversos libros importantes, sagas literarias y al Cosmere en su totalidad.

El análisis del grafo del Cosmere proporcionará una comprensión más profunda de las relaciones y Conexiones entre los personajes y los mundos que componen este vasto universo. Además, se examinará la relación entre las métricas de *pagerank*, *eigenvector* y grado de nodo, con el objetivo de determinar el grado de "protagonismo"de cada personaje. Esta medida la llamamos *fractal protagonism*(*Fractal Protagonists*, 2023) evalúa la capacidad de acción de un personaje, el *eigenvector* mide su prestigio y el grado representa el número de interacciones que tiene. Esta definición se basa en el hecho de que un protagonista debe tener un gran impacto en la trama, moverse con frecuencia y sus acciones deben tener consecuencias en quienes lo rodean.

Además de las métricas mencionadas anteriormente, también se calculará el coeficiente de *clustering* y la media de los caminos cortos para evaluar si los mundos del Cosmere se ajustan a la propiedad de los "mundos pequeños". Esto permitirá comprender si los personajes y los mundos del Cosmere están altamente interconectados o si prevalecen estructuras más aisladas.

Este análisis detallado proporcionará una visión panorámica del Cosmere y sus componentes, permitiendo una mejor comprensión de la complejidad y la importancia de cada libro, saga y el conjunto del Cosmere en sí mismo. A través de este estudio, se espera obtener información valiosa sobre la estructura y la dinámica de este fascinante universo literario, así como la identificación de los personajes más prominentes y los mundos más interconectados.

A continuación, se presentarán los resultados y el análisis correspondiente a cada métrica evaluada, con el objetivo de arrojar luz sobre la importancia de estas medidas en el contexto del Cosmere.

6.1 El aliento de los dioses

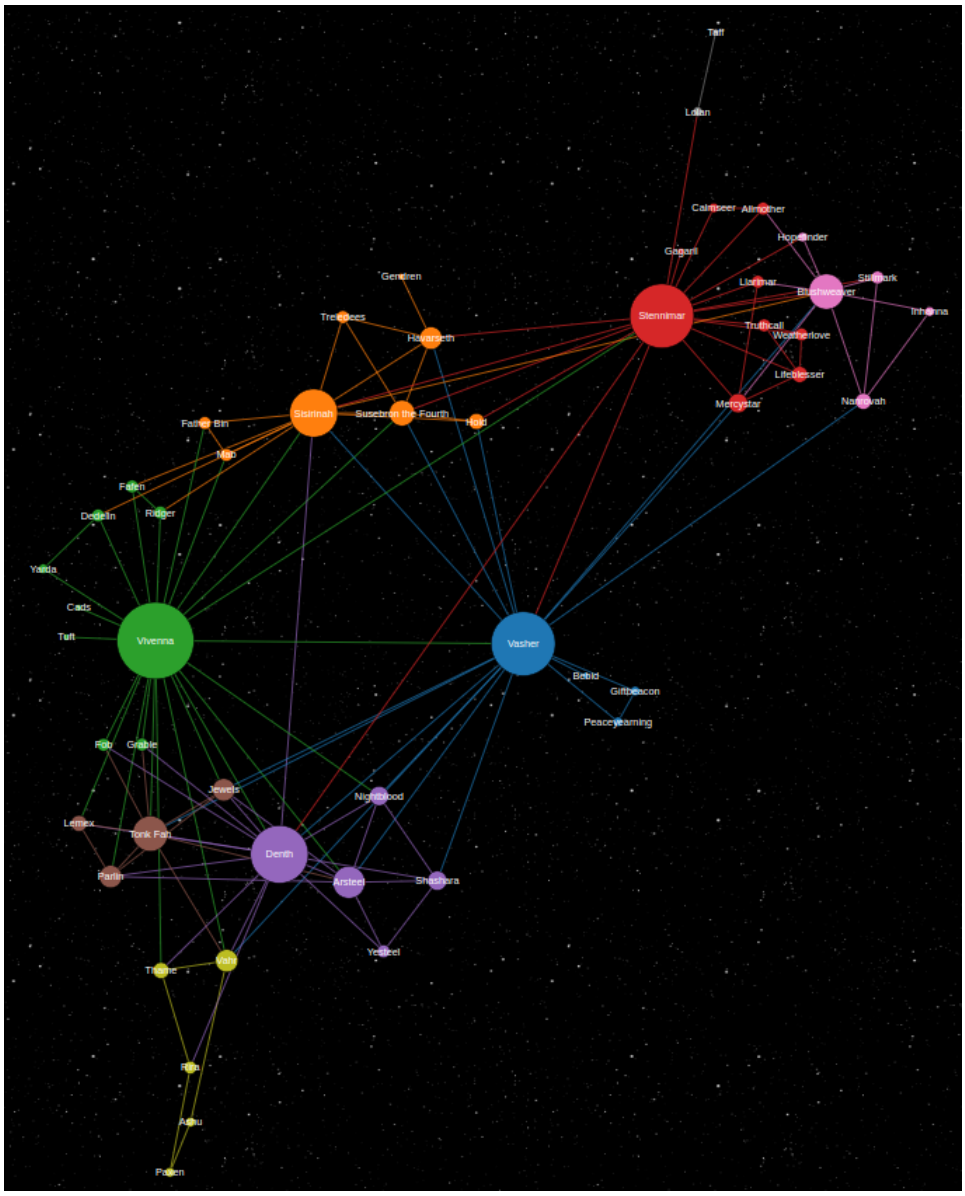

Figura 6.1: Grafo de la red de *El aliento de los dioses*

Al observar el grafo(figura 6.1), se puede identificar fácilmente a cinco personajes principales: Vasher, Deth, Vivenna, Siri y Sondeluz. Esta información se confirma con el *pagerank*(figura 6.2), ya que ocupan las cinco primeras posiciones. Cabe destacar que Vivenna ocupa el primer puesto, mientras que su hermana, Siri, queda relegada a la quinta posición, a pesar de tener varios puntos de vista en la historia.

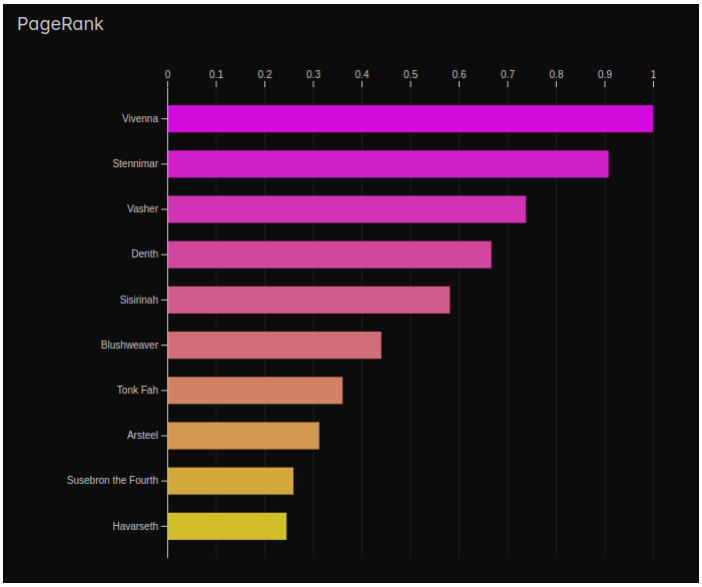

Figura 6.2: Pagerank de *El aliento de los dioses*

El resto de las estadísticas corroboran lo que se intuía, los mismos cinco personajes ocupan siempre las primeras posiciones siendo Vivenna la primera en todas salvo en el *betweenness* donde es Sondeluz quien la ocupa. Esto indica que Sondeluz es el personaje con más influencia en las interacciones del resto de personajes debido a que muchos de los caminos de un nodo a otro pasan por él.

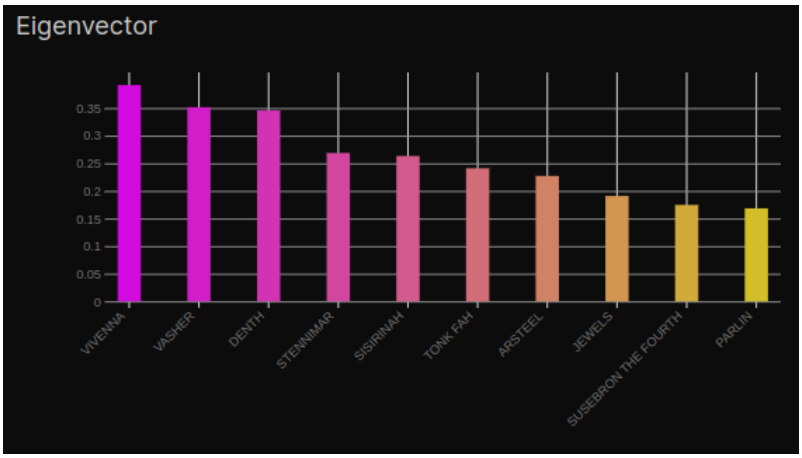

Figura 6.3: Eigenvector de *El aliento de los dioses*

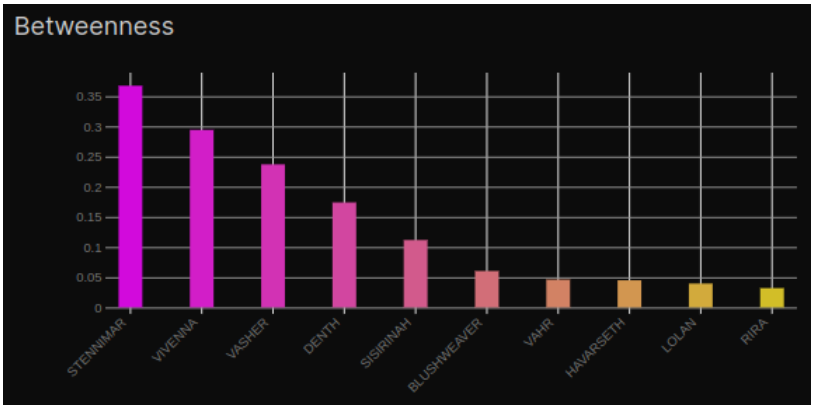

Figura 6.4: Betweenness de *El aliento de los dioses*

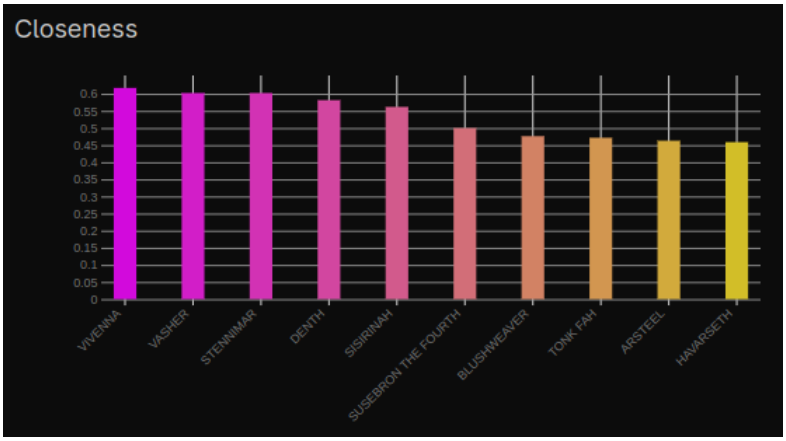

Figura 6.5: Closeness de *El aliento de los dioses*

Con un coeficiente de *clustering* de 0,62 y un coeficiente de *average shortest path* de 4, podemos considerar la red de *El aliento de los dioses* de mundo pequeño. Esta medida indica que el camino medio para llegar de un personaje a cualquier otro es pequeño y que, además, los personajes suelen estar en grupos.

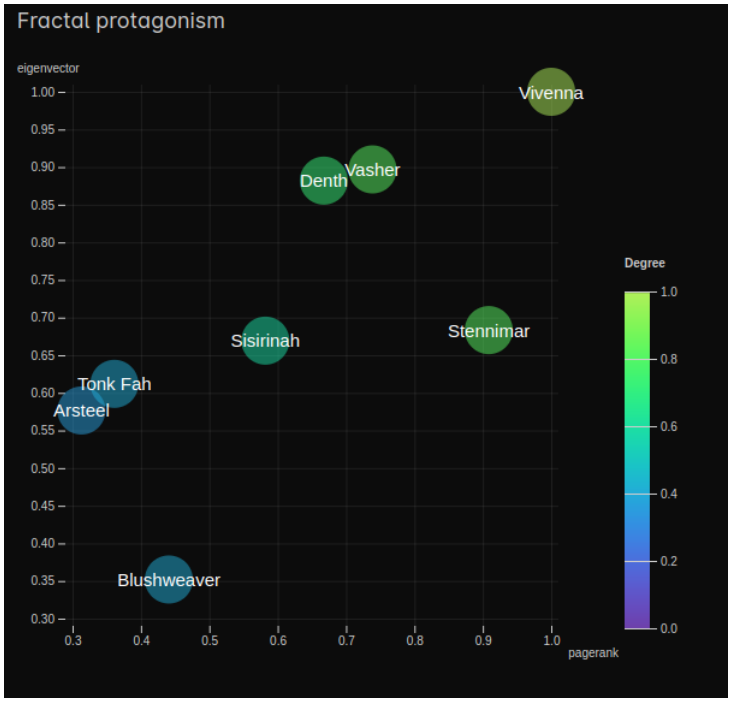

Figura 6.6: Fractal protagonism de *El aliento de los dioses*

En conclusión, se puede observar claramente que Vivenna es la protagonista de *El aliento de los dioses* según el *fractal protagonism*.

6.2 Nacidos de la bruma: Era 1

La primera trilogía de *Nacidos de la bruma* está compuesta por *El imperio final*, *El pozo de la ascensión* y *El Héroe de la eras*. Durante el transcurso de los libros hay personajes que mueren, otros ganan protagonismo y algunos quedan relegados a un segundo plano. Vamos a comprobar si lo que se siente en los libros se refleja en nuestra red. Comenzaremos analizando cada uno de los libros por separado para, finalmente, hacer una lectura de la trilogía en su conjunto.

6.2.1. El imperio final

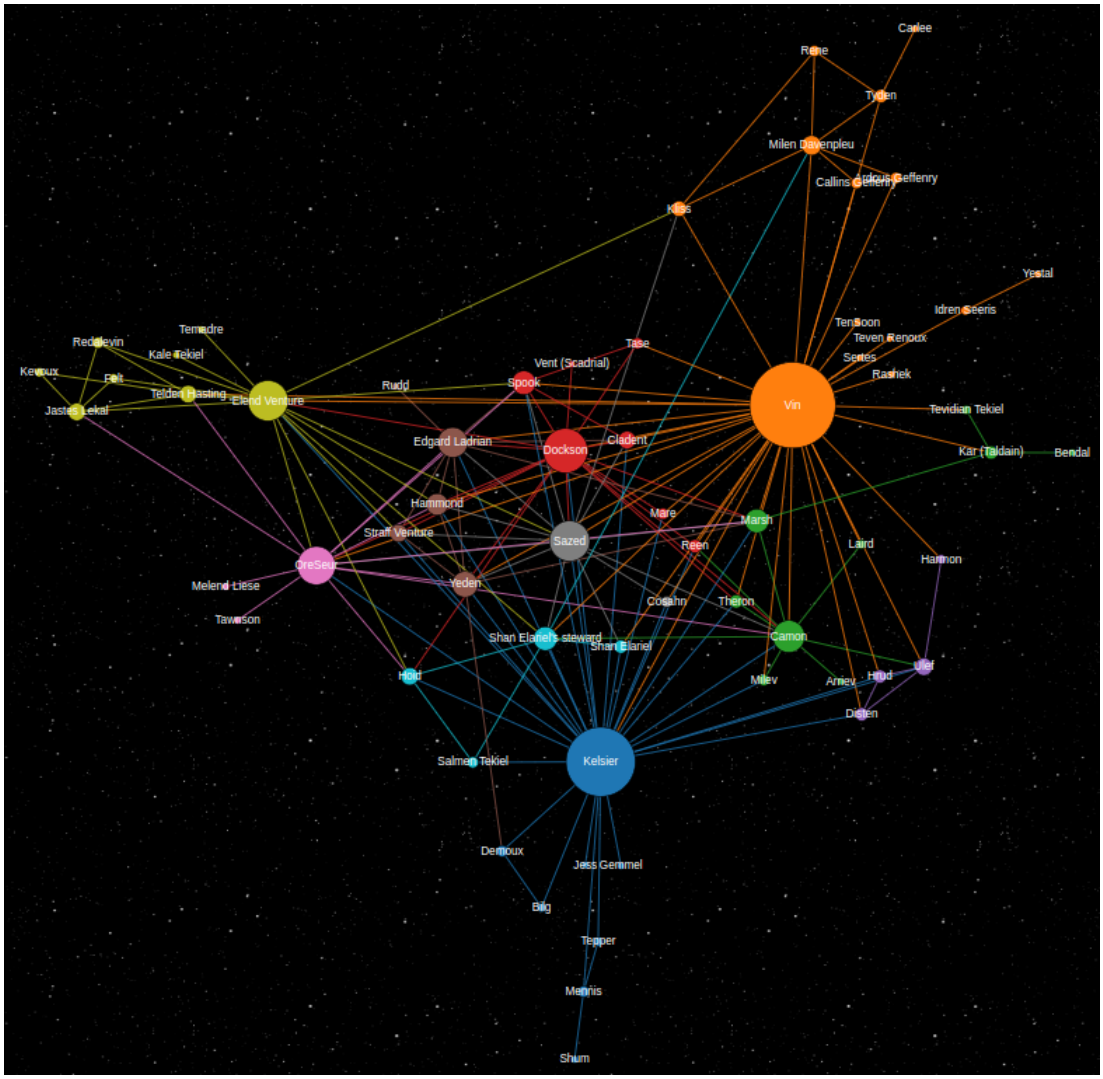

Figura 6.7: Grafo de la red de *El imperio final*

En el grafo(figura 6.7), se puede observar cómo hay dos nodos que destacan por encima del resto. Vin y Kelsier son los personajes con más relaciones en la red. De entre los dos, Vin es la que ocupa los primeros puestos en todas las métricas, lo que indica que Vin es la protagonista indiscutible de este primer libro.

Cabe destacar el personaje de Elend Venture, el noble que más tarde acabará proclamándose como el emperador del imperio ya tiene desde este primer libro una influencia notable en el grafo a pesar de no tener mucho impacto en la historia, salvo por su conexión con Vin.

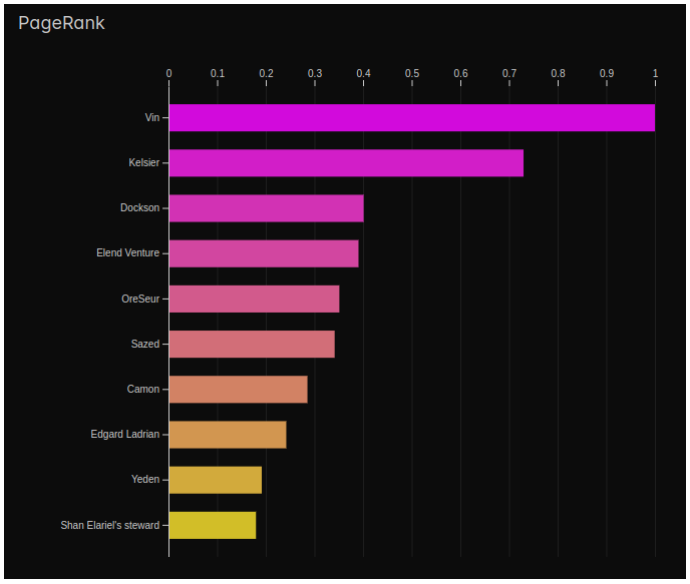

Figura 6.8: *Pagerank* de *El imperio final*

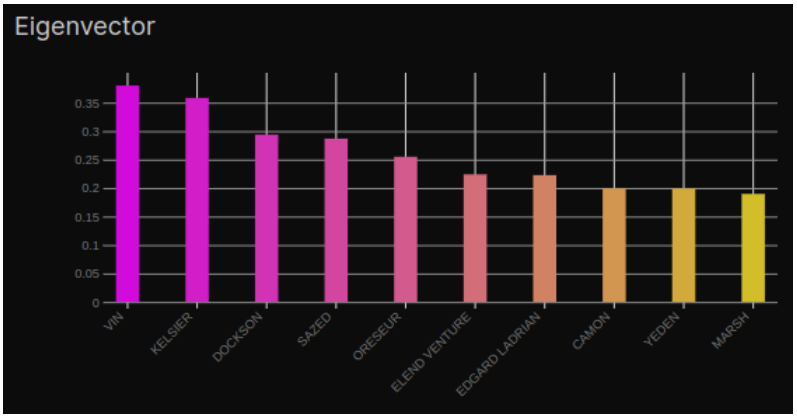

Figura 6.9: *Eigenvector* de *El imperio final*

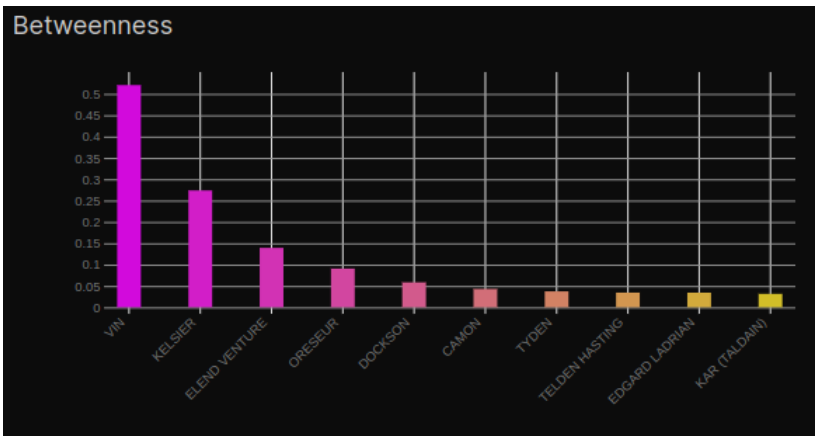

Figura 6.10: *Betweenness* de *El imperio final*

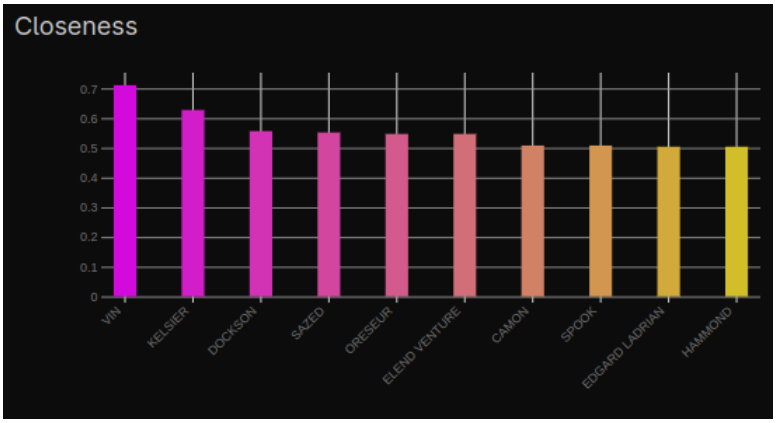

Figura 6.11: *Closeness* de *El imperio final*

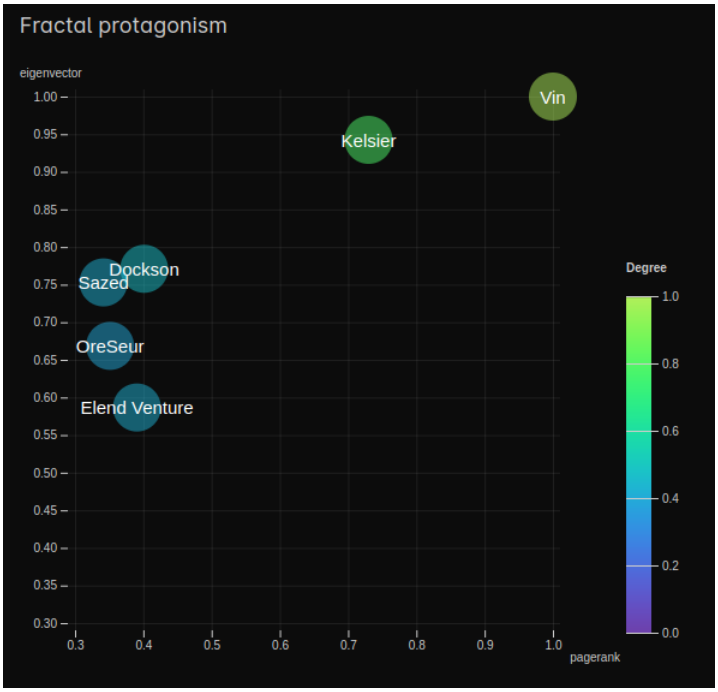

Figura 6.12: *Fractal protagonism* de *El imperio final*

6.2.2. El pozo de la ascensión

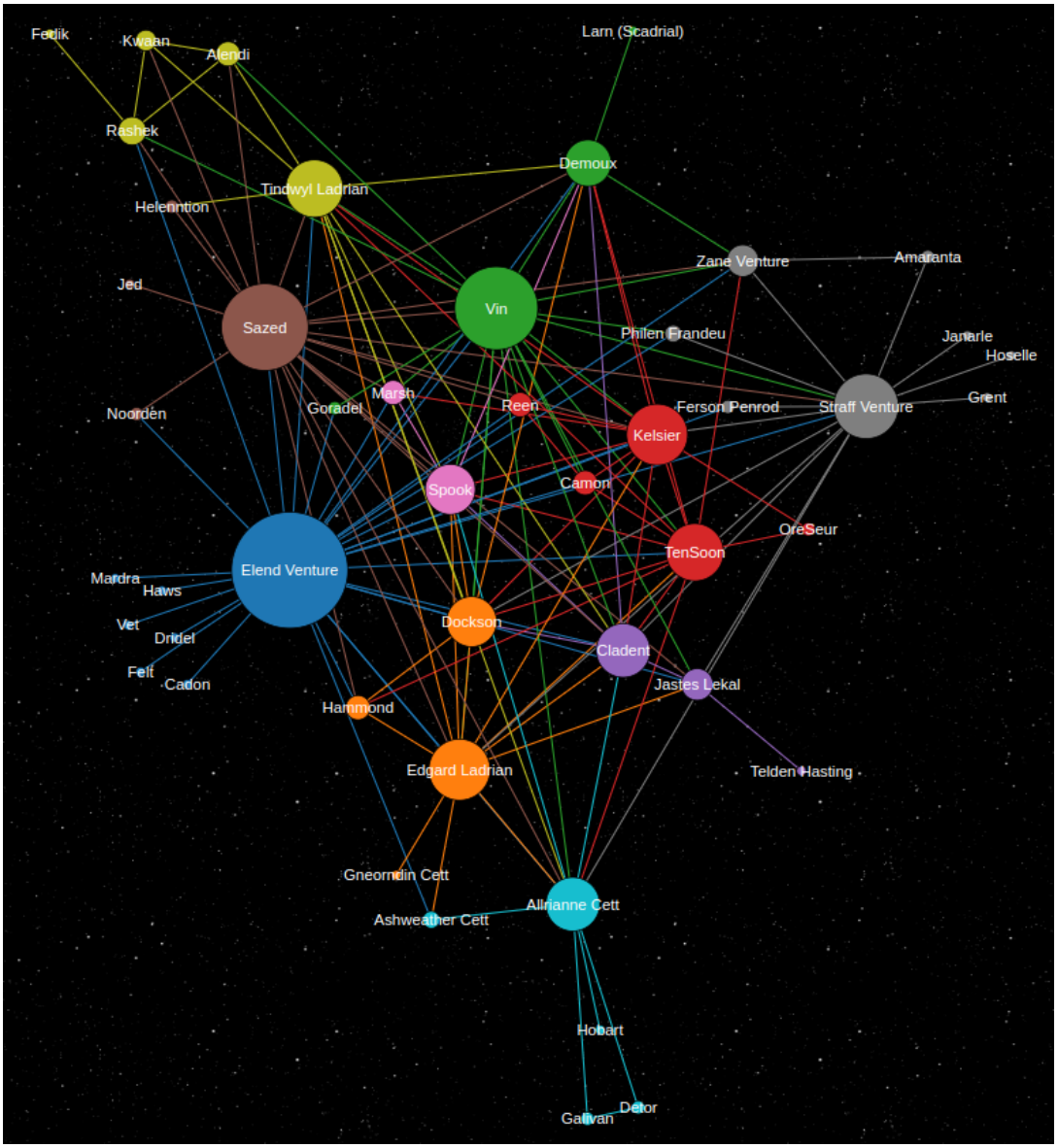

Figura 6.13: Grafo de la red de *El pozo de la ascensión*

En este segundo libro Elend Venture se convierte en el emperador de Luthadel y del libro ya que es el nodo más conectado de la red. No solo eso, sino que ha sustituido a Vin como protagonista ocupando el primer puesto en todas las métricas.

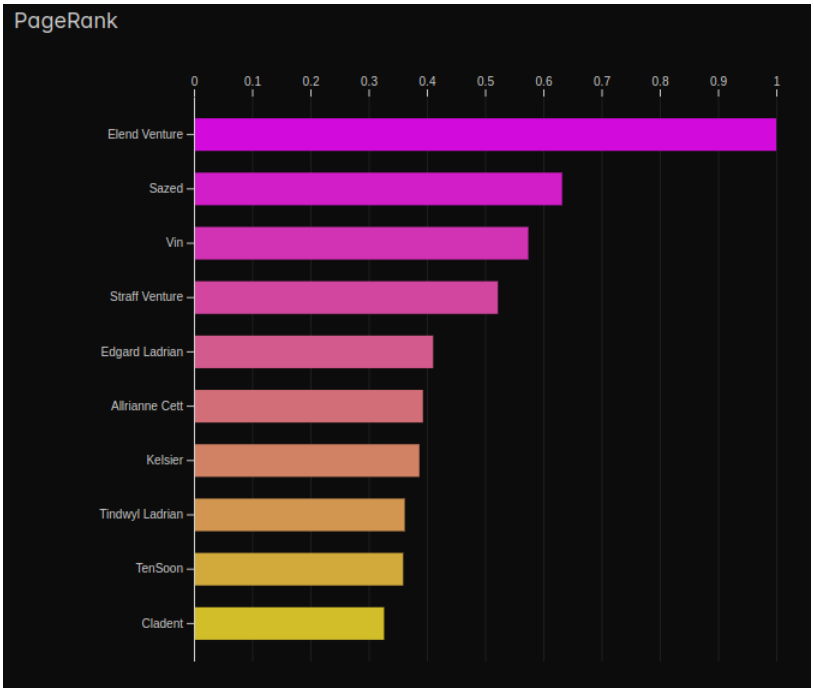

Figura 6.14: *Pagerank* de *El pozo de la ascensión*

Otro aspecto a destacar es Sazed, que ha pasado del sexto puesto en el *pagerank*(figura 6.14) al segundo, por encima de Vin. También tenemos la aparición de Straff Venture, el padre de Elend, que es uno de los antagonistas de este libro. También es importante mencionar el personaje de Kelsier que, aun estando muerto, tiene una influencia notable en este segundo libro. Esto puede ser debido a que su filosofía y creencias que han quedado en las mentes de los protagonistas y la fundación de la iglesia del Superviviente alrededor de su persona.

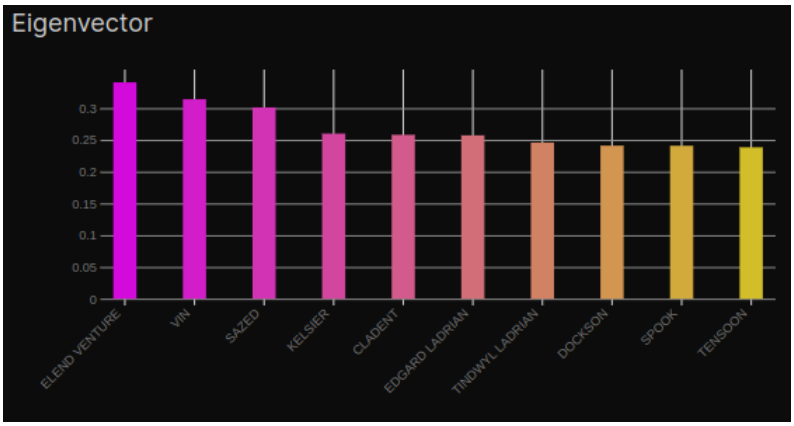

Figura 6.15: *Eigenvector* de *El pozo de la ascensión*

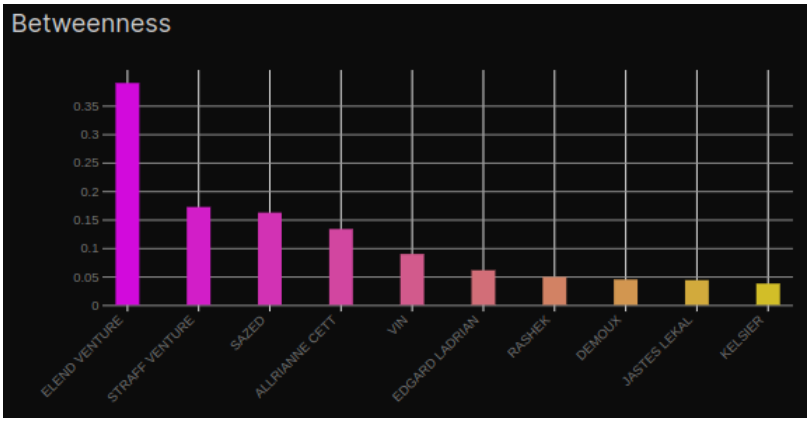

Figura 6.16: *Betweenness* de *El pozo de la ascensión*

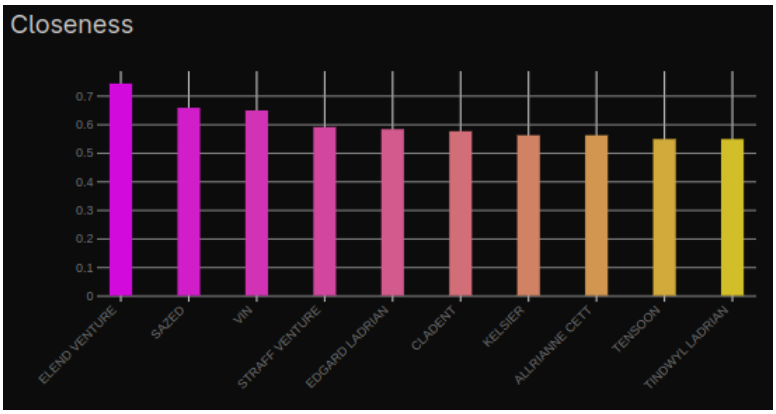

Figura 6.17: *Closeness* de *El pozo de la ascensión*

Es necesario tener en cuenta que los datos no son del todo correctos debido a ciertos problemas como personajes con los mimos nombres o mismos títulos. Esto se agrava aún más con los Kandras, una especie de Scadrial que se dedica a suplantar a otros personajes. El caso más claro ocurre con OreSeur y TenSoon. En cierto punto de la historia que no está del todo claro TenSoon asesina a Oreseur, el kandra de Vin, y pasa a suplantar su identidad por lo que a partir de ese momento toda persona que conozca a OreSeur en realidad está conociendo a TenSoon.

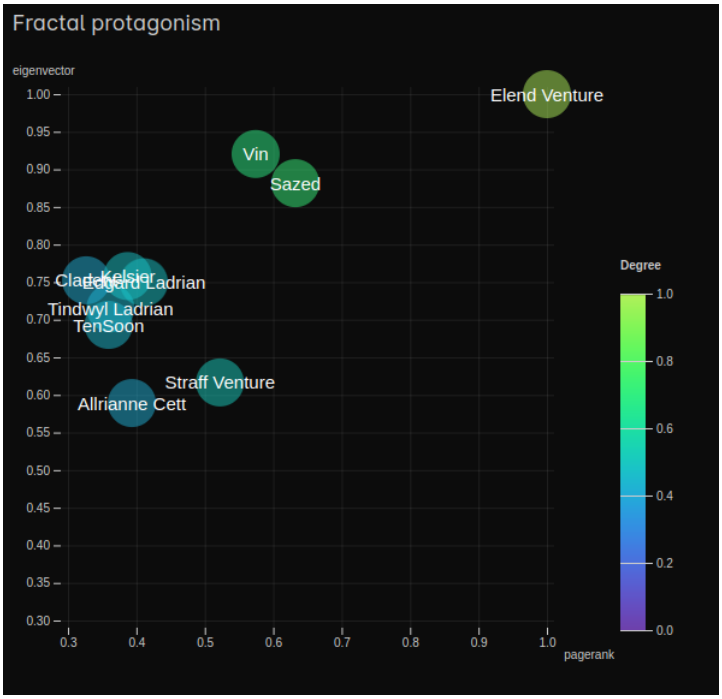

Figura 6.18: *Fractal protagonism* de *El pozo de la ascensión*

6.2.3. El héroe de las eras

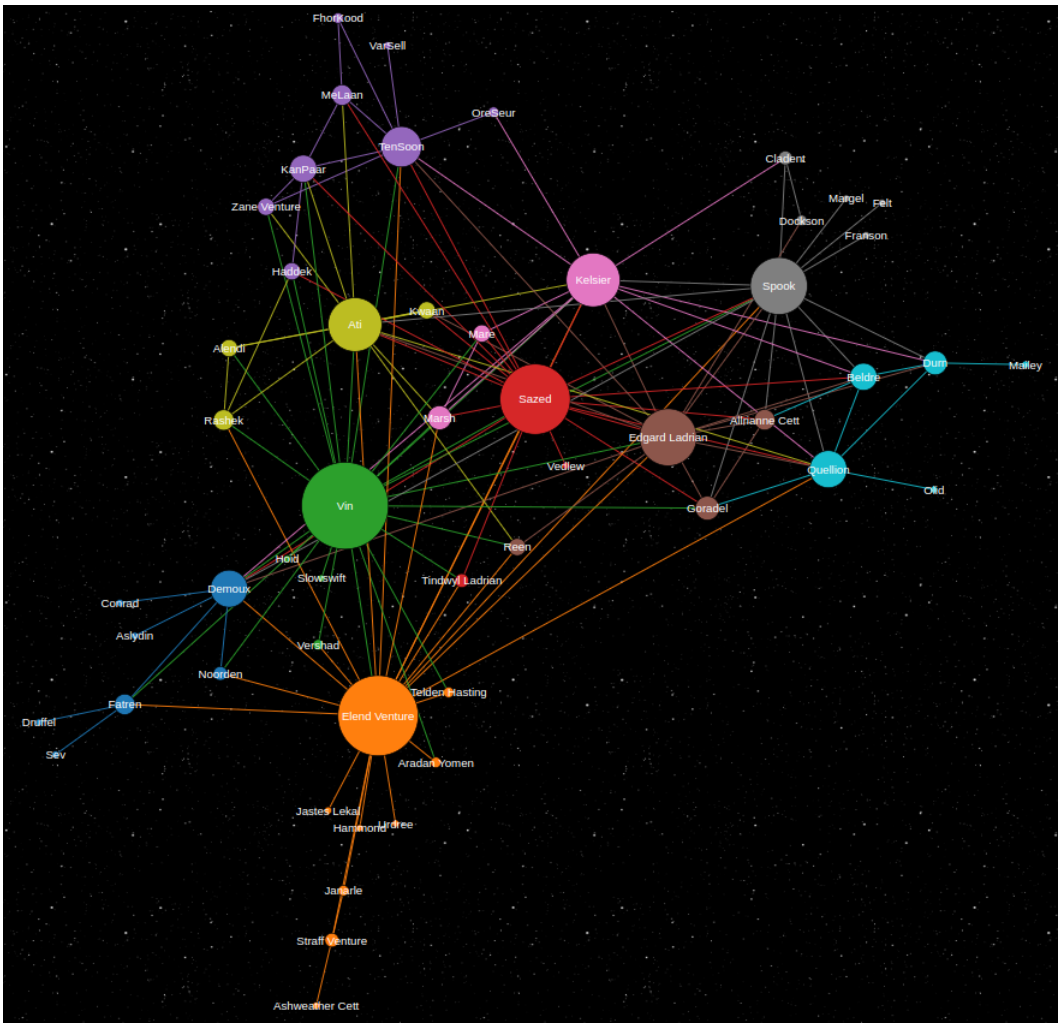

Figura 6.19: Grafo de la red de *El héroe de las eras*

En el primer libro, Vin asumió el papel protagónico, mientras que, en el segundo libro, dicho papel fue otorgado a Elend. Sin embargo, en este tercer libro, ambos personajes han obtenido prácticamente el mismo nivel de *pagerank*(figura 6.20).

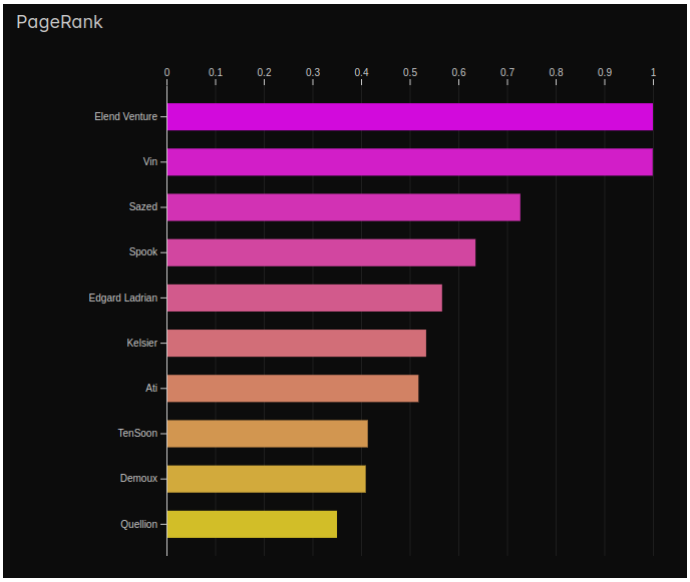

Figura 6.20: *Pagerank* de *El héroe de las eras*

Además, es interesante notar que en la métrica de *fractal protagonism*(figura 6.21), también presentan valores bastante similares, aunque Vin sobresale ligeramente por encima de Elend. En cuanto al cuarto lugar del *pagerank*, este es ocupado por Spook, un personaje previamente considerado secundario, pero que ahora ha adquirido un protagonismo significativo al ser reconocido como el superviviente de las llamas.

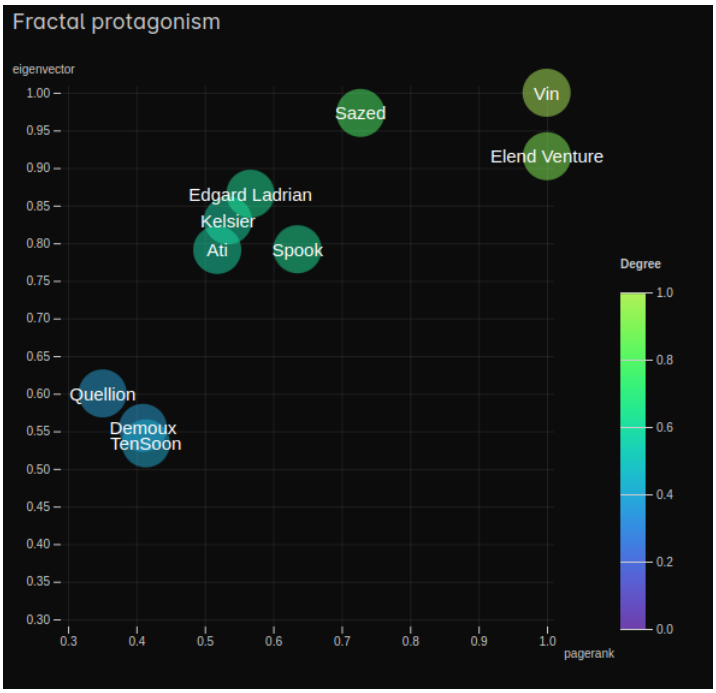

Figura 6.21: *Fractal protagonism* de *El héroe de las eras*

También merece mención la inclusión de la comunidad de kandras en el grafo de personajes, una presencia novedosa que no había sido introducida en entregas anteriores. Es importante comentar que en este libro ocurre algo similar al caso de OreSeur y TenSoon. Sucede que Ati, Ruina, se hace pasar por Kelsier cuando ha Fantasma le clavan un punzón hemalúrgico. Por lo que, aunque no se dice en ninguna parte de forma explícita, se ha considerado que Fantasma a conocido a Ati.

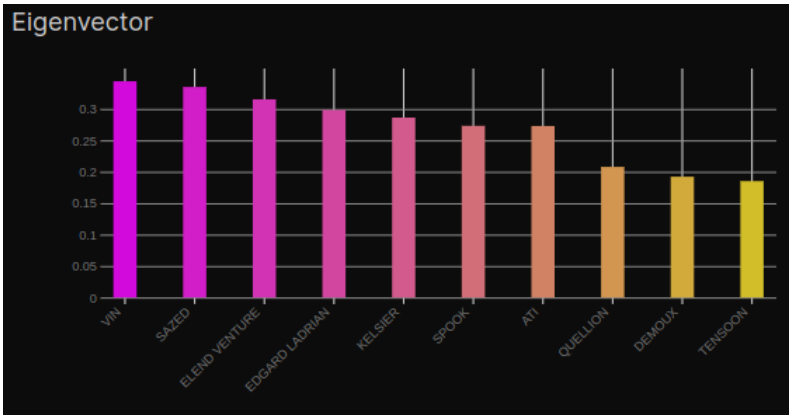

Figura 6.22: *Eigenvector* de *El héroe de las eras*

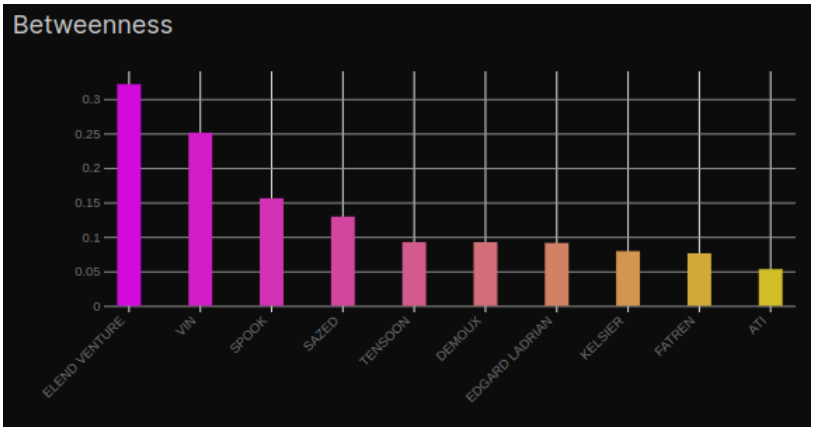

Figura 6.23: *Betweenness* de *El héroe de las eras*

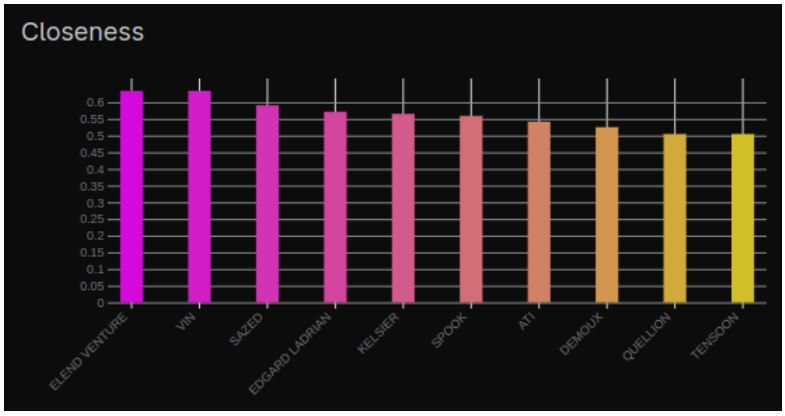

Figura 6.24: *Closeness* de *El héroe de las eras*

6.2.4. Nacidos de la bruma: Era 1

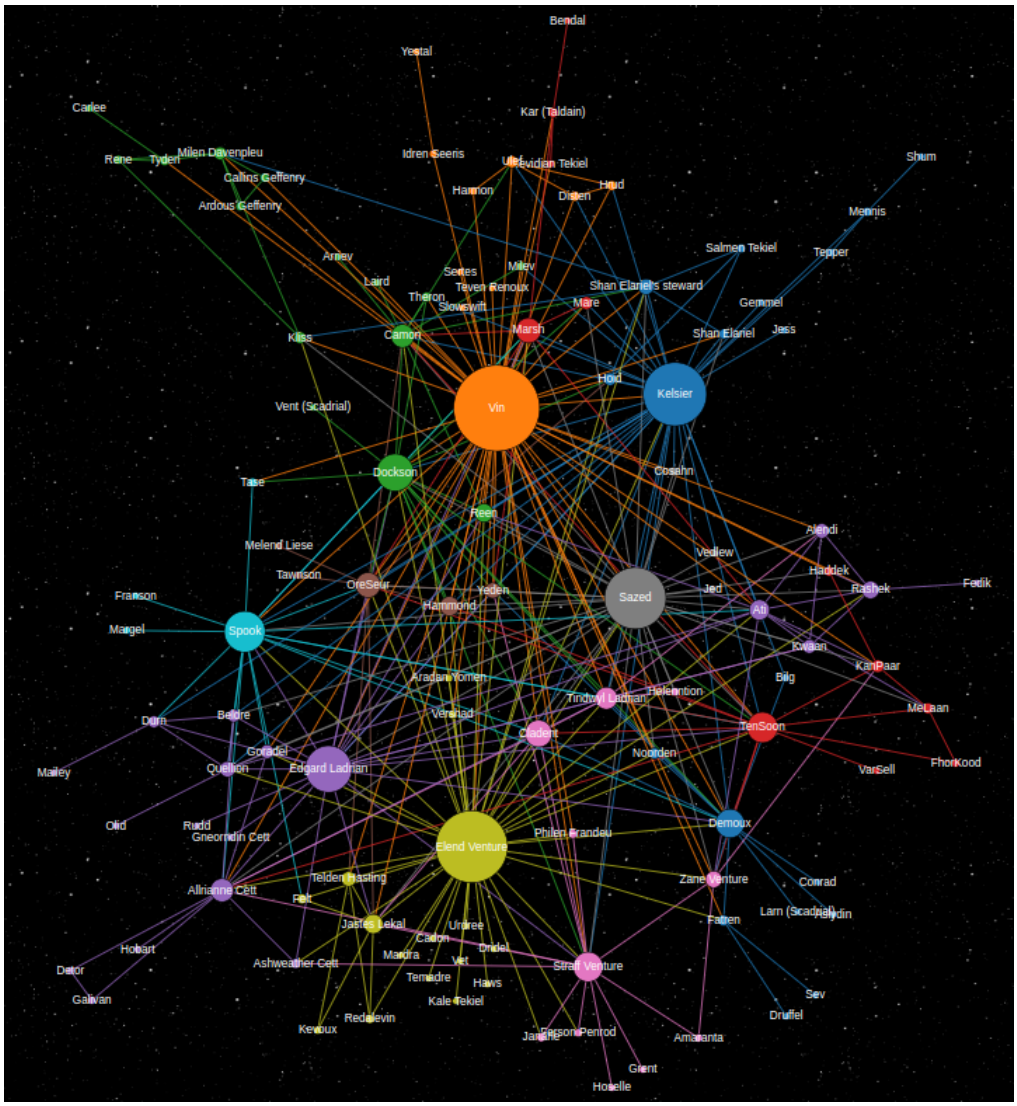

Figura 6.25: Grafo de la red de *Nacidos de la bruma: Era 1*

Una vez realizado el análisis de los libros por separado, procederemos a hacerlo con los tres libros en conjunto para determinar quién es el protagonista de la primera trilogía de *Nacidos de la bruma*.

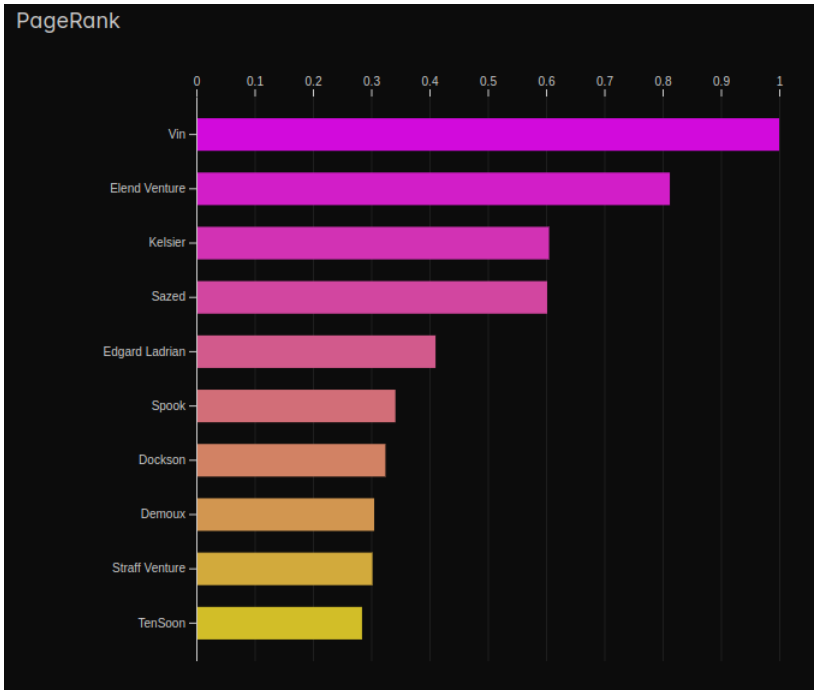

Figura 6.26: *Pagerank* de *Nacidos de la bruma: Era 1*

Observando el grafo(figura 6.25), destacan cuatro nodos por encima del resto: Vin, Elend Venture, Kelsier y Sazed. Aunque Elend ha ido adquiriendo protagonismo a lo largo de los libros, Vin ocupa el primer lugar en todas las métricas, lo que nos lleva a concluir que Vin es la protagonista de la primera era de *Nacidos de la bruma*.

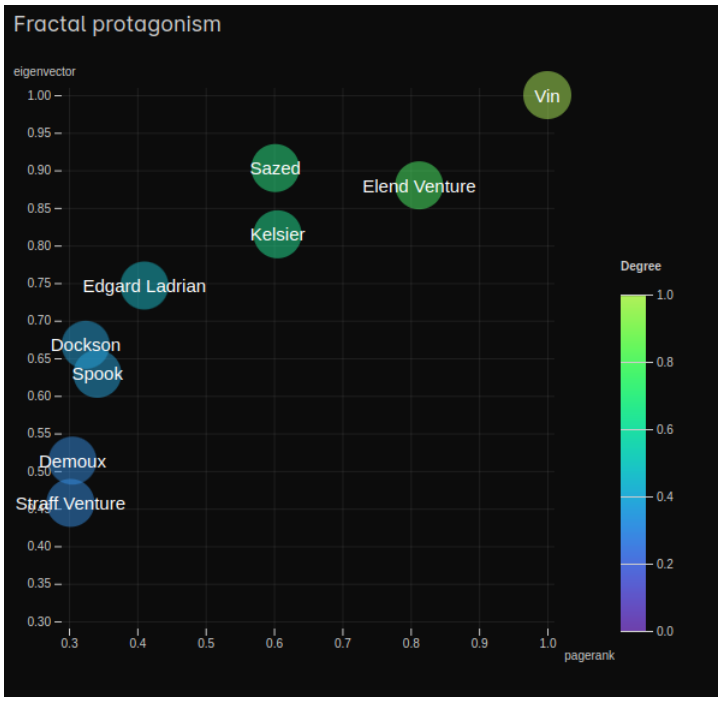

Figura 6.27: *Fractal protagonism* de *Nacidos de la bruma: Era 1*

A continuación, realizaremos el análisis de mundo pequeño. La red de los tres primeros libros de *Nacidos de la bruma* tiene un coeficiente de *clustering* de 0,5 y un coeficiente de *average shortest path* de 3,87. Dado que presenta un alto coeficiente de agrupamiento y una

longitud media de camino inferior a 6, se puede considerar que cumple con la definición de mundo pequeño.

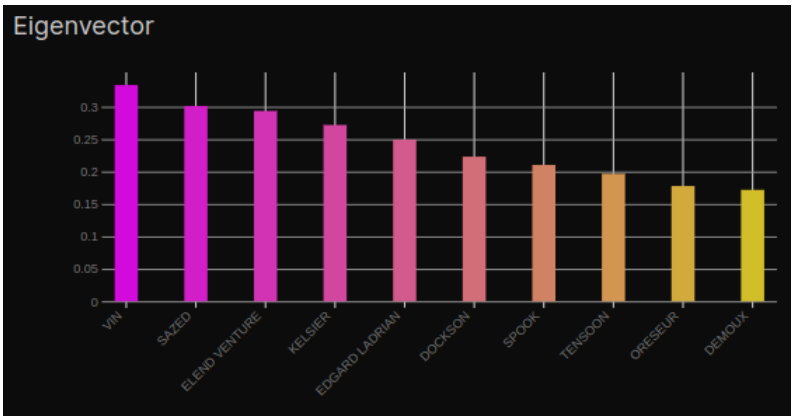

Figura 6.28: *Eigenvector* de *Nacidos de la bruma: Era 1*

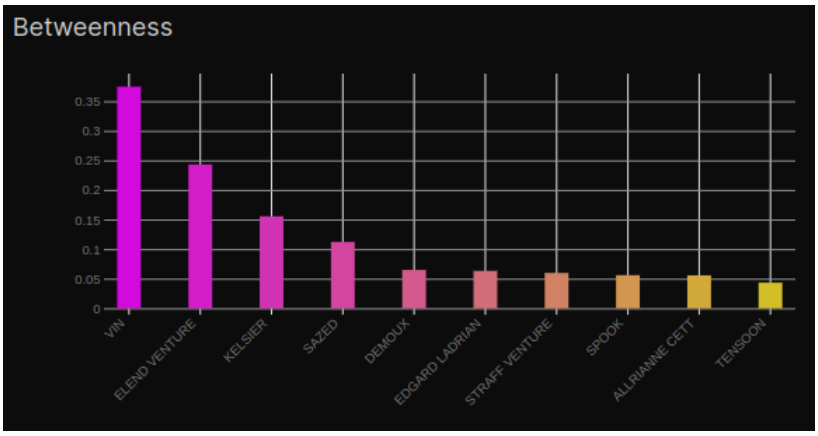

Figura 6.29: *Betweenness* de *Nacidos de la bruma: Era 1*

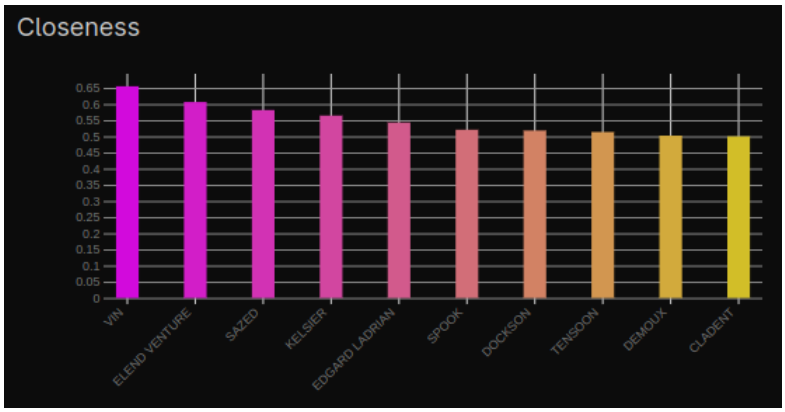

Figura 6.30: *Closeness* de *Nacidos de la bruma: Era 1*

6.3 Nacidos de la bruma: Era 2

Transcurridos tres siglos desde el renacimiento del mundo y con Sazed convertido en el portador de las esquirlas de Ruina y Conservación, nos encontramos ante la introducción de numerosos nuevos personajes. Sin embargo, también tendremos el placer de

reencontrarnos con algunos de nuestros antiguos amigos. La segunda era se compone de cuatro libros: *Aleación de ley*, *Sombras de identidad*, *Brazales de duelo* y *El metal perdido* aunque nuestro análisis será solo de los tres primeros.

6.3.1. Aleación de ley

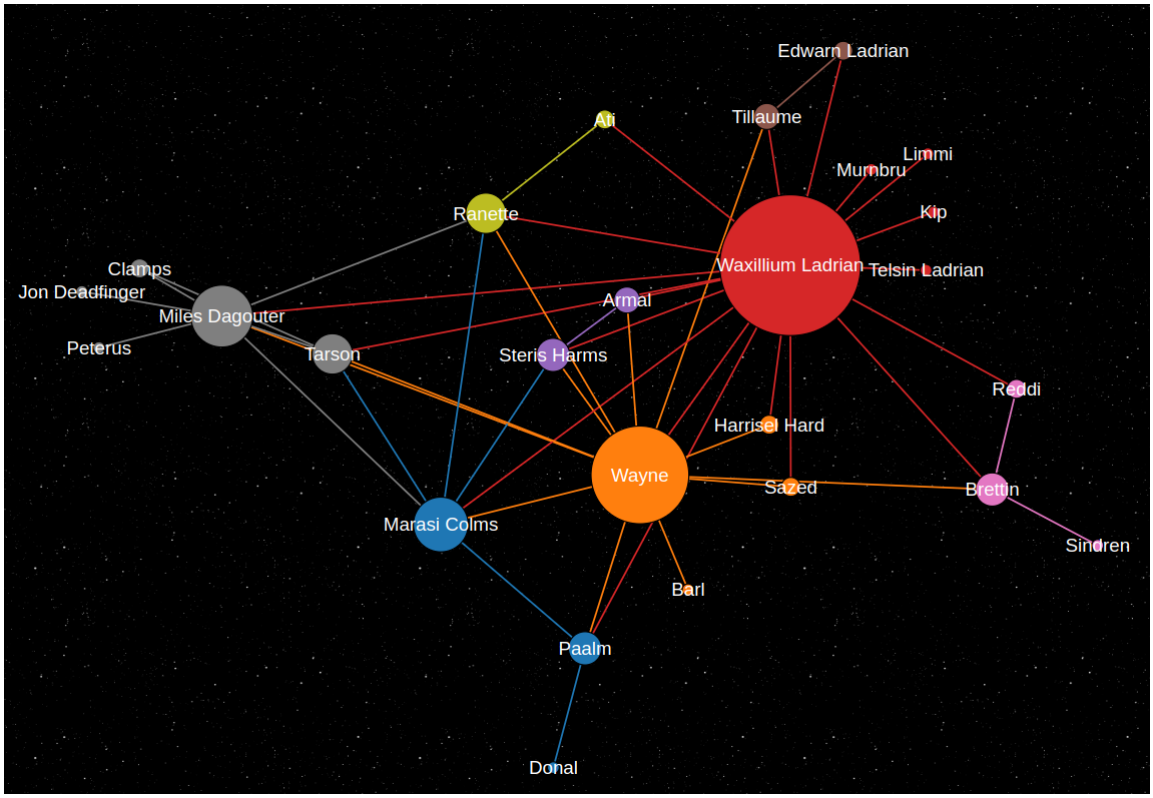

Figura 6.31: Grafo de la red de *Aleación de ley*

Inicialmente, este libro estaba destinado a ser un volumen único que se ubicaría entre la primera y la segunda era de *Nacidos de la bruma*. Sin embargo, debido a su gran acogida entre los lectores, se convirtió en el primer libro de la segunda era. Es posible que esta circunstancia, junto con su longitud relativamente corta en comparación con otras obras de Brandon Sanderson, explique por qué es el libro con menos personajes hasta el momento.

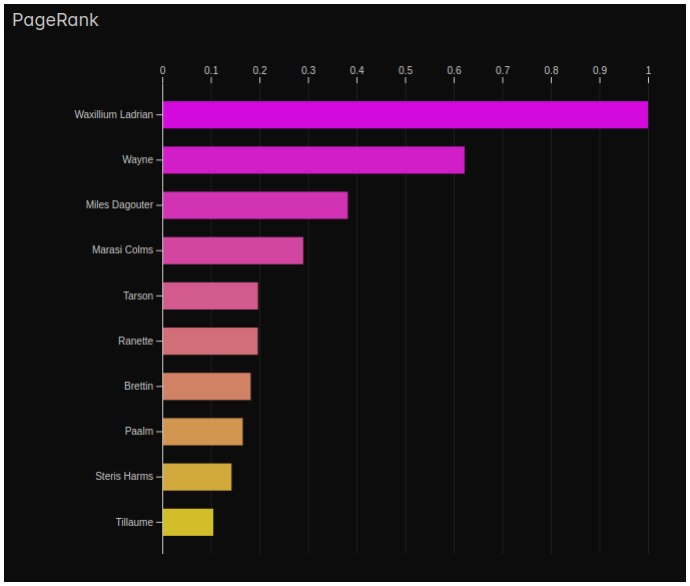

Figura 6.32: *Pagerank* de *Aleación de ley*

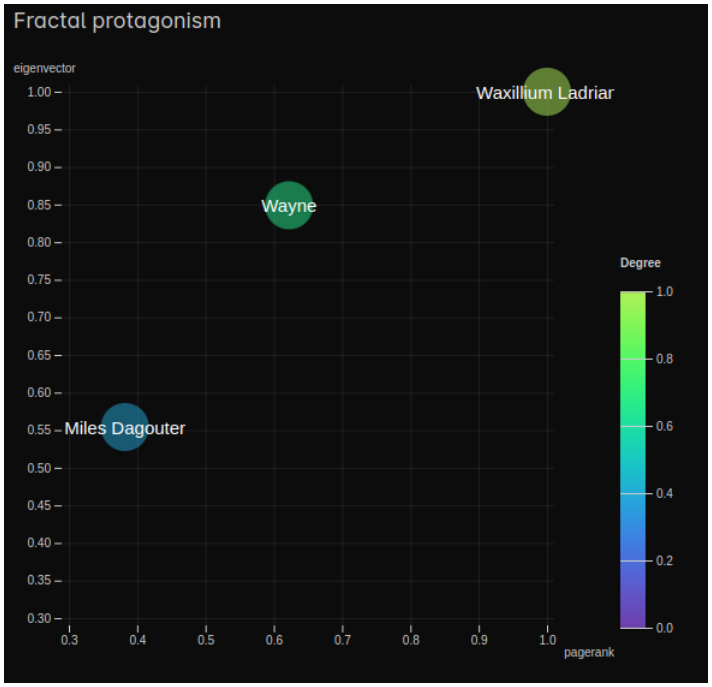

Figura 6.33: *Fractal protagonism* de *Aleación de ley*

Tanto como lector como al observar el grafo(figura 6.31), resulta evidente que Waxilium es el protagonista de este libro. Su compañero, Wayne, ocupa el segundo lugar en la lista, mientras que Miles Dagouter, el antagonista, se sitúa en el tercer puesto. De hecho, esto es tan notable que en el gráfico de *fractal protagonism*(figura 6.33) solo aparecen estos tres personajes.

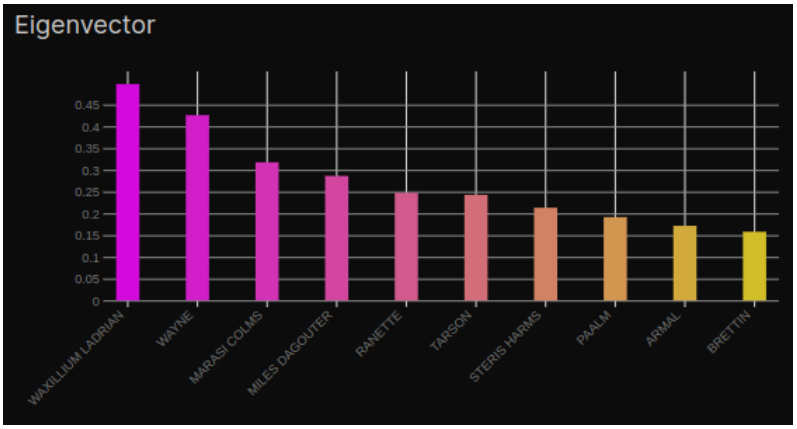

Figura 6.34: *Eigenvector* de *Aleación de ley*

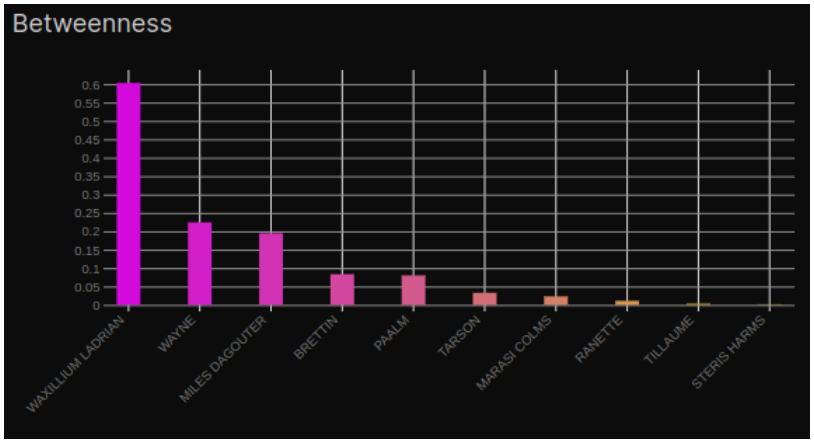

Figura 6.35: *Betweenness* de *Aleación de ley*

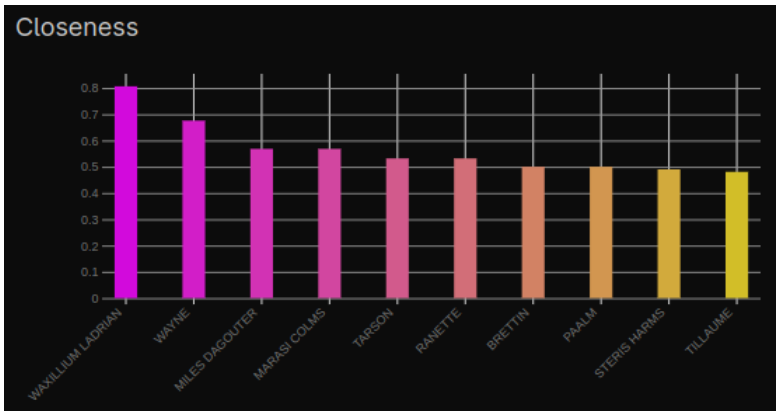

Figura 6.36: *Closeness* de *Aleación de ley*

6.3.2. Sombras de identidad

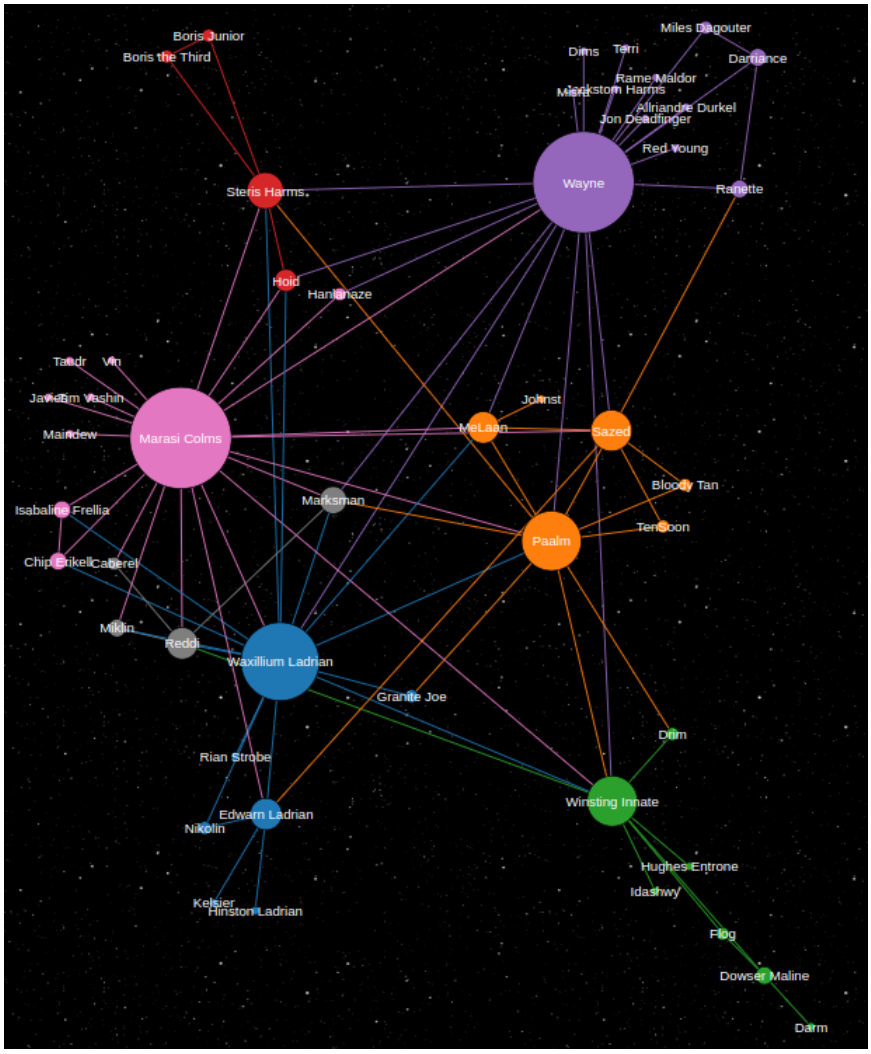

Figura 6.37: Grafo de la red de *Sombras de identidad*

En este segundo libro Waxilium Ladrian pasa a un segundo plano mientras que Marasi y Wayne rellenan el hueco que Wax ha dejado. Los dos personajes se intercambian las posiciones entre la primera y la segunda en las diferentes métricas. Tanto es así que para la medida del *pagerank*(figura 6.38) es Wayne el ganador pero para el *eigenvector*(figura 6.39) lo es Marasi. Si a esto le sumamos que tienen un número de relaciones similar nos encontramos en el primer libro donde tenemos no a un solo protagonista sino a una pareja de protagonistas.

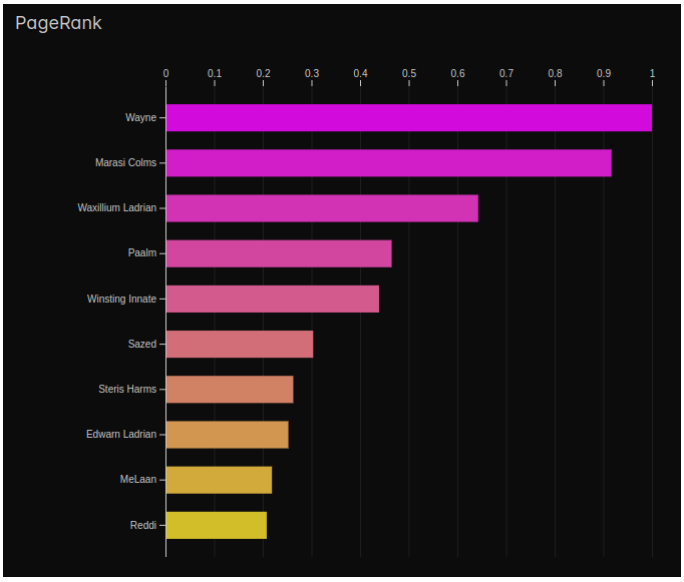

Figura 6.38: *Pagerank* de *Sombras de identidad*

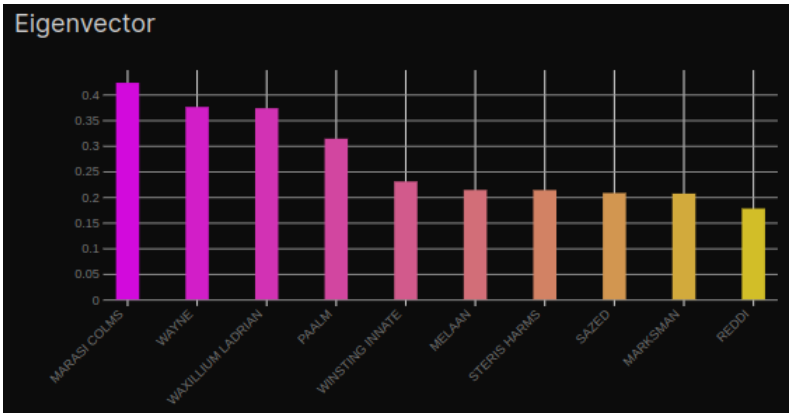

Figura 6.39: *Eigenvector* de *Sombras de identidad*

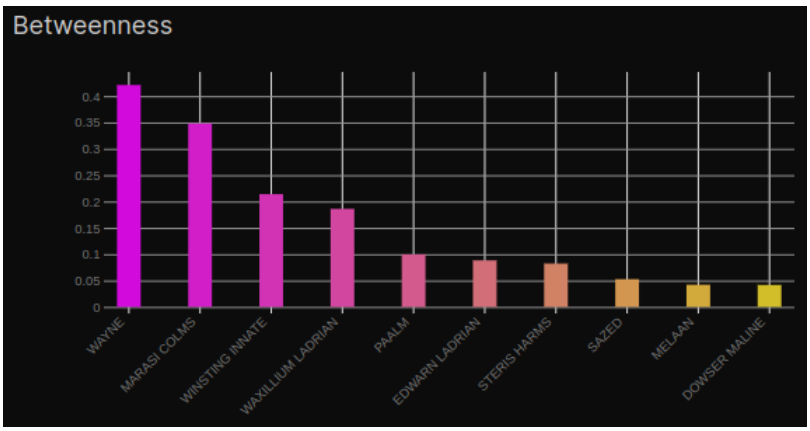

Figura 6.40: *Betweenness* de *Sombras de identidad*

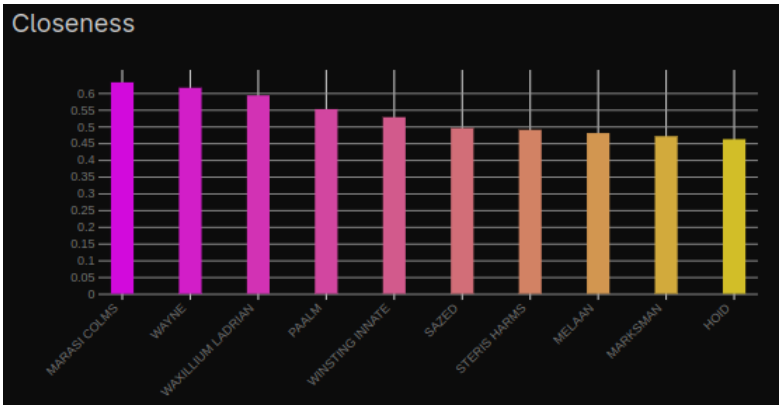

Figura 6.41: *Closeness* de *Sombras de identidad*

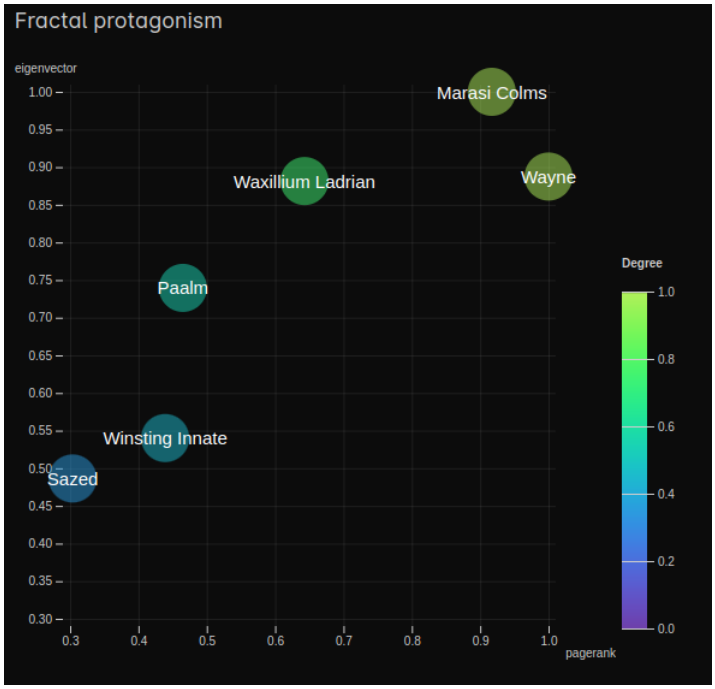

Figura 6.42: *Fractal protagonism* de *Sombras de identidad*

6.3.3. Brazales de duelo

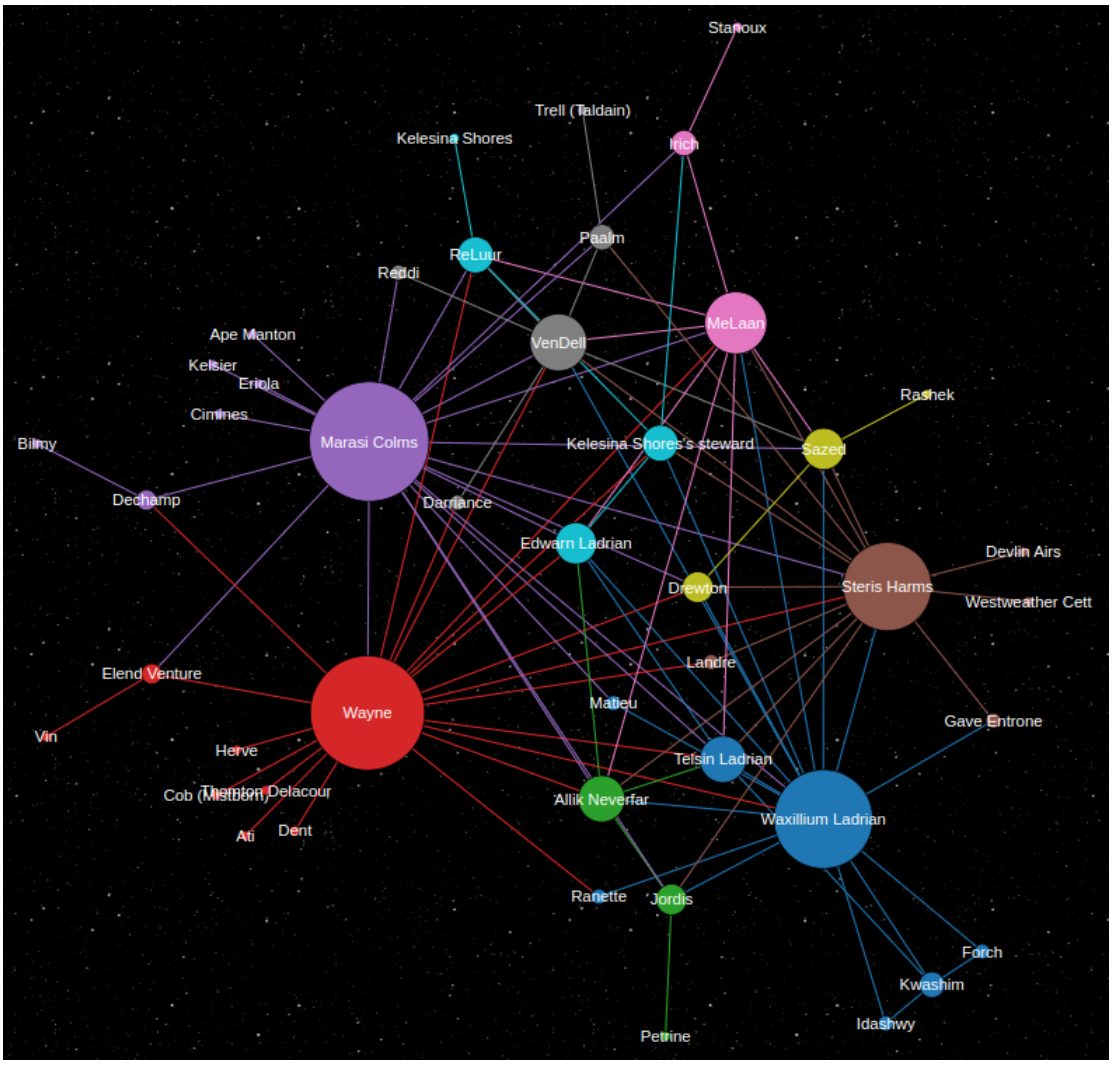

Figura 6.43: Grafo de la red de *Brazales de duelo*

En el tercer libro sí que es Marasi el personaje con mayor *pagerank* y *eigenvector*(figura 6.48) lo que la convierte en la protagonista de esta tercera historia. Destaca el personaje de Steris Harms, hermanastra de Marasi y mujer de Waxilium, que ha tenido que esperar hasta esta tercera entrega para ocupar un lugar relevante en la historia. Se puede ver un paralelismo entre Steris y Fantasma, los dos han ganado protagonismo en el tercer libro de sus respectivas sagas, pasando de ser meros personajes secundarios a figuras con un peso notorio en la trama.

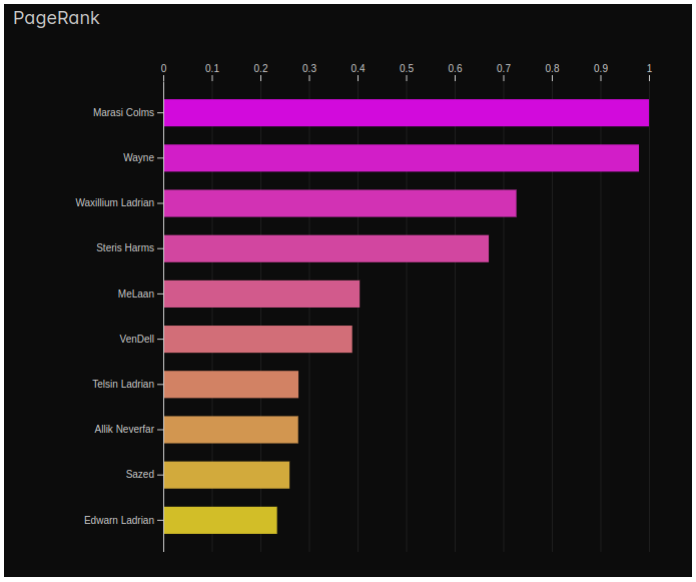

Figura 6.44: *Pagerank* de *Brazales de duelo*

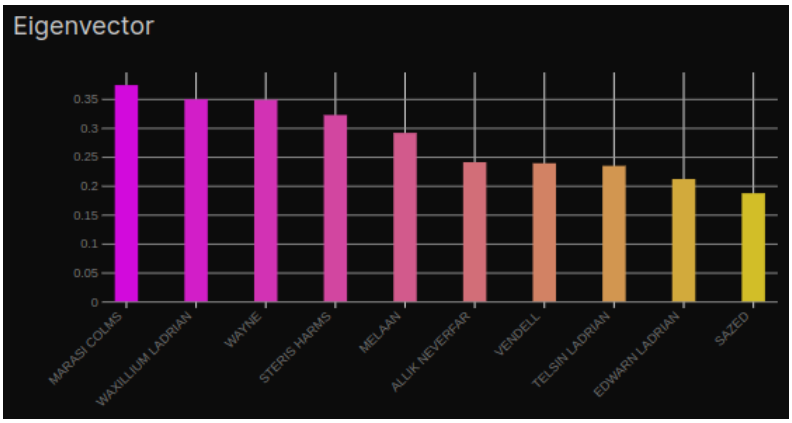

Figura 6.45: *Eigenvector* de *Brazales de duelo*

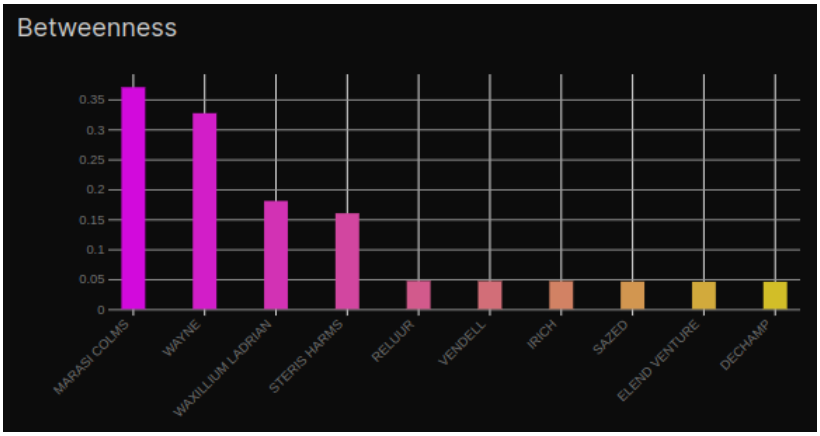

Figura 6.46: *Betweenness* de *Brazales de duelo*

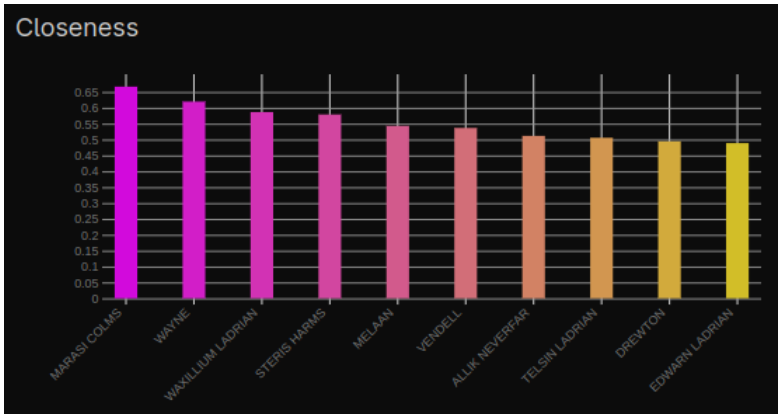

Figura 6.47: *Closeness* de *Brazales de duelo*

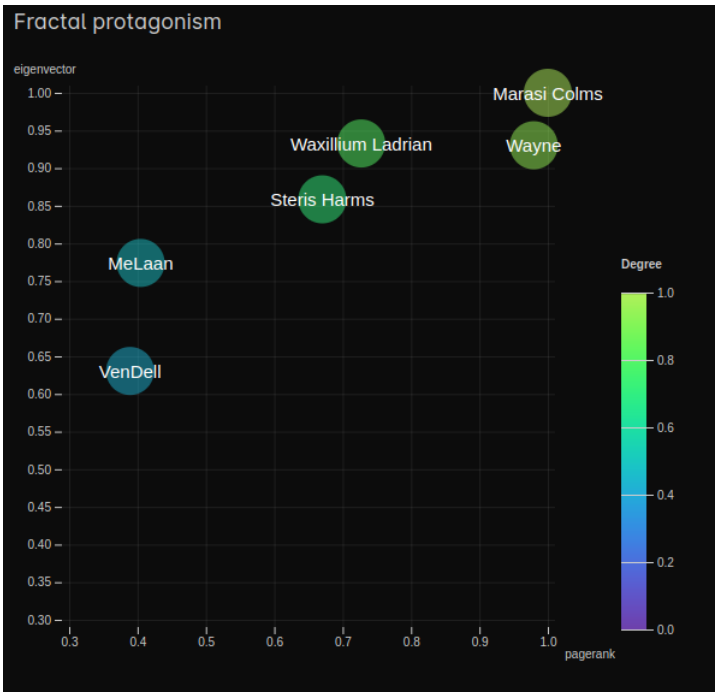

Figura 6.48: *Fractal protagonism* de *Brazales de duelo*

6.3.4. Era 2

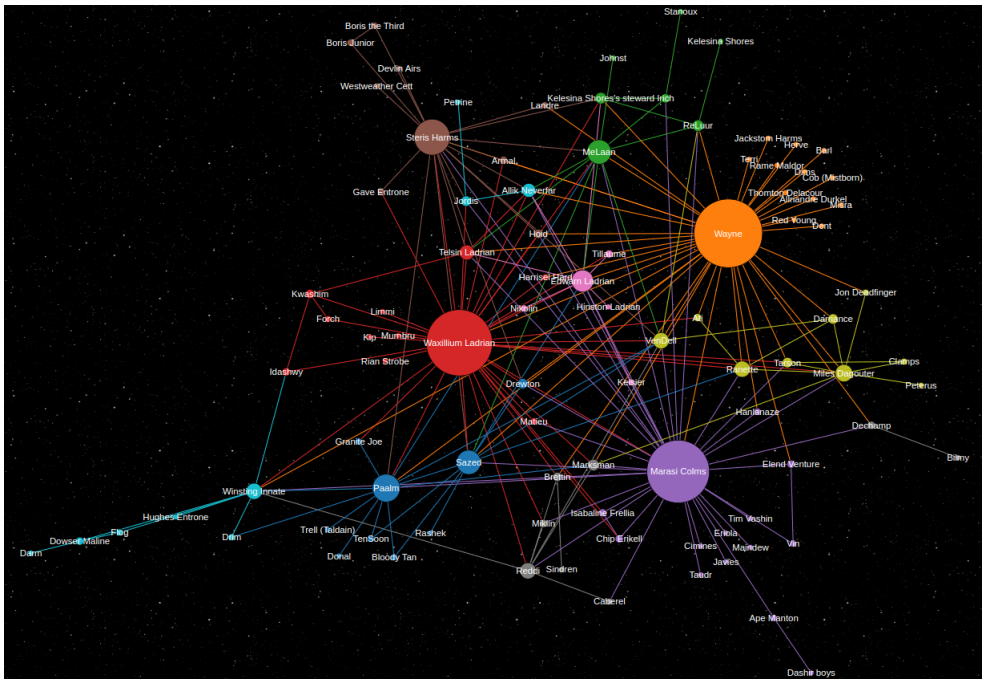

Figura 6.49: Grafo de la red de *Nacidos de la bruma: Era 2*

A falta de analizar el *El metal perdido*, el cuarto y último libro, podemos observar(figura 6.51) que los tres protagonistas son: Waxilium Ladrian, Wayne y Marasi Colms. Es curioso que Marasi, un personaje que apenas aparecia en el primer libro y que nunca ha sido la protagonista clara esté en una posición tan alta. Wayne es el personaje con mejor *fractal protagonism*(figura 6.51), siendo el personaje con mayor *pagerank*(figura 6.50) y casi con el mayor *eigenvector*(figura 6.52) solo un poco por debajo de Wax. No solo eso sino que además es el personaje con mayor *betweenness*(figura 6.53) y *closseness*(figura 6.54), por lo tanto, hemos concluido que Wayne es el protagonista de la segundo era de *Nacidos de la bruma*.

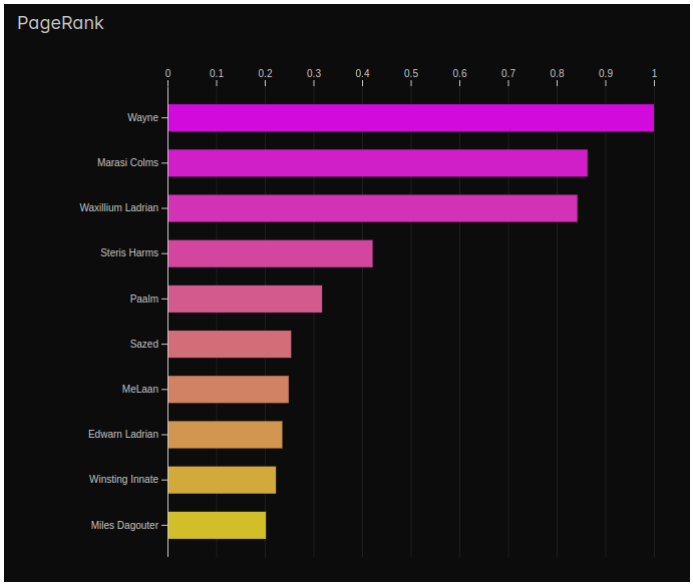

Figura 6.50: *Pagerank* de *Nacidos de la bruma: Era 2*

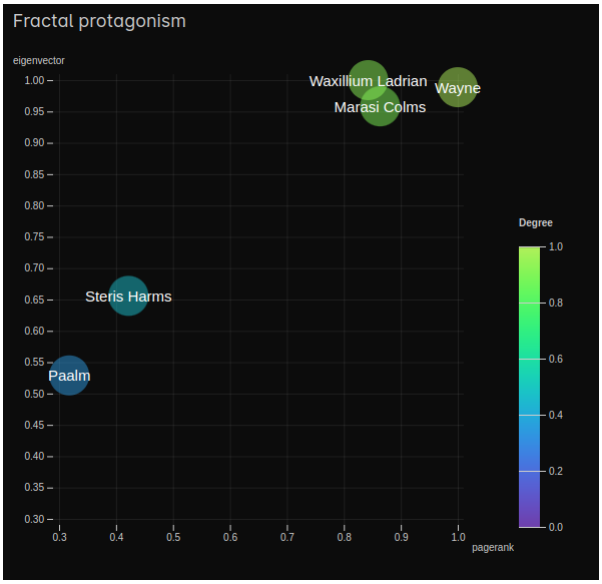

Figura 6.51: *Fractal protagonism* de *Nacidos de la bruma: Era 2*

Respecto a las métricas generales del grafo, la segunda era de *Nacidos de la bruma* obtiene un coeficiente de *clustering* 0,42 y un 4,10 en el coeficiente de *average shortest path* por lo que se puede considerar una red de mundo pequeño.

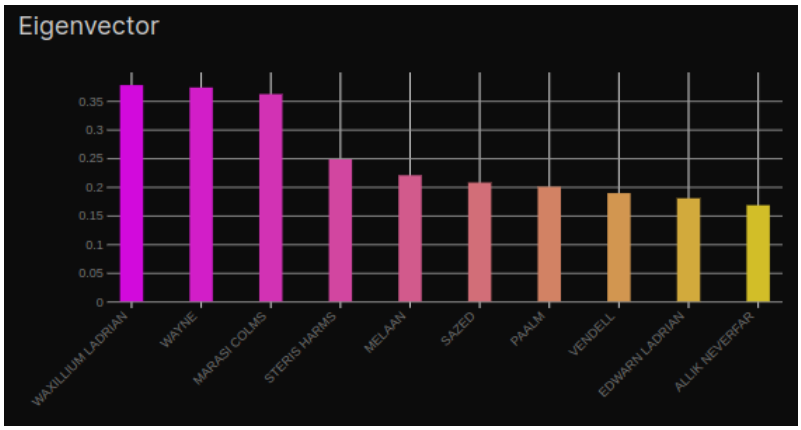

Figura 6.52: *Eigenvector* de *Nacidos de la bruma: Era 2*

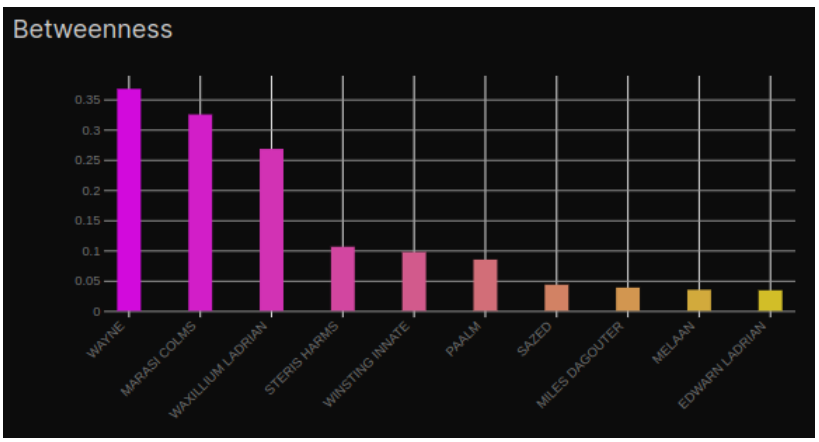

Figura 6.53: *Betweenness* de *Nacidos de la bruma: Era 2*

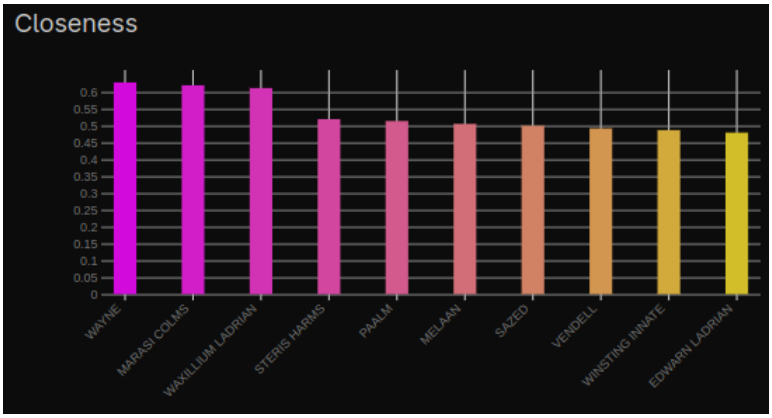

Figura 6.54: *Closeness* de *Nacidos de la bruma: Era 2*

6.4 El archivo de las tormentas

El archivo de las tormentas es la obra magna de Brandon Sanderson. La historia se desarrolla en Roshar, un planeta crucial en la historia del Cosmere, ya que ha sido habitado por tres de las esquirlas. Cada uno de los libros revela el pasado de uno de los personajes principales, lo que nos lleva a una hipótesis: el protagonista del libro debería ser aquel del que se cuenta su historia. Además, esta saga presenta numerosos personajes que han aparecido en otros libros, lo que la convierte en un caso de especial interés para el análisis de redes. Los libros del archivo son los más extensos de Brandon, con un promedio de alrededor de 1350 páginas, lo que implica que el número de personajes que aparecen es mucho mayor que en el resto de sus obras. Además, debido a las múltiples tramas en paralelo, el análisis de comunidades adquiere aún más valor en esta saga.

6.4.1. El camino de los reyes

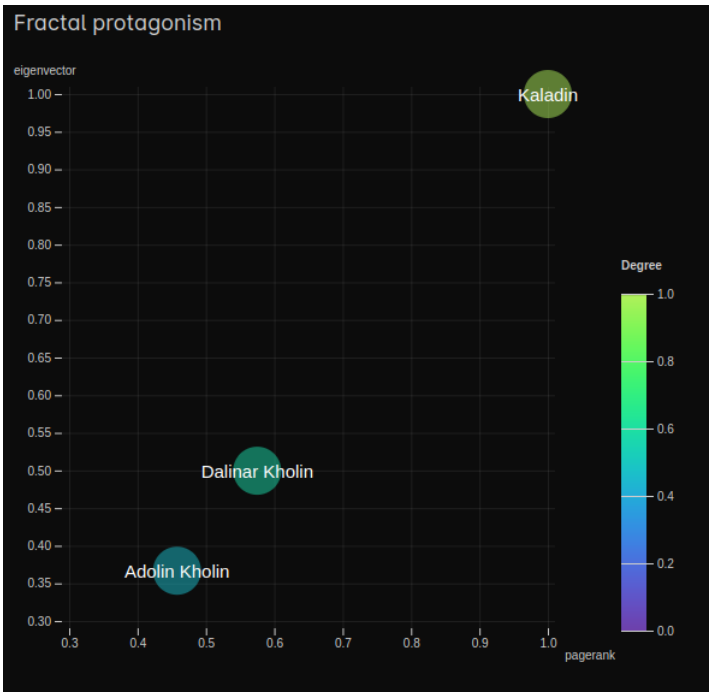

Figura 6.55: *Fractal protagonism* de *El camino de los reyes*

El camino de los reyes es el primer libro de *El archivo de las tormentas*. En él se desvela el pasado de Kaladin, uno de los personajes más queridos de la saga. Como era de esperar, los datos(figura 6.55) respaldan la hipótesis y Kaladin se presenta como el claro protagonista de esta primera entrega.

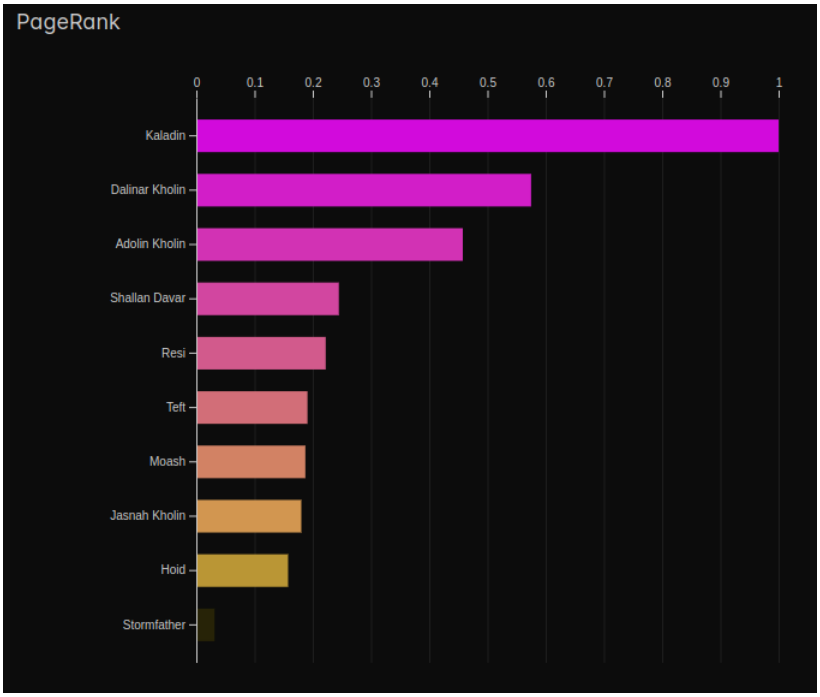

Figura 6.56: *Pagerank* de *El camino de los reyes*

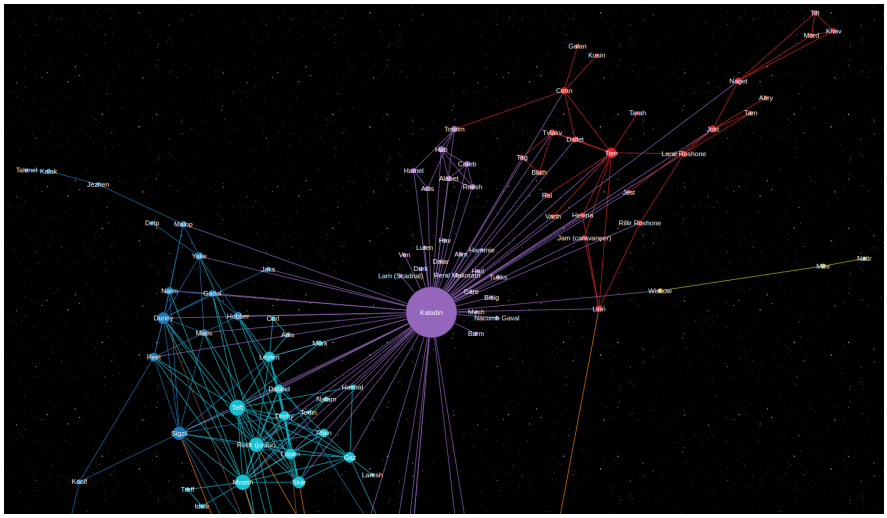

Figura 6.57: Comunidad de Kaladin en *El camino de los reyes*

Lo más destacable de este libro son las comunidades. Se puede observar en el grafo perfectamente los diferentes grupos que se forman durante la historia y como cada uno de los personajes principales encabeza una comunidad distinta. La red(figura 6.57) muestra a la perfección los tres grandes grupos de amigos que ha tenido Kaladin a lo largo de su vida, desde niño pasando por su adolescencia hasta la época actual. Alrededor de Kaladin se crea una comunidad con los compañeros que tuvo durante su estancia en el ejército de Amaram. Cerca de él encontramos a todo el puente cuatro como otra comu-

nidad, y a Tien, su hermano fallecido, con la comunidad de su pueblo cuando era niño. Las comunidades parecen indicar agrupaciones temporales.

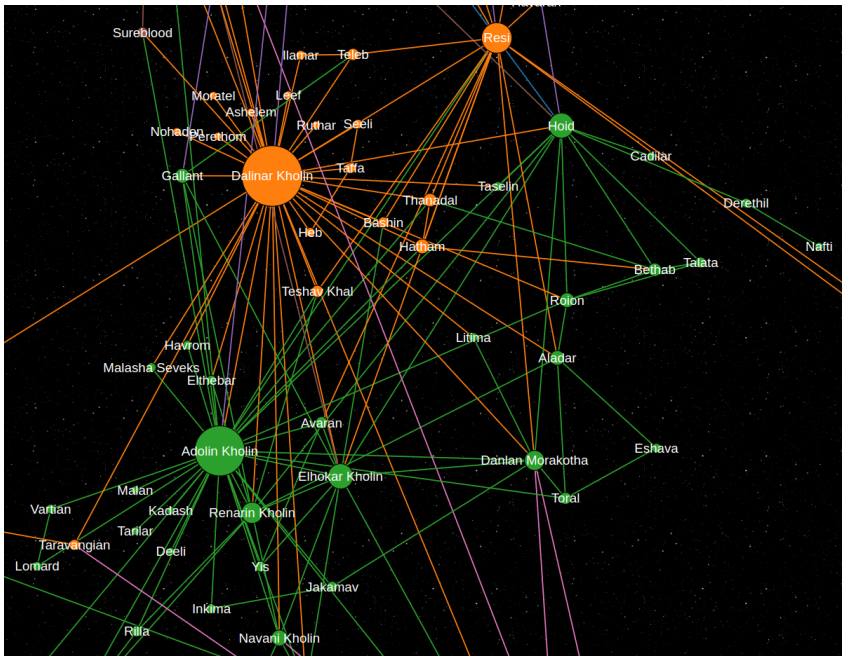

Figura 6.58: Comunidad de los Kholin en *El camino de los reyes*

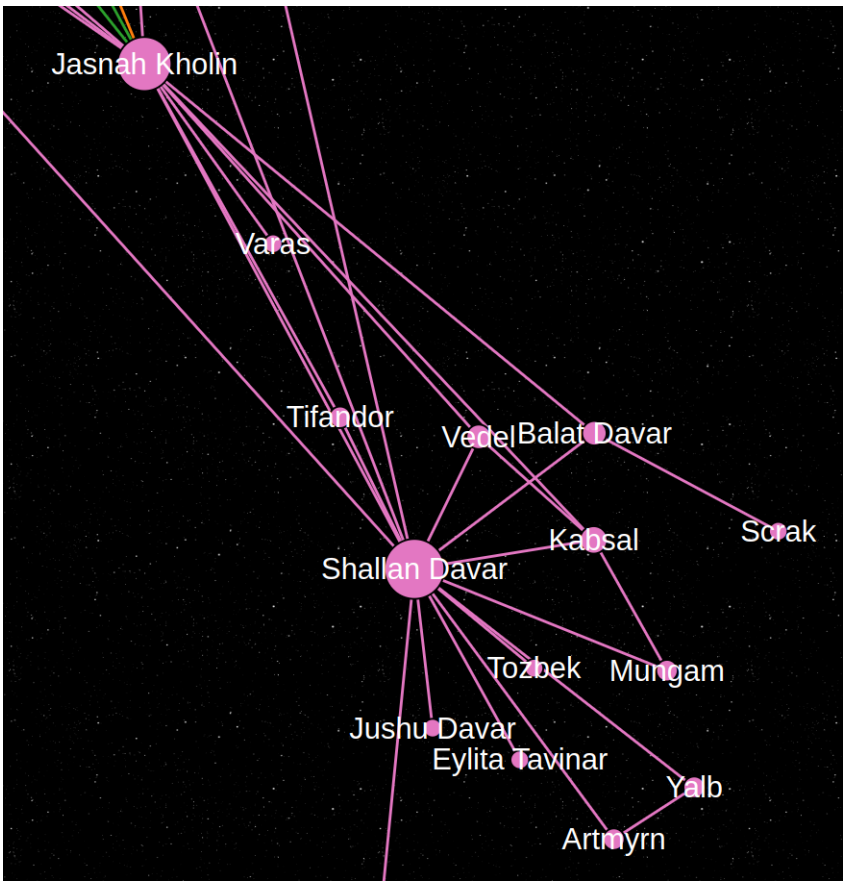

Figura 6.59: Comunidad de Shallan Y Jasnah en *El camino de los reyes*

Luego, tenemos otras dos grandes comunidades: la de Dalinar Kholin y la de su hijo, Adolin Kholin(figura 6.58). Debido a su estrecha relación y al tiempo que pasan juntos, estas dos comunidades están muy entrelazadas. Aunque no se destaque como una comunidad distinta, se puede observar por separado a Szeth, el asesino de blanco. Es curioso cómo, desde el primer libro, existe una relación indirecta desde Kaladin hasta Szeth, pasando por Sigzil, Koolf, Amark, Took y finalmente llegando a Szeth. Por último, tenemos a Jasnah Kholin y Shallan Davar(figura 6.59). Al observar la red, podemos intuir las diferentes tramas, ya que por un lado tenemos a Kaladin como esclavo. Más tarde encontramos al puente cuatro y el pasado de Kaladin. En los campamentos de las llanuras quebradas, nos adentramos en la historia de Dalinar y Adolin, y finalmente, en Kharbranth, conocemos a Jasnah y Shallan.

6.4.2. Palabras radiantes

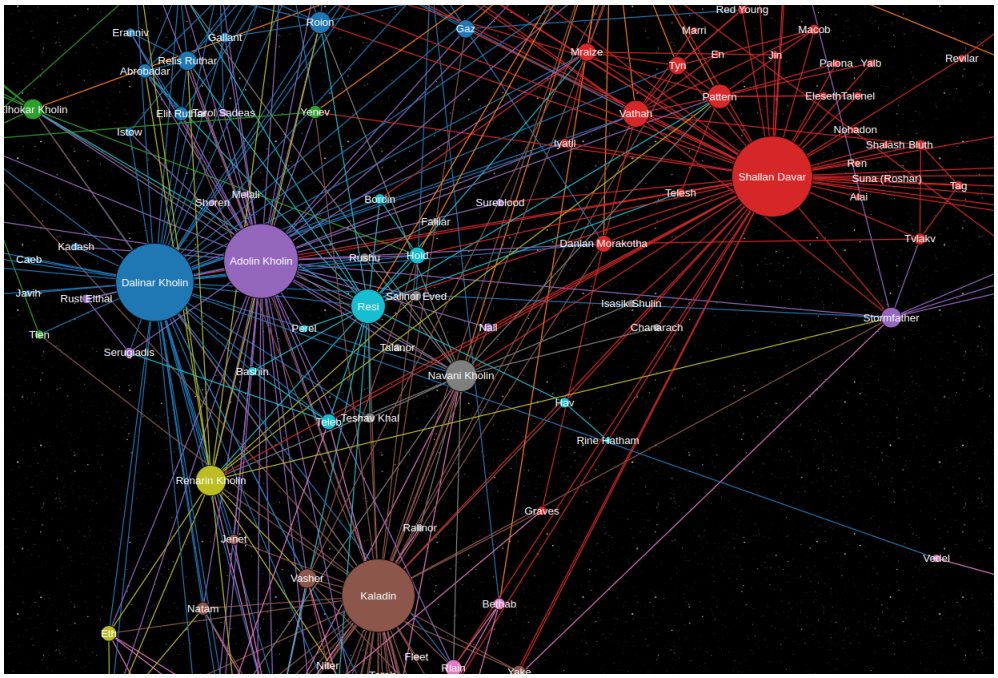

Figura 6.60: Grafo de los personajes principales de *Palabras radiantes*

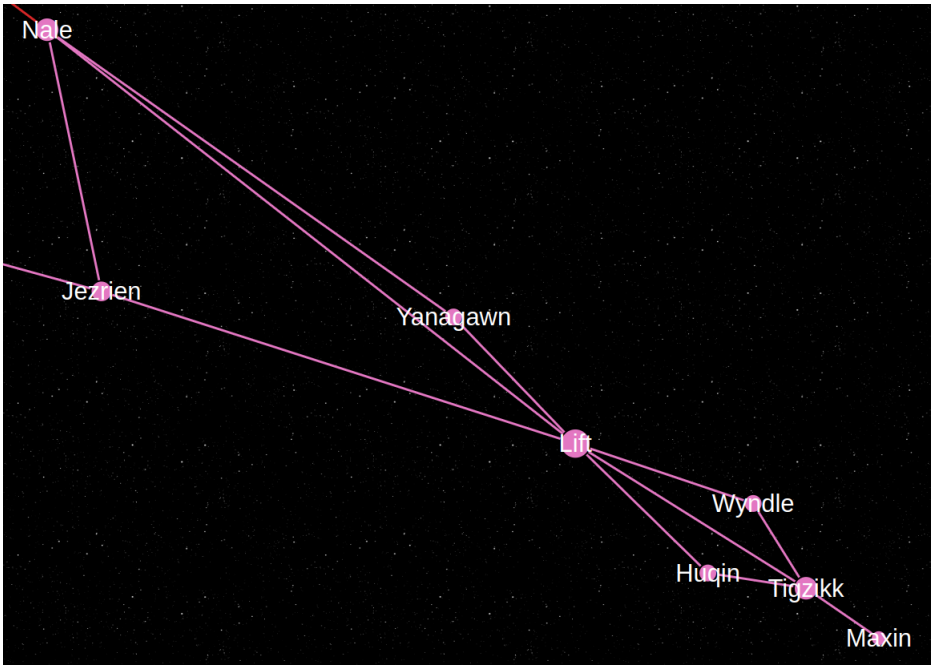

Figura 6.61: Comunidad de Lift en *Palabras radiantes*

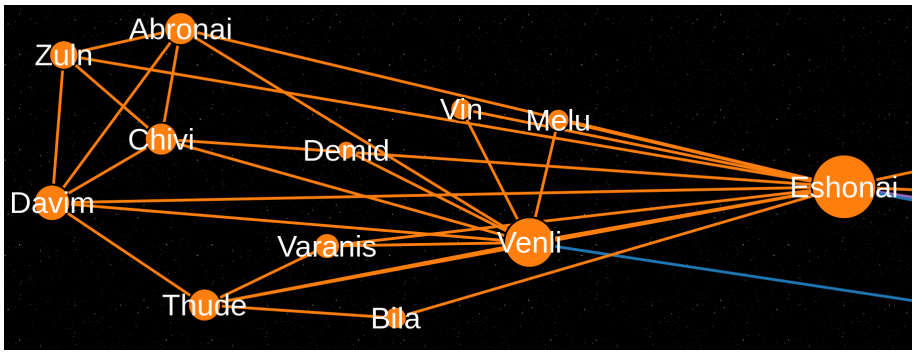

Figura 6.62: Comunidad de Eshonai en *Palabras radiantes*

En el segundo libro, *Palabras radiantes*, la mayoría de los personajes principales ya se conocen, lo que hace que sus comunidades se entrelacen(figura 6.60). Sin embargo, también se introducen nuevos personajes, como Lift(figura 6.61) y Eshonai(figura 6.62), junto con sus respectivos grupos.

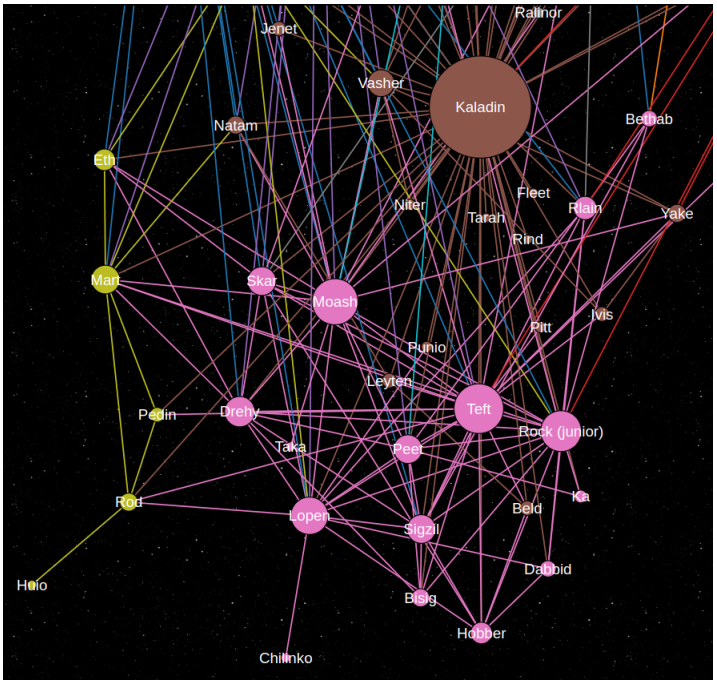

Figura 6.63: Comunidad del puente cuatro en *Palabras radiantes*

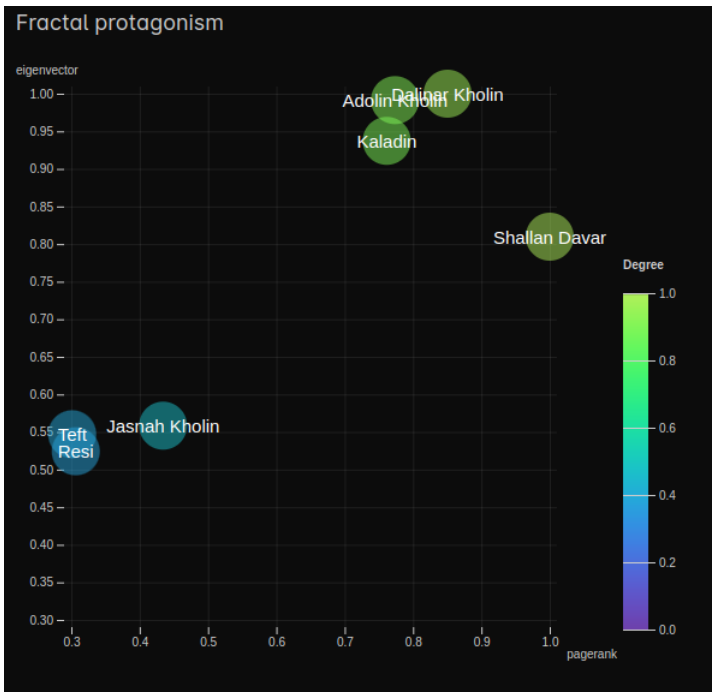

Figura 6.64: *Fractal protagonism* de *Palabras radiantes*

El puente cuatro(figura 6.63) adquiere mayor protagonismo, estableciendo Conexiones con más personajes. Esto se refleja en el número de relaciones que tienen estos personajes. De hecho, uno de los miembros del puente cuatro, Teft, incluso aparece en la métrica de *fractal protagonism*(figura 6.64). Por otro lado, encontramos la comunidad de los Parshendi, con Eshonai como nodo principal, y los familiares de Shallan, representados en color morado.

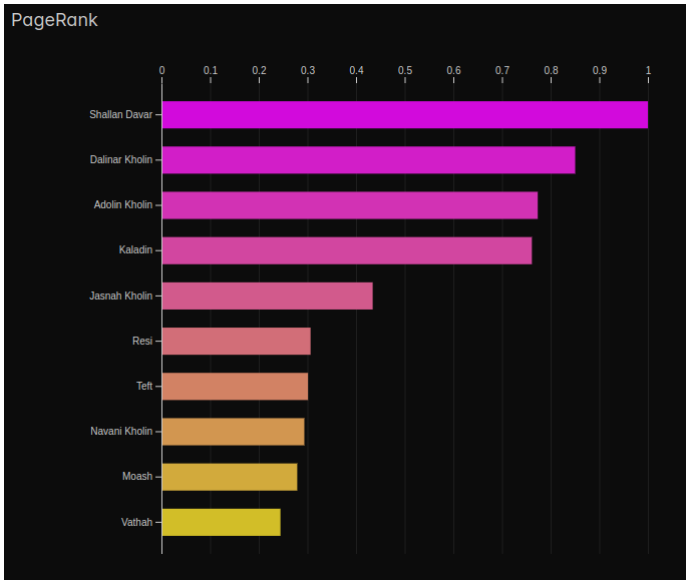

Figura 6.65: *Pagerank* de *Palabras radiante*

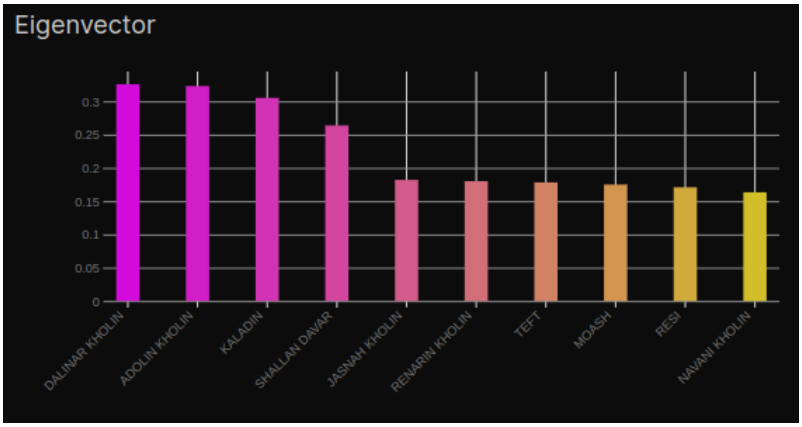

Figura 6.66: *Eigenvector* de *Palabras radiantes*

En este libro también se cumple nuestra hipótesis aunque no de forma tan clara como en el anterior. Shallan tiene el *pagerank*(figura 6.65) y el grado más elevado pero ocupa la cuarta posición en la gráfica de *eigenvector*(figura 6.66). Aun así se puede considerar que Shallan Davar es la protagonista del segundo libro de *El archivo de las tormentas*.

6.4.3. Juramentada

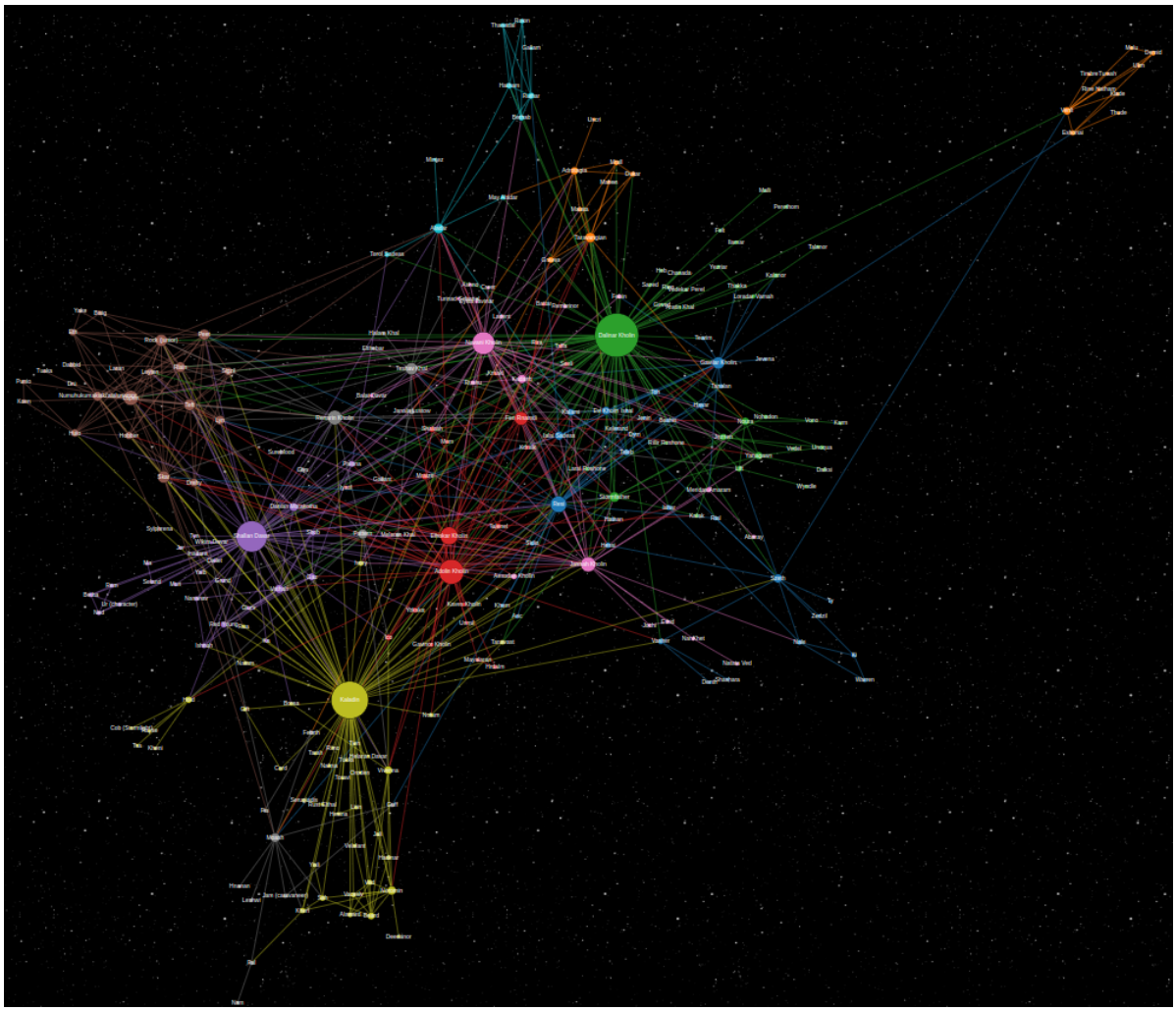

Figura 6.67: Grafo de la red de *Juramentada*

En este tercer libro no hay demasiados cambios respecto al anterior. Nuevamente se cumple la hipótesis siendo Dalinar el personaje con mayor *fractal protagonism*(figura 6.69). Destaca el personaje de Navani Kolin que hasta el momento tenia un papel más secundario y ahora se posiciona como quinta en el *pagerank*(figura 6.68). También es imporate remarcar algo que se ve claramente en nuestra red, la aparición de dos personajes que ya conocemos de otro libro, Vivenna y Vasher, aunque Vasher ya había aparecido en *Palabras Radiantes*.

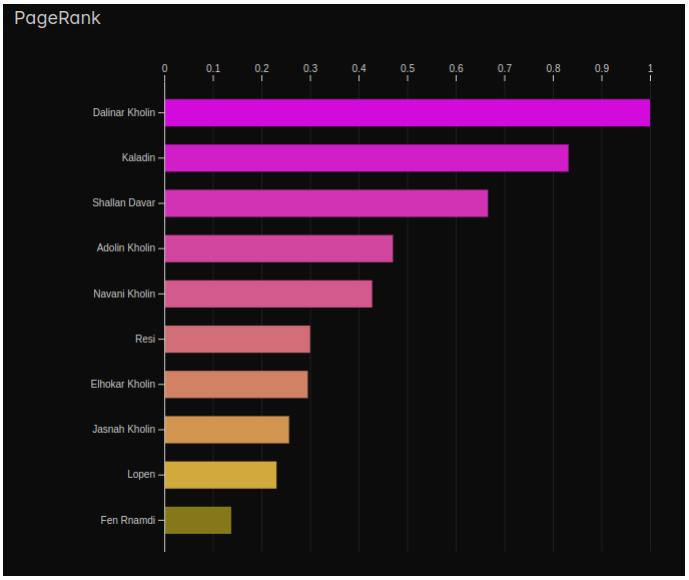

Figura 6.68: *Pagerank* de *Juramentada*

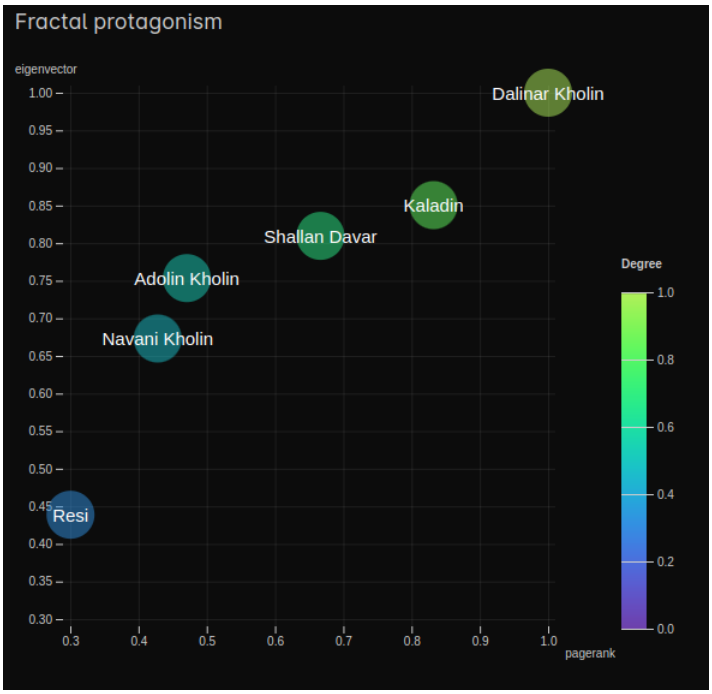

Figura 6.69: *Fractal protagonism* de *Juramentada*

6.4.4. El ritmo de la guerra

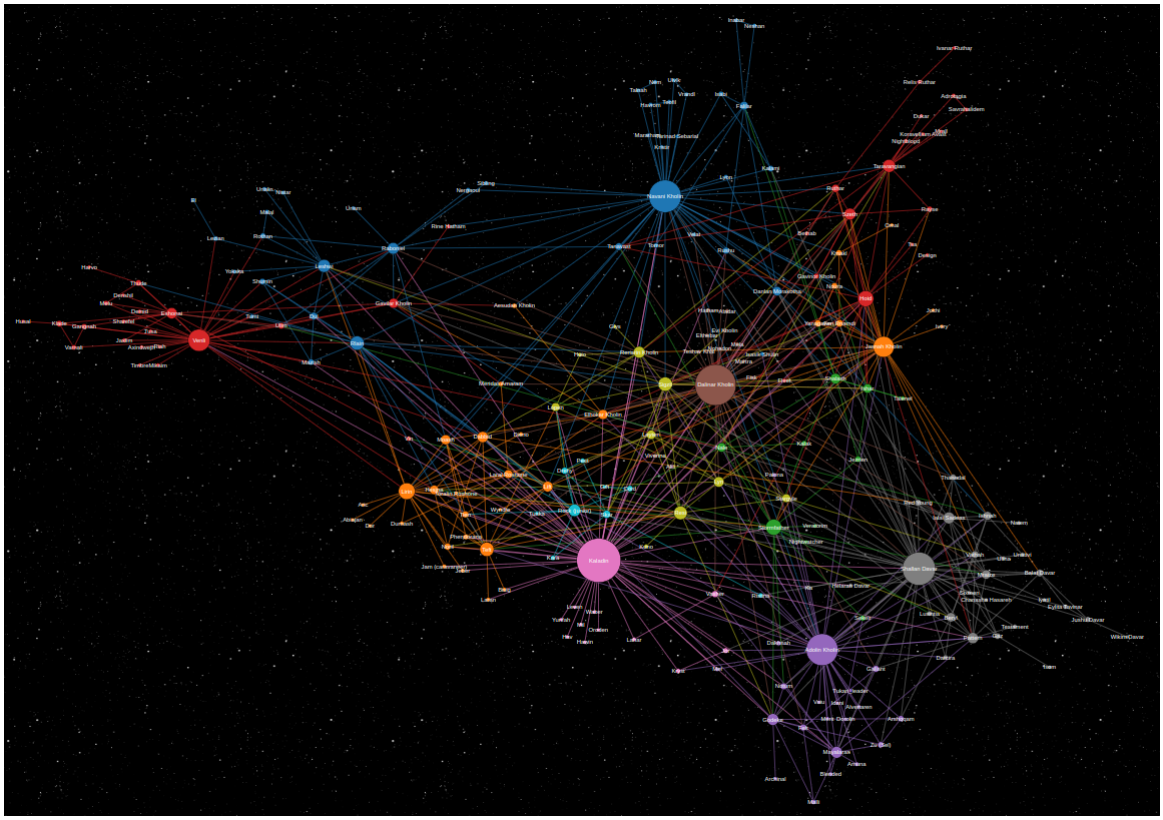

Figura 6.70: Grafo de la red de *El ritmo de la guerra*

El ritmo de la guerra es el cuarto y último libro de *El archivo de las tormetas* escrito hasta la fecha de publicación de esta memoria. En principio iba a ser el libro de Venli y, de hecho, se cuenta su pasado. Pero al parecer, Brandon Sanderson ya había abordado mucho sobre los oyentes en Juramentada, por lo que Venli no termina teniendo tanto protagonismo y se le da mayor importancia a Navani. Esto se refleja en el *pagerank*(figura 6.71), Navani sube a la tercera posición y Venli a la quinta. A pesar de esto, ambas tienen un peso notable en la trama. No obstante, Kaladin vuelve a ser el protagonista en esta cuarta entrega, ya que ocupa el primer lugar en todas las métricas(figura 6.72).

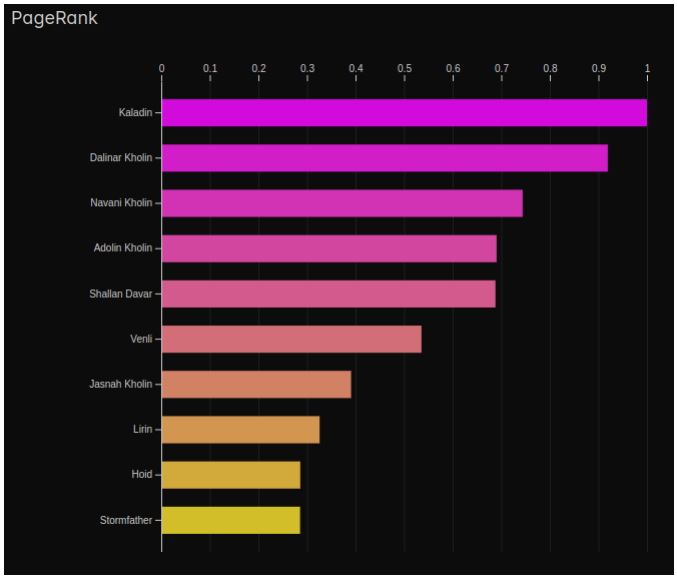

Figura 6.71: *Pagerank* de *El ritmo de la guerra*

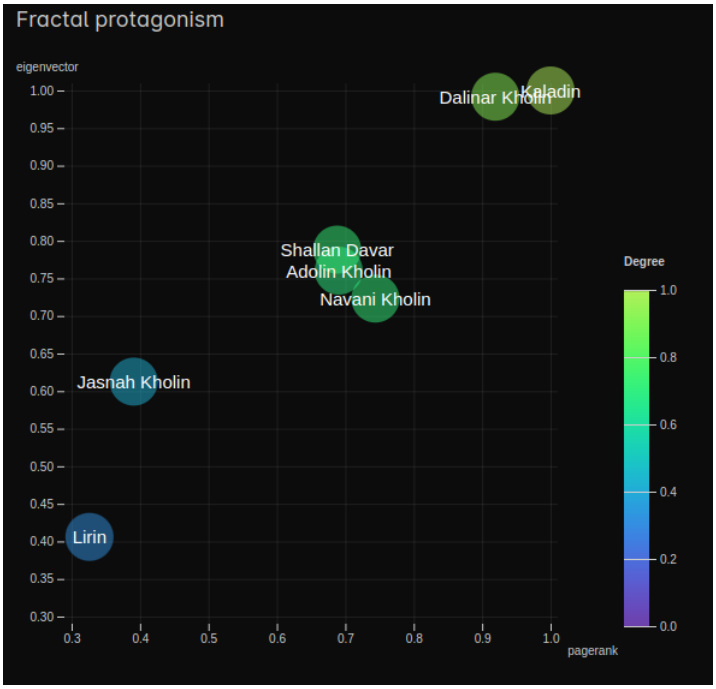

Figura 6.72: *Fractal protagonism* de *El ritmo de la guerra*

Realizando el análisis de comunidades vuelve a ocurrir como en el primer libro, podemos intuir las diferentes tramas observando la red. En este libro nuestros protagonistas se vuelven a dividir y esto se refleja en la red pintando las comunidades casi como las diversas historias que ocurren simultáneamente en diferentes puntos del planeta. Por una parte tenemos a Kaladin resistiendo en la torre, a Venli(figura 6.73) con los Retornados, a Navani(figura 6.74) cautiva e investigando y el resto pintados en naranja completamente cautivos. Luego vemos a Shallan y Adolin(figura 6.75) que están en Shadesmar y que, además, se diferencia sus subtramas, una con los Sangre Espectral y otro con Maya. Por último, están Dalinar Kolin aprendiendo sobre sus potencias, Jashna recién ascendida al trono de Alezkar y Taravagian con sus conspiraciones.

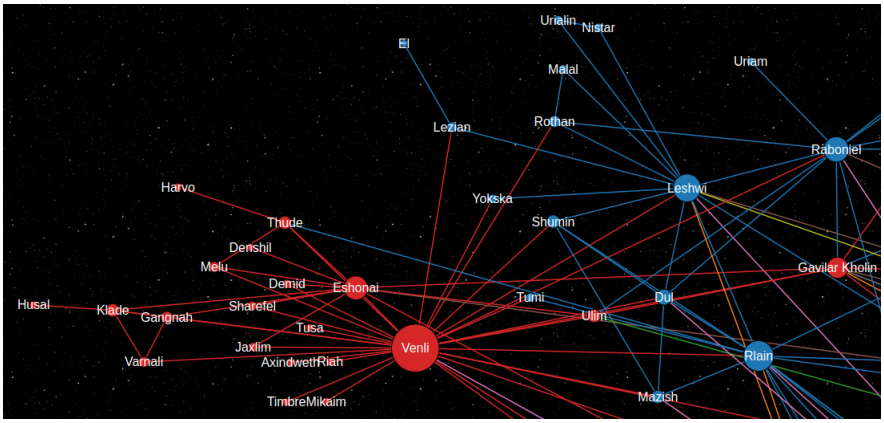

Figura 6.73: Comunidad de Venli en *El ritmo de la guerra*

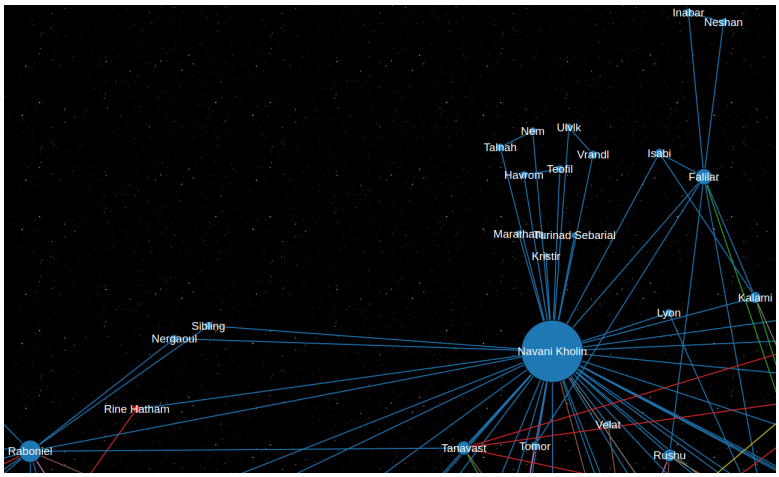

Figura 6.74: Comunidad de Navani en *El ritmo de la guerra*

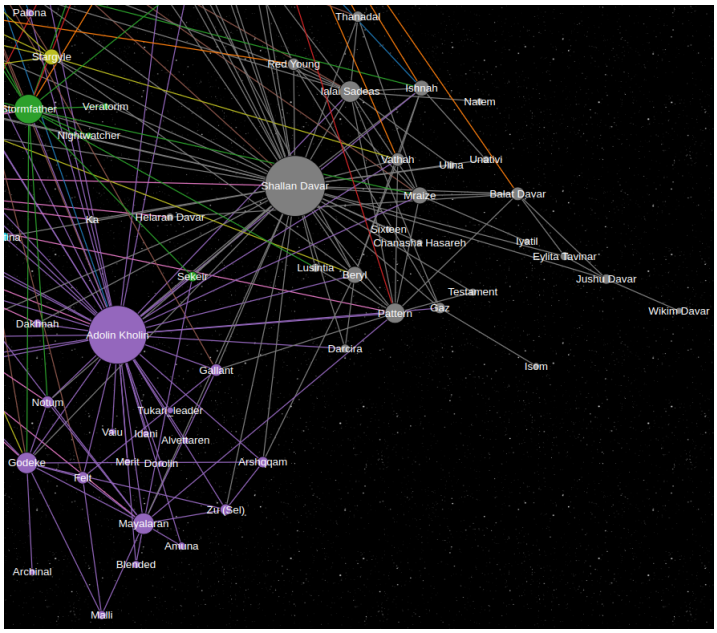

Figura 6.75: Comunidad de Shallan y Adolin en *El ritmo de la guerra*

6.4.5. El archivo de las tormentas en conjunto

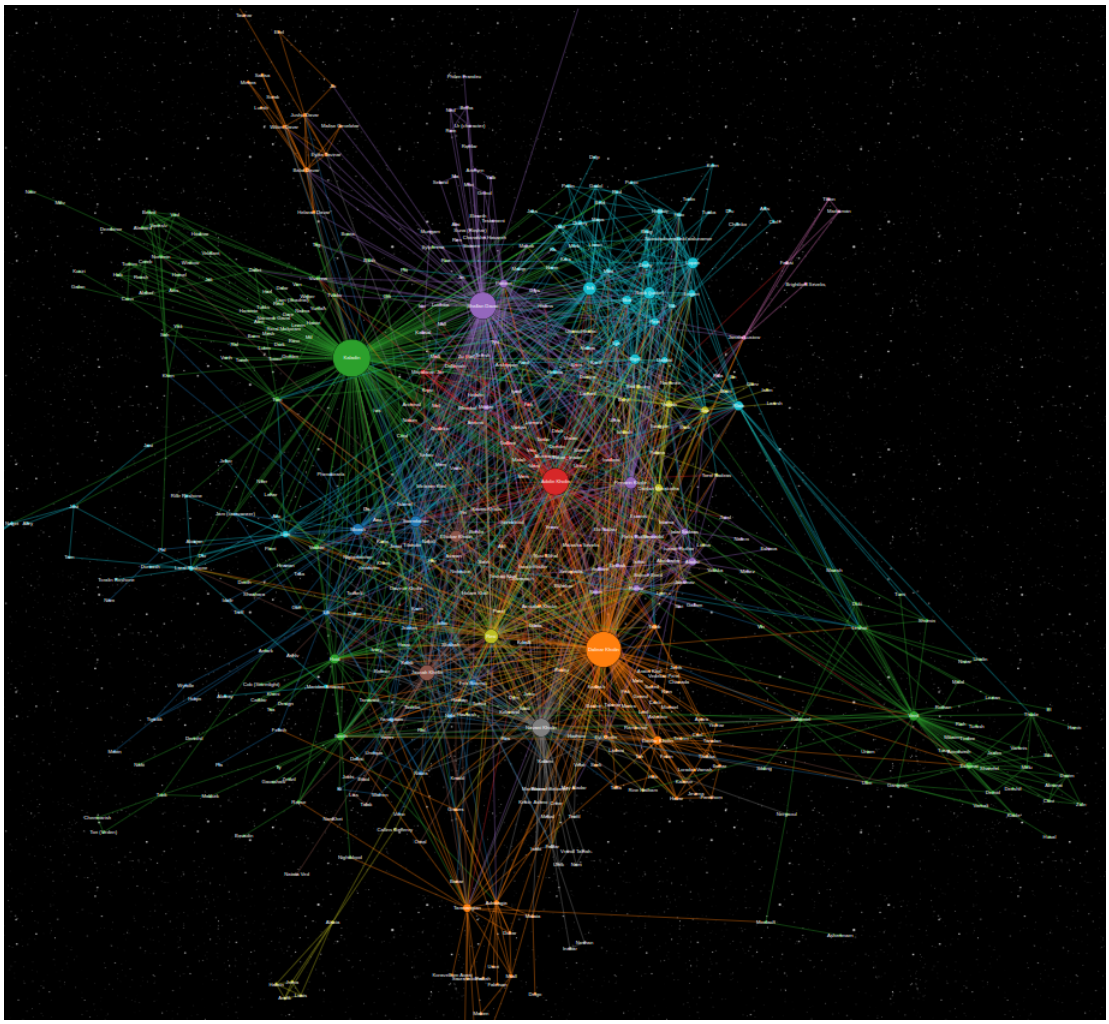

Figura 6.76: Grafo de la red de *El archivo de las tormentas*

Aunque el quinto libro no esté aun publicado para analizar toda la primera parte de *El archivo de las tormentas* se pueden sacar conclusiones de los cuatro primeros libros. Primero, comentar los cuatro nodos que más destacan de la red: Kaladin, Shallan, Dalinar y Adolin. Los tres primeros no deberían sorprender, cada uno ha sido protagonista en uno que otro libro, pero destaca Adolin, un personaje que siempre ha estado muy presente pero que nunca ha conseguido sobresalir respecto a los demás. Nuevamente es muy resaltable las comunidades. La red muestra muy bien y diferenciadas los conjuntos de personajes.

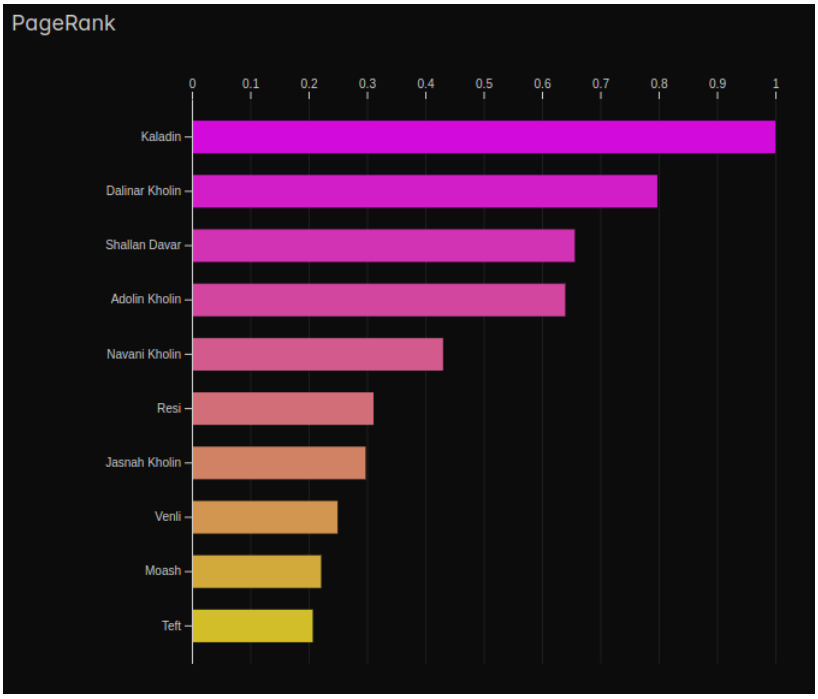

Figura 6.77: *Pagerank* de *El archivo de las tormentas*

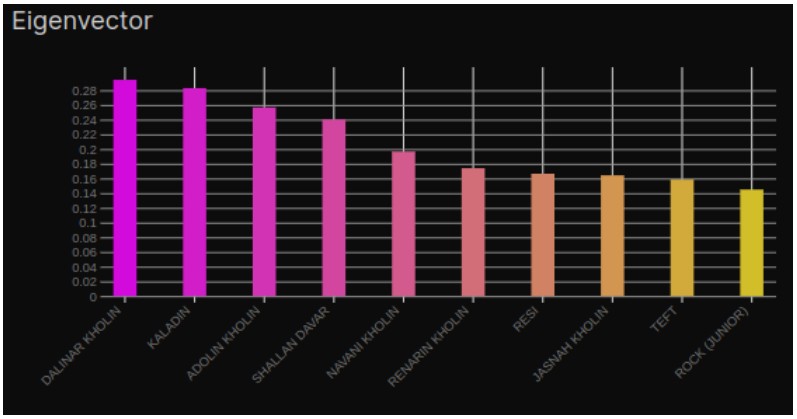

Figura 6.78: *Eigenvector* de *El archivo de las tormentas*

Para terminar con el análisis de *El archivo de las tormentas* y a falta de la última entrega de la primera parte, Kaladin se convierte en el protagonista de la saga con la primera posición en el *pagerank*(figura 6.77) y una muy justa segunda posición en el *eigenvector*(figura 6.78).

6.5 El Cosmere en su inmensidad

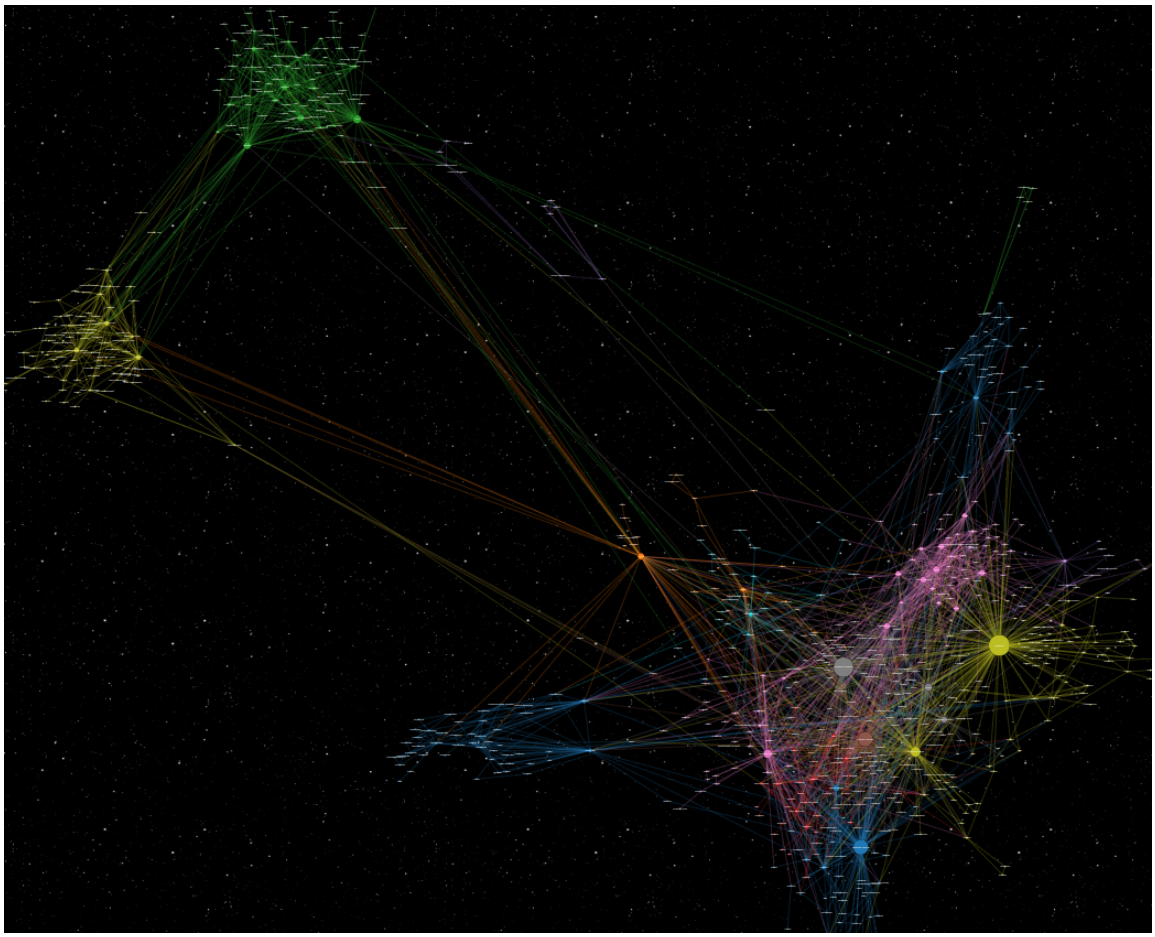

Figura 6.79: Grafo de la red del Cosmere

Lo primero que salta a la vista al observar la red del Cosmere son los cuatro(figura 6.79) grandes grupos de nodos. Cada uno de ellos pertenece a una saga de libros: Nacidos de la bruma era 1 y 2, *El aliento de los dioses* y *El archivo de las tormentas*. No solo eso, sino que además el algoritmo de Louvain asigna a cada saga un grupo distinto, salvo al AT, que al tener tantos personajes lo conforman varias comunidades. También destaca que los nodos más grandes están en *El archivo de las tormentas*, esto no es de extrañar ya que en un solo libro del AT hay más personajes que en sagas enteras.

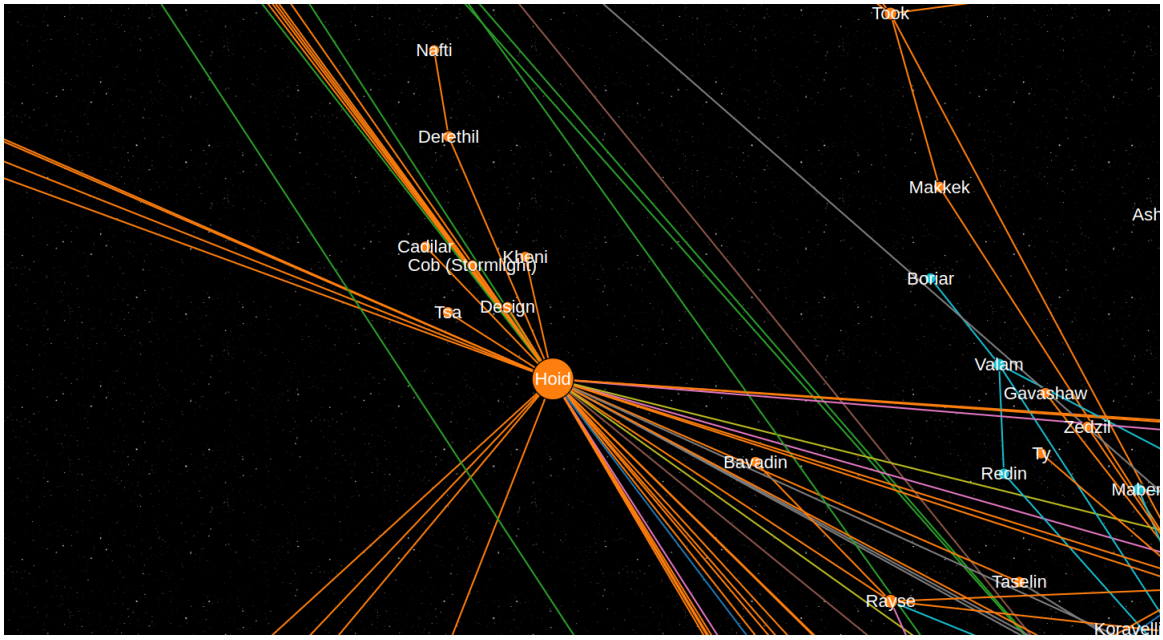

Figura 6.80: Hoid en la red del Cosmere

Hasta ahora no hemos dado mucha importancia a las métricas de *betweenness* y *closeness*. Esto se debe a que casi siempre eran los personajes principales los que mejor valor tenían en estas medidas. Pero esto cambia cuando juntamos todos los libros del Cosmere porque aparecen personajes capaces de conectar los libros. El personaje más importante en esta tarea es Hoid(figura 6.80), quedando en segunda y tercera posición en el *betweenness*(figura 6.84) y *closeness*(figura 6.85) respectivamente. Hoid es un personaje que aparece en todas las sagas del Cosmere, aunque se suele relacionar solo con los personajes más importantes de cada una y con diferentes nombres. En la red destaca por representar un papel central siendo el único que conecta las cuatro sagas y que probablemente obtendría el primer puesto en ambas medidas de hacer el análisis de todos los libros del Cosmere. Es importante mencionar que, aunque en general no hay personajes con el mismo nombre dentro de un mismo libro, si los hay en diferentes sagas por lo que podría haber relaciones entre libros que no sean ciertas debido a que el algoritmo no lo ha podido detectar y la revisión manual no lo ha corregido.

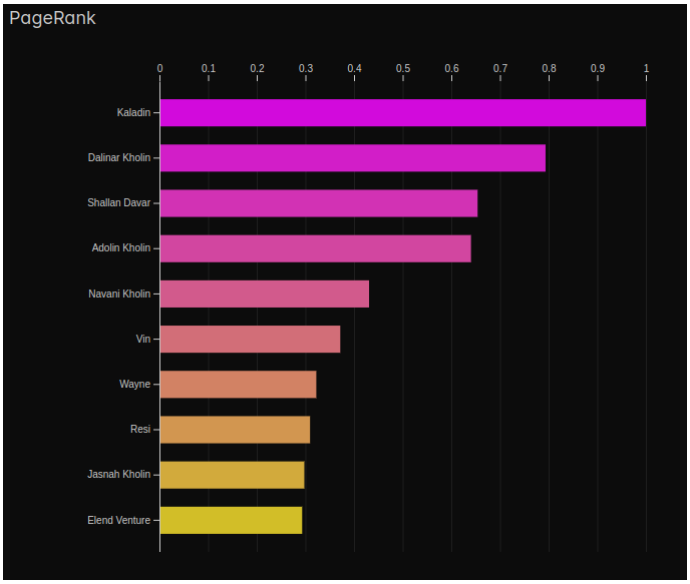

Figura 6.81: *Pagerank* del Cosmere

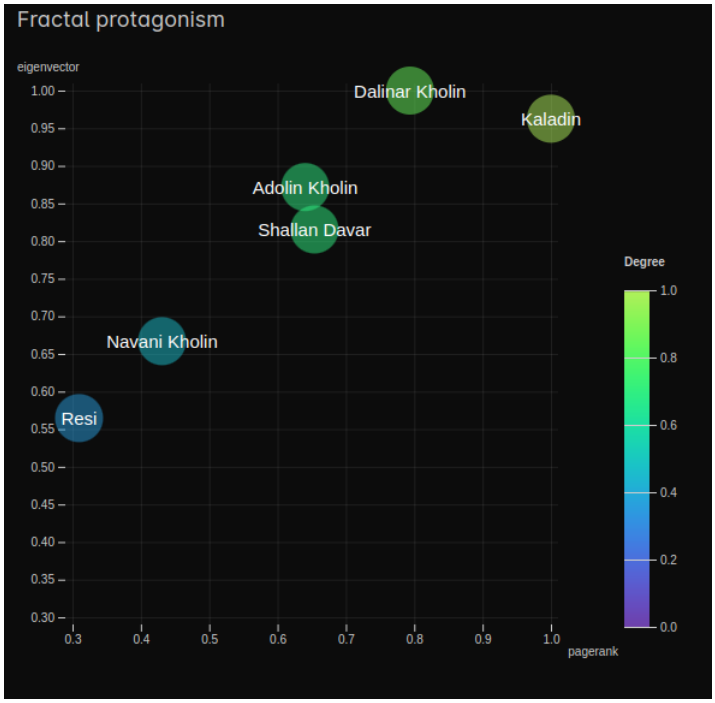

Figura 6.82: *Fractal protagonism* del Cosmere

Han aparecido personajes que han ganado protagonismo durante el transcurso de las historias mientras que otros la han ido perdiendo, pero solo uno puede ser el protagonista de todo el Cosmere. Por ocupar prácticamente el primer puesto en todas las métricas(figura 6.82), Kaladin Bendito por la Tormenta se alza con el título de protagonista de todo el Cosmere.

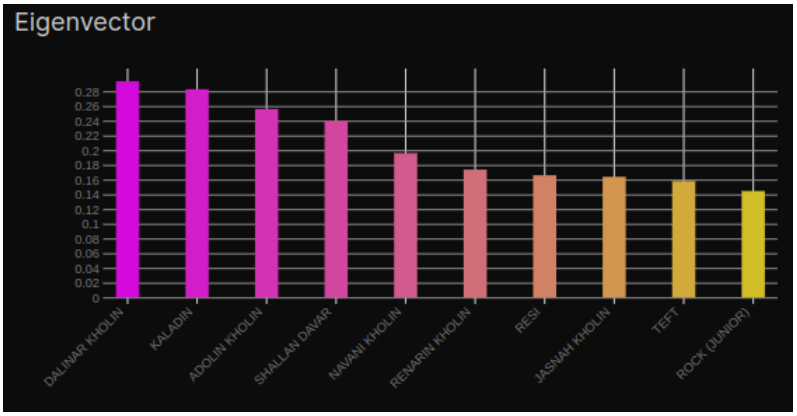

Figura 6.83: *Eigenvector* del Cosmere

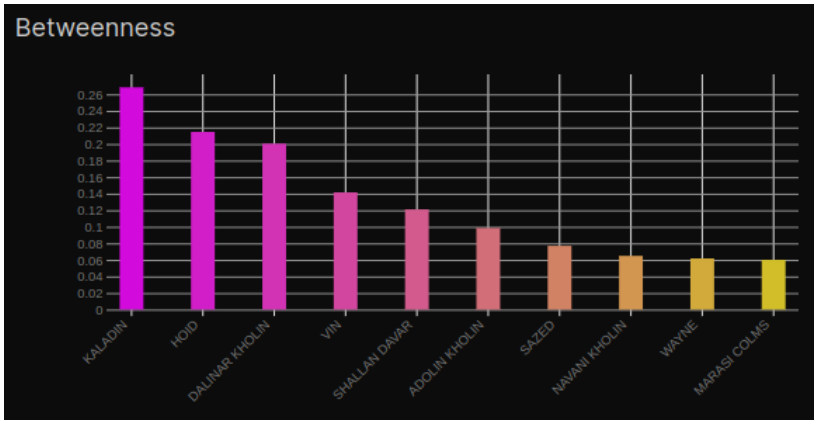

Figura 6.84: *Betweenness* del Cosmere

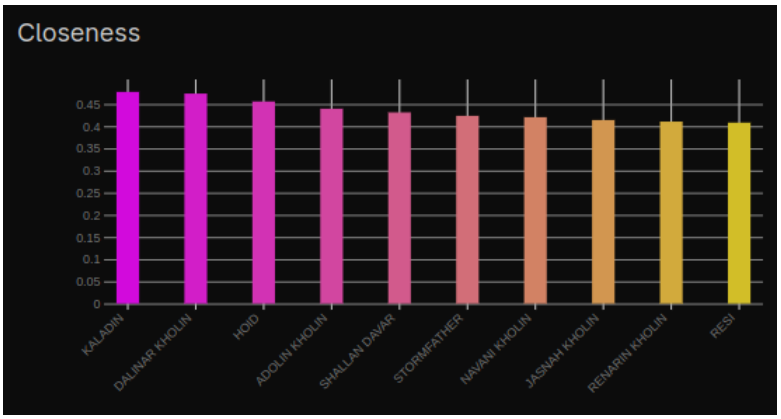

Figura 6.85: *Closeness* del Cosmere

CAPÍTULO 7
Conclusiones

Brandon Sanderson es actualmente uno de los escritores más importantes del mundo. No es solo por las increíbles historias que plasma en las hojas, sino que además ha demostrado ser capaz de estar a la altura de las expectativas sin bajar el ritmo de producción. Sin duda es un ejemplo a seguir para muchos futuros escritores que actualmente están cursando sus clases.

Para conseguir nuestros objetivos hemos realizado un *script* capaz de entrar en la *coopermind* y extraer los alias de los personajes del Cosmere. Hemos hecho un algoritmo que recoge la lista de personajes y alias para, a partir de los archivos nlp del preanálisis, analizar los libros e incluso otros textos. Luego hemos empleado esa información para mostrarla en una página web como una red y gráficas. Finalmente, hemos podido lanzar la aplicación para que así cualquier fan como nosotros pueda disfrutar de investigar las Conexiones del Cosmere.

Además, observando la información plasmada en la web, hemos realizado un análisis de la red para sacar nuevos datos del universo de Brandon Sanderson. Por ejemplo, hemos descubierto que en las terceras entregas de *Nacidos de la bruma* Sanderson tiende a dar imporantancia a algún personaje secundario. También hemos podido deducir las diferentes tramas de *El camino de los reyes* al observar las comunidades resultantes del grafo. Pero, por encima de toda la información que hemos descubierto, la más relevante es que hemos visto que Kaladin Bendito por la Tormenta es el protagonista del universo del Cosmere.

7.1 Trabajos futuros

Durante el proyecto han salido muchas ideas que no se han podido llevar a cabo por limitaciones de tiempo. Vamos a enumerar algunas de ellas que se quedan como trabajos futuros:

Análisis de sentimientos : Se pretende realizar un análisis similar al análisis de relaciones, pero en lugar de buscar personajes de los libros buscar palabras que tengan sentimiento. Esto servirá para ver que libros son más tristes, más alegres y la evolución de un libro a lo largo de la historia.

Guía de lectura : La idea es crear una guía de lectura en forma de grafo. Actualmente existen muchas guías diferentes así que pretendemos juntarlas todas y añadir información que permita al lector saber por cual libro empezar o seguir.

Configuración : Ahora mismo solo se puede configurar la web para filtrar los libros. El objetivo final es tener un panel de configuración rica en opciones. Una de las más destacables seria poder limpiar la red para dejar solo los personajes protagonistas.

Mejorar el análisis : Aunque hemos obtenido un resultado más que decente, queremos mejorar el algoritmo de análisis. Se han tenido que hacer ajustes manualmente que se debería automatizar, mitigar el problema de los alias y los personajes con mismo nombre, etc.

Rendimiento : El rendimiento de la aplicación aún tiene margen de mejora

Grafo : Añadir funcionalidades como desplazar los nodos, resaltar los arcos de un nodo o redirigir a la *coopermind* al clicar en un nodo.

Autoanálisis de la red : Seria genial si se pudiera implementar un algoritmo, ya sea convencional o de inteligencia artificial, que sea capaz de analizar la red y explicar los resultados al lector.

CAPÍTULO 8
Bibliografia

Referencias

Beveridge, A., y Shan, J. (2016). Network of thrones. *Math Horizons*, *23*, 18 - 22.

Bostock, M. (2012). *D3.js - data-driven documents.* Descargado de http://d3js.org/

Documentation - git. (s.f.). Descargado de https://git-scm.com/doc

Documentation - tailwind css. (2023, Jun). Descargado de https://v2.tailwindcss.com/docs

Evans, E. (2004). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley.

Fields, S., Cole, C. L., Oei, C., y Chen, A. T. (2022, 08). Using named entity recognition and network analysis to distinguish personal networks from the social milieu in nineteenth-century Ottoman–Iraqi personal diaries. *Digital Scholarship in the Humanities*, *38*(1), 66-86. Descargado de https://doi.org/10.1093/llc/fqac047 doi: 10.1093/llc/fqac047

Flanagan, D., y Ferguson, P. (1998). *Javascript: The definitive guide* (3rd ed.). USA: O'Reilly Associates, Inc.

Fractal protagonists. (2023, Jun). Descargado de https://networkofthrones.wordpress.com/fractal-protagonists/

Hexagonal architecture. (2023, Jun). Descargado de https://fideloper.com/hexagonal-architecture

Honnibal, M., y Montani, I. (2017). *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing.* Descargado de https://spacy.io/api (To appear)

Jain, N., Bhansali, A., y Mehta, D. (2014). Angularjs: A modern mvc framework in javascript. *Journal of Global Research in Computer Science*, *5*(12), 17–23.

La coppermind wiki - 17th shard, the official brandon sanderson fansite. (2023, Jun). Descargado de https://es.coppermind.net/wiki

Lough, C. (2017, Jun). *Mapping brandon sanderson's cosmere raises so many questions about his future books.* Descargado de https://www.tor.com/2017/06/06/brandon-sanderson-worldhoppers

Office, U. C. (2023, Jun). *Welcome to the u.s. copyright office.* Descargado de https://www.copyright.gov/

Plique, G. (2023, marzo). *Graphology, a robust and multipurpose Graph object for JavaScript*. Zenodo. Descargado de https://doi.org/10.5281/zenodo.7695163 doi: 10.5281/zenodo.7695163

Rxjs reactive extensions library for javascript. (2023, Jun). Descargado de https://rxjs.dev/api

Websites for you and your projects. (2023, Jun). Descargado de https://pages.github.com/

Wiki oficial del cosmere en español. (2023, Jun). Descargado de https://cosmere.es/

Wiki oficial del cosmere y la literatura de brandon sanderson. (2023, Jun). Descargado de https://www.brandonsanderson.com/

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**	X			
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**		X		
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				X

Nuestro proyecto se ha centrado en el desarrollo de herramientas de código abierto para el análisis de redes y el renderizado de gráficos en la web. El software resultante ha sido creado con la intención de ser utilizado por futuros estudiantes y emprendedores interesados en llevar a cabo proyectos similares. Además, creemos que nuestro software podría ser de utilidad tanto para el fandom de Brandon Sanderson como para empresas que deseen utilizarlo como punto de partida para sus propias iniciativas. Estamos convencidos de que nuestro proyecto puede tener un impacto significativo en la mejora de la educación y la industria informática.

En primer lugar, nuestro proyecto contribuye al ODS 4, .Ed"ucación de calidad". Al proporcionar herramientas de análisis de redes y renderizado de gráficos, estamos facilitando el acceso a recursos y tecnologías que pueden ayudar a los estudiantes a comprender mejor la estructura y las relaciones dentro de los libros de Brandon Sanderson. Esto puede fomentar un aprendizaje más profundo y significativo, alentando a los estudiantes a desarrollar habilidades de análisis y síntesis que son fundamentales para una educación de calidad. Además, al ser un software de código abierto, estamos promoviendo la colaboración y el intercambio de conocimientos entre los usuarios, lo que puede enriquecer aún más la experiencia educativa.

Por otro lado, nuestro proyecto también tiene implicaciones en el ODS 9, Ïndustria, innovación e infraestructuras". La industria informática es un campo en constante evolución y crecimiento, y creemos que nuestro software puede contribuir a impulsar la innovación

en esta área. Al proporcionar una plataforma y herramientas para el análisis de redes y el renderizado de gráficos, estamos facilitando el desarrollo de nuevas soluciones y aplicaciones en este campo. Además, al ser de código abierto, nuestro software permite a los usuarios adaptarlo y mejorarlo según sus necesidades, fomentando así la creatividad y la innovación en la industria informática.

Sin embargo, es importante reconocer que nuestro proyecto no cumple con todos los Objetivos de Desarrollo Sostenible. Por ejemplo, no abordamos directamente el ODS 1, "Fin de la pobreza". Si bien nuestro software puede ser utilizado de forma gratuita, no aborda directamente las cuestiones económicas y estructurales subyacentes que perpetúan la pobreza. Reconocemos la importancia de este objetivo y su interconexión con otros aspectos del desarrollo sostenible, y alentamos a otros proyectos y esfuerzos a abordar directamente esta problemática.

Nuestro proyecto tampoco se enfoca en el ODS 2, "Hambre cero". Si bien la educación y el acceso a herramientas pueden ser factores importantes para abordar el hambre y la inseguridad alimentaria a largo plazo, reconocemos que nuestro proyecto no se dirige directamente a este desafío. Es fundamental que se realicen esfuerzos adicionales y coordinados para abordar las causas subyacentes de la falta de acceso a alimentos adecuados y la inseguridad alimentaria.

Tampoco abordamos directamente el ODS 3, "Salud y bienestar". Aunque nuestro software puede tener un impacto positivo en el bienestar emocional y mental de los usuarios al facilitar el análisis y la comprensión de las obras de Brandon Sanderson, es importante destacar que no está diseñado específicamente para abordar problemas de salud o promover prácticas saludables.

Resumiendo, nuestro proyecto de desarrollo de software para el análisis de redes y el renderizado de gráficos en la web tiene un impacto positivo en la educación y la industria informática, contribuyendo a los ODS 4 y 9, .[Ed]ucación de calidad.[e]Industria, innovación e infraestructuras", respectivamente. Sin embargo, también reconocemos que nuestro proyecto no cumple con otros objetivos, como el ODS 1, "Fin de la pobreza", el ODS 2, "Hambre cero", y el ODS 3, "Salud y bienestar". Es importante que otros proyectos y esfuerzos se centren en abordar directamente estos desafíos y trabajar hacia un futuro más sostenible y equitativo.