Facultad de Ingeniería, Universidad de la República

Visualización y análisis de interacciones por medio de mensajes en entornos educativos

Autora:
Leticia Errandonea Mattos

Tutora:
Dra. Ing. Libertad Tansini

Usuario responsable:
Ing. Antonio López Arredondo

Informe de Proyecto de Grado presentado al Tribunal Evaluador como requisito de graduación de la carrera Ingeniería en Computación

Uruguay, 4 de noviembre de 2021

Resumen

El presente proyecto se desarrolla en el contexto de la institución educativa EviMed y su plataforma de aprendizaje RedEMC. Hasta la fecha esta plataforma cuenta con más de ciento treinta cursos y cuarenta y cinco mil personas distribuidas en veintisiete países, siendo los foros en la misma la principal vía de comunicación entre los usuarios. El principal objetivo de este proyecto es, entonces, proveer una forma de visualizar las interacciones de los usuarios a través de mensajes en los foros, con el fin de analizar el relacionamiento entre ellos.

Considerando los requerimientos iniciales, se describe un análisis de herramientas, así como de los datos provistos para el diseño y desarrollo de la solución. La misma comprende la elección de la base de datos orientada a grafos Neo4j, el modelado de la realidad, la migración de datos del modelo relacional al de grafos, y el desarrollo de consultas que permitan extraer información usando distintos criterios. Por otro lado, se investiga y propone una alternativa visual para la generación de consultas a la base de datos, sin necesidad de contar con conocimientos técnicos previos. Finalmente, se provee una forma gráfica de analizar los resultados obtenidos, que permite un análisis tanto cuantitativo como cualitativo de las interacciones.

Palabras clave: Bases de datos de grafos, Cypher, Neo4j, EviMed, RedEMC.

Tabla de Contenidos

Glosario

API En inglés *Application programming interface* (interfaz de programación de aplicaciones), es una capa de abstracción ofrecida por un programa o biblioteca, que le permite a otro programa comunicarse con el mismo.

Base de datos de grafos Base de datos que utiliza un modelo basado en teoría de grafos. Optimiza operaciones típicas de grafos, permitiendo atravesar el mismo sin que la *performance* se degrade.

Base de datos relacional Base de datos que usa el modelo relacional. Este es un modelo que prioriza describir cómo cierto conjunto de datos debe ser mostrado al usuario por sobre cómo es almacenado entonces, los datos se llevan a relaciones y cada relación es un conjunto de datos. Cada fila es un conjunto de campos, siendo cada campo el valor de un atributo que representa la realidad.

Cypher Lenguaje de consultas utilizado por Neo4j.

Driver Software que permite la conexión entre una aplicación y la base de datos. Es específico para el par lenguaje - base de datos.

Endpoint URL de una API que responde a una petición específica. Se utiliza uno para la autenticación y otro para consultas.

EviMed Institución educativa uruguaya fundada en 2004 en el marco de la cual se realiza el presente proyecto.

Grafo Formado por vértices y aristas. Los vértices representan objetos distintos, mientras que las aristas conectan (relacionan) estos objetos. Ambos pueden tener propiedades.

Linter Herramienta de análisis de código estático para detectar patrones que puedan causar problemas.

Neo4j Implementación de base de datos de grafos muy utilizada desde sus inicios.

RedEMC Red de Educación Médica Continua, plataforma de aprendizaje de EviMed. Brinda cursos en línea para profesionales de la salud de diversas especialidades. Desde sus comienzos hasta la fecha ha ofrecido ciento treinta cursos a más de cuarenta y cinco mil personas distribuidas en veintisiete países.

Script Programa relativamente simple que ejecuta una serie de comandos. Utilizado para automatizar tareas repetitivas, como la carga de datos semanalmente.

Servidor Conjunto de computadoras que atiende peticiones de clientes y le retorna una respuesta. También se le llama *servidor* al programa que se ejecuta en estas computadoras y es responsable de efectivamente atender las peticiones.

1. Introducción

1.1. Motivación

A medida que el acceso a la tecnología se ha universalizado, han surgido nuevas propuestas educativas que se apoyan en el uso de computadoras. En ese marco, el aprendizaje a distancia se ha popularizado, siendo cada vez más frecuente realizar cursos a demanda (*on demand*) desde el hogar. Algunos ejemplos de propuestas educativas son los siguientes:

- **Entorno Virtual de Aprendizaje** [13]: utilizado por la Universidad de la República para compartir material, realizar evaluaciones, y posibilitar el intercambio entre docentes y estudiantes.
- **Plan Ceibal** [34]: disponible para estudiantes de Uruguay desde niños de educación inicial hasta adolescentes de bachillerato.
- **Stanford Online** [24]: provisto por la Universidad de Stanford para ofrecer sus cursos en línea.
- **Prisma Live** [37]: plataforma educativa diseñada para cursos en línea para niños en edad escolar de Estados Unidos.

Habitualmente, era el docente quien estaba atento a detectar estudiantes que participaran poco, parecieran atrasados respecto al programa, o simplemente no asistieran a clases. Con el surgimiento de cursos a distancia en plataformas educativas, estos datos pueden ser tomados directamente por el sistema y accedidos posteriormente haciendo uso de consultas.

EviMed [15] es una institución educativa uruguaya, que trabaja en la educación médica continua para profesionales y equipos de salud con el objetivo de colaborar y promover la mejora en la atención médica, haciendo uso de conocimiento nuevo. Dado que tanto sus docentes como estudiantes están distribuidos en distintos países de América Latina, cuenta con una plataforma llamada **RedEMC** (Red de Educación Médica Continua), que provee sus cursos (ciento treinta desde 2018) para diversas especialidades médicas. Estos cursos, además de contar con material gráfico y escrito para introducir la temática correspondiente, incluyen foros de discusión. En los foros los estudiantes y docentes intercambian opiniones y se realizan consultas, basados tanto en los conocimientos del curso como en su propia experiencia.

EviMed ha decido almacenar información sobre la participación de sus estudiantes y docentes, a fin de poder analizar el funcionamiento de los cursos,

más allá de las posibles encuestas u opiniones. Fue así que comenzó a guardar datos de qué mensajes de su foro fueron vistos, qué respuestas tuvo, cómo fueron las valoraciones, quiénes realizaron esas acciones, et cetera. El problema con este tipo de datos es que si bien son relativamente fáciles de guardar, puede tornarse complejo extraerlos de forma que su análisis sea útil.

La motivación de este proyecto es entonces proveer una forma de analizar a través de la visualización de estos datos que en los últimos años **EviMed** ha recogido pero no existe una manera eficiente de obtener porque la tecnología usada hasta el momento (base de datos relacional) no es suficiente para tales fines.

Las bases de datos relacionales son muy buenas para almacenar y acceder información sobre entidades, pero no para evaluar cómo se relacionan personas a través de mensajes. Las bases de datos de grafos fueron creadas especialmente para este tipo de casos, con la promesa de no solo poder obtener estos datos sino hacerlo de forma eficiente. Por esto último, se decide proponer un proyecto de grado que investigue las herramientas para visualización y análisis de los datos mencionados.

1.2. Objetivos

Es importante señalar que el relevamiento de herramientas se realizó de forma simultánea al análisis de requerimientos.

Dado que **EviMed** ya cuenta con gran cantidad de datos respecto al comportamiento de sus usuarios[1] en la plataforma **RedEMC**, el objetivo principal de este proyecto es proveer una forma de hacer uso de estos datos para su análisis. De esta manera, el equipo de **EviMed** contaría con la información necesaria para evaluar la actual experiencia de aprendizaje. Es importante destacar que, en etapas tempranas el proyeco tuvo un fuerte componente exploratorio con el fin de evaluar qué información sería útil y de qué manera usarla.

Una vez fueron entendidas las necesidades de **EviMed**, surgieron nuevos objetivos. El primero de ellos es analizar cómo modelar la realidad en una base de datos de grafos e investigar cómo exportar datos del modelo relacional a grafos, para luego encontrar las consultas correctas para obtener la información requerida. Estos datos es clave que sean mostrados de una forma que sea posible realizar un

[1] A lo largo del informe se utilizará el término *usuario* para hacer referencia a estudiantes y docentes. De ser necesario diferenciarlos, se hará explícito, haciendo uso de *estudiantes* y *docentes* según corresponda.

análisis valioso rápidamente. Por otro lado, es de interés que el prototipo desarrollado pueda ser integrado con la plataforma ya existente con el menor esfuerzo y costo posible.

Los objetivos, entonces se pueden resumir de la siguiente manera:

- Relevar necesidades de **EviMed** en cuanto a la información sobre el intercambio de mensajes.
- Representar información en una base de datos de grafos.
- Cargar datos de la base relacional a la de grafos.
- Proveer una forma simple de acceder a los datos que permita filtrar los resultados.
- Mostrar los resultados de forma que sea fácil de analizar.
- Facilitar la futura integración con **RedEMC**.

1.3. Resultados

Los principales resultados obtenidos se pueden resumir en:

- Análisis de la realidad y necesidades de **EviMed**, luego utilizado para la realización de un modelo que está representado en la base de datos.
- Investigación de técnicas de pasaje de información de bases de datos relacionales a la base de datos orientada a grafos **Neo4j**.
- Investigación e implementación del uso de base de datos de grafos para obtener información en el ámbito educativo.
- Implementación motor de traducción de filtros seleccionables a consultas en **Cypher**[2], lo que permite a usuarios sin conocimientos técnicos específicos de esta tecnología obtener grafos refinados de acuerdo a sus necesidades. Esto demuestra ser especialmente valioso para aquellas consultas que requieren del uso de conocimientos avanzados, como por ejemplo nodos virtuales.
- Implementación de visualización del grafo resultado, que provee información general a través del uso de colores y tamaños distintos en sus elementos, lo que permite realizar un primer análisis cualitativo sin necesidad de interacción. Posteriormente, al interactuar con los elementos se obtienen datos extra sobre los distintos componentes.
- Prototipo incluyendo posibilidad de integración en la plataforma actual, haciendo este último paso relativamente sencillo para el **EviMed**.

[2]Lenguaje de consultas utilizado por **Neo4j**

- Redacción de un manual de usuario, que puede ser utilizado para ayudar al equipo de operaciones a hacer uso de la aplicación rápidamente.

1.4. Organización del documento

A continuación se describe brevemente la forma en la que esta organizado el documento.

El capítulo 2 presenta el contexto del proyecto. Esto incluye información sobre **EviMed** y su realidad, así como conceptos teóricos que serán utilizados a lo largo del informe.

Posteriormente, en el capítulo 3 se describe la metodología de trabajo utilizada durante el desarrollo del proyecto, con especial énfasis en la interacción con **EviMed**. Además, se comenta la organización del equipo para cumplir con los objetivos intermedios.

El capítulo 4 explica el problema a resolver y se detallan los requerimientos finales.

Luego, durante el capítulo 5 se realiza un análisis de las herramientas necesarias, junto con la fundamentación de la elección final.

A continuación, en el capítulo 6 son definidos la arquitectura y diseños elegidos. Además, se describen los datos, considerando el volumen, formato y características, así como el modelado elegido y consultas base.

El capítulo 8 se presentan los resultados obtenidos en el proyecto.

En el capítulo 7 se exponen las pruebas realizadas para validar tanto el ajuste del prototipo a los requerimientos como la correctitud de la solución alcanzada.

Finalmente, en el capítulo 9 se resumen las conclusiones finales, y se proponen líneas de investigación futuras.

Se cuenta con dos anexos que proveen información útil respecto a la aplicación y la API. El primero de ellos cuenta con dos partes: una enfocada en la configuración, mientras la otra lo hace en el uso de la aplicación.

2. Contexto

En este capítulo se presentan conceptos clave que serán utilizados a lo largo del informe. Por otro lado, se introduce la institución educativa **EviMed**, mencionando su objetivo y funcionamiento.

2.1. Grafos

En esta sección se definen los conceptos relativos a los grafos que serán utilizados a lo largo del presente informe.

A continuación se definen conceptos necesarios, extraídos del libro *Graph theory* [12]:

- Sea V un conjunto vacío no finito. Un **grafo dirigido** (o **digrafo**) **G sobre V** está formado por los elementos de V, llamados *vértices* o *nodos* de G, y un subconjunto E de VxV, conocido como las *aristas* (dirigidas) (o *arcos*) de G. Si $a, b \in V$ y $(a, b) \in E$, entonces existe una arista de a a b. El vértice a es el *origen* o *fuente* de la arista, y b es el *término*, o *vértice terminal*, y decimos que b *es adyacente desde* a y que a *es adyacente hacia* b. Además, si $a \neq b$, entonces $(a, b) \neq (b, a)$. Una arista de la forma (a, a) es un *lazo* (en a).
- Un **grafo denso** es aquel en el que la cantidad de aristas se acerca a la cantidad máxima posible. Esto es considerando que no se admiten múltiples aristas entre los mismos nodos (solo dos en el caso de grafos dirigidos).
- Un **grafo disperso** es aquel en el que existen muy pocas aristas.

2.2. EviMed

Fundada en 2004, **EviMed** [15] es una institución educativa uruguaya especializada en la educación médica continua para profesionales y equipos de salud. Su objetivo es colaborar en la evolución de la atención médica, a través de la aplicación del nuevo conocimiento científico en el menor tiempo posible. Para ello se basa en tecnología de la información y comunicación, posibilitando la realización de cursos *online*.

Inicialmente, contaba con cursos para Uruguay. Luego, en 2012 se consolida como una opción para tomar cursos en toda América Latina, con especial foco en países hispanohablantes. Un año después, se comenzó a trabajar en cursos en

portugués e inglés. Es valioso destacar que, si bien el objetivo era América Latina, se acercaron profesionales de otras zonas del mundo. Este crecimiento hacia distintas zonas geográficas permitió ampliar el equipo docente a médicos de otros países, lo que fue un aporte a los cursos.

Por otro lado, EviMed cuenta con respaldo de importantes sociedades científicas, lo que hace que los certificados emitidos al final de los cursos sean de autoridades académicas.

Desde 2018 cuenta con su plataforma virtual de aprendizaje a distancia llamada **RedEMC** (Red de Educación Médica Continua [39]), en la que se proveen permanentemente cursos para profesionales de la salud de diversas especialidades. En ese sentido, hasta la fecha ha ofrecido ciento treinta cursos para más de cuarenta y cinco mil personas en veintisiete países.

En la elaboración y dictado de cursos participan equipos multidisciplinarios, liderados por coordinadores académicos con vasta experiencia, así como fuertes lazos con la universidad, lo que los hace referentes.

Los cursos siguen un modelo pedagógico propio de **RedEMC**, que consta de cuatro pilares:

1. **Aprendizaje tácito**: se cuenta con foros de discusión donde se intercambian conocimientos aprendidos a lo largo de la práctica de la profesión.
2. **Aprendizaje por descubrimiento**: se proponen ejercicios de simulación clínica interactiva, que le permiten a los participantes tomar decisiones respecto a los siguientes pasos a realizar en un hipotético caso clínico. Además, están disponibles ejercicios de autoevaluación, que cumplen con el objetivo de detectar conocimientos aprendidos así como los puntos que requieren de mayor dedicación.
3. **Aprendizaje situado**: se fomenta la aplicación en el contexto laboral o social de los conocimientos generados en el curso. Estas aplicaciones pueden ser tanto intervenciones en la clínica como cambios en protocolos o guías.
4. **Comunidades de práctica**: un sello distintivo de **EviMed**. El aprendizaje se considera un proceso de participación social, por lo que se propician estos encuentros. Además de los foros donde se pueden plantear preguntas y generar discusiones, se cuenta con una *Wiki* para la generación de material de forma colaborativa. Por otro lado, en los cursos se promueve la presentación de casos reales, lo que aporta un intercambio sobre realidades y opiniones.

Resulta importante mencionar que los cursos no solo son brindados a distancia, sino que es posible realizarlos en el horario que le sea más conveniente al

participante. El intercambio mencionado anteriormente funciona entonces de forma asíncrona, mediante los foros, *Wiki*, y comentarios en los recursos (videos, imágenes, documentos). Dada su gran relevancia, los mensajes en los foros cuentan con algunas funcionalidades tales como ser respondidos o valorados (indicando "me gusta").

Por fuera de los cursos, **RedEMC** se presenta como una red social, en la que un usuario puede "seguir" a otro, creando comunidades que se mantienen a lo largo del tiempo. Actualmente, esta información es presentada en un grafo a cada usuario, en el que puede ver su comunidad.

De esta forma, **RedEMC** es a la vez una plataforma de aprendizaje y una red social. Esta innovadora combinación, en conjunto con su equipo y el modelo pedagógico, convierte a **EviMed** en una empresa líder en el ámbito.

3. Metodología de trabajo

En este capítulo se presenta el proceso de trabajo utilizado para obtener los requerimientos y validar los progresos con **EviMed**. A su vez, se explica de qué forma el equipo se organizó para llevar un registro de las tareas pendientes y su avance.

3.1. Reuniones

A medida que fue necesario y posible, se realizaron reuniones con Ing. Antonio López Arredondo, director de **EviMed**. Habitualmente, concurría acompañado por parte de su equipo de desarrollo. Las primeras fueron de forma presencial, pero luego debido a la situación de pandemia, se realizaron a través de videollamadas.

En períodos en los que no hubo reuniones, fue común la comunicación por email, ya fuera con representantes de **EviMed** o la tutora.

3.1.1. Octubre de 2019

La primera reunión se realizó en octubre de 2019. De esta reunión también participaron estudiantes que trabajarían en otro proyecto. El objetivo fue conocer a **EviMed**, sus representantes, su contexto, y cuáles eran sus siguientes metas.

De lo que se habló, se decidió trabajar en identificar cómo se relacionan los estudiantes y docentes, usando una base de datos de grafos y mostrando los resultados de forma amigable. Se resolvió investigar distintas bases de datos de grafos y, a la siguiente reunión, evaluar propuestas concretas para los proyectos.

3.1.2. Diciembre de 2019

En esta reunión se discutieron detalles respecto a la información de la que **EviMed** disponía de los usuarios, cursos, recursos, y comentarios. En base a ello, surgieron dos proyectos: uno centrado en los usuarios y las redes formadas por ellos directamente (*usuario A sigue a usuario B*); el otro centrado en cómo se relacionan los usuarios a través de los mensajes.

Por otro lado, en cuanto a herramientas a utilizar, se decidió que **Neo4j**[30] fuera la base de datos de grafos.

Finalmente, **EviMed** se comprometió a compartir un usuario de prueba para que se tomara más contexto respecto a cómo los usuarios usan **RedEMC**.

A partir de este momento, quedó definido que el presente proyecto tendría como objetivo mostrar las interacciones entre usuarios a través de los mensajes, proporcionando formas de filtrar y agrupar.

3.1.3. Agosto de 2020

El objetivo de esta reunión fue mostrar los avances del proyecto, así como obtener *feedback* y sugerencias respecto a en qué trabajar a continuación.

En esta fecha, el producto tenía una arquitectura definida, y contaba con las funcionalidades:

- mostrar un grafo con usuarios, mensajes, y relaciones entre ellos
- filtrar sobre características de usuarios - país y especialidad
- filtrar sobre el tipo de relación entre el usuario y el mensaje
- agrupar usuarios - país o especialidad

En la reunión se recibieron sugerencias e ideas sobre qué podía ser agregado o modificado:

- agregar el concepto de *curso*. Los mensajes *tienen* un curso y los usuarios *toman* o *dictan* cursos
- agregar la propiedad *género* a los usuarios
- agregar propiedades a los mensajes
- agregar que aquellos usuarios que *toman* cursos tienen un *estado*
- implementar posibilidad de filtrar por *id* de persona o mensaje
- implementar posibilidad de filtrar según rol del usuario
- implementar agrupar por estado en el curso
- realizar cambios en el diseño del grafo mostrado

Posterior a la reunión se realizó el pedido de datos reales de **RedEMC** para validar el funcionamiento del producto para grandes cargas de datos.

3.1.4. Marzo de 2021

Se convocó una nueva reunión con los objetivos de recibir los datos reales así como mostrar los últimos avances.

El *feedback* fue positivo y solo se discutieron ligeras modificaciones a la forma de mostrar algunos datos de los usuarios.

Finalmente, se acordó exactamente qué datos y en qué plazo serían enviados.

3.1.5. Abril de 2021

Una vez obtenidos los datos y realizadas las modificaciones necesarias, se agendó una nueva reunión con la finalidad de discutir los próximos pasos para poder integrar el proyecto a **RedEMC**. La forma de realizar la integración estaría a cargo del equipo de **EviMed**.

Por otro lado, debido a que los datos no podían ser de público acceso, se decidió agregar una capa simple de autenticación a la aplicación. De esta forma, solo usuarios con credenciales validadas por **EviMed** podrían tener acceso a la aplicación y, como consecuencia, sus datos.

Otro importante aspecto de los datos era cómo actualizarlos. En ese sentido, se acordó agregar un *script* que permitiera actualizar los datos semanalmente de forma automática.

3.1.6. Julio de 2021

Luego de la reunión, el equipo se encargó de poner la aplicación completa en funcionamiento en el servidor. Esto implicó la implantación de la misma, así como el desarrollo del *script* para actualizar los datos.

Por otro lado, se avanzó en el desarrollo de la autenticación aunque no se incorporó a la aplicación alojada en el servidor. Esto se debió a que no era posible el uso del *endpoint* provisto por **EviMed** con los datos disponibles. En particular, el *endpoint* esperaba la contraseña del usuario encriptada, pero los métodos propuestos (*MD5* o *DES*) resultaban en mensajes de error.

El objetivo de la reunión fue entonces realizar pruebas para integrar la aplicación en **RedEMC**, embebiéndola en una nueva página. Desafortunadamente, recién en este momento se detectó que el servidor no contaba con la configuración *SSL*, por lo que Wordpress (sistema que **RedEMC** usa) no permitía embeber la aplicación.

Se acordó dividir la tarea de configurar *SSL* en el servidor (**EviMed**) y aplicación (equipo). Una vez realizada, sería responsabilidad del equipo de

EviMed integrar la aplicación a RedEMC.

Con respecto a la autenticación, su integración quedó sujeta a que las instrucciones para el correcto uso del *endpoint* creado para tales fines le fueran provistas al equipo.

3.2. Proceso de desarrollo

Para el desarrollo del proyecto se utilizó un proceso iterativo incremental. De esta forma, a medida que se presentaron avances, se obtuvo *feedback* y se trabajó tanto en mejoras como en nuevas funcionalidades. [41]

Dado que el código fuente sería versionado, se decidió utilizar la herramienta Git [18] y tenerlo almacenado en GitHub [20].

Inicialmente, no se contaba con un proceso de seguimiento de tareas, y simplemente se tomaba nota de qué debía realizarse y se realizaba. A medida que el alcance estuvo definido, se hizo necesario contar con una organización de la tarea, principalmente dividiendo el trabajo restante de modo que fuera posible hacer un seguimiento del estado de las distintas funcionalidades.

Debido a que el código del proyecto estaba en GitHub, se decidió crear un *issue* por cada funcionalidad faltante o *bug*, agregando una etiqueta para indicar prioridad. Posteriormente, se hizo uso de GitHub projects [2], lo que permitió tener un tablero de estilo Kanban para cada conjunto de funcionalidades que restaba finalizar.

Este tablero Kanban se organizaba en cuatro columnas:

- **Pendiente**: *issues* creados que debían ser atendidos.
- **Próximo**: *issues* que eran prioritarios para terminar una cierta funcionalidad.
- **En progreso**: *issues* en los que se estaba trabajando o que requerían de alguna interacción por parte del equipo de **EviMed**.
- **Hecho**: *issues* correspondientes a tareas ya finalizadas.

Si bien la organización a lo largo del proyecto fue prácticamente la misma, no siempre se utilizó el mismo tablero. Esto se debió a que se decidió que cada uno contemplara distintas fases del proyecto. Finalmente, se usaron tres:

- **Inicial**: versión inicial de todo el conjunto de funcionalidades que cumplían con el alcance inicial y tomaba en cuenta el *feedback* de agosto de 2020.

- **Completo**: versión final, con los cambios necesarios para usar los datos reales, así como aquellos relacionados a la privacidad de datos.
- **Puesta en producción**: tareas necesarias para la correcta integración e implantación del proyecto. Incluía aspectos tales como configuración de servidor, y desarrollo de *script* para cargar los datos.

Respecto a nuevas ideas que pudieran surgir por parte de **EviMed**, tutora, o equipo, se tomó nota y cada una de ellas fue agregada como *issue*. Dependiendo de la relevancia, se le asignó la etiqueta *necesario* o *deseable*. Para la completitud del proyecto, se realizaron todas aquellas indicadas como *necesario* y más adelante en este informe se presentan aquellas que fueron *deseables* pero no se trabajó en ellas.

A modo de que el proyecto fuera visible para las partes interesadas sin necesidad de la presencia del equipo, se decidió tener siempre la aplicación disponible con datos de prueba en un entorno público. Dado que a medida que nuevas funcionalidades eran integradas, ese entorno era actualizado, fue también de utilidad para mostrar avances en las reuniones.

Otro aspecto positivo de contar con ese entorno fue que propició la resolución de problemas frecuentes relacionados a la puesta en producción de una aplicación. Como consecuencia, cuando efectivamente se puso en producción el proyecto, los problemas encontrados fueron únicamente relacionados a la configuración del servidor en el entorno de **EviMed**.

4. Requerimientos

En este capítulo se realiza una descripción del problema que el proyecto pretendía resolver. Una vez finalizada esa parte, se plantea el alcance acordado con **EviMed**.

4.1. Descripción del problema

Dado el contexto de este proyecto, en la presente sección se explica el problema a resolver.

Como se comentó en la sección 2.2, **RedEMC** permite inscribirse a los cursos, acceder a recursos (tanto videos como documentos), realizar tareas, intercambiar opiniones en foros dedicados al tema, y crear redes con otros usuarios.

Los foros tienen un rol clave en la vida de un usuario de un curso de **RedEMC**. Al ser cursos en línea, es importante generar un espacio que permita a los estudiantes interactuar entre ellos así como con los docentes, de la misma manera que lo harían si el curso fuera presencial.

Al comenzar un curso, cada estudiante debe presentarse en el foro de Introducción. Esto permite que los estudiantes se empiecen a conocer aún antes de que haya tareas específicas del curso para realizar. Por otro lado, cada estudiante puede participar en un foro publicando, respondiendo, o indicando que le gusta un comentario. Naturalmente, este tipo de funcionalidades permite que se den discusiones de gran valor en los foros.

Al momento de iniciar el presente proyecto, **EviMed** disponía de la totalidad de la información relativa a los comentarios, aunque el único uso que se le daba era mostrar los comentarios con sus respuestas y cuántos estudiantes habían indicado que les gustaba. Debido a esto era de interés buscar formas de poder usar esos datos para entender las redes entre estudiantes. A través de una mejor comprensión de cómo se relacionan los estudiantes, se pueden elaborar estrategias para potenciar el uso de foros y, de esta manera, lograr el objetivo de capacitar a los profesionales de la salud a la vez que se favorece el desarrollo de redes.

Se decidió que este proyecto investigara cómo se relacionan los usuarios a través de los mensajes, tanto de forma individual como según su país de origen o especialización. Para los estudiantes existe también el concepto de *estado* en el marco de un curso (qué tan avanzado está respecto a las tareas que debe realizar). Como consecuencia, se optó por agrupar por estado, permitiendo analizar cómo se relacionan los estudiantes según qué tan involucrados están en el curso.

4.2. Alcance

De acuerdo a lo convenido, el producto a construir debía permitir analizar de forma visual el comportamiento de los usuarios a través de su relación con los mensajes en el marco de un curso. Además, por seguridad, sería deseable requerir autenticación para acceder a los datos.

La aplicación a realizar tendría, entonces, tres etapas: autenticación, búsqueda y filtrado, y visualización de los datos obtenidos.

4.2.1. Autenticación

La aplicación sería responsable de obtener las credenciales por parte del usuario y enviarlas a una API provista por **EviMed**, que sería quien finalmente validara las mismas, para permitir el acceso a los datos.

4.2.2. Búsqueda y filtrado

Los datos serían accedidos realizando una búsqueda de una entidad (curso, persona, mensaje) y, a su vez, aplicando filtros sobre entidades o relaciones asociadas.

Por curso

La selección del curso debía poder hacerse por ID o nombre. El fundamento detrás de esto es que un mismo curso puede ser dictado en distintos idiomas y, en ese caso, el ID es distinto pero el nombre es el mismo.

Los filtros serían sobre usuarios, mensajes, y la relación entre estos.

Sobre los usuarios se podría elegir rol en el curso (*docente*, *estudiante*), especialidad, género, país de origen, años de ejercicio de la profesión, y estado en el curso (*On track*, *Audit*, *Behind*, *Out*).

En el caso de los mensajes, los filtros serían según cantidad de "me gusta", respuestas, impresiones, y semana del curso a la que pertenece.

Finalmente, se podría elegir qué relaciones entre usuarios y mensajes considerar a la hora de extraer datos y mostrarlos. Estas relaciones son tres:

publicó, *indicó que le "gusta"*, o *respondió* un mensaje.

Agrupamiento

Al momento de analizar las interacciones de un curso, sería útil agrupar los usuarios según sus características y así obtener información que permitiera otro tipo de análisis.

Se agruparía por país, especialidad, o estado en el curso. En todos los casos serían aplicables los demás filtros sobre usuarios (género o años de ejercicio), así como sobre las relaciones.

En ningún caso sería posible filtrar por la característica elegida como criterio de agrupación, dado que la consulta carecería de valor. Tampoco sería posible filtrar por rol al agrupar por estado, ya que los únicos usuarios que tienen un estado son los que tienen el rol de *estudiante*.

Por persona

Debido a que es frecuente que los usuarios tomen más de un curso, también sería posible indicar el ID de un usuario y, de esa manera, ver sus cursos y mensajes.

En este caso también sería posible filtrar los mensajes (usando las mismas opciones que para las consultas por curso), así como los usuarios con los cuales se relacionó a través de dichos mensajes.

Nuevamente, se podría elegir las relaciones entre el usuario identificado por el ID y los mensajes.

Por mensaje

Así como sería posible analizar las interacciones entre un usuario y sus mensajes, también estaría la opción de elegir un mensaje por ID y visualizar qué usuarios interactuaron a través de él.

Se podría filtrar a los usuarios por las mismas características que en las consultas por curso.

4.2.3. Visualización

Sin agrupar

En este caso sería conveniente mostrar los datos de cada usuario (mensaje o curso). Además de los datos usados para filtrar, sería de interés tener acceso a ID y nombre.

Agrupado

Ya que no todos los grupos incluirían a la misma cantidad de usuarios, sería útil que el tamaño de los nodos fuera representativo del tamaño del grupo. Del mismo modo sucedería con el grosor de las aristas y la cantidad de relaciones.

5. Relevamiento de herramientas

En este capítulo se comenta el ambiente de **EviMed** y se presentan las herramientas utilizadas, justificando su elección.

En todos los casos la decisión estuvo basada en los siguientes criterios:

- comunidad
- documentación
- posibilidad de ser usada en el ambiente de **EviMed**
- existencia de tutoriales

5.1. Ambiente de EviMed

Al momento de poner en producción una plataforma, es necesario contar con un servidor donde estén disponibles los recursos necesarios para que los usuarios puedan acceder a la misma. Actualmente, es frecuente el uso de proveedores de servidores virtuales en la nube, que suelen garantizar, entre otros, alta disponibilidad [1], variedad de versiones y distribuciones de sistemas operativos, y posibilidad de aumentar recursos de la máquina fácilmente.

En el caso de **EviMed**, se hace uso del servicio **EC2** de Amazon Web Services (popularmente conocido como **AWS** [4]). Este servicio es plenamente configurable en términos de procesador, almacenamiento, conectividad, y sistema operativo, lo que hace que los servidores puedan llevar a cabo distintas tareas, desde *machine learning* hace publicar una página web.

5.2. Base de datos

El uso de una base de datos de grafos fue una condición del proyecto. De cualquier manera, fue necesario realizar una investigación al inicio para decidir qué implementación era la adecuada.

En esta sección se introducen las bases de datos de grafos, así como las implementaciones consideradas y los detalles de la elegida.

[1] La máquina en la que es ejecutado el código que sirve los recursos no sufre fallas ni problemas de conectividad.

5.2.1. Bases de datos de grafos

Las bases de datos permiten representar objetos y las relaciones entre ellos. Habitualmente, las consultas están centradas en los objetos y solo se usan las relaciones para complementar la información. Inclusive, es posible representar un grafo en una base de datos relacional.

Suponiendo que se cuenta con una tabla de *personas* y otra que relaciona los pares (con la semántica de que se conocen entre ellas), la consulta para probar la teoría de los *Seis grados de separación* [40] implica realizar aproximadamente 12 *JOIN*. Aún si la base de datos hubiera sido optimizada con índices, la consulta es costosa.

La dificultad en la resolución de este problema en una base de datos relacional radica en que la representación de las relaciones es deficiente. Esto es poco notorio en casos en los que solamente se quiere obtener información simple (como ser "¿a quién conoce esta persona?"), pero debido a que el desempeño se degrada rápidamente, información que requiera atravesar el grafo es imposible de obtener.

Para casos como el presentado anteriormente, las bases de datos de grafos son ideales. En ellas la relación entre objetos es el centro y, por lo tanto, poseen formas de navegar a través de ellas de forma inmediata. Esto se debe a que cuando un objeto "apunta" a otro, este guarda una referencia de la dirección de memoria en la que se encuentra el nodo. Como consecuencia, acceder de uno al otro es inmediato ya que no requiere traducir la referencia a una dirección de memoria.

5.2.2. Elección de la implementación

De acuerdo al *ranking* realizado por la web DB-Engines [10], la base de datos de grafos con mayor popularidad es **Neo4j** [30] (ver 5.1). Este *ranking* analiza los siguientes seis parámetros (entre paréntesis las páginas web utilizadas para obtener los datos):

- cantidad de menciones en páginas web (*Google*, *Bing*)
- interés (*Google Trends*)
- frecuencia de discusiones técnicas (*Stack Overflow*, *DBA Stack Exchange*)
- ofertas laborales (*Indeed*, *Simple Hired*)
- menciones en perfiles laborales (*LinkedIn*)

- relevancia en redes sociales (*Twitter*)

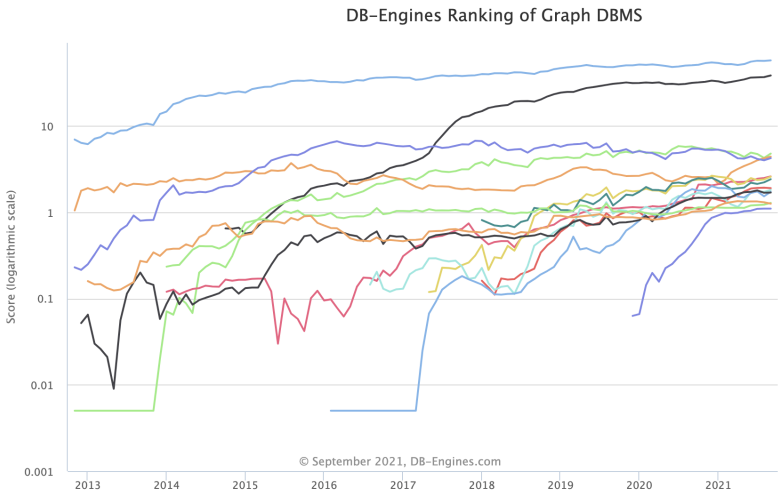

Figura 5.1: Gráfico de popularidad de las bases de datos de grafos. En azul claro en la parte superior se muestra **Neo4j**. [23]

Neo4j cuenta con documentación que guía al usuario desde cómo diseñar la estructura hasta cómo optimizar consultas. A través de una gran comunidad, se han desarrollado *drivers* para distintos lenguajes. Además, existen extensiones que resuelven problemas comunes y simplifican la resolución de otros.

De forma gratuita se puede instalar **Neo4j Desktop** [31] que permite en un ambiente local tener múltiples bases de datos, instalar extensiones, realizar consultas, y visualizar resultados en un grafo o tabla.

La puesta en producción de una base de datos **Neo4j** en **AWS** es simple. En el *marketplace* de AWS está disponible una imagen oficial de **Neo4j** que provee lo necesario.

Por último, brinda la posibilidad de realizar cursos gratuitos a través de su plataforma **GraphAcademy** [22].

También se analizaron Amazon Neptune [3] y Titan [43]. En ambos casos se detectó que la comunidad entorno a la tecnología era significativamente menor a la de **Neo4j** y, como consecuencia, las herramientas disponibles eran menos.

Considerando que ni **EviMed** ni el equipo tenían conocimientos previos sobre bases de datos de grafos, resultaba muy riesgoso optar por una tecnología con una comunidad reducida, por lo que se decidió usar **Neo4j**.

5.2.3. Neo4j

Debido a la inexperiencia en la herramienta por parte del equipo, se decidió tomar el curso *Introduction to Neo4j 4.0*. El mismo es provisto por GraphAcademy de forma gratuita y contiene una introducción a **Neo4j**, consultas en **Cypher**, creación de nodos y aristas, uso de índices, e importación de datos.

Cypher

El lenguaje de consultas de **Neo4j** es **Cypher** [8], y es utilizado para realizar todas las acciones sobre una base de datos de este tipo.

Está inspirada en SQL, y tiene como objetivo que las consultas se puedan realizar de forma intuitiva, así como que al escribir una consulta se indique **qué** datos se desean y no **cómo** obtenerlos.

Además de proporcionar formas de crear, editar, obtener, y borrar nodos y aristas, es posible agregar procedimientos y funciones.

Existen librerías que proveen procedimientos para resolver problemas frecuentes en bases de datos de grafos. Para este proyecto fue de interés utilizar **APOC** [5].

APOC

Awesome Procedures on Cypher (APOC) es la librería de Neo4j más utilizada, ya que cuenta con más de cuatrocientos cincuenta procedimientos y funciones para conversiones de texto, creación de nodos y aristas virtuales, cargar datos a partir de un archivo *JSON*, et cétera.

En el caso de este proyecto, la librería debió ser utilizada para realizar consultas que requerían de agrupar datos.

5.3. Servidor

En caso del servidor, en la elección de qué tecnología usar fue necesario tomar en cuenta los *drivers* existentes para la base de datos. De acuerdo a la guía para desarrolladores [32], si bien hay *drivers* disponibles para prácticamente todos los lenguajes o *frameworks* más usados, solo cinco de ellos son oficiales (*.Net* [1], *Java* [11], *JavaScript* [27], *Go* [42], y *Python* [45]).

Debido a que el equipo contaba con experiencia previa en desarrollo de servidores web en JavaScript, se decidió usar ese lenguaje.

Para que un servidor escrito en JavaScript pueda ejecutarse, se debe utilizar Node.js [33], un entorno de ejecución enfocado en eventos asíncronos que fue diseñado para facilitar el manejo de aplicaciones.

Para resolver los problemas comunes de un servidor web (tales como crear una respuesta HTTP correcta) se decidió utilizar un *framework*. El caso de uso de esta aplicación era simple, sin necesidad de usar modelos de entidades, capas de lógica de negocio, ni interacción con servicios externos. Teniendo en cuenta lo mencionado, se optó por usar el *framework* más liviano, que solo proporcionara las herramientas esenciales para el desarrollo y correcto funcionamiento del servidor. Este *framework* fue Express [16].

5.3.1. API

El servidor contaría con dos *endpoints*: uno para iniciar sesión y otro para realizar consultas. Principalmente en el caso del *enpoint* para obtener los datos, se preveía que se debería recibir distintos *query params* y, por lo tanto, se entendió que era necesario realizar validaciones respecto a qué valores se aceptaría en cada parámetro.

En JavaScript, lo habitual para este caso de uso es instalar la librería Joi [29], que permite definir esquemas y realizar validaciones de la *request* (entre otras funcionalidades). Las validaciones integradas fueron referentes al tipo de datos admitido (por ejemplo, *number* para cantidad de impresiones, y *string* para país), así como opciones disponibles para casos como rol y criterio de agrupación.

5.4. Cliente

Finalmente, restaba tomar la decisión de cómo implementar el cliente. Dado que se trata de una aplicación web, las opciones eran: usar un *framework*, o implementar usando casi exclusivamente la API nativa de la web.

Analizando los requerimientos, se destacaba la necesidad de contar con una herramienta que permitiera dibujar el grafo resultado de la consulta. Si bien también era necesario tener campos para seleccionar filtros, esto realiza utilizando elementos HTML, por lo que no presentaba dificultades.

Luego de investigar las opciones para mostrar un grafo en un aplicación web, las opciones se redujeron a **Popoto.js** [36] y **D3.js** [9].

5.4.1. Popoto.js

Popoto.js es una librería creada especialmente para escribir consultas en bases de datos **Neo4j** y mostrar el grafo. También cuenta con algunas funcionalidades extra como la construcción de consultas simples a partir de la selección de un nodo.

Inicialmente, Popoto.js era una buena opción ya que tenía implementadas parte de las funcionalidades requeridas para este proyecto. El uso de esta librería permitía centrar el esfuerzo en resolver problemas específicos de la realidad de **EviMed**. De cualquier manera, Popoto.js contaba con la desventaja de que se conecta directamente con la base de datos. Como consecuencia, las credenciales para conectarse a la base de datos es de fácil acceso, simplemente mirando el código fuente de la página que se carga en el navegador. Esto resultaba inaceptable, ya que **EviMed** había manifestado su interés de incorporar los resultados de este proyecto en su propia aplicación web.

5.4.2. D3.js

Esta librería que debe su nombre a *Data-Driven Documents* (documentos orientados a datos) proporciona una amplia gama de opciones para presentar datos. Una de las posibilidades es lo que sus creadores llaman *force*, lo que son grafos.

Dado que D3.js no era una librería pensada para mostrar grafos de **Neo4j**, no contaba con ninguna de las funcionalidades extra que Popoto.js sí proporcionaba. El uso de D3.js implicó que el equipo se debió hacer responsable del desarrollo de las funciones necesarias para convertir los datos en un grafo que fuera de fácil comprensión por parte de los usuarios.

En el caso de esta librería, el grafo era presentado apenas recibía los datos, posteriormente modificando la ubicación de nodos y aristas para optimizar su visualización en el espacio disponible. Para lograr este funcionamiento resultaba útil contar con un *framework* que simplificara el código que requería ser escrito por el equipo.

Para la elección de este *framework* nuevamente se tuvieron en cuenta los

mismos criterios que fueron presentados al comienzo de esta sección. En ese sentido, las opciones que resultaban más convenientes eran React [38] y Vue.js [44].

En principio, el equipo no tenía una preferencia marcada por una de las dos opciones, por lo que se decidió investigar cómo se usaba D3.js con cada uno de ellos. Luego de búsquedas, se encontró que existían tutoriales completos para React, por lo que este fue finalmente el *framework* elegido.

Una vez estuvo decidido el uso de **React** con **D3.js**, se tomaron los cursos *Introduction to data visualization with d3.js v4* y *Data visualization for React developers*, ambos en la plataforma Frontend Masters [17].

5.5. Calidad

Considerando que era de interés de **EviMed** incorporar el proyecto a su campus **RedEMC**, se buscó que el código de la aplicación fuera mantenible y siguiera convenciones así como mejores prácticas.

En el marco de lo anteriormente mencionado, además de considerar el tamaño de la comunidad de las distintas herramientas y la documentación disponible, se incorporó un *linter*.

Tanto para el servidor como para el cliente, la herramienta elegida fue ESLint [14], usando el conjunto de reglas recomendado. En el caso del cliente, se agregó, además, la configuración recomendada para aplicaciones React, que agrega reglas específicas para dicho *framework*.

La forma de garantizar que en cada nueva funcionalidad se mantuviera la calidad del código fue configurar un GitHub Action [19] que corriera al crear un nuevo *pull request* en el repositorio. En dicho *hook*, se verificaban las reglas y, en caso de que alguna no se cumpliera y el arreglo fuera simple, se creaba automáticamente un *commit* que lo corrigiera.

6. Diseño e implementación de la solución

En este capítulo se presenta la arquitectura y diseño de la solución. A continuación, se describen los datos con los que se trabajó tanto en la fase de desarrollo como en la fase final.

6.1. Arquitectura

En esta sección se describe el diseño de la solución, presentando su arquitectura.

Luego de evaluar los requerimientos, se decidió utilizar una arquitectura cliente-servidor. En la figura 6.1 se presenta un esquema de la arquitectura.

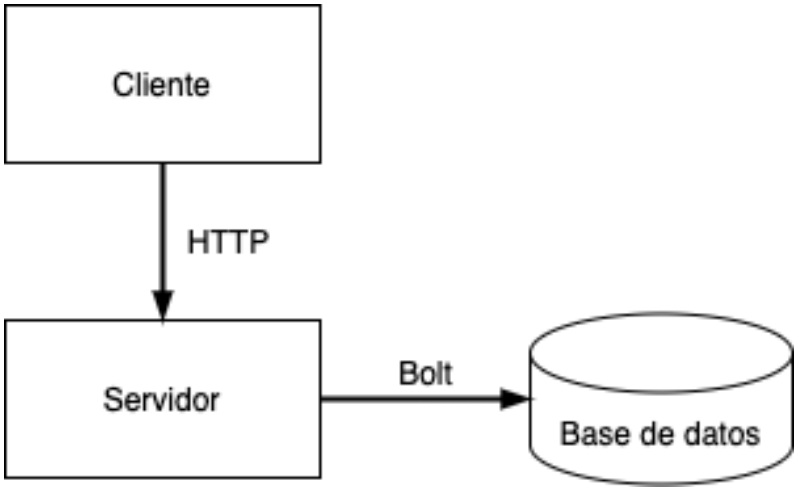

Figura 6.1: Arquitectura de la solución.

6.1.1. Cliente

El cliente es una aplicación web que corre en el navegador de usuario. Las responsabilidades son obtención de credenciales (para autenticar al usuario en el servidor), selección de datos y filtros, y visualización de resultados.

Una vez que el cliente tiene información de qué datos desea obtener el usuario, se los envía al servidor usando *query params* en una *HTTP request*.

A través de una *HTTP response* obtiene los datos en formato *JSON*, los cuales procesa para dibujar un grafo y mostrárselo al usuario.

6.1.2. Servidor

El servidor es el núcleo de la aplicación. Es el responsable de conectarse con el servicio de **EviMed** que autentica usuarios, así como transformar los parámetros de consulta obtenidos del cliente en una consulta **Cypher** válida, que devuelva los datos que el usuario pretendía.

6.1.3. Base de datos

Se destaca que el protocolo usado para la comunicación entre el servidor y la base de datos es **Bolt** [6], ya que es el utilizado por los *drivers* de **Neo4j**.

6.1.4. Puesta en producción

La arquitectura presentada así como el diseño de la solución son flexibles y admiten distintas formas de implantación.

Durante la etapa de desarrollo, se decidió mantener la aplicación disponible en línea, para realizar demostraciones, y posibilitar su acceso a EviMed o tutora según resultara conveniente. Para ello se usó **Neo4j Sandbox** para la base de datos, **Heroku** para el servidor, y **Netlify** para el cliente.

Al momento de hacer la puesta en producción, el equipo de **EviMed** proveyó una única instancia **EC2** en **AWS**. Se decidió que en esa instancia estuvieran disponibles la base de datos en un puerto, y un único servidor en otro. Este último cambio implicó realizar algunos ajustes para que los archivos correspondientes al cliente también fueran servidos por el mismo servidor que la API.

6.2. Diseño

Esta sección define las decisiones tomadas respecto al diseño tanto del servidor como de la aplicación.

6.2.1. Servidor

Como se explicó en la sección anterior, parte de las responsabilidades del servidor es la de realizar la transformación de los parámetros a la consulta. Con ese objetivo, se escribieron funciones específicas de acuerdo al criterio de búsqueda

(curso, persona, mensaje) y si era necesario agrupar.

Las funciones creadas fueron:

- **buildByCourseQuery**: crea la consulta por curso sin agrupar. Toma curso (ya sea nombre o ID) y filtros.
- **buildByPersonIdQuery**: crea la consulta por persona. Toma ID de la persona y filtros.
- **buildByMessageIdQuery**: crea la consulta por persona. Toma ID del mensaje y filtros.
- **buildGroupQuery**: crea la consulta por curso agrupando. Toma curso (ya sea nombre o ID) y filtros.

Para los casos en que se decidiera filtrar según una propiedad fuera menor, igual, o mayor a un valor dado, se necesitó escribir una función específica. Esta convierte los parámetros correspondientes a signo y valor en una sentencia de **Cypher** válida.

Un problema encontrado es que los resultados de la consulta a la base de datos se expresan en forma de tabla. Esto implica que si *Usuario 1* y *Usuario 2* tomaron *Curso A*, se obtienen dos filas, cada una de ellas con el nodo completo correspondiente al *Curso A*. Esto provocaba que las respuestas del servidor al cliente fueran innecesariamente grandes. Para evitar este problema, se implementaron funciones que convierten los resultados en *JSON* mínimos que se le envían al cliente. Estos contienen personas, mensajes, cursos, y relaciones. Cada uno de los nodos y aristas aparece una única vez.

6.2.2. Aplicación

Al momento de diseñar la experiencia de usuario, fue necesario definir una serie de aspectos, incluyendo cuántas vistas distintas tendría la aplicación, qué información estaría siempre disponible, qué datos serían visibles al momento de cargar el grafo y cuáles al interactuar con él. La mayoría de estos aspectos fueron propuestos por el equipo a través de un prototipo y aceptados por **EviMed**.

Dado que la aplicación cuenta con autenticación y que sin ella no se puede realizar ninguna acción, lo intuitivo fue crear una vista inicial para solicitar las credenciales.

Considerando que existían varias características por las que buscar y filtrar, se decidió tener una vista dedicada a esta etapa del uso de la aplicación. Una vez el usuario presionara el botón para realizar la consulta, la aplicación mostraría otra

vista, en la cual se vería el grafo. Junto con el grafo estaría un botón para acceder nuevamente a la vista anterior en caso que se deseara modificar la consulta.

En la vista del grafo, se optó por tomar aspectos útiles de la visualización en **Neo4j Desktop** y agregar otros para ayudar a mejorar la información presentada al usuario. Así es como se muestra siempre dentro de un nodo la propiedad que los identifica (**nombre** cuando fuera posible o **id**), expandiendo la información en un *tooltip* al interactuar con él. Además, cada tipo de nodo o relación contaría con un color distinto, por lo que junto al grafo estaría visible un cuadro de referencias, explicando la semántica. Un detalle que se consideró fue conservar el color de los nodos de personas al agruparlas, pero modificar el de las aristas, dado que al agrupar se contienen distintos tipos de relaciones. A pedido de **EviMed**, al agrupar los nodos tienen un tamaño proporcional a la cantidad de personas que lo integran. Análogamente sucede con las aristas, cuyo espesor es proporcional a la cantidad de relaciones que la componen.

6.3. Descripción de datos

Durante la etapa de desarrollo del proyecto se decidió usar datos sintéticos, provistos por el equipo. Posteriormente, en fase final del proyecto, se obtuvo una copia de los datos de producción de **RedEMC**, lo que permitió realizar pruebas sobre lo desarrollado.

En este capítulo se describen los datos así como la estructura utilizada en la base de datos.

6.3.1. Modelado

Dado que la base de datos es un grafo en sí misma, se decidió representar las entidades y relaciones de forma que las posteriores consultas fueran más eficientes.

De esta manera, el grafo tendría una representación intuitiva de la realidad, lo que además permitiría que las consultas para extraer información relevante fueran más simples.

En la figura 6.2 se muestra un diagrama de la estructura de la base de datos.

La estructura elegida contaría con los nodos (entidades) **persona**, **curso**, y **mensaje**.

Las propiedades de las personas serían:

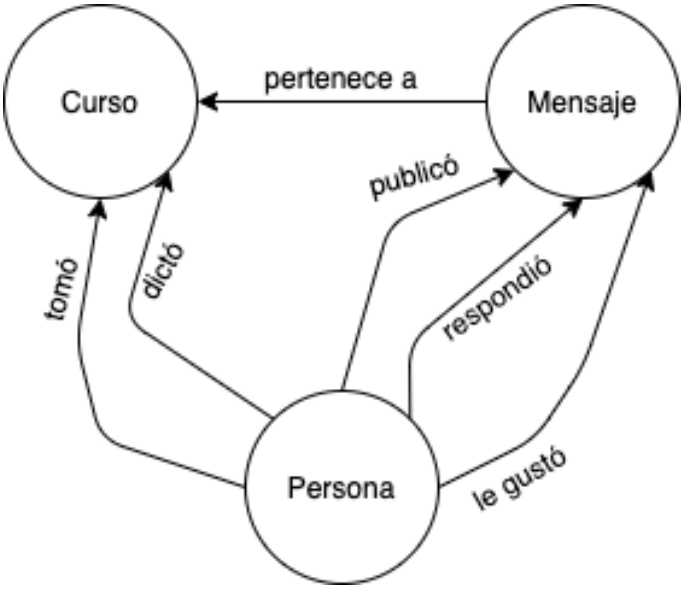

Figura 6.2: Representación de la estructura de la base de datos.

- **id**: identificador en la base de datos original de RedEMC.
- **name**: texto basado en el email con el que se registró en RedEMC.
- **country**: país de origen.
- **gender**: género.
- **specialty**: especialidad médica o área de trabajo en el ámbito de la salud.
- **yearsActive**: años ejerciendo su especialidad.

Las propiedades de los cursos serían:

- **id**: identificador en la base de datos original de RedEMC.
- **name**: nombre del curso.

Las propiedades de los mensajes serían:

- **id**: identificador en la base de datos original de RedEMC.
- **impressionCount**: cantidad de veces que usuarios interactuaron con el mismo.
- **likeCount**: cantidad de veces que usuarios indicaron que les "gusta".
- **responseCount**: cantidad de respuestas.
- **week**: semana del curso a la que pertenece.

Respecto a las relaciones, se añadiría:

- **included_in**: relación entre curso y mensaje. La semántica es "el mensaje está incluido en el curso".

- **took**: relación entre curso y persona. La semántica es "la persona tuvo el rol de estudiante en el curso". Se añade la propiedad **status**: estado del estudiante en el curso (*On track*, *Behind*, *Audit*, *Out*).
- **taught**: relación entre curso y persona. La semántica es "la persona tuvo el rol de docente en el curso".
- **liked**: relación entre persona y mensaje. La semántica es "la persona indicó que le gusta el mensaje".
- **published**: relación entre persona y mensaje. La semántica es "la persona publicó el mensaje".
- **replied**: relación entre persona y mensaje. La semántica es "la persona respondió el mensaje".

6.3.2. Volumen

Previo a contar con los datos reales, se decidió trabajar con un volumen mínimo de datos que permitiera agregar funcionalidades y probarlas. De esta forma, en la base de datos había 2 cursos, 26 mensajes y 15 personas.

En marzo el equipo de **EviMed** compartió los datos que consistían de 146 cursos, 89.643 mensajes, 73.008 personas.

Respecto a cómo se relacionan personas, cursos, y mensajes entre sí, los datos indicaban 94.162 inscripciones a cursos, 1.401 docencias en cursos, 106.680 "me gusta", y 14.533 respuestas a mensajes.

6.3.3. Formato

Una parte importante de la implementación fue la decisión de cómo cargar los datos a partir de la base de datos relacional de **RedEMC**.

Las opciones evaluadas fueron tres:

1. Por cada nodo o arista generar consultas **Cypher** y luego correrlas en la base de datos.
2. Por cada tipo de nodo y arista, generar un CSV y correr una consulta **Cypher** que cargara dicho archivo y creara o actualizara lo necesario en la base de datos.
3. Por cada tipo de nodo y arista, generar un CSV especial, con encabezados que pueden ser interpretados por **Neo4j** y usar la herramienta *neo4j-admin import*.

La opción 1 fue descartada debido a que requería miles de accesos a la base de datos, uno por cada nodo o arista. Esto tiene problemas de desempeño, lo que hacía que el proceso de carga fuera muy costoso. Debido a que en el análisis teórico ya se anticipaba que se trataba de una solución subóptima, no se implementó.

Respecto a la segunda opción, se consideró que era una solución que podía ser factible, por lo que se decidió que era conveniente realizar una prueba con todos los datos, a modo de verificar la hipótesis. De acuerdo a la documentación de **Neo4j** [25], esta forma de importar es adecuada para CSVs de hasta 10 millones de filas. Con esta opción, la carga completa de los datos insumió alrededor de 8 segundos [1]. La solución fue evaluada como satisfactoria y, finalmente, el *script* de carga de datos fue implementado con ella.

Por otro lado, el caso de uso ideal para la tercera opción es carga completa de decenas de millones de nodos o aristas. Debido a esto, se requiere limpiar la base antes de crear los datos, lo que presenta un riesgo si posteriormente los mismos no pudieran ser creados por, por ejemplo, un error en el archivo. Esto último presenta una desventaja frente a la segunda opción. Ya que la cantidad de datos es del orden de decenas de miles, la principal ventaja resultaba indistinta, por lo que se decidió descartarla.

Luego de decidido que la opción a ser utilizada era la segunda, el formato de cada archivo CSV fue definido:

- **courses.csv**: *id_curso*, *nombre_curso*.
- **messages.csv**: *comment_ID*, *impressionCount*, *responseCount*, *likeCount*, *semana_curso*.
- **people.csv**: *ID*, *nombre*, *especialidad*, *pais*, *genero*, *ejercicio*.
- **included.csv**: *comment_ID*, *id_curso*.
- **liked.csv**: *id_usuario*, *id_comentario*.
- **published.csv**: *user_id*, *comment_ID*.
- **replied.csv**: *user_id*, *comment_ID*.
- **taught.csv**: *id_usuario*, *id_curso*.
- **took.csv**: *id_usuario*, *id_curso*, *estado*.

[1] Prueba realizada en computadora MacBook Pro con sistema operativo macOS Catalina. Procesador 2GHz Quad-Core Intel Core i5. Memoria RAM 16 GB DDR4.

6.3.4. Características y calidad de los datos

Se identificaron propiedades de las personas con valor vacío. Las únicas propiedades para las que esto no sucedió fueron ID y *name*, ya que eran las únicas no obligatorias, por lo tanto se decidió no realizar ninguna acción sobre los datos. Por otro lado, en el campo ***specialty*** se detectaron valores ligeramente diferentes. Un ejemplo de esto último es "nefrologia"(sic) y "nefrología". En este caso, la discrepancia fue señalada, aunque su impacto era mínimo en el desarrollo del proyecto, por lo que se decidió no trabajar en la normalización de los mismos al momento de realizar la carga de datos.

Respecto a los cursos, existían cursos con distintos ID y el mismo valor para *name*. Luego de consultar con el equipo de **EviMed** se supo que esto se debía a que algunos cursos se dictaron en distintos lenguajes y, por lo tanto, tienen el mismo nombre. Dado que esto tiene una explicación basada en la lógica de negocio, simplemente se tomó nota para la correcta elaboración de las consultas.

Inicialmente, los datos provistos por **EviMed** tenían múltiples relaciones *took* para el mismo usuario en el curso. Esto se debía a que el estado de una persona en un curso está enmarcado en el contexto de una semana. Se decidió que, a los efectos del presente trabajo, lo mejor era tener en la base de datos únicamente una relación con el último valor de estado.

6.3.5. Consultas

Las consultas **Cypher** escritas para obtener los datos requeridos por el usuario tuvieron distintas complejidades según fuera necesario agrupar. A continuación se comenta qué fue necesario en cada caso y se explican los detalles más importantes.

Debido a que eran las consultas que en principio aparentaban ser más directas y similares a ejemplos típicos en cursos y tutoriales, se decidió comenzar con aquellas que no implicaban la agrupación de personas. En efecto, bastó con usar filtros simples para obtener los resultados deseados.

```
1 MATCH (p:Person)-[r]->(m:Message), (p)-[role]-(c:Course { id:
  ↪ $courseId }), (m)--(c)
2 RETURN m, r, p, role
```

Ejemplo 1: Consulta sin agrupar.

Por el contrario, la generación de consultas que agruparan personas resultó notoriamente más compleja y requirió de una sintaxis más avanzada, haciendo uso de la librería **APOC**.

Lo primero a destacar es que fue necesario crear nodos virtuales [2] para los grupos que serían retornados al final de la consulta. Por ejemplo, si se deseaba agrupar por países, el primer paso era obtener todos los distintos países y crear un nodo virtual para cada uno, guardando la cantidad de personas pertenecientes al mismo. A su vez, se creaba un objeto con referencias a cada nodo virtual.

Por otro lado, dado que los nodos virtuales solo existen durante la ejecución de la consulta que los crea, fue necesario hacer dos operaciones *MATCH*: una para obtener la lista de países, y otra para los cursos, personas, y mensajes que cumplieran los criterios de filtro.

Finalmente, se creaba una relación virtual [3] por cada par de países entre los que hubiera interacciones, indicando la cantidad de estas.

Corresponde aclarar que para cada par de grupos *G1, G2*, la consulta retorna una relación virtual de *G1* a *G2* y otra de *G2* a *G1*. Esto se debe a que la consulta itera sobre las personas y, al encontrar un grupo, busca los grupos con los que tiene una relación partiendo de sí mismo [4]. De cualquier manera, esto no afecta el resultado que retorna la *API* ya que en código **JavaScript** se juntan en una sola relación, sumando la cantidad de interacciones.

[2] Los nodos virtuales son nodos que se crean como parte de una consulta y existen solo durante la ejecución de la misma. Pueden tener propiedades como los nodos comunes. Su *ID* asignada automáticamente toma un valor negativo, y es ese el criterio que suele usarse para distinguir a estos nodos de los otros.

[3] Equivalente a un nodo virtual.

[4] Vale la pena recordar que en **Neo4j** existen únicamente aristas unidireccionales.

```
MATCH (p:Person)-[]-(:Course {id:$courseId})
WITH DISTINCT p.country AS country, collect(p.country) AS countryList
WITH apoc.map.groupBy(collect(apoc.create.vNode([\"Country\"],{
  ↪ name:country, value: size([c IN countryList WHERE c = country
  ↪ ])})), \"name\" ) AS countries
MATCH (p1:Person)-[]-(m:Message)-[]-(p2:Person), (p1)-[r1]-(c:Course
  ↪ { id: $courseId })-[r2]-(p2), (m)--(c)
WHERE p1.id < p2.id AND EXISTS(p1.country) AND EXISTS(p2.country)
WITH p1.country AS cFrom, p2.country AS cTo, count(*) AS count,
  ↪ countries
RETURN countries[cFrom] AS from, countries[cTo] AS to,
  ↪ apoc.create.vRelationship(countries[cFrom], \"KNOWS\", { count:
  ↪ count }, countries[cTo]) AS rel
```

Ejemplo 2: Consulta agrupando.

7. Validación y pruebas

A lo largo del proyecto se realizaron distintas pruebas para detectar desperfectos o posibles problemas y resolverlos.

Se realizaron pruebas sobre los datos, unitarias del módulo encargado de la traducción a consultas Cypher, y estrés tanto de *backend* como de *frontend*. Por otro lado, se realizaron pruebas de tipo cualitativo sobre la visualización.

En la presente sección se describen las pruebas realizadas y los resultados obtenidos. En caso de corresponder, se indican los cambios realizados para resolver los problemas encontrados.

7.1. Datos

Las pruebas sobre datos fueron dos: análisis de datos proporcionados en los CSV, y verificación de la correctitud de datos cargados.

Respecto al análisis de datos, inicialmente se detectaron anomalías tales como repetición de valores en algunas propiedades para nodos con distinto ID. Luego de validar con el equipo de desarrollo de **EviMed** que estas anomalías eran efectivamente errores, se obtuvieron nuevas versiones de los archivos correspondientes.

Una vez resueltos dichos problemas, las pruebas se centraron en la verificación de los datos creados. Realizando estas pruebas, se notó que cada persona que tomaba un curso tenía varias relaciones distintas con el mismo, a veces con distintos valores en el campo *estado*. Al comentarlo con el equipo de **EviMed**, se entendió que, si bien no era un error, a los efectos del presente proyecto era innecesario contar con un estado por cada semana y bastaba con tomar el más reciente.

Por otro lado, el uso de la aplicación con los datos reales permitió un mejor entendimiento de **RedEMC** y, como consecuencia, propició la corrección de errores. Algunos de estos errores fueron cambiar campos en la aplicación para que, en vez de ser abiertos, se ofreciera una lista de opciones de la que se podía seleccionar una. Esto sucedió tanto con *especialidad* como con *años de ejercicio*.

También se encontraron problemas en las consultas **Cypher** en las que se asumía que campos como *specialty* y *country* siempre tenían valores. Para algunas personas esos valores eran *null*, y las consultas mencionadas fallaban. Esto sucedía principalmente con aquellas necesarias para agrupar personas.

Analizando los valores para las distintas propiedades, se notó que existen distintas opciones que seguramente en realidad sean la misma. Un ejemplo de esto es en el campo *especialidad* algunas personas tienen *nefrología* y otros lo tienen sin tilde. Algo similar sucedió con años de ejercicio, que cuenta con la opción 25, así como 25+.

Finalmente, al cargar los datos se verificó que todos los nodos (cursos, personas, y mensajes) fueron creados correctamente. Las aristas, en tanto, no fueron creadas en su totalidad y se realizó un análisis de las causas. En la tabla 7.1 se presenta la diferencia entre valores creados y esperados, así como la causa.

Relación	Diferencia	Causa
Took	238	usuario_id correspondiente a un usuario no existente.
Taught	4	usuario_id correspondiente a un usuario no existente.
Included	1807	curso_id correspondiente a un curso no existente.
Published	2971	usuario_id o id_comentario correspondientes a usuario o comentario no existente, respectivamente.
Liked	200	usuario_id o id_comentario correspondientes a usuario o comentario no existente, respectivamente.
Replied	108	id_comentario correspondiente a un comentario no existente.

Cuadro 7.1: Causas para las diferencias en cantidad de relaciones creadas.

Esto último fue reportado al equipo de **EviMed**, aunque su corrección inmediata no fue solicitada, ya que no ocasionaba dificultades para el desarrollo de este proyecto. Al momento de la puesta en producción se recordó este problema.

7.2. Unitarias

Para realizar las pruebas unitarias se utilizó el *framework* Jest [28] y el patrón **Arrange-Act-Assert**.

Arrange-Act-Assert propone dividir las pruebas en tres etapas: ***arrange***, ***act***, y ***assert***. En la etapa ***arrange*** se realiza la preparación del escenario que se desea probar. Luego, en ***act***, se invoca la función que se está probando, haciendo uso del escenario previamente preparado. Finalmente, en ***assert*** se compara el resultado obtenido con el esperado.

Las pruebas unitarias estuvieron centradas en el núcleo de la aplicación: la conversión de parámetros a consulta **Cypher**. Como se mencionó en la sección 6.2, hay cuatro funciones dedicadas a esta conversión. Cada una de ellas fue probada en distintos casos de uso.

En las siguientes secciones se describen los casos probados (*arrange* para cada función. La etapa *act* fue la llamada a la función indicada en cada sección. Por último, *assert* en todos los casos implicó la verificación de la consulta **Cypher** generada.

Curso

Las pruebas realizadas para búsquedas por ID del curso (función **buildByCourseQuery**) se detallan a continuación:

- **Por defecto**: no se aplica ningún filtro.
- **Único filtro**: de a uno por vez, se filtra por
  - Persona: país, especialidad, género, años de ejercicio, estado, o rol.
  - Mensaje: cantidad de impresiones, "me gusta", y respuestas, o semana.
  - Relación.
- **Múltiples filtros**: se aplican todos los anteriormente mencionados, pero esta vez juntos.

El proceso es simétrico para búsqueda por nombre del curso.

Persona

Las pruebas realizadas para búsquedas por ID de la persona (función **buildByPersonQuery**) se detallan a continuación:

- **Por defecto**: no se aplica ningún filtro.
- **Sin incluir otras personas**: se filtra indicando o no relaciones con los mensajes.
- **Incluyendo otras personas**:
  - Único filtro: de a uno por vez, se filtra por

        - Persona: país, especialidad, género, años de ejercicio, estado, o rol.
        - Mensaje: cantidad de impresiones, "me gusta", y respuestas, o semana.
        - Relación.
    - Múltiples filtros: se aplican todos los anteriormente mencionados, pero esta vez juntos.

Mensaje

Las pruebas realizadas para búsquedas por ID del mensaje (función **buildByMessageQuery**) se detallan a continuación:

- **Por defecto**: no se aplica ningún filtro.
- **Único filtro**: de a uno por vez, se filtra por
    - Persona: país, especialidad, género, años de ejercicio, estado, o rol.
    - Relación.
- **Múltiples filtros**: se aplican todos los anteriormente mencionados, pero esta vez juntos.

Agrupando

Las pruebas realizadas para búsquedas por ID del curso agrupando (función **buildGroupQuery**) se detallan a continuación:

- **Agrupado según país**:
    - Por defecto: no se aplica ningún filtro.
    - Único filtro: de a uno por vez, se filtra por
        - Persona: especialidad, género, años de ejercicio, o estado.
        - Relación.
- **Agrupado según especialidad**:
    - Por defecto: no se aplica ningún filtro.
    - Único filtro: de a uno por vez, se filtra por
        - Persona: país, género, años de ejercicio, o estado.
        - Relación.
- **Agrupado según estado**:
    - Por defecto: no se aplica ningún filtro.
    - Único filtro: de a uno por vez, se filtra por

- Persona: país, género, años de ejercicio, o especialidad.
- Relación.

El proceso es simétrico para búsqueda por nombre del curso.

7.3. Estrés

En etapas finales del proyecto, se realizaron pruebas de estrés para evaluar el desempeño con gran cantidad de datos. Estas pruebas eran fundamentales debido a que la cantidad de datos usados para el desarrollo era varias magnitudes menor que el real. Todas las pruebas presentadas fueron realizadas en un entorno local[1]. Resulta de interés destacar que parte de la configuración inicial de la base de datos fue agregar índices sobre **ID** de curso, mensaje, y persona, así como de **nombre** de curso.

La primera prueba fue sobre la carga de datos. Como se explicó en la sección 6.3.3, la carga total con el método elegido insumió alrededor de 8 segundos, lo que se consideró satisfactorio, por lo que no se realizaron modificaciones.

Peor caso

Inicialmente, la planificación indicaba que se realizarían pruebas incrementales hasta llegar a la evaluación de la totalidad de datos. A efectos de tener nociones del peor caso (carga total de datos sin realizar modificaciones), se decidió hacer una prueba preliminar con todos los datos, haciendo búsquedas sobre el curso con mayor cantidad de usuarios asociados. Dicho curso es *Enfermeria-HD*, el mismo cuenta con 6521 usuarios (6498 estudiantes, 23 docentes), 2081 mensajes. En la tabla 7.2 se presentan tiempos obtenidos para distintas búsquedas [2].

A partir de los resultados obtenidos en esa prueba, se concluyó que el cuello de botella se encontraba en la visualización del grafo. Investigando el funcionamiento de la librería *d3.js*, se comprendió que el motivo para el pobre desempeño era que el algoritmo es de orden $O(n^2)$, siendo n la cantidad de nodos. Por otro lado, si bien el servidor demora 1.4 segundos, la respuesta es de tamaño 4 MB, lo cual está por encima de lo habitualmente recomendado.

[1] Computadora MacBook Pro con sistema operativo macOS Catalina. Procesador 2GHz Quad-Core Intel Core i5. Memoria RAM 16 GB DDR4.

[2] Se entiende *Tiempo - servidor* como tiempo en segundos transcurrido desde que se envía la *request* al servidor hasta que se obtiene la respuesta. *Tiempo - consulta* es el tiempo en segundos transcurrido desde que se presiona el botón **Consultar** hasta que se muestra una imagen del grafo.

Consulta	Tiempo - servidor (s)	Tiempo - aplicación (s)
Por defecto (por nombre)	1.4	28
Por defecto (por ID)	1.4	28
Filtrado por docente	0.820	4
Filtrado por estudiantes de UY	0.630	1
Agrupamiento por estado	4.94	5
Filtrado por relación de publicado/respondido	0.465	7

Cuadro 7.2: Tiempos obtenidos para búsquedas en el curso Enfermeria-HD.

Aunque se presentan los resultados obtenidos para las consultas por defecto, los requerimientos del proyecto estaban centrados en visualización con filtros y agrupamientos, para lo cual los resultados fueron aceptables, aún en el caso de un curso que tiene más usuarios que la media (ubicada entorno a los 800).

Debido a los motivos expuestos, se consideró una opción aceptable agregar una aclaración al manual del usuario explicando este comportamiento.

Caso promedio

Para evaluar el caso promedio se buscaron proyectos que tuvieran alrededor de 800 usuarios involucrados. Se encontró que el curso *Enfermedad-Cardiovascular-y-Rinon* cuenta con 801 usuarios (781 estudiantes, 20 docentes), y 1706 mensajes, por lo que se decidió trabajar con el mismo. En la tabla 7.3 se presentan tiempos obtenidos para distintas búsquedas, con igual semántica que en 7.3.

De los tiempos obtenidos se concluye que el desempeño para los cursos promedio, realizando el tipo de consultas objetivo de este proyecto, es aceptable.

7.4. Visualización

La validación de la visualización de los datos se enfocó en dos puntos:

- calidad de la información inicial
- calidad de la información luego de la interacción

Consulta	Tiempo - servidor (s)	Tiempo - aplicación (s)
Por defecto (por nombre)	0.553	10
Por defecto (por ID)	0.550	10
Filtrado por docente	0.228	2.04
Filtrado por estudiantes de UY	0.099	1.59
Agrupamiento por estado	1.18	1.65
Filtrado por relación de publicado/respondido	0.295	5

Cuadro 7.3: Tiempos obtenidos para búsquedas en el curso Enfermedad-Cardiovascular-y-Rinon.

7.4.1. Calidad de información presentada inicialmente

En este aspecto lo primero que se destaca es que tanto los nodos como las aristas tienen colores con una semántica explicada en un cuadro de referencias, siempre visible en la esquina superior izquierda de la página. A modo de evitar confusiones, las aristas toman distintos colores según la vista sea de datos agrupados o no, pero los nodos de personas permanece en rojo. Ver figuras 7.1 y 7.2.

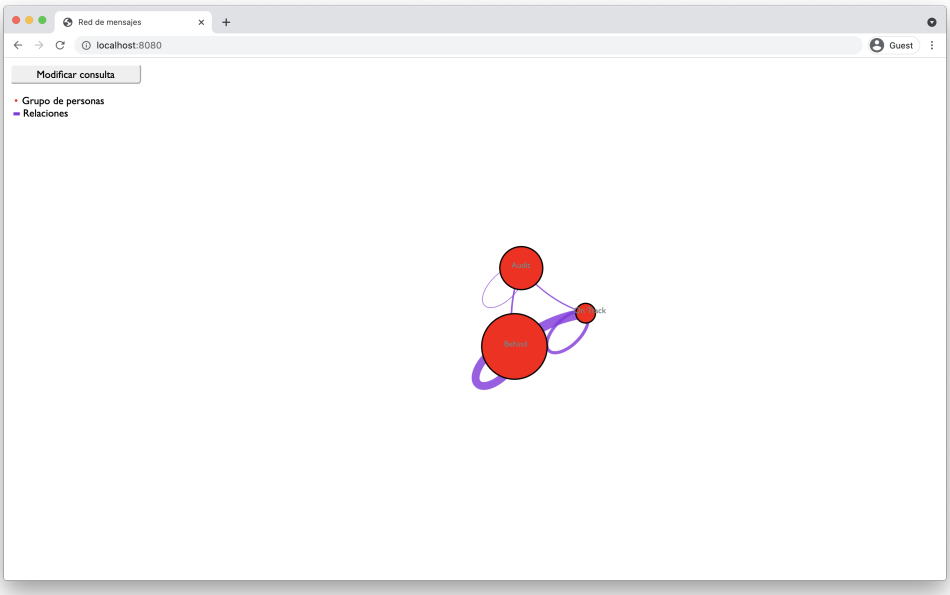

Figura 7.1: Vista del grafo con personas agrupadas.

Por otro lado, cada nodo tiene visible a simple vista un dato único:

- persona: email

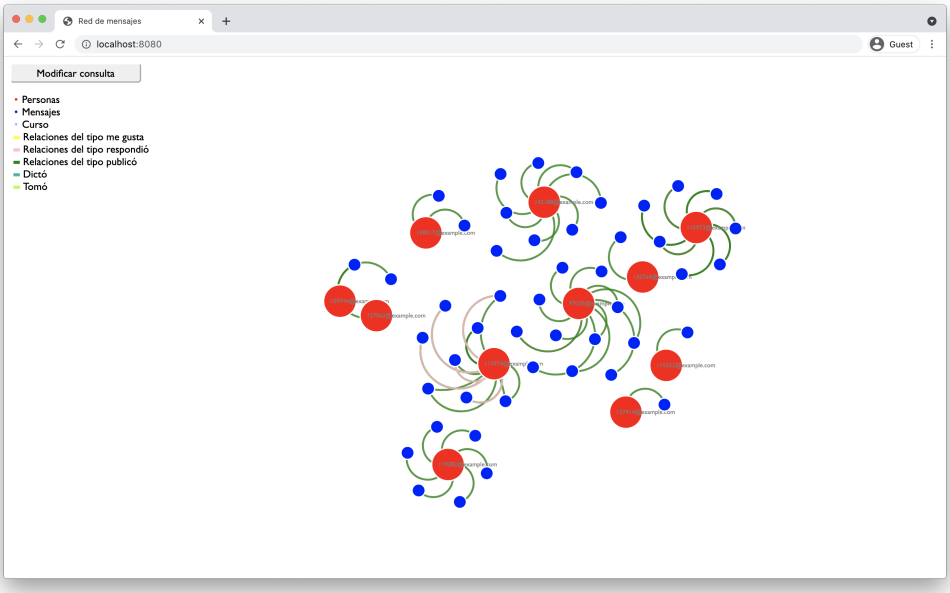

Figura 7.2: Vista del grafo con personas por separado.

- mensaje: id
- curso: nombre
- grupo de personas por país: nombre de país
- grupo de personas por especialidad: nombre de especialidad
- grupo de personas por estado en el curso: estado

Además, al agrupar personas, los nodos tienen un tamaño proporcional a la cantidad de individuos que lo integran. De forma análoga sucede para las aristas, cuyo espesor es proporcional a la cantidad de relaciones que la componen.

Estas tres decisiones de diseño permiten al usuario de la aplicación rápidamente entender qué está viendo y, aún antes de interactuar con el grafo, esbozar conclusiones cualitativas sobre el comportamiento de los usuarios. Esto es especialmente cierto para el caso en que se decide agrupar personas.

Se observa que en algunos casos la elección de colores puede presentar dificultades por falta de contraste.

7.4.2. Calidad de información presentada luego de interacción

Al posicionarse sobre un nodo o arista, se muestra una descripción emergente (también conocida como *tooltip*) que aporta datos extra sobre el mismo. En todos los casos, se muestran todos los datos disponibles en la base de datos.

A modo de ejemplo, cuando no se ha decidido agrupar personas, al posicionarse sobre el nodo correspondiente a un individuo, se obtienen datos de enorme valor como ser su id. Esto permite, dentro de la aplicación realizar una búsqueda sobre el mismo y analizar sus interacciones. Estos datos, por tanto son de gran utilidad, ya que también facilitan la búsqueda de más información en la base de datos de **EviMed**. Ver figura 7.3.

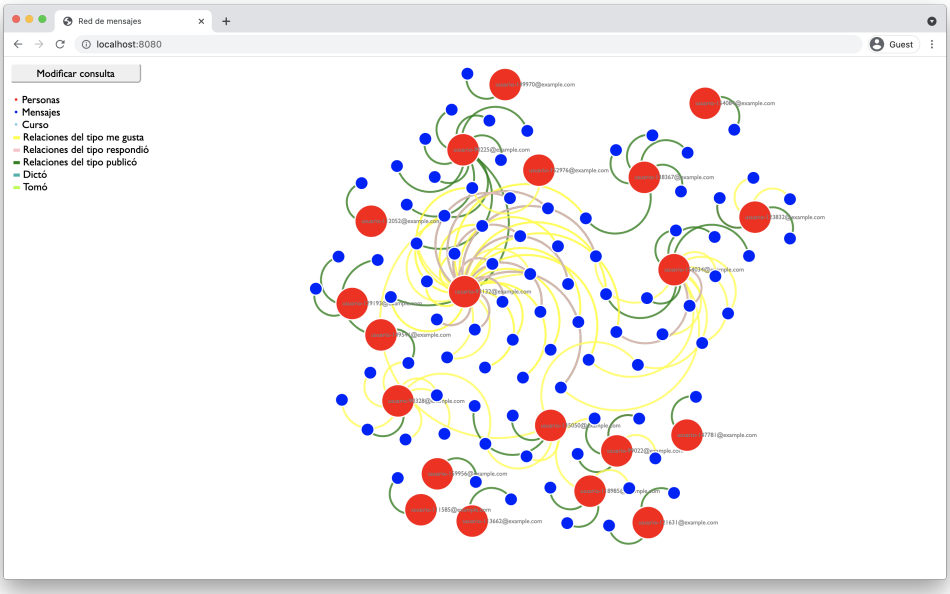

Figura 7.3: Vista del grafo con personas por separado al interactuar.

En el caso de un grafo correspondiente a personas agrupadas, las aristas proporcionan información sobre qué dos grupos son los que están en los extremos. Esto permite paliar las dificultades que pueden producirse cuando son muchas las relaciones entre los grupos y no es claro qué nodos unen las aristas. Ver figura 7.4.

Con respecto a la información obtenida al interactuar, sería de utilidad destacar los nodos/aristas relacionados al posicionarse en uno. De esta forma, aún cuando hubiera muchos nodos y aristas en el grafo, se podría mejorar la visibilidad sin pérdida de información respecto al panorama general.

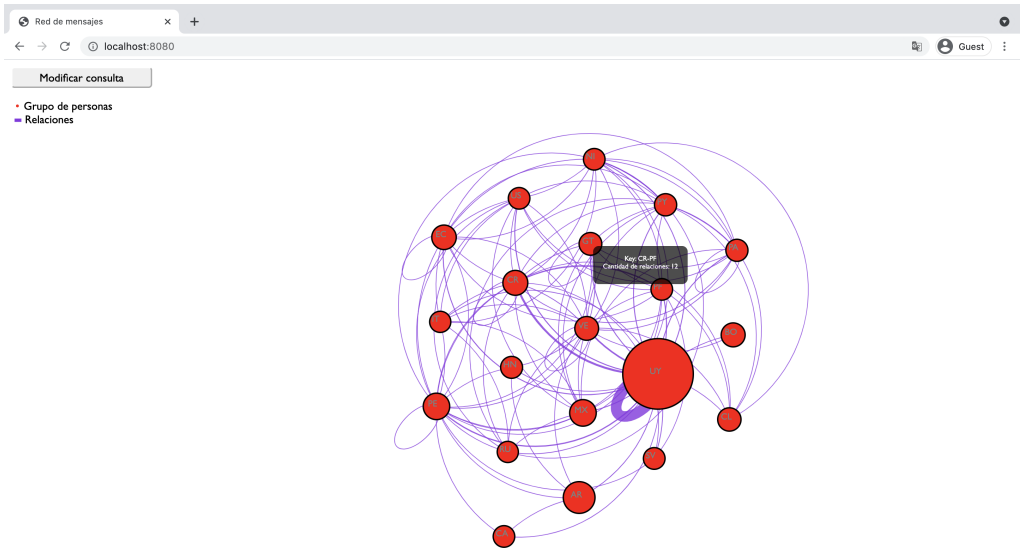

Figura 7.4: Vista del grafo con personas agrupadas al interactuar.

7.4.3. Umbral de visibilidad

Al realizar consultas sin agrupar ni filtrar, se produce un grafo resultado con miles de nodos y aristas, lo que dificulta su análisis. En el caso de las aristas, las relaciones de tipo *le gusta* agregan ruido, aunque la elección de color amarillo para las mismas hace que no resalten tanto.

Encontrar un umbral es complejo, ya que además de la cantidad de nodos y relaciones, es importante qué tan denso es el grafo. Grafos menos densos tienen tendencia a tener una mejor visibilidad que otros con la misma cantidad de nodos y grafos.

Habitualmente, los grafos correspondientes a consultas agrupando personas por país o especialidad son más densos, por lo que si hay muchas relaciones las aristas pueden quedar superpuestas y dificultar su comprensión. Ver figura 7.5.

Se decidió comparar el comportamiento de la aplicación con la herramienta **Neo4j Desktop**. Una observación importante es que esta por defecto limita la cantidad de nodos que se dibujan a 300 por vez. De cualquier manera, grafos con esta cantidad de nodos presentan cierta dificultad para ser leídos. Ver figuras 7.7 y 7.6.

Podría ser conveniente limitar la cantidad de nodos que se muestran por defecto y advertirle al usuario de esto, recomendando usar filtros.

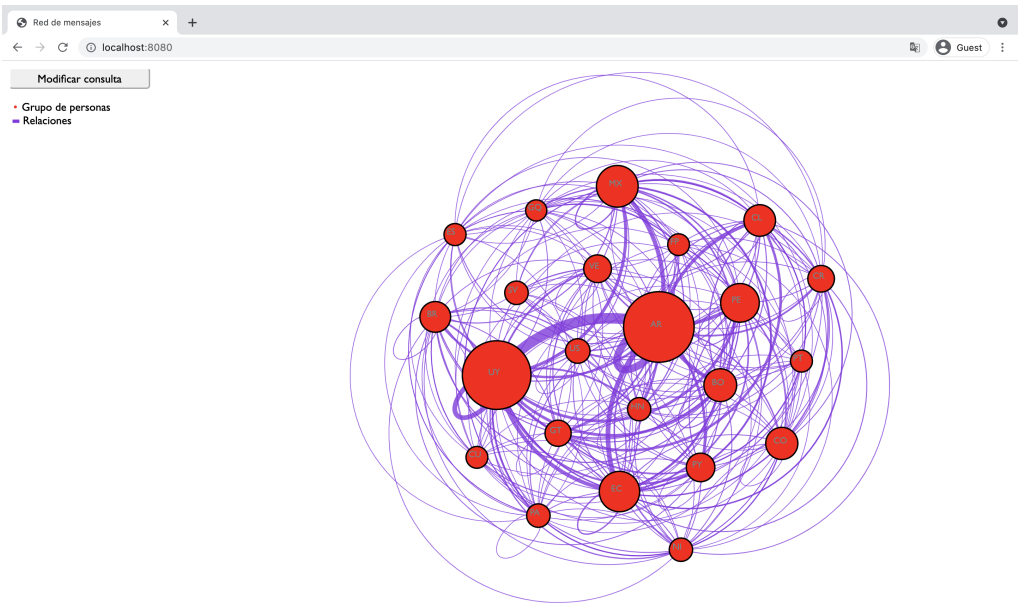

Figura 7.5: Vista de un grafo muy denso con personas agrupadas.

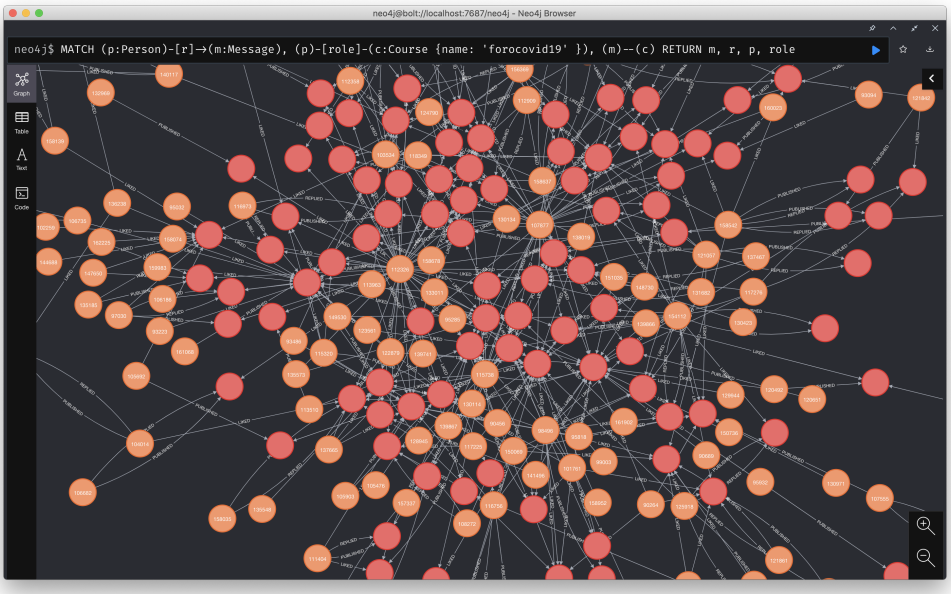

Figura 7.6: Vista del grafo del curso *forocovid19* en Neo4j Desktop.

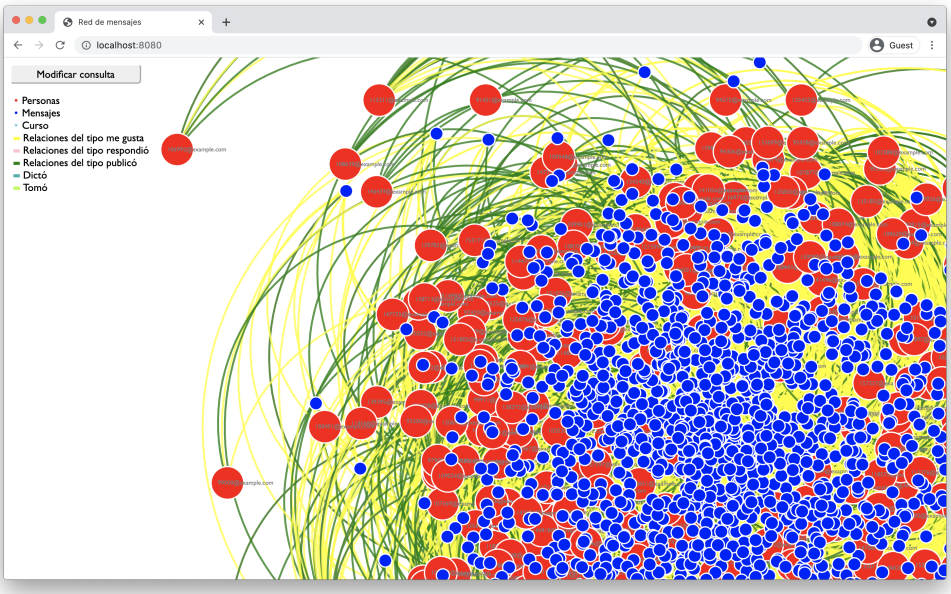

Figura 7.7: Vista del grafo del curso *forocovid19* en la aplicación.

Si bien Neo4j Desktop también provee la posibilidad de arrastrar nodos para separarlo de los demás, no parece aportar claridad por sobre el grafo original. De cualquier manera, podría ser interesante investigar esta posibilidad.

7.5. Validación por parte de EviMed

En agosto de 2021, momento en el que el proyecto se encontraba prácticamente finalizado, se decidió realizar una encuesta a autoridades de la institución educativa en el marco de la cual se desarrolló el mismo.

La finalidad de la encuesta fue conocer su opinión respecto al trabajo realizado por parte del equipo, así como validar que el proyecto cumplía con las expectativas trazadas al comienzo del mismo.

En ese marco, las preguntas elegidas fueron las siguientes:

1. ¿Considerás que el objetivo del proyecto se logró?
2. ¿Qué utilidad imaginás que le van a dar?
3. ¿Resultó productivo el proceso de trabajo?

En el recuadro 7.1 se presentan las respuestas obtenidas.

¿Considerás que el objetivo del proyecto se logró?

El objetivo inicial ha sido alcanzado ampliamente, y se han logrado resultados que no estaban previstos. El proyecto original estaba planteado con un objetivo de análisis de información (modelado en base de datos de grafos, importación de grandes volúmenes de datos, generación de consultas complejas, evaluación de performance, etc) pero al final se terminó implementando no solo eso sino que además se realizó un front-end que permite realizar análisis interactivo de la información de una manera muy visual y centrada en el usuario (cursante), que le permite "ver" cómo ha sido su interacción en las actividades educativas con los mensajes de los foros de discusión desde múltiples perspectivas (geográfica, por especialidad, etc).

¿Qué utilidad imaginás que le van a dar?

En EviMed ya estamos trabajando para automatizar la carga de datos en la base de datos de grafos y para disponibilizar el front-end antes mencionado para que los propios cursantes lo puedan utilizar. Esto -creemos- contribuirá significativamente a dar visibilidad al principal diferenciador de la plataforma educativa de EviMed respecto de otras plataformas de e-learning, que es el uso intensivo de herramientas de social learning analytics que si bien existen y mejoran significativamente la experiencia de los cursantes, no es fácil mostrar cómo funcionan. Este front-end va a dar visibilidad de los fundamentos de los algoritmos que soportan dichas herramientas.

¿Resultó productivo el proceso de trabajo?

El proceso de trabajo fue excelente . La interacción con Leticia ha sido muy fluida y dinámica desde el inicio. A diferencia de otros proyectos donde se realizan reuniones periódicas, en este caso tuvimos reuniones a demanda, en función de las necesidades, y casi siempre a distancia. Esta metodología resultó eficiente y efectiva para atender las necesidades de este proyecto, quizás por el hecho de que se trataba de un sólo estudiante.

Recuadro 7.1: Respuestas a encuesta por parte de Ing. Antonio López Arredondo.

Como se observa en el cuadro, la opinión de **EviMed** es positiva en todos los aspectos, tanto respecto a los resultados como en cuanto a la forma de trabajo e interacción. Se destaca que los resultados superaron las expectativas iniciales y el interés por su parte de integrar lo realizado en este proyecto lo antes posible. Finalmente, el proceso de trabajo no solo resultó efectivo, sino también eficiente.

8. Resultados

En el presente capítulo se realiza una presentación de las funcionalidades implementadas, así como la puesta en producción.

8.1. Funcionalidades

La aplicación tiene su núcleo en las consultas y la visualización de sus resultados. Más allá de eso, a pedido del equipo de EviMed se añadió una capa de seguridad para el acceso a los datos.

8.1.1. Autenticación

Al acceder a la aplicación, se requiere iniciar sesión para poder realizar consultas y ver los resultados. Esto se debe a que los datos son sensibles y, por lo tanto, no pueden ser accesibles públicamente.

El usuario se autentica proveyendo las mismas credenciales que usa para iniciar sesión en **RedEMC**. Luego, en el servidor, se consulta un *endpoint* (provisto por **EviMed**) con el email y contraseña, para verificar su validez. Si los datos son correctos, el usuario accede a la aplicación.

Ya que la autenticación en definitiva la realiza **EviMed**, en cada *request* al servidor, se debe enviar el email y contraseña, de forma que se pueda validar que son correctos.

Para evitar que el cliente envíe repetidamente las credenciales del usuario en texto plano, cuando el usuario inicia sesión, se crea un *token* que luego el servidor recibe junto con la consulta a realizarse en la base de datos. En el *token* están (codificados) el email y contraseña, por lo que el servidor puede decodificarlo y validar las credenciales con el *endpoint* de **EviMed**.

Para simplificar el desarrollo de la autenticación, se decidió que cuando el usuario recarga la página, su sesión será cerrada. De la misma manera, si cambia la contraseña en **RedEMC** y luego realiza una consulta, sus credenciales no serán validadas por el *endpoint* y su sesión será cerrada.

8.1.2. Consultas

Nota: En el apéndice A.2 se realiza una explicación en detalle de las funcionalidades presentadas en esta sección.

De acuerdo a lo definido en el alcance, las consultas a realizar consideran búsqueda por curso, persona, o mensaje, y, a su vez, permiten aplicar filtros las propiedades de las entidades y relaciones asociadas. Debido a la complejidad y flexibilidad de los filtros y sus combinaciones, es posible realizar consultas que obtienen precisamente los datos deseados.

Por curso

En todos los casos aplican los mismos filtros al buscar por ID o nombre del curso.

A continuación se detallan los campos disponibles para el filtrado por **persona**, indicando las opciones para cada uno en caso de corresponder[1].

- Rol
  - Docente
  - Estudiante
- Especialidad: listado de todas las disponibles (55 opciones)
- Sexo
  - Hombre
  - Mujer
- País de origen: campo de texto
- Años de ejercicio
  - 0-5
  - 6-15
  - 16-25
  - 25+
- Estado en el curso
  - *Audit*

[1]El campo **Estado** no es aplicable al filtrar por el **Rol** docente.

- *Behind*
- *On track*
- *Out*

Los campos disponibles para filtrar **mensajes** son los siguientes.

- Cantidad de "me gusta": listado de comparaciones entre números (mayor, menor, igual), y campo de texto
- Cantidad de impresiones: listado de comparaciones entre números (mayor, menor, igual), y campo de texto
- Cantidad de respuestas: listado de comparaciones entre números (mayor, menor, igual), y campo de texto
- Semana del curso: campo de texto

Finalmente, existen filtros sobre las tres **relaciones** existentes entre mensajes y personas:

- Le gustó
- Publicó
- Respondió

Las restricciones aplicadas al agrupar personas se detallan en la tabla 8.1.

Criterio	Restricciones
País	Rol y País no disponibles. No hay filtros sobre mensajes.
Especialidad	Rol y Especialidad no disponibles. No hay filtros sobre mensajes.
Estado	Rol y Estado no disponibles. No hay filtros sobre mensajes.

Cuadro 8.1: Restricciones sobre filtros al agrupar personas.

Por persona

La búsqueda se realiza por ID de la persona.

A continuación se detallan los campos disponibles para el filtrado por **persona** (con quienes interactuó el que se identifica con la ID), indicando las opciones para

cada uno en caso de corresponder[2].

- Rol
  - Docente
  - Estudiante
- Especialidad: listado de todas las disponibles (55 opciones)
- Sexo
  - Hombre
  - Mujer
- País de origen: campo de texto
- Años de ejercicio
  - 0-5
  - 6-15
  - 16-25
  - 25+
- Estado en el curso
  - *Audit*
  - *Behind*
  - *On track*
  - *Out*

Los campos disponibles para filtrar **mensajes** son los siguientes.

- Cantidad de "me gusta": listado de comparaciones entre números (mayor, menor, igual), y campo de texto
- Cantidad de impresiones: listado de comparaciones entre números (mayor, menor, igual), y campo de texto
- Cantidad de respuestas: listado de comparaciones entre números (mayor, menor, igual), y campo de texto
- Semana del curso: campo de texto

[2]El campo **Estado** no es aplicable al filtrar por el **Rol** docente.

Finalmente, existen filtros sobre las tres **relaciones** existentes entre mensajes y personas:

- Le gustó
- Publicó
- Respondió

Los filtros sobre mensajes y personas están disponibles solo al incluir a quienes interactuaron con el usuario por el que se realiza la búsqueda. En otro caso, únicamente se pueden filtrar sobre las relaciones.

Por mensaje

La búsqueda se realiza por ID del mensaje.

A continuación se detallan los campos disponibles para el filtrado por **persona** (quienes interactuaron con el mensaje buscado), indicando las opciones para cada uno en caso de corresponder[3].

- Rol
  - Docente
  - Estudiante
- Especialidad: listado de todas las disponibles (55 opciones)
- Sexo
  - Hombre
  - Mujer
- País de origen: campo de texto
- Años de ejercicio
  - 0-5
  - 6-15
  - 16-25
  - 25+
- Estado en el curso
  - *Audit*

[3]El campo **Estado** no es aplicable al filtrar por el **Rol** docente.

- *Behind*
- *On track*
- *Out*

Finalmente, existen filtros sobre las tres **relaciones** existentes entre mensajes y personas:

- Le gustó
- Publicó
- Respondió

Ejemplo de consulta sobre curso

A continuación se presenta un ejemplo de una consulta sobre cursos. En la misma se busca obtener información sobre la interacción de los docentes en los mensajes del curso *vacunaspediatria*, incluyendo únicamente los mensajes publicados o respondidos por ellos, como se muestra en la figura 8.1. El grafo resultado 8.2 permite concluir rápidamente que los docentes de este curso publicaron más mensajes de los que respondieron y, además, que sobre cada mensaje hubo interacciones solo por parte de un docente a la vez.

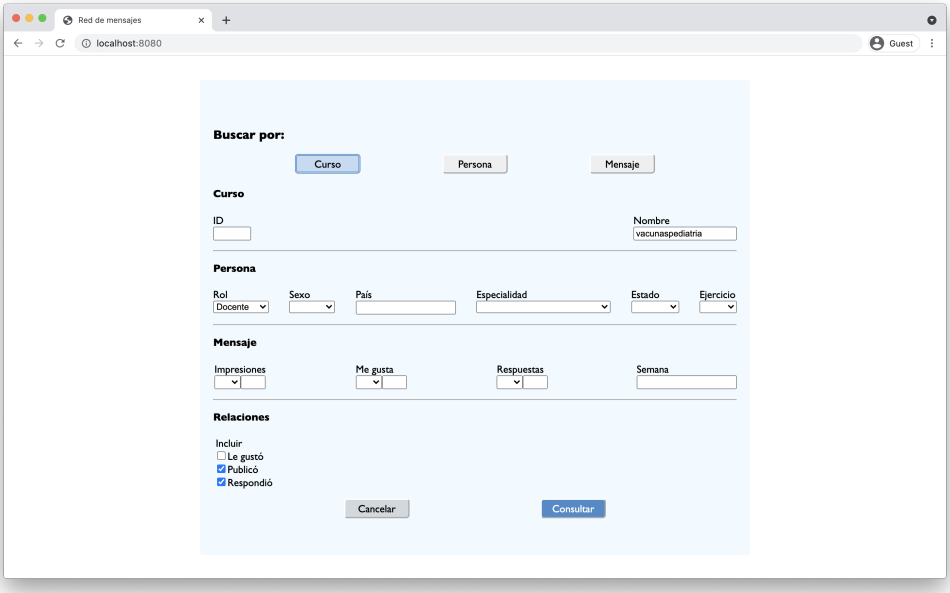

Figura 8.1: Consulta sobre el curso *vacunaspediatria*.

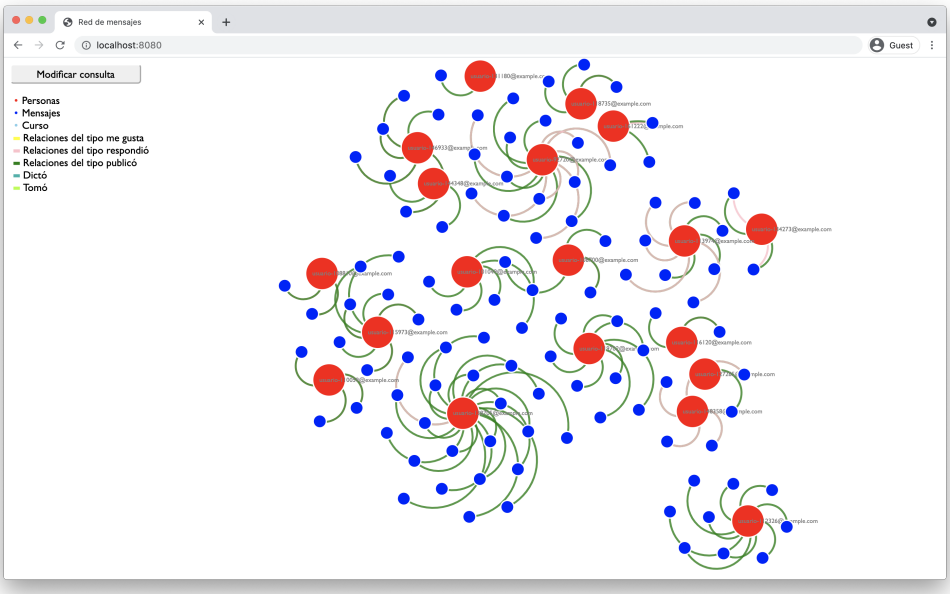

Figura 8.2: Grafo resultado.

8.2. Puesta en producción

Sobre el final del proyecto, se hizo foco en todo lo necesario para garantizar que la aplicación podría ser utilizada en producción, como parte de **RedEMC**.

8.2.1. Configuración

Para la puesta en producción, el equipo de desarrollo de **EviMed** puso a disposición un servidor Amazon Elastic Compute Cloud (EC2). En ese mismo servidor se instaló **Neo4j** e implantó el servidor web de la aplicación.

Base de datos

Se instaló en el servidor la base de datos **Neo4j** (a cargo del equipo de **EviMed**), así como la librería **APOC**. Posteriormente, se creó un *cron job*, encargado de cargar los datos semanalmente. Este *cron job* ejecuta un *script* que, dado los CSVs con los nodos y aristas, corre las consultas necesarias para realizar los cambios en el grafo.

Servidor

Para poner en funcionamiento el servidor, se instaló la librería **PM2** [35] en la instancia EC2 provista. Luego de realizada la configuración del servidor, se

agregaron las variables de entorno necesarias para conectarlo con la base de datos.

Cliente

Para poner en producción el cliente, se decidió convertir la aplicación en un sitio web estático, que pudiera ser servido por el mismo servidor que funcionaba como *backend*. Para realizar esto, debió ser levemente modificado el *script* que se utilizaba para mantener el entorno de pruebas durante el desarrollo.

8.2.2. Integración con RedEMC

La integración de la aplicación con **RedEMC** es muy simple ya que puede funcionar perfectamente por fuera de otro sistema. De querer embeber la aplicación en otra página web se puede lograr haciendo uso de un *iframe*, sin requerir otra configuración. Esto es muy importante ya que uno de los objetivos trazados al inicio era posibilitar la integración con bajo esfuerzo y costo, lo cual sucedió.

9. Conclusiones

En este último capítulo se comentan las conclusiones adquiridas a lo largo del proyecto junto con ideas de otros trabajos relacionados a este que se podrían realizar en el futuro.

9.1. Conclusiones

El presente proyecto tuvo como principal objetivo investigar y realizar un prototipo del uso de base de datos de grafos para obtener datos que en una base de datos relacional son de muy difícil análisis.

En ese contexto, parte del trabajo realizado por el equipo se centró en el diseño de la base de datos, así como la correcta escritura de las consultas. Por otro lado, fue de interés del equipo presentar un prototipo que no solo permitiera la visualización del grafo, sino que le brindara al usuario la posibilidad de obtener datos sin necesidad de conocer el lenguaje de consultas.

Aunque inicialmente el alcance del proyecto no incluía la transformación de filtros de un formulario a consultas **Cypher**, el equipo entendió que era una arista del problema que valía la pena investigar. En buena medida, esta característica del proyecto es un diferencial respecto a otras opciones que existen en la web: sin saber **Cypher**, se pueden realizar consultas de relativa complejidad, que permitan obtener datos para su análisis.

Respecto al diseño del producto, se decidió mantener la lógica de consultas en el *backend* siempre que fuera posible. Es un servidor simple, que expone un único *endpoint*, y tiene solo una funcionalidad: dada un conjunto de filtros, devolver los nodos y aristas del grafo resultante. Esto último permite, potencialmente, que si EviMed decidiera desarrollar otra aplicación o servicio para consumir los datos, no sería necesario lidiar con la complejidad relativa al paradigma de la base de datos de grafos.

Pasando al análisis de la metodología de trabajo, resulta interesante la opinión de **EviMed** en este proyecto, señalando que la misma fue "excelente". Debido a que el equipo estaba integrado por una sola estudiante, garantizar avances de forma sostenida era difícil y, por otro lado, era de interés de todas las partes hacer un uso eficiente del tiempo. De esta forma, la comunicación se mantuvo en buena medida por *email* y, cuando fue necesario se coordinaron reuniones. Para el equipo

esta forma de trabajo resultó muy buena, ya que permitió presentar avances en la medida que estuvieran lo suficientemente completos como para que fuera razonable mostrar su funcionamiento. A su vez, si surgían dudas puntuales respecto a la lógica de negocio, la comunicación por *email* era rápida ya que no era necesario coordinar las agendas de las partes.

Se destaca la excelente opinión de EviMed en los tres puntos de la encuesta, que así como en los distintos intercambios, fue de satisfacción con los resultados. Para el equipo esto es muy gratificante ya que además de los objetivos formales del proyecto, era de su interés trabajar en un proyecto que agregara valor a una institución educativa con una propuesta innovadora como lo es **EviMed**.

9.2. Trabajo Futuro

Debido a la variedad de aristas cubiertas en el proyecto, que el equipo estaba conformado por una única estudiante, y el tiempo acotado, se decidió trabajar en un prototipo sólido sobre el que se pudieran realizar mejoras, así como basarse para otras aplicaciones. De esta forma es que en la presente sección se detalla trabajo futuro relacionado a profundizar en lo construido para la realidad de **EviMed**, proyectos similares para otras instituciones, y nuevas posibilidades que busquen objetivos distintos.

En primer lugar, dado que el proyecto se enfocó en las relaciones entre personas y mensajes, puede ser valioso investigar otras entidades entre las que sea de utilidad analizar cómo se relacionan. Datos relacionados a los recursos disponibles para el aprendizaje y las actividades de evaluación podrían aportar información complementaria que permita comprender mejor las interacciones en los mensajes de los foros. Por otro lado, los mensajes tienen contenido, que por el momento es ignorado (ni siquiera se exporta a la base de datos **Neo4j**). Sería interesante aplicar alguna técnica de procesamiento de lenguaje natural, a los efectos de agregar más filtros sobre los mensajes, relativos a qué etiquetas se le pueden aplicar. Un ejemplo interesante sería cómo interactúan los distintos roles según si el mensaje es de opinión o pregunta. En la misma línea, el análisis de sentimientos permitiría conocer el tono del mensaje, lo que impacta el el ciclo de vida de las comunidades en línea [7]. El análisis del grafo completo es otra arista que sería interesante investigar. De esta forma, se podrían obtener datos al respecto de qué grupos de usuarios se generan en la interacción a través de los mensajes. Para esto se consideraría grupo a todos los usuarios que interactúan con un mismo mensaje. Posteriormente, se pueden tomar métricas para la evaluación de la estructura global de la red.

Como se mencionó en la sección 6.3.4, los datos provistos presentaban

características relacionadas a cierta inconsistencia en el uso de categorías. Esto parece indicar que una posible línea de investigación a futuro es la mejora en la integridad de los datos de la base de datos relacional con la que ya contaba previamente el sistema. A los efectos del presente proyecto, luego del análisis se concluyó que no eran inconsistencias que afectaran en gran medida el desarrollo, aunque a futuro sería deseable normalizar los datos, siendo uno particularmente relevante el de las especialidades, en donde se destaca la presencia de la misma especialidad con y sin tilde, *null*, *Ninguna*, y *Sin especialidad*, *Otra* y *Otros*. Respecto a los filtros, inicialmente se optó por esconder los filtros sobre mensajes al agrupar para disminuir la complejidad de la consulta, aunque esta decisión podría ser modificada.

En la sección 7.4 se detallan los problemas en el grafo dibujado cuando la cantidad de nodos y aristas es muy grande. Cuando la cantidad de nodos es tan grande que dificulta su correcta visualización, una posible mejora sería explicar al usuario su motivo y recomendar el uso de filtros. Según el caso, también se podría evaluar la utilidad de hacer *zoom* en el grafo, a modo mejorar la visibilidad de ciertas partes. Cuando el grafo es denso y, por lo tanto, la cantidad de aristas es la raíz del problema, una solución podría ser resaltar los nodos y aristas relacionados con aquel con el que el usuario interactúa. Por otro lado, dado que no era una prioridad, se decidió realizar los esfuerzos en otros aspectos, lo que relegó la experiencia de usuario como por ejemplo considerar aspectos de accesibilidad. Si bien la aplicación se puede utilizar y en el presente informe se cuenta con un manual de usuario completo, antes de darle acceso a todos los usuarios sería deseable estudiar este aspecto más a fondo, de modo que se puedan resolver problemas frecuentes tales como falta de contraste en los colores, o tamaño de letra muy pequeño.

Si bien el presente proyecto se desarrolló en el contexto de una institución educativa determinada y, por lo tanto, la solución fue hecha a su medida, la investigación realizada y los conceptos base pueden ser de utilidad para otras aplicaciones. Inicialmente, resulta intuitivo pensar en llevar una propuesta similar a la plataforma **EVA**[1]. En especial en tiempos en los que el dictado de cursos fue únicamente virtual, tener acceso a una forma rápida y simple de obtener datos sobre la interacción de los distintos usuarios (estudiantes y docentes) hubiera sido de utilidad para, por ejemplo, detectar a aquellos que participaban poco de los foros y estaban atrasados con las actividades del curso. Aunque el ejemplo más rápido es el de la plataforma **EVA**, existe una gran variedad de problemas[2] que

[1]Entorno Virtual de Aprendizaje. Es utilizado por la Universidad de la República para los cursos de las distintas carreras que se dictan.

[2]De acuerdo a la web de **Neo4j** [21], algunos de los casos de uso incluyen detección de fraude,

son resueltos por bases de datos de grafos. Por otro lado, siempre que sea necesario realizar consultas similares repetidas veces por parte de distintos usuarios, contar con un motor que convierta filtros (entendidos por personas) en consultas (entendidas por la base de datos) será de utilidad.

Finalmente, para usuarios no familiarizados con lenguajes de consultas de bases de datos, podría resultar útil contar con una forma de obtener automáticamente la consulta **Cypher** que provee el grafo deseado a partir de filtros seleccionados. De esta forma, a través del análisis de la consulta obtenida, podrían crear nuevas consultas no previstas por los filtros, facilitando el uso y aprendizaje.

motor de recomendaciones de productos, grafos de redes sociales, prevención de crímenes, grafos de conocimientos, entre otros.

Bibliografía

[1] *.NET | Free. Cross-platform. Open Source.* https://dotnet.microsoft.com/. [Online; accedido 05-Octubre-2021].

[2] *About project boards - GitHub Docs.* https://docs.github.com/en/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards. [Online; accedido 18-Octubre-2021].

[3] *Amazon Neptune.* https://aws.amazon.com/neptune/. [Online; accedido 14-Abril-2021].

[4] *Amazon Web Services (AWS).* https://aws.amazon.com/. [Online; accedido 14-Abril-2021].

[5] *APOC.* https://neo4j.com/labs/apoc/4.1/. [Online; accedido 19-Abril-2021].

[6] *Bolt.* https://boltprotocol.org/. [Online; accedido 14-Abril-2021].

[7] *CyberEmotions.* http://www.cyberemotions.eu/. [Online; accedido 27-Setiembre-2021].

[8] *Cypher Query Language.* https://neo4j.com/developer/cypher/. [Online; accedido 11-Mayo-2021].

[9] *D3.js.* https://d3js.org/. [Online; accedido 15-Abril-2021].

[10] *DB-Engines - Knowledge Base of Relational and NoSQL Database Management Systems.* https://db-engines.com/en/. [Online; accedido 21-Setiembre-2021].

[11] *Dev.java: The Destination for Java Developers.* https://dev.java/. [Online; accedido 05-Octubre-2021].

[12] R Diestel. *Graph theory.* Springer-Verlag Berlin Heidelberg, 2017. ISBN: 978-3-662-53622-3.

[13] *Entorno Virtual de Aprendizaje.* https://proeva.udelar.edu.uy/eva/. [Online; accedido 13-Setiembre-2021].

[14] *ESLint - Pluggable JavaScript linter.* https://eslint.org/. [Online; accedido 15-Mayo-2021].

[15] *EviMed.* https://es.evimed.net/home?r_done=1. [Online; accedido 10-Marzo-2021].

[16] *Express.* https://expressjs.com/. [Online; accedido 15-Abril-2021].

[17] *Frontend Masters.* https://frontendmasters.com/. [Online; accedido 12-Mayo-2021].

[18] *Git.* https://git-scm.com/. [Online; accedido 18-Octubre-2021].

[19] *GitHub Actions Documentation - GitHub Docs.* https://docs.github.com/actions. [Online; accedido 15-Mayo-2021].

[20] *GitHub: Where the world builds software · GitHub*. https://github.com/. [Online; accedido 18-Octubre-2021].

[21] *Graph Database Use Cases & Solutions: Where to Use a Graph Database*. https://neo4j.com/use-cases/. [Online; accedido 27-Setiembre-2021].

[22] *GraphAcademy*. https://neo4j.com/graphacademy. [Online; accedido 14-Abril-2021].

[23] *historical trend of graph DBMS popularity*. https://db-engines.com/en/ranking_trend/graph+dbms. [Online; accedido 21-Setiembre-2021].

[24] *Home | Learning for a Lifetime | Stanford Online*. https://online.stanford.edu/. [Online; accedido 13-Setiembre-2021].

[25] *Importing CSV Data into Neo4j*. https://neo4j.com/developer/guide-import-csv/. [Online; accedido 14-Mayo-2021].

[26] *Importing CSV Files: Neo4j Desktop and Sandbox*. https://neo4j.com/developer/kb/import-csv-locations/. [Online; accedido 19-Abril-2021].

[27] *JavaScript.com*. https://www.javascript.com/. [Online; accedido 05-Octubre-2021].

[28] *Jest - Delightful JavaScript Testing*. https://jestjs.io/. [Online; accedido 14-Mayo-2021].

[29] *joi.dev*. https://joi.dev/. [Online; accedido 15-Mayo-2021].

[30] *Neo4j*. https://neo4j.com/. [Online; accedido 10-Marzo-2021].

[31] *Neo4j Desktop*. https://neo4j.com/developer/neo4j-desktop/. [Online; accedido 14-Abril-2021].

[32] *Neo4j developer guides*. https://neo4j.com/developer/language-guides/. [Online; accedido 14-Abril-2021].

[33] *Node.js*. https://nodejs.org/. [Online; accedido 15-Abril-2021].

[34] *Plan Ceibal*. https://www.ceibal.edu.uy/es. [Online; accedido 13-Setiembre-2021].

[35] *PM2 - Home*. https://pm2.keymetrics.io/. [Online; accedido 17-Octubre-2021].

[36] *Popoto.js*. http://www.popotojs.com/. [Online; accedido 15-Abril-2021].

[37] *Prisma LIVE - A live learning platform purpose-built for kids.* https://www.joinprisma.com/prisma-live. [Online; accedido 13-Setiembre-2021].

[38] *React*. https://reactjs.org/. [Online; accedido 15-Abril-2021].

[39] *RedEMC*. https://redemc.net/. [Online; accedido 10-Marzo-2021].

[40] *Seis grados de separación*. https://es.wikipedia.org/wiki/Seis_grados_de_separación. [Online; accedido 11-Mayo-2021].

[41] *Software Development Life Cycle Models and Methodologies - Mohamed Sami*. https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/. [Online; accedido 05-Octubre-2021].

[42] *The Go Programming Language*. https://golang.org/. [Online; accedido 05-Octubre-2021].

[43] *Titan*. https://titan.thinkaurelius.com/. [Online; accedido 14-Abril-2021].

[44] *Vue.js*. https://vuejs.org/. [Online; accedido 15-Abril-2021].

[45] *Welcome to Python.org*. https://www.python.org/. [Online; accedido 05-Octubre-2021].

Anexos

A. Manual de usuario

A.1. Base de datos

A continuación se presenta la información necesaria para la configuración de la base de datos, así como las cargas de datos necesarias.

En primer lugar se describe la configuración, incluyendo librerías e índices. En segunda instancia se comenta cómo realizar la carga inicial de datos. Finalmente, se sugiere cómo mantener los datos actualizados.

A.1.1. Configuración

Luego de instalar **Neo4j**, es necesario instalar la librería **APOC** [5], usada para las consultas que implican agrupar personas. Para funcionar correctamente, la versión a instalar debe ser la misma que la de Neo4j.

Dado que las búsquedas se realizan sobre los campos **id** de los nodos, o el campo ***name*** de ***Course***, se recomienda agregar índices a los mismos. Esto se hace simplemente corriendo la siguiente consulta:

```
1  CREATE INDEX ON :Course(id);
2  CREATE INDEX ON :Course(name);
3  CREATE INDEX ON :Message(id);
4  CREATE INDEX ON :Person(id);
```

Consulta 3: Índices.

A.1.2. Cargas de datos

Luego de creada y configurada la base de datos, se está en condiciones de cargar los datos. Antes de presentar las consultas que efectivamente cargan los datos, es necesario cumplir con ciertas precondiciones.

La primera condición es tener los archivos CSV, con todos los campos requeridos:

- **courses.csv**: *id_curso*, *nombre_curso*.

- **messages.csv**: *comment_ID*, *impressionCount*, *responseCount*, *likeCount*, *semana_curso*.
- **people.csv**: *ID*, *nombre*, *especialidad*, *pais*, *genero*, *ejercicio*.
- **included.csv**: *comment_ID*, *id_curso*.
- **liked.csv**: *id_usuario*, *id_comentario*.
- **published.csv**: *user_id*, *comment_ID*.
- **replied.csv**: *user_id*, *comment_ID*.
- **taught.csv**: *id_usuario*, *id_curso*.
- **took.csv**: *id_usuario*, *id_curso*, *estado*.

La segunda condición es tener los CSVs en una URL o ubicación del *filesystem* a la que **Neo4j** tenga acceso. Neo4j cuenta con documentación respecto a desde cuáles ubicaciones se pueden importar datos [26]. En las consultas se deberá sustituir *<UBICACIÓN_DEL_ARCHIVO>* por la ubicación del archivo, teniendo en cuenta que si es dentro del *filesystem* se debe agregar el prefijo *file* .

Dadas estas dos condiciones, se pueden ejecutar las consultas provistas en las siguientes secciones. Cabe destacar que es una consulta por cada archivo CSV.

Sea por primera vez o en actualizaciones de datos, se recomienda que el orden de las consultas siga el criterio de primero ejecutar aquellas referentes a nodos y luego a aristas (relaciones).

Si bien se sugiere realizar la carga inicial de datos con las consultas de la sección A.1.2, es posible realizarla con las de la sección A.1.2, aunque el tiempo de ejecución será mayor.

Inicial

A continuación se presentan las consultas **Cypher** que se deben ejecutar para cargar los datos.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
CREATE (c:Course { id: line.id_curso, name: line.nombre_curso })
```

Consulta 4: Cursos.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
CREATE (m:Message { id: line.comment_ID, impressionCount:
↪  toInteger(line.impressionCount), responseCount:
↪  toInteger(line.responseCount), likeCount:
↪  toInteger(line.likeCount), week: line.semana_curso })
```

Consulta 5: Mensajes.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
CREATE (p:Person { id: line.ID, name: line.nombre, country: CASE
↪  trim(line.pais) WHEN "NULL" THEN null ELSE line.pais END,
↪  gender: CASE trim(line.genero) WHEN "NULL" THEN null ELSE
↪  line.genero END, specialty: CASE trim(line.especialidad) WHEN
↪  "NULL" THEN null ELSE line.especialidad END, yearsActive: CASE
↪  trim(line.ejercicio) WHEN "NULL" THEN null ELSE line.ejercicio
↪  END })
```

Consulta 6: Personas.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.comment_ID })
MATCH (c:Course { id: line.id_curso })
CREATE (m)-[:INCLUDED_IN]->(c)
```

Consulta 7: Mensaje incluido en curso.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (p:Person { id: line.id_usuario })
MATCH (c:Course { id: line.id_curso })
CREATE (p)-[:TAUGHT]->(c)
```

Consulta 8: Persona dictó curso.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (p:Person { id: line.id_usuario })
MATCH (c:Course { id: line.id_curso })
CREATE (p)-[:TOOK { status: CASE trim(line.estado) WHEN "A" THEN
↪  "Audit" WHEN "B" THEN "Behind" WHEN "O" THEN "Out" WHEN "T"
↪  THEN "On Track" END }]->(c)
```

Consulta 9: Persona tomó curso.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.comment_ID })
MATCH (p:Person { id: line.user_id })
CREATE (p)-[:PUBLISHED]->(m)
```

Consulta 10: Persona publicó mensaje.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.comment_ID })
MATCH (p:Person { id: line.user_id })
CREATE (p)-[:REPLIED]->(m)
```

Consulta 11: Persona respondió mensaje.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.id_comentario })
MATCH (p:Person { id: line.id_usuario })
CREATE (p)-[:LIKED]->(m)
```

Consulta 12: Persona le gustó mensaje.

Posteriores

En caso de que los CSVs contengan únicamente los nodos y aristas que deben ser agregados a la base de datos, se puede ejecutar las mismas consultas detalladas en la sección anterior. Si, por el contrario, los CSVs presentan todos los datos, se

debe ejecutar las nuevas consultas propuestas a continuación, ya que se evitará crear nodos o aristas que ya existieran y se actualizarán aquellos que presenten cambios.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MERGE (c:Course { id: line.id_curso })
ON CREATE
  SET c.name = line.nombre_curso
ON MATCH
  SET c.name = line.nombre_curso
```

Consulta 13: Cursos.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MERGE (m:Message { id: line.comment_ID })
ON CREATE
  SET
    m.impressionCount = toInteger(line.impressionCount),
    m.responseCount = toInteger(line.responseCount),
    m.likeCount = toInteger(line.likeCount),
    m.week = line.semana_curso
ON MATCH
  SET
    m.impressionCount = toInteger(line.impressionCount),
    m.responseCount = toInteger(line.responseCount),
    m.likeCount = toInteger(line.likeCount),
    m.week = line.semana_curso
```

Consulta 14: Mensajes.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MERGE (p:Person { id: line.ID })
ON CREATE
  SET
    p.name = line.nombre,
    p.country = CASE trim(line.pais) WHEN "NULL" THEN null ELSE
↪   line.pais END,
    p.gender = CASE trim(line.genero) WHEN "NULL" THEN null ELSE
↪   line.genero END,
    p.specialty = CASE trim(line.especialidad) WHEN "NULL" THEN
↪   null ELSE line.especialidad END,
    p.yearsActive = CASE trim(line.ejercicio) WHEN "NULL" THEN
↪   null ELSE line.ejercicio END
ON MATCH
  SET
    p.name = line.nombre,
    p.country = CASE trim(line.pais) WHEN "NULL" THEN null ELSE
↪   line.pais END,
    p.gender = CASE trim(line.genero) WHEN "NULL" THEN null ELSE
↪   line.genero END,
    p.specialty = CASE trim(line.especialidad) WHEN "NULL" THEN
↪   null ELSE line.especialidad END,
    p.yearsActive = CASE trim(line.ejercicio) WHEN "NULL" THEN
↪   null ELSE line.ejercicio END
```

Consulta 15: Personas.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.comment_ID })
MATCH (c:Course { id: line.id_curso })
MERGE (m)-[:INCLUDED_IN]->(c)
```

Consulta 16: Mensaje incluido en curso.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (p:Person { id: line.id_usuario })
MATCH (c:Course { id: line.id_curso })
MERGE (p)-[:TAUGHT]->(c)
```

Consulta 17: Persona dictó curso.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (p:Person { id: line.id_usuario })
MATCH (c:Course { id: line.id_curso })
MERGE (p)-[r:TOOK]->(c)
ON CREATE
  SET r.status = CASE trim(line.estado) WHEN "A" THEN "Audit" WHEN
↪  "B" THEN "Behind" WHEN "O" THEN "Out" WHEN "T" THEN "On Track"
↪  END
ON MATCH
  SET r.status = CASE trim(line.estado) WHEN "A" THEN "Audit" WHEN
↪  "B" THEN "Behind" WHEN "O" THEN "Out" WHEN "T" THEN "On Track"
↪  END
```

Consulta 18: Persona tomó curso.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.comment_ID })
MATCH (p:Person { id: line.user_id })
MERGE (p)-[:PUBLISHED]->(m)
```

Consulta 19: Persona publicó mensaje.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.comment_ID })
MATCH (p:Person { id: line.user_id })
MERGE (p)-[:REPLIED]->(m)
```

Consulta 20: Persona respondió mensaje.

```
:auto USING PERIODIC COMMIT 1000
LOAD CSV WITH HEADERS
FROM <UBICACIÓN_DEL_ARCHIVO>
AS line
MATCH (m:Message { id: line.id_comentario })
MATCH (p:Person { id: line.id_usuario })
MERGE (p)-[:LIKED]->(m)
```

Consulta 21: Persona le gustó mensaje.

A.2. Uso de la aplicación

En esta sección se describe la aplicación y su funcionamiento.

Nota: para la generación de imágenes se utilizó la base de datos provista por EviMed, anonimizando datos para a fin de preservar la privacidad de los mismos.

La interfaz de usuario es simple, contando con tres etapas: autenticación, consulta, y visualización del resultado. La etapa de autenticación consiste de simplemente dos campos para indicar usuario y contraseña utilizados en la plataforma **RedEMC** (A.1). Al validar las credenciales, se pasa a la etapa de consulta, que cuenta con tres vistas, según se busque por curso, persona, o mensaje. En tanto, la etapa de visualización cuenta con dos vistas según se haya agrupado o no.

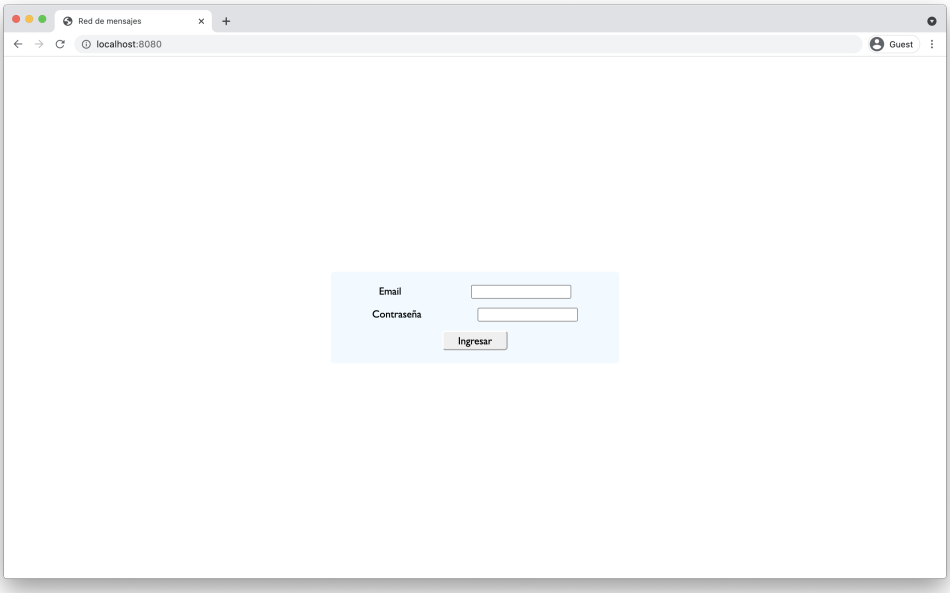

Figura A.1: Vista de autenticación.

A continuación se presentan las vistas de consulta por curso (A.2), persona (A.3), y mensaje (A.4), así como las de visualización agrupando (A.5), y por separado (A.6).

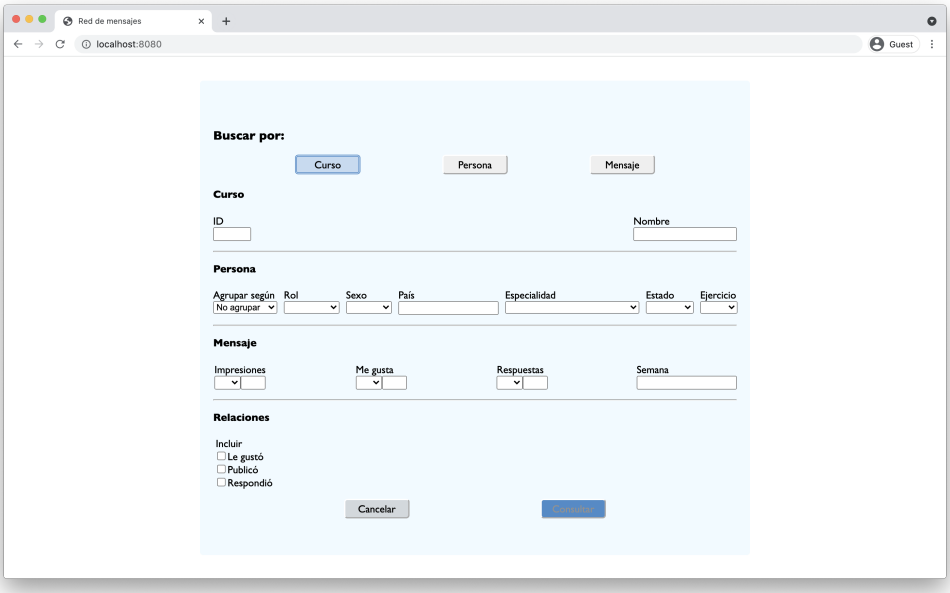

Figura A.2: Vista de consulta por curso.

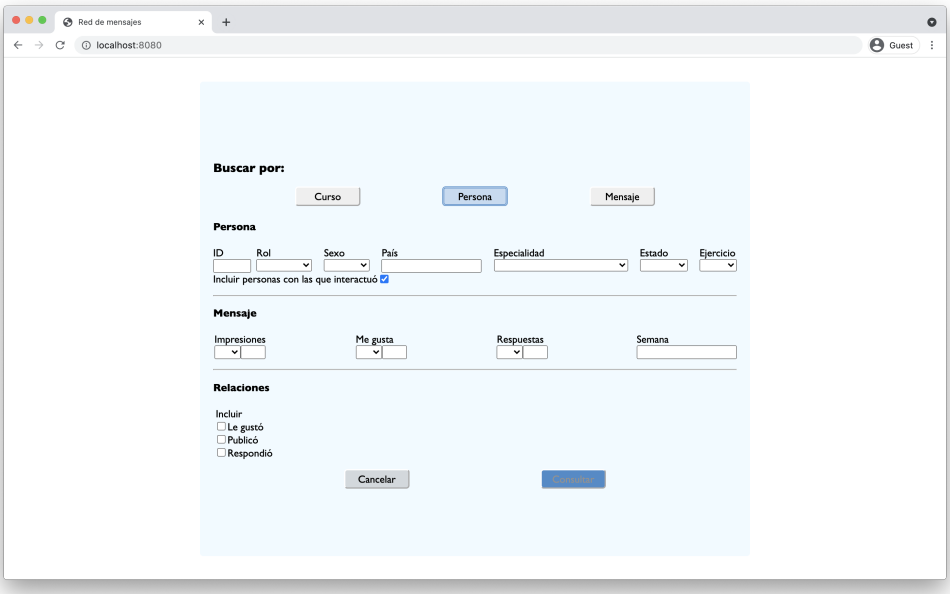

Figura A.3: Vista de consulta por persona.

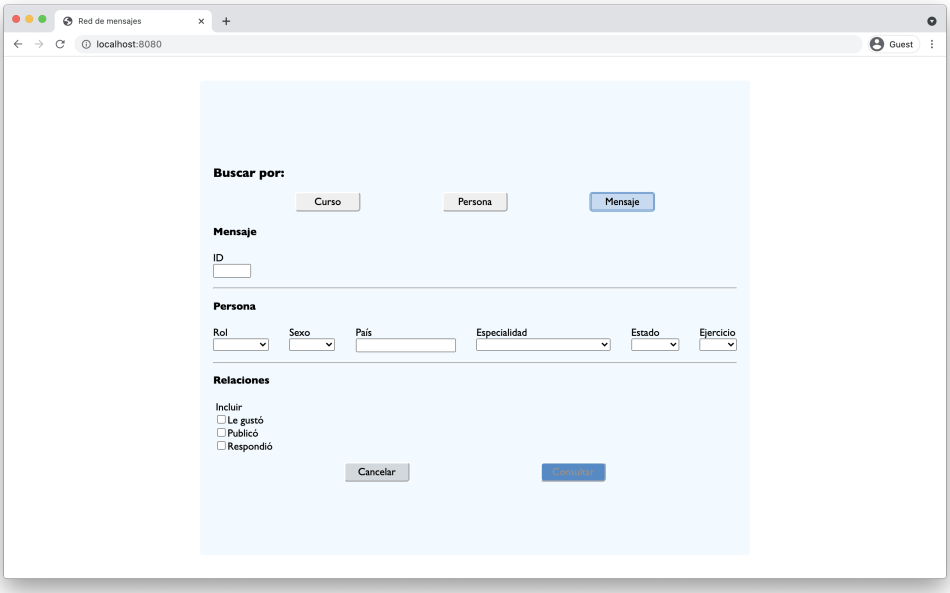

Figura A.4: Vista de consulta por mensaje.

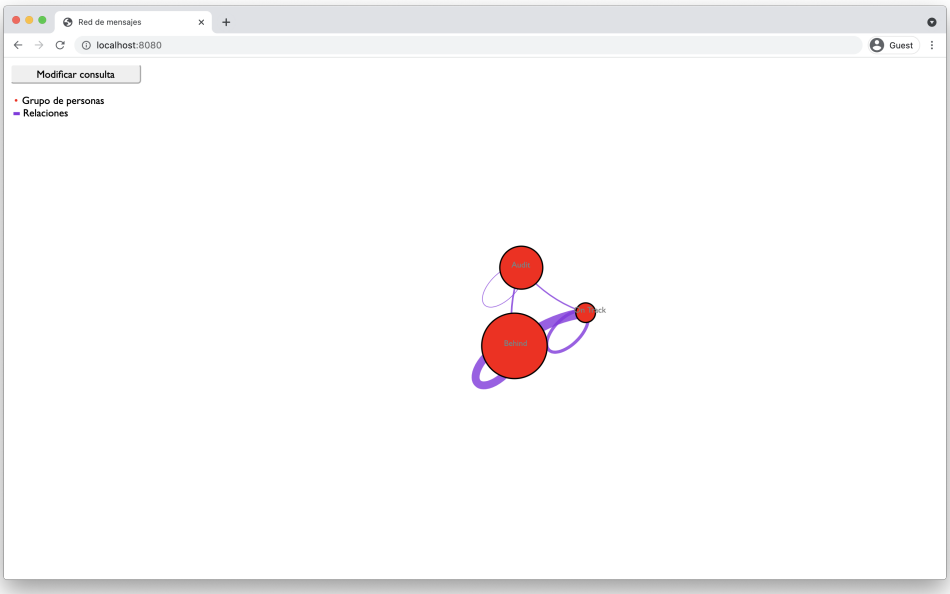

Figura A.5: Vista del grafo con usuarios agrupados.

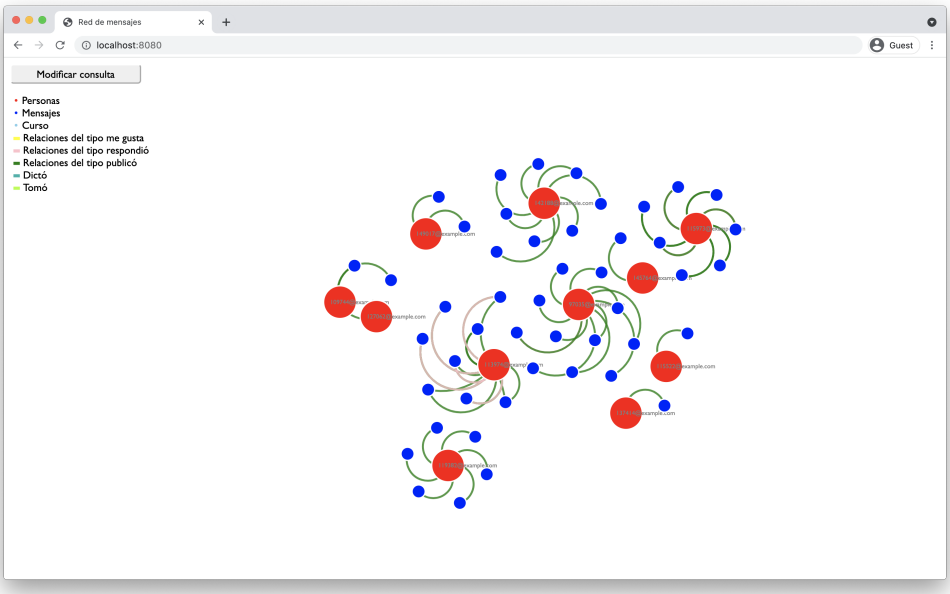

Figura A.6: Vista del grafo con usuarios por separado.

A modo de ejemplificar su uso, se presentarán posibles casos de uso, para luego explicar cómo obtener la información buscada, y mostrar cómo se ve el grafo resultado.

Nota: para la realización de este manual se utilizaron los datos reales proporcionados por **EviMed**.

A.2.1. Por curso

En esta sección se introducen las búsquedas por **nombre** o **id** del curso, para luego mostrar la posibilidad de agrupar los individuos.

Al decidir hacer una consulta por curso, será necesario proveer el **nombre** o la **id** del mismo. Mientras ambos campos estén vacíos, el botón **Consultar** permanecerá deshabilitado.

Búsqueda por nombre

Podría ser de interés, ver la interacción de los docentes con los mensajes publicados para el curso de nombre *vacunaspediatria*.

Para ello, en la vista de consulta, se selecciona **Por curso**, escribe el nombre, y elige el rol *Docente*:

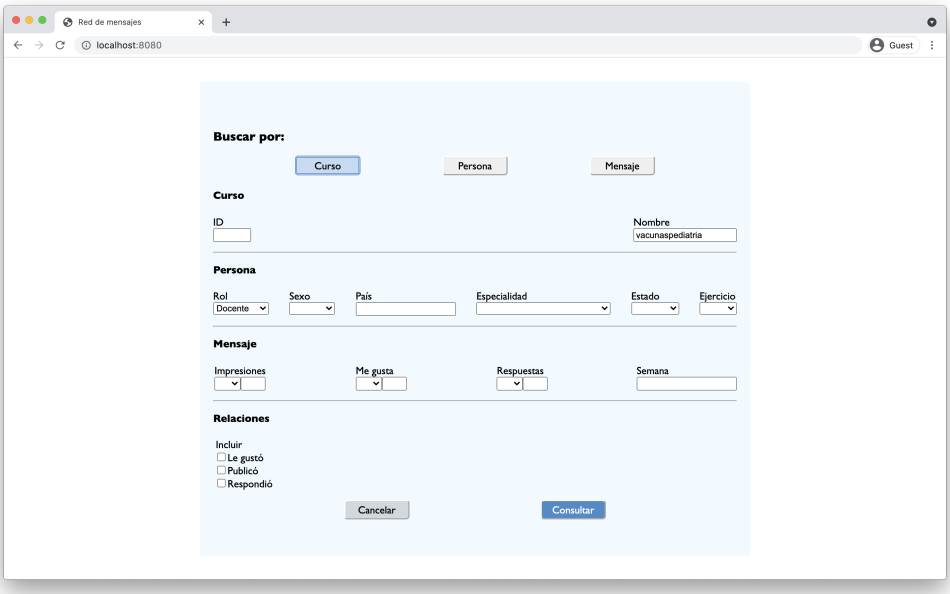

Figura A.7: Consulta inicial.

Al hacer *click* en **Consultar**, el resultado es el siguiente:

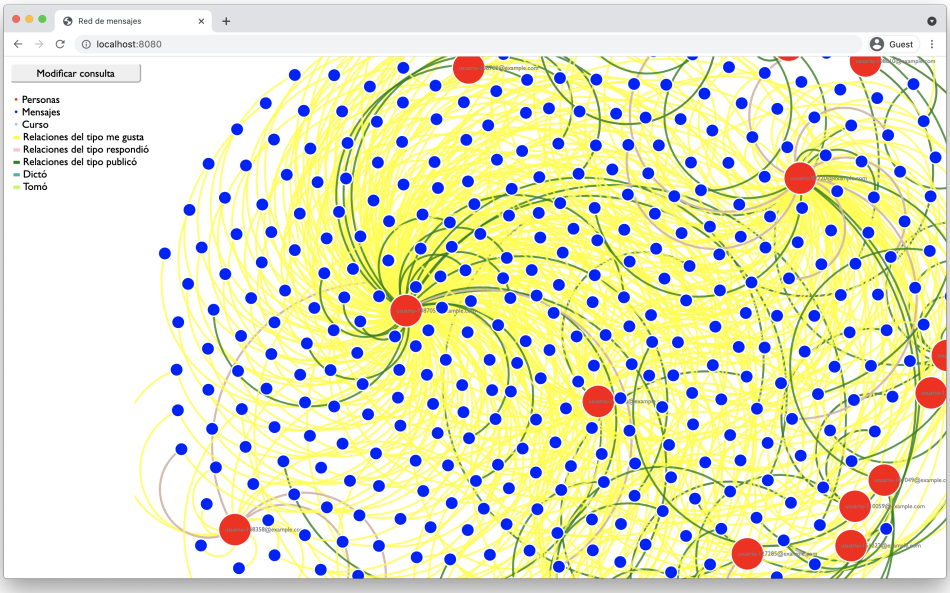

Figura A.8: Grafo resultado inicial.

Se puede ver que la cantidad de aristas amarillas dificulta su visibilidad. Al revisar las referencias (arriba a la izquierda), se sabe que estas aristas corresponden a relaciones del tipo “le gusta”.

Lo anteriormente mencionado proporciona información cualitativa de que los

docentes suelen indicar que les gustan comentarios, pero dificulta interpretar cómo son las demás interacciones.

Con el objetivo de refinar la consulta, se decide ajustar los filtros. Para ello es necesario hacer *click* en **Modificar consulta** (arriba a la izquierda, junto a las referencias).

Una vez allí, se seleccionan todas las relaciones excepto "le gustó" y nuevamente se hace *click* en **Consultar**, obteniendo un nuevo grafo:

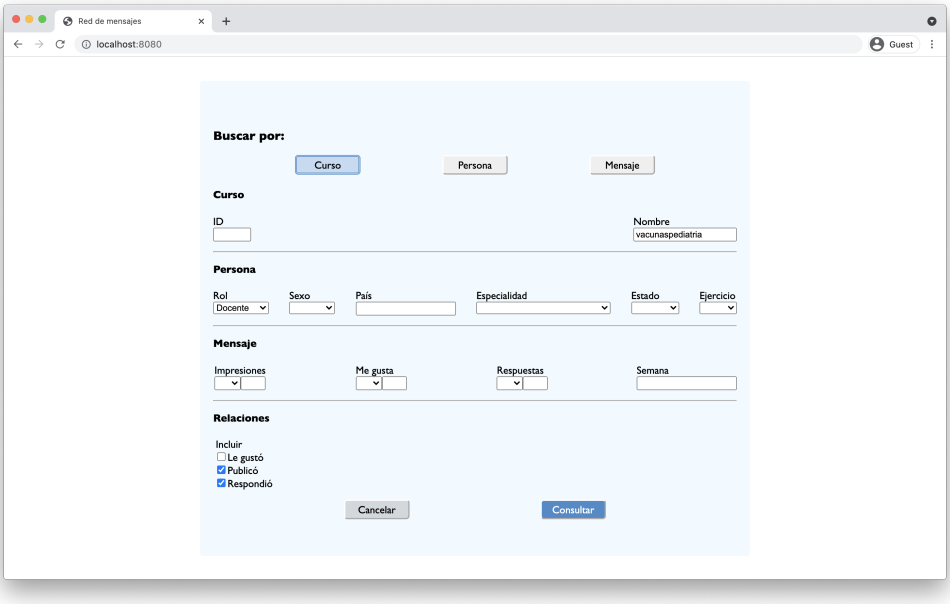

Figura A.9: Consulta refinada.

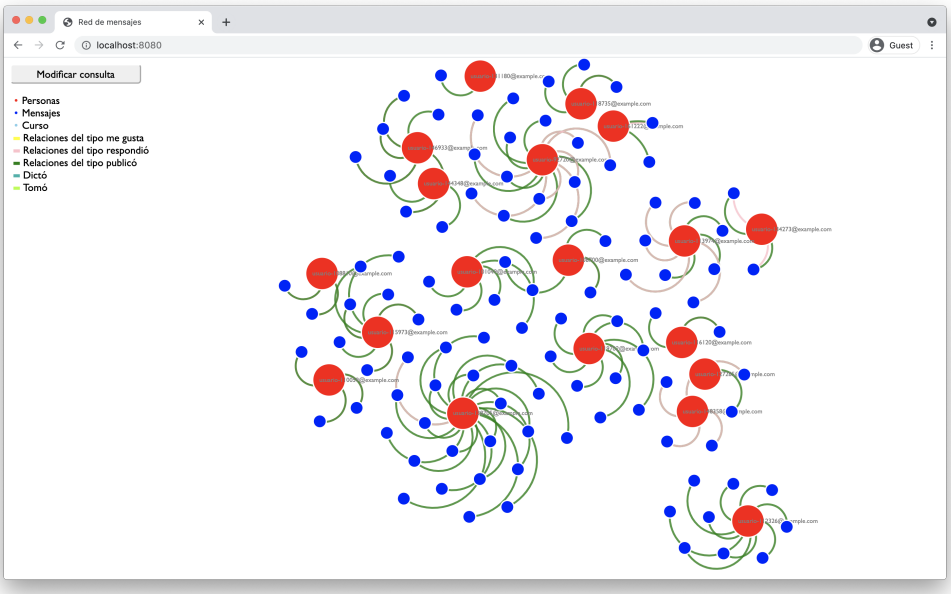

Figura A.10: Grafo resultado refinado.

Finalmente, se tiene un grafo que proporciona información respecto a cómo interactuaron los docentes con demás usuarios a través los mensajes a lo largo del curso. Inclusive, se puede sacar conclusiones respecto a cómo hace uso de la herramienta cada docente, ya que se puede apreciar que hay quienes mayoritariamente publicaron mensajes, mientras otros únicamente respondieron.

Búsqueda por ID

Anteriormente se explicó cómo extraer información de un curso usando su nombre. Dado que hay cursos en la plataforma RedEMC que se dictan en más de un idioma y todos ellos tienen el mismo nombre, resulta de interés poder realizar consultas haciendo uso del **id** para poder distinguirlos.

En este caso, se presenta una consulta sobre cómo interactúan los usuarios del curso *nefrometabolismo2020* para su versión en portugués (se supondrá **ID** 1234).

En primera instancia, la interacción que se tiene en cuenta es entre hombres, sin importar su rol en el curso.

Análogamente a lo realizado en la búsqueda por nombre, se selecciona **Por curso**, escribe la **id** del curso y selecciona el género *Hombre*:

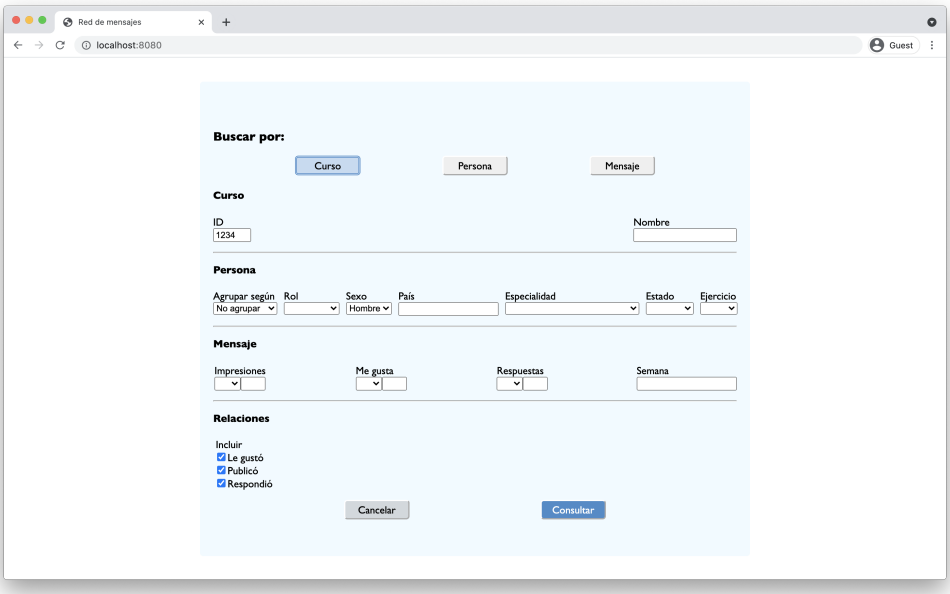

Figura A.11: Consulta.

El grafo que se obtiene es el siguiente:

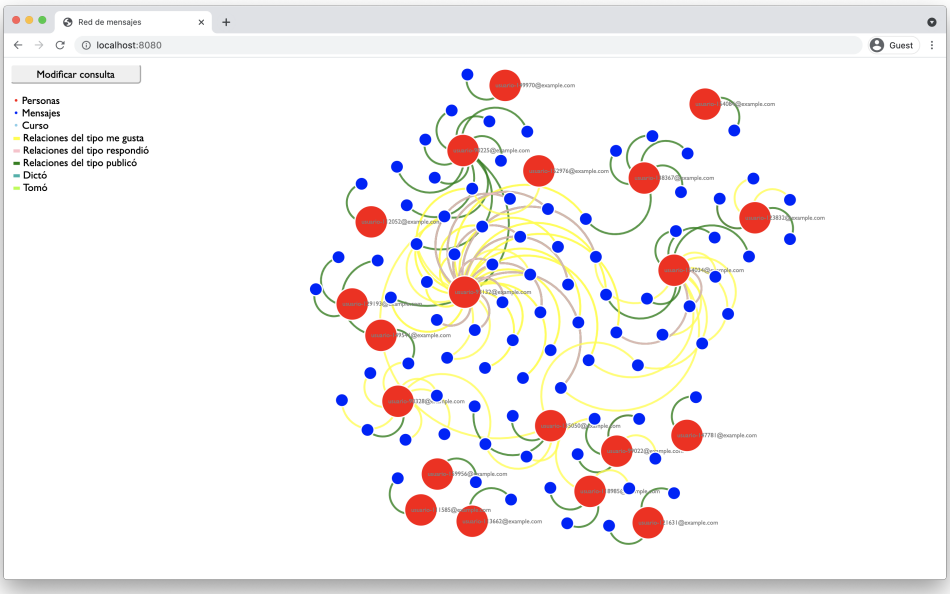

Figura A.12: Grafo resultado.

Filtro por mensajes

Los ejemplos presentados hasta el momento han hecho uso de filtros sobre las personas. En esta oportunidad, el filtro será sobre los mensajes.

Se tomará el curso *nefrometabolismo2020* para su versión en portugués (se supondrá **ID** 1234).

La información buscada es qué usuarios tuvieron acciones sobre los mensajes con más de 250 impresiones en la semana 6 del curso.

En el panel de consultas, se selecciona **Por curso**, escribe la **ID**, selecciona el símbolo $\geq$, a su derecha la cantidad de impresiones y, finalmente, la semana:

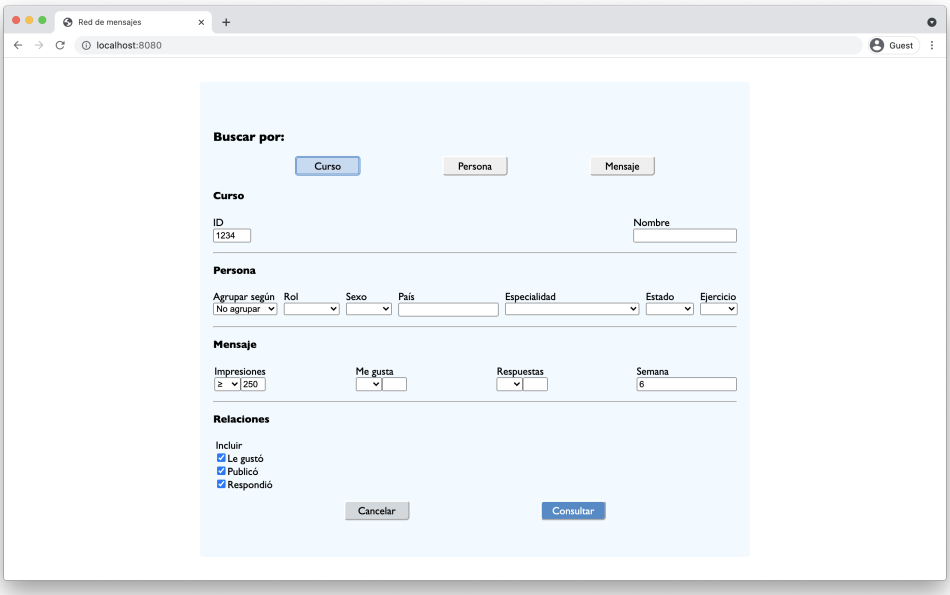

Figura A.13: Consulta.

El grafo que se obtiene es el siguiente:

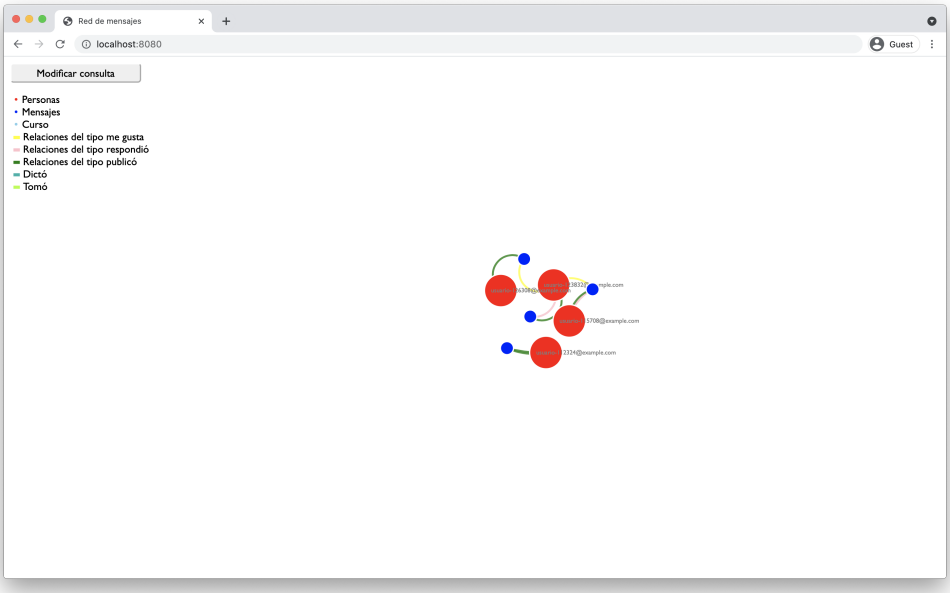

Figura A.14: Grafo resultado.

Se observa que mensajes con alta cantidad de impresiones no generaron interacción entre usuarios en este curso. Esta conclusión se desprende de dos hechos: solo cuatro usuarios están involucrados en estos cuatro mensajes, y solo uno de ellos tuvo una acción distinta de publicar.

Nota: los filtros por cantidad de "me gustaz respuestas son análogos al de cantidad de impresiones.

Agrupando resultados

Existen casos en los que interesa saber cómo se comportan grupos de personas entre sí.

Para ejemplificar este escenario, se busca cómo se relacionaron las personas con especialidad en enfermedades infecciones de acuerdo a su estado en el curso. Para esto se usa el curso *forocovid19*.

A modo de realizar la consulta, se selecciona **Por curso**, escribe el **nombre**, agrupa según *Estado*, y elige filtrar por especialistas en *Enfermedades infecciosas*:

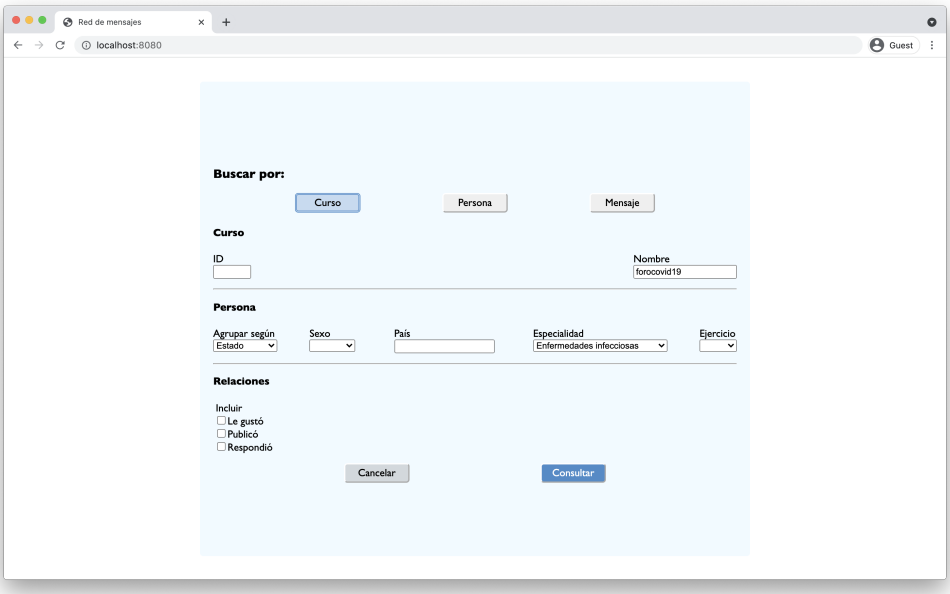

Figura A.15: Consulta.

El grafo que se obtiene es el siguiente:

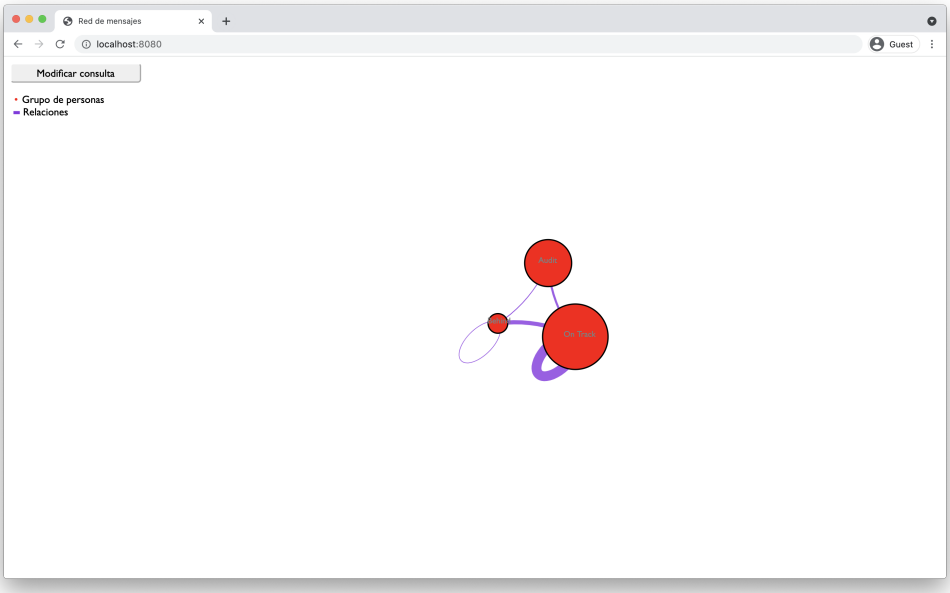

Figura A.16: Grafo resultado.

Los distintos tamaños de los nodos son representativos de la cantidad de usuarios pertenecientes a ese grupo. De esta forma, rápidamente se toma conocimiento de que la mayoría de los usuarios llevan el curso al día.

El espesor de las aristas tiene una semántica similar a la del tamaño del nodo.

Las aristas más gruesas implican que hay mayor cantidad de relaciones.

En este grafo no está presente un nodo para los usuarios que están *Out*. Cuando esto sucede, tiene dos posibles explicaciones: no existen usuarios que tengan ese estado, o sí existen pero no interactuaron con otros.

Con el objetivo de responder esa interrogante, se puede realizar una nueva consulta que no agrupe y únicamente filtre según estado del usuario:

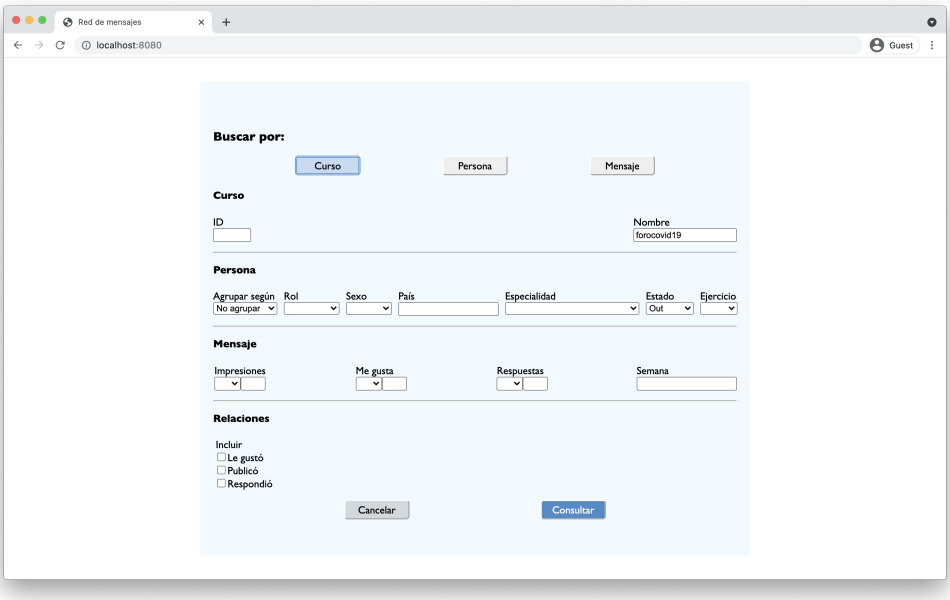

Figura A.17: Consulta de usuarios *Out*.

El grafo resultado permite identificar el motivo de lo observado anteriormente al agrupar:

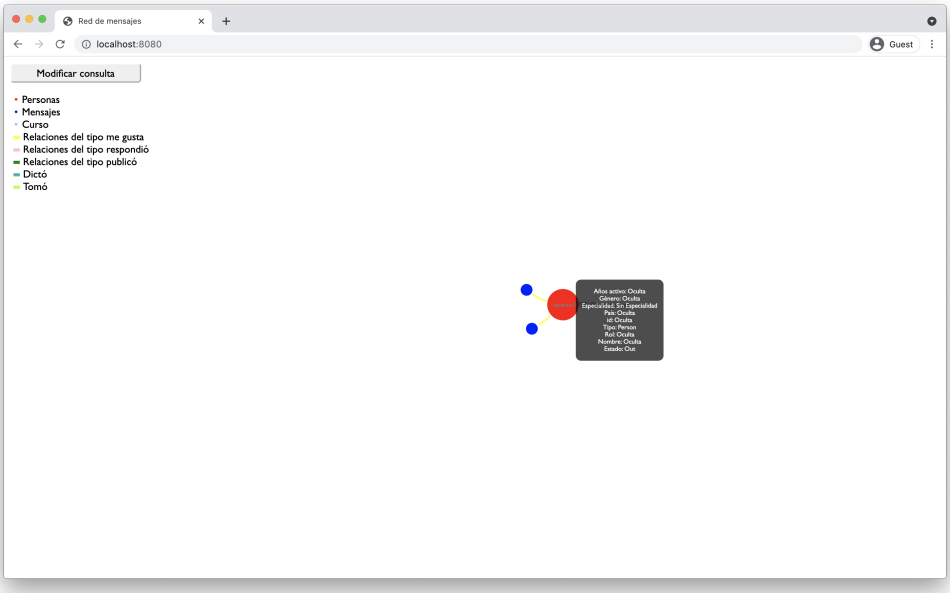

Figura A.18: Grafo resultado de usuarios *Out*.

Hay un único usuario con estado *Out* en el curso y es uno que no es especialista en enfermedades infecciosas, por eso no había un nodo para dicho estado al agrupar.

También se puede agrupar por especialidad o país de origen, de forma análoga a la realizada en este ejemplo.

A.2.2. Por persona

Otro posible caso de uso es analizar la interacción de un usuario en particular en todos sus cursos. Para ello se cuenta con la funcionalidad de buscar por la **id** del usuario, nuevamente con distintos filtros disponibles.

Para este ejemplo se supondrá un usuario con **id** 1234.

En el panel de consulta, se selecciona **Por persona**, escribe la **id**:

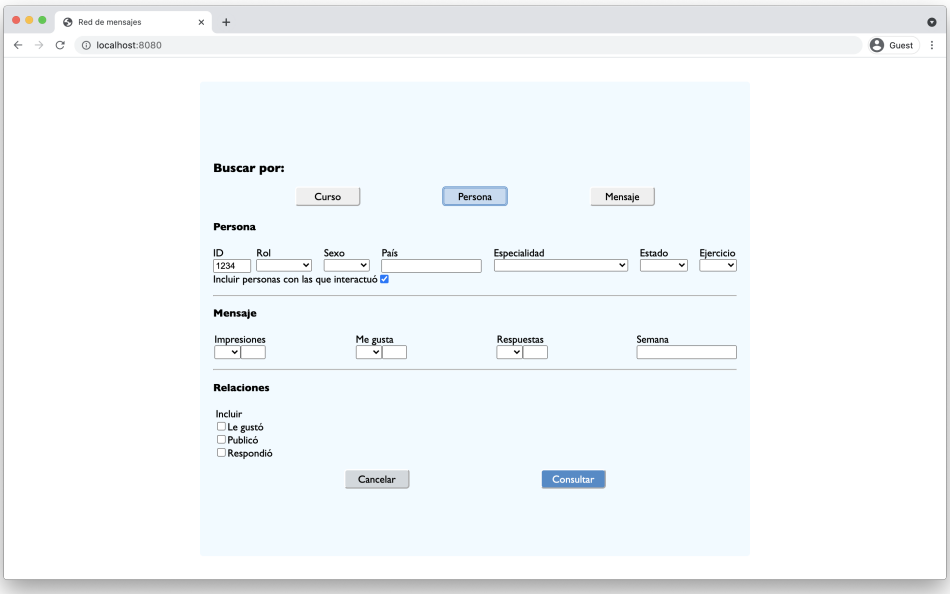

Figura A.19: Consulta inicial.

Al hacer *click* en **Consultar**, el resultado es el siguiente:

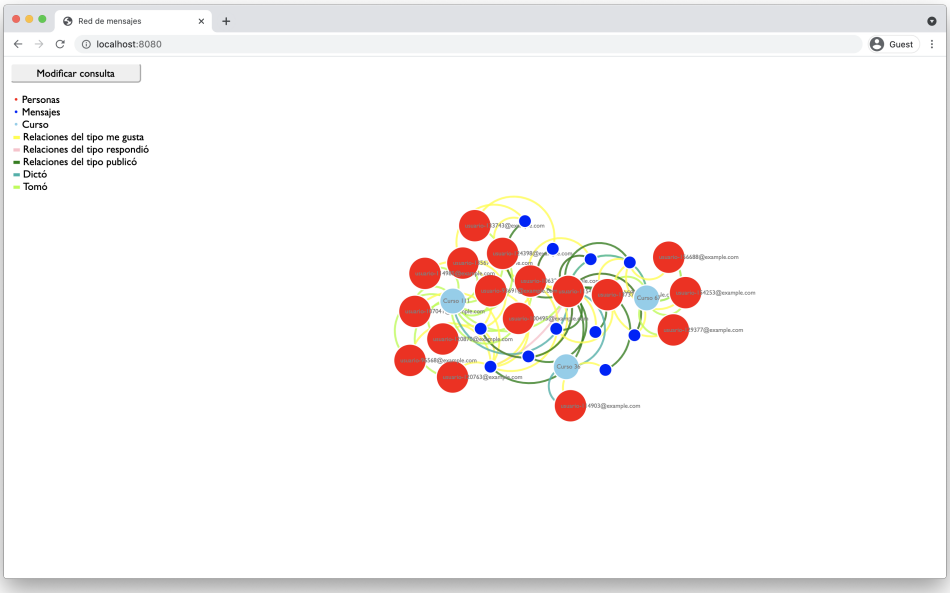

Figura A.20: Grafo resultado inicial.

En la figura A.19 se ve que por defecto está seleccionada la opción de incluir todos aquellos usuarios con los que interactuó el de la **id** provista. Debido a esto, el grafo resultado presenta más de un nodo de tipo persona.

Si bien el grafo anterior proporciona información valiosa como ser con quiénes y

de qué manera interactuó el usuario elegido, puede ser difícil ver exactamente qué tipo de acciones suele tener él con respecto a los mensajes. Para ello se realiza una nueva consulta, desmarcando la opción mencionada:

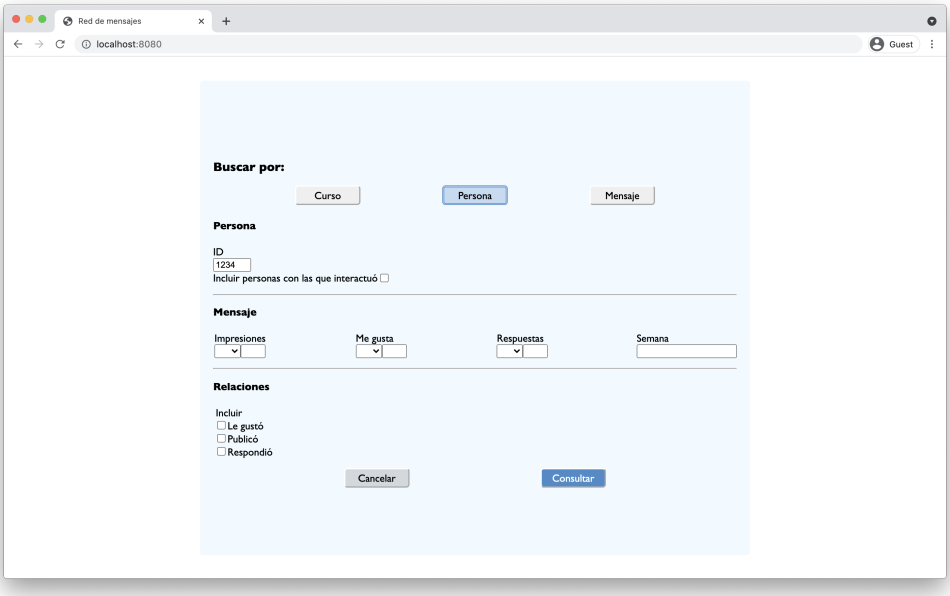

Figura A.21: Consulta sin otros usuarios.

El grafo resultado es:

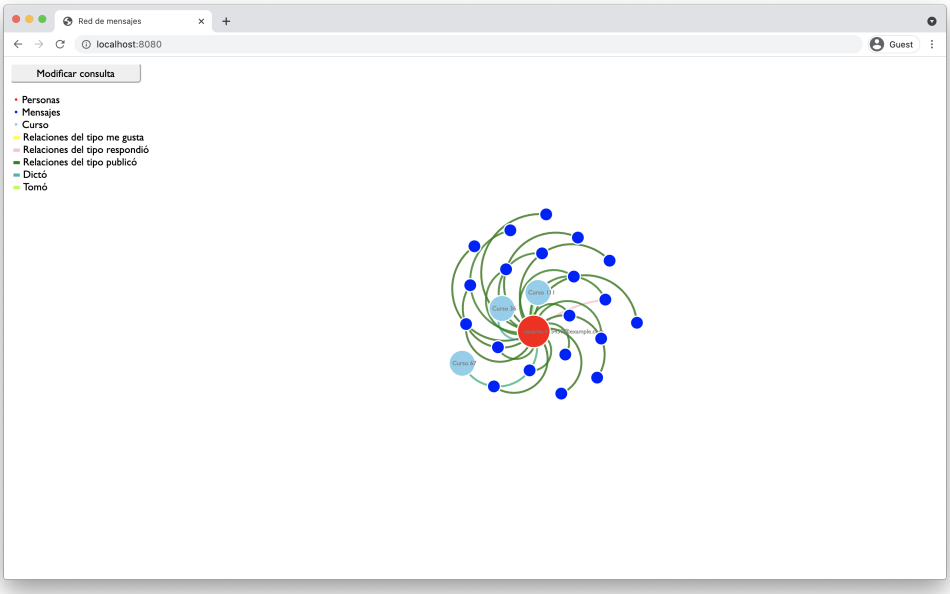

Figura A.22: Grafo resultado sin otros usuarios.

Comparando las imágenes A.19 y A.21 se aprecia que, al desmarcar la opción de incluir a otros usuarios, se eliminan las demás opciones disponibles en la

sección *Persona*. Esto se debe a que esos filtros aplican sobre las personas con las que el usuario elegido haya interactuado.

Finalmente, los filtros sobre mensajes y relaciones tienen el mismo comportamiento que el explicado en la sección A.2.1.

A.2.3. Por mensaje

Por último, la aplicación proporciona una forma de realizar búsquedas por la **id** de un mensaje. Esto facilita ver la interacción entre usuarios a través de un mensaje en particular.

Para este ejemplo se supondrá un mensaje con **id** 1234.

En el panel de consulta, se selecciona **Por mensaje**, escribe la **id**:

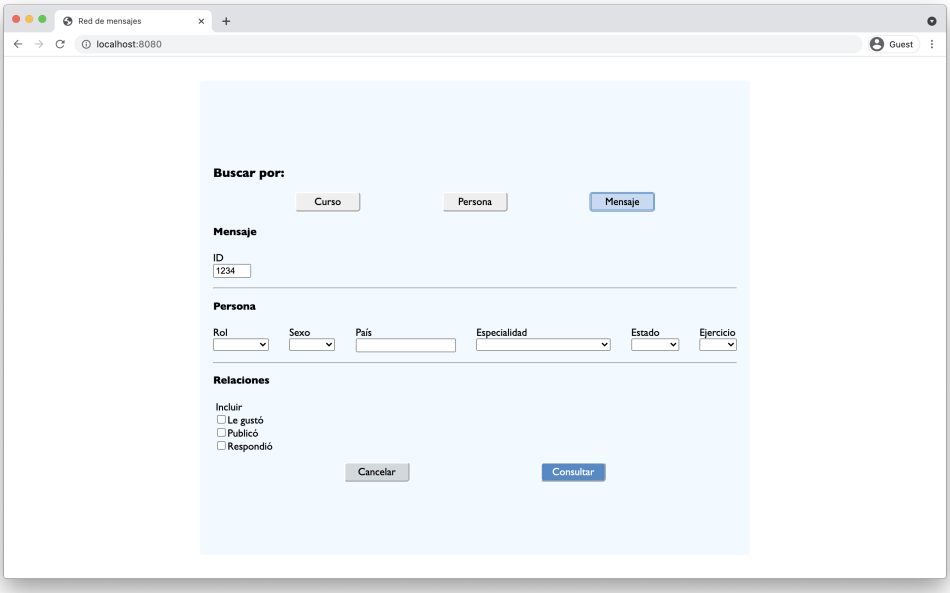

Figura A.23: Consulta.

El grafo resultado de esa consulta es:

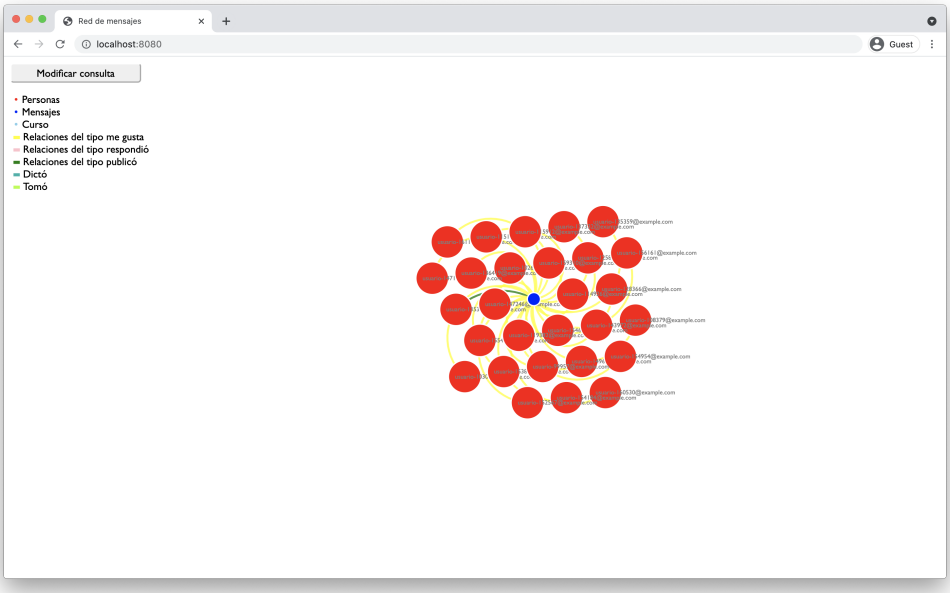

Figura A.24: Grafo resultado.

Los filtros sobre personas y relaciones tienen el mismo comportamiento que el explicado en la sección A.2.1.

B. API

En este anexo se documenta la API construida como parte del proyecto y utilizada por la aplicación.

B.1. Autenticación

Antes de poder acceder a los datos el usuario debe autenticarse con el objetivo de obtener un *token* que le permita realizar futuras *requests* al servidor.

POST /api/login

B.1.1. Cabeceras

Content-Type: application/json

B.1.2. Parámetros

- **email (requerido)**: string no vacío. *Email* del usuario en RedEMC.
- **password (requerido)**: string no vacío. Contraseña del usuario en RedEMC.

B.1.3. Respuestas

- **200**: Usuario autenticado correctamente.
- **400**: Alguno de los parámetros tiene errores.
- **401: Not allowed**: Credenciales provistas no son correctas.

B.1.4. Ejemplo

Request

```
curl --location --request POST 'localhost:3000/api/login' \
--header 'Content-Type: application/json' \
--data-raw '{
    "email": "leticia@ejemplo.com",
```

```
    "password": "password"
}'
```

Respuesta

```
{
    "token": "token-de-respuesta"
}
```

B.2. Consultas

Importante: Se debe contar con un *token* para poder hacer uso de este *endpoint*.

Permite obtener los resultados de la base de datos a partir de los *query parameters* provistos.

GET /api/query

B.2.1. Cabeceras

Authorization: Bearer <**TOKEN**>

B.2.2. Query parameters

- **courseId (opcional)**: número. **ID** del curso por el que se quiere buscar.
- **courseName (opcional)**: string. Nombre del curso por el que se quiere buscar.
- **personId (opcional)**: número. **ID** del usuario por el que se quiere buscar.
- **messageId (opcional)**: número. **ID** del mensaje por el que se quiere buscar.
- **country (opcional)**: string no vacío. Nombre del país por el cual filtrar personas.

- **specialty (opcional)**: string no vacío. Nombre de la especialidad por la cual filtrar personas.
- **status (opcional)**: string, únicamente *audit*, *behind*, *on_ track*, o *out*. Estado del estudiante en el curso.
- **role (opcional)**: string, únicamente *student* o *teacher*. Rol de la persona en el curso.
- **gender (opcional)**: string, únicamente *female* o *male*. Género de la persona.
- **yearsActive (opcional)**: string, únicamente *0-5*, *6-15*, *16-25*, o *25*. Rango de años en actividad de la persona.
- **grouped (opcional)**: string, únicamente *country*, *specialty*, o *status*. Criterio por el que se quiere agrupar personas.
- **impressionCount (opcional)**: número. Cantidad de impresiones del mensaje.
- **impressionCountSign (opcional)**: string, únicamente *less*, *less_ or_ equal*, *equal*, *equal_ or_ greater*, o *greater*. Criterio de comparación entre el parámetro *impressionCount* y la cantidad de impresiones del mensaje.
- **likeCount (opcional)**: número. Cantidad de "me gusta" del mensaje.
- **likeCountSign (opcional)**: string, únicamente *less*, *less_ or_ equal*, *equal*, *equal_ or_ greater*, o *greater*. Criterio de comparación entre el parámetro *likeCount* y la cantidad de "me gusta" del mensaje.
- **responseCount (opcional)**: número. Cantidad de respuestas del mensaje.
- **responseCountSign (opcional)**: string, únicamente *less*, *less_ or_ equal*, *equal*, *equal_ or_ greater*, o *greater*. Criterio de comparación entre el parámetro *responseCount* y la cantidad de respuestas del mensaje.
- **week (opcional)**: número. Semana a la que pertenece el mensaje.
- **include_other_people (opcional)**: booleano. Decisión de incluir otras personas al filtrar por una.
- **relationships (opcional)**: string de relaciones (*liked*, *published*, o *replied*) separadas por coma. Relaciones entre usuarios y mensajes a ser incluídas.

B.2.3. Respuestas

- **200**: Usuario autenticado correctamente.
- **400**: Alguno de los *query parameters* tiene errores.
- **401: Not allowed**: Token inválido o el usuario no tiene permisos.
- **500: There was an error**: Error al procesar la *request* y realizar la consulta.

Estructura sin agrupar

- **people**: arreglo de personas. Dentro de *properties* están las propiedades de la persona, cual fueran presentadas en 6.3.1.
- **messages**: arreglo de mensajes. Dentro de *properties* están las propiedades del mensaje, cual fueran presentadas en 6.3.1.
- **relationships**: arreglo de relaciones entre entidades. Los campos *start* y *end* referencian la *identity* de las entidades (única, ya que es creada internamente por **Neo4j**).
- **courses**: arreglo de cursos. Dentro de *properties* están las propiedades del curso, cual fueran presentadas en 6.3.1. No disponible al realizar la búsqueda por courseId o courseName.

Estructura agrupando

- **nodes**: arreglo de personas agrupadas. Incluye las siguientes propiedades:
  - **name**: Nombre del grupo.
  - **count**: Cantidad de personas en el grupo.
  - **key**: Clave numérica que permite identificar al grupo.
- **relationships**: arreglo de relaciones entre entidades. Incluye las siguientes propiedades:
  - **start**: Número que referencia la *key* de uno de los grupos.
  - **end**: Número que referencia la *key* del otro grupo.
  - **key**: Nombre de la relación. Generado usando el nombre de ambos grupos separados por un guión.
  - **count**: Cantidad de relaciones entre ambos grupos.

B.2.4. Ejemplo

Request

```
curl --location --request GET
↪  'http://localhost:3000/api/query?courseName=nombre-curso' \
--header 'Authorization: Bearer <TOKEN>'
```

Respuesta sin agrupar


```
{
    "people": [
        {
            "identity": 1,
            "labels": [
                "Person"
            ],
            "properties": {
                "name": "leticia@example.com",
                "country": "UY",
                "specialty": "Medicina General",
                "yearsActive": "0-5",
                "id": "1",
                "gender": "F"
            },
            "inCourse": {
                "role": "Estudiante",
                "status": "Audit"
            }
        }
    ],
    "messages": [
        {
            "identity": 2,
            "labels": [
                "Message"
            ],
            "properties": {
                "likeCount": 15,
                "responseCount": 8,
                "week": "1",
                "id": "12",
                "impressionCount": 25
            }
        }
    ],
    "relationships": [
        {
            "identity": 3,
            "start": 1,
            "end": 2,
            "type": "PUBLISHED",
            "properties": {}
        }
```

```
    ]
}
```

Respuesta agrupando

```
{
    "nodes": [
        {
            "name": "PY",
            "count": 2,
            "key": 0
        },
        {
            "name": "UY",
            "count": 4,
            "key": 1
        }
    ],
    "relationships": [
        {
            "start": 1,
            "end": 1,
            "key": "UY-UY",
            "count": 8
        },
        {
            "start": 0,
            "end": 1,
            "key": "PY-UY",
            "count": 2
        }
    ]
}
```