



X



Índice:

1. Introducción
2. Modelado de Datos
3. Investigación
4. Funciones
5. Bonita
6. Manual de Instalación
7. Manual de Usuario
8. Conclusiones
9. Referencias, Bibliografía y Anexo

1. *Introducción:*

La teoría de grafos es una disciplina matemática originada en el siglo dieciocho, y a lo largo de los últimos años, el ser humano se ha dado cuenta que podía aplicar esa disciplina a escenarios reales.

En los últimos años se han desarrollado las herramientas que nos acercan a esa relación grafo con la realidad y las que trabajaremos en este proyecto serán Neo4Java y Bonita Soft, las cuales nos permite la primera, modelar bases de datos mediante grafos, y la segunda permite modelar procesos de negocio, entre otras aplicaciones [1].

Una vez ya establecidas las herramientas, denotaremos la temática del proyecto, para poder aprovechar al máximo estas herramientas. Crearemos una base de datos centrada en procesos de negocio, permitiéndonos que una vez creada la base de datos en Neo4Java, podamos trasladarla a Bonita Soft y poder mostrar esos procesos de negocio que no son visibles en un grafo. En este caso he decidido crear una base de datos de una red social, debido en gran parte a que son las mayores beneficiarias en los últimos años de esta técnica de modelado de grafos, y luego ver que procesos de negocio y sus flujos que podamos mostrar en Bonita Soft (especial mención a las asignaturas de ADDA y Matemáticas Discretas, que fomentan todo esto, en 2º de Ingeniería del Software).

Universidades privadas como INESEM en Granada, en su artículo [2], indican ya desde 2017, que las redes sociales han increpado el uso de Big Data, y poder generar grafos a partir de estos datos. Obviamente, no centramos el proyecto en el Big Data, de ahí la pequeña cantidad de información, también por realizarse por una sola persona. El artículo marca también lo que dijo Mark Zuckerberg, ya en 2007, donde aclaraba que el uso como él dice de “Grafos Sociales”, conectan a las personas y todas sus interacciones, con actividades, perfiles, amigos, mensajes, etc.

La metodología para aplicar será, primero voy a mostrar una investigación personal sobre las herramientas en Internet y YouTube, para documentarme y contextualizarme de las herramientas que voy a manejar. Modelare un UML, para poder tener una base donde empezar, y empezar a modelar los grafos en la herramienta correspondiente, y concluiremos creando la base de datos y visualizándola en varios procesos de negocio.

En el proyecto encontraremos como, generar una mini red social, nos permitirá sacar conclusiones de las relaciones interpersonales de los participantes de la web y poder sacar conclusiones, de estas, como que poder recomendar a otros usuarios, o ver si alguna publicación está en tendencia.

En resumen, es interesante y útil, la herramienta Neo4Java y BonitaSoft, para poder tener una idea rápida, que rendimiento tiene la aplicación a estudiar.

2. *Modelado de Datos:*

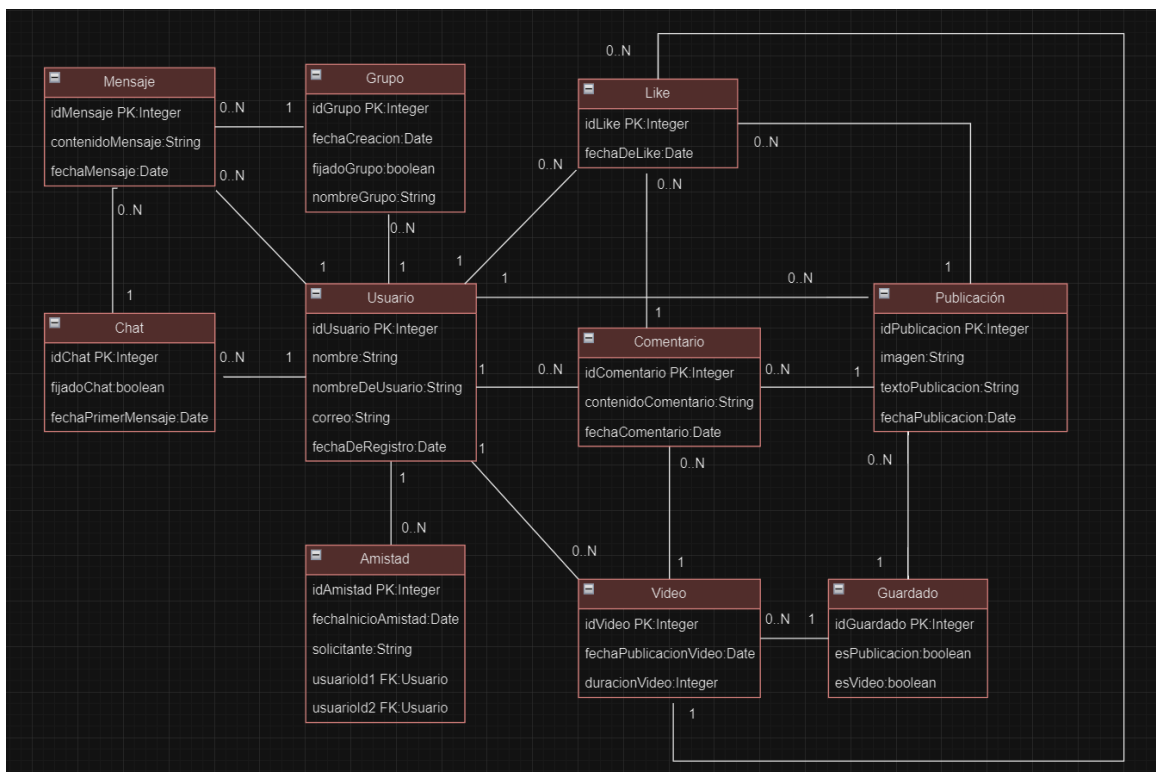


Imagen 1: Modelado UML, base para empezar a crear la base de datos de grafos

[3] <https://drive.google.com/file/d/1pFnXDtfqG5fEUQ6VsCrsS1BcSCztdrv6/view?usp=sharing>

Hemos creado un modelado clásico, que puede ser aplicado en una base de datos SQL, pero obviamente, cuando importemos las clases con aristas y vértices, se vera de forma alterada, y podremos decir que el modelado en grafos está basado en este, y también nos ayudara en las tablas que crearemos para poder visualizar todo mejor también Bonita Soft.

(Si una palabra esta entre comillas (")) hace referencia a un elemento del entorno Neo4Java)

Este modelado al ser el de una red social, coincide con los estándares vistos en las redes sociales más comunes, incluyendo 'Video', como entidad, que hace referencia a la tendencia de las nuevas aplicaciones de tener unas secciones de videos cortos y deslizantes hacia el siguiente.

Como se puede apreciar es una base sencilla con 10 entidades: 'Usuario', 'Mensaje', 'Chat', 'Grupo', 'Like', 'Amistad', 'Comentario', 'Publicación', 'Video' y 'Guardado'.

No todos de ellos serán vértices ya que posiblemente, intentare hacer que 'Amistad' solo sea implementado como arista, al igual que 'Like'.

Además, las relaciones, me pueden ser útiles para cuando veamos los grafos se corresponda y se visualice de un usuario por ejemplo todos sus chats, siendo un vértice de tipo 'Usuario', unido a muchos vértices de tipo 'Chat'.

La base de datos resultante en Neo4Java:

Primero crearemos los 20 usuarios que corresponderán a nuestro grafo, con el comando

```
neo4j$ CREATE (Eduardo:Persona {surname: 'Pizarro López',nickname: 'theRock',mail: 'edu@gmail.com',fechaRegistro: datetime('2024-04-20T01:00:00Z'}})
```

Imagen 2: Comandos en Neo4Java

CREATE (Eduardo:Persona {surname: 'Pizarro López',nickname: 'theRock',mail: 'edu@gmail.com',fechaRegistro: datetime('2024-04-20T01:00:00Z'}}) [1]

Como resultado nos dará:

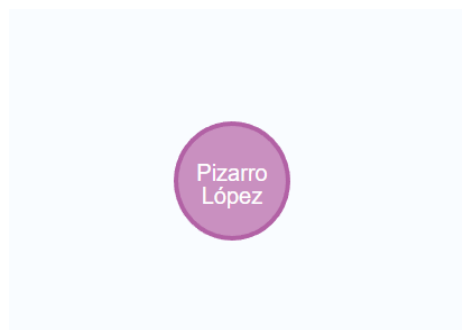


Imagen 3: Creación del primer vértice del grafo

La primera apreciación que tuvimos fue que, pensando que colocar 'Eduardo:Persona', cogería 'Eduardo' como nombre, y como funciona Neo4Java lo escoge como variable a buscar, aunque da igual, se podrá añadir mas adelante atributos a algun nodo si es necesario con el comando `MATCH(n{nickname:'theRock'}) SET n.nombre = 'Edu'`

Con el comando `MATCH(n) WHERE n.nickname='theRock' RETURN n`, podremos en un futuro seleccionar solo una arista.

La generación del resto de vertices como es lo mismo pero cambiando los valores he hecho uso de la Inteligencia artificial de chatGPT para ahorrarme esos minutos de sustitución, en general solo lo usare salvo casos especificos para ahorrarme el mismo proceso repetidas veces [\[2'\]](#)

Se visualiza todo mediante `MATCH(n) RETURN(n)`:

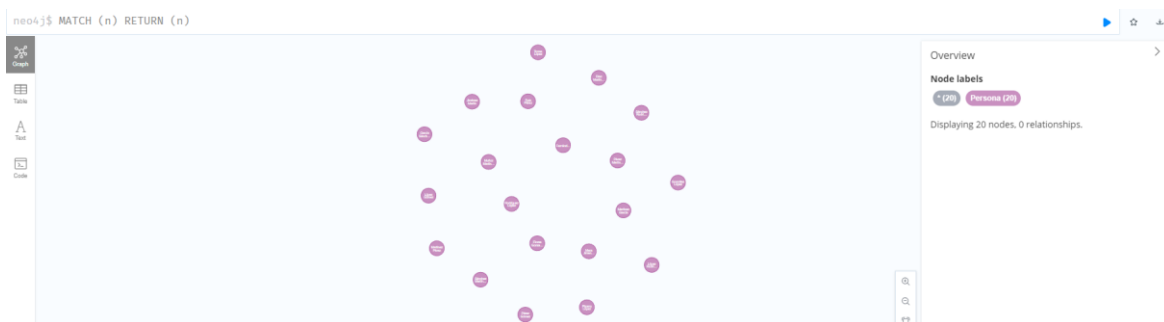


Imagen 4: Generación de todos los vértices de tipo 'Persona' del proyecto

Como no podemos visualizar correctamente lo que he creado, cogeremos uno cualquiera, en este caso, voy a escoger a 'juani', por su 'nickname', con el comando: `MATCH (p:Persona) WHERE p.nickname = 'juani' RETURN p`

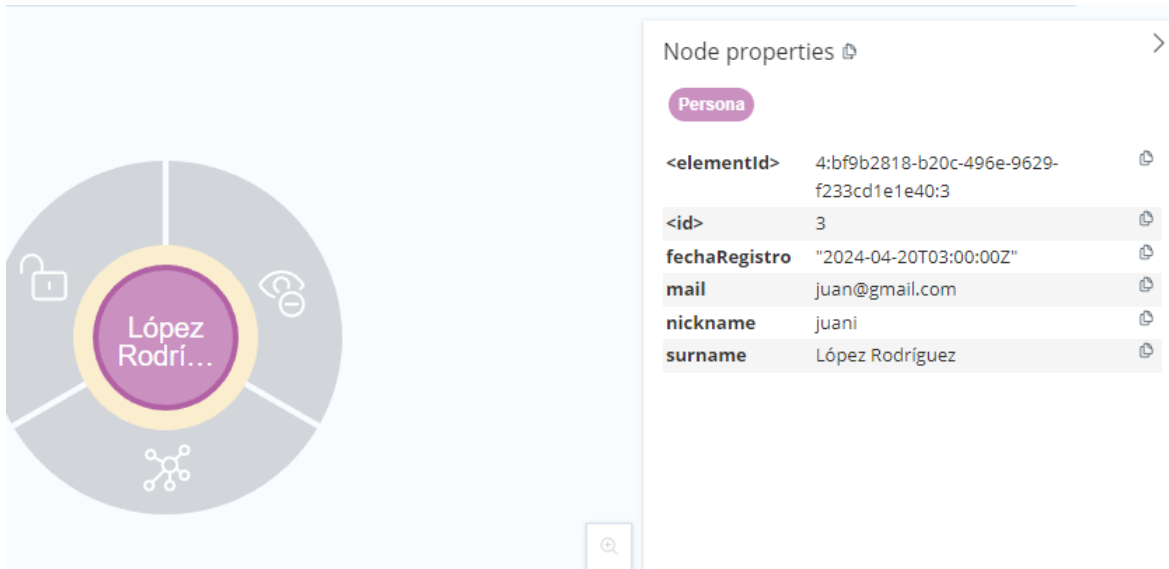


Imagen 5: Visualizamos los atributos del vértice de tipo 'Persona'

Ahora crearemos las aristas para generar las amistades en el UML definidas, con el comando:

`MATCH(n{nickname:'lolita3'}),(m{nickname:'sari'}) CREATE (n)-[:Amistad]->(m):`

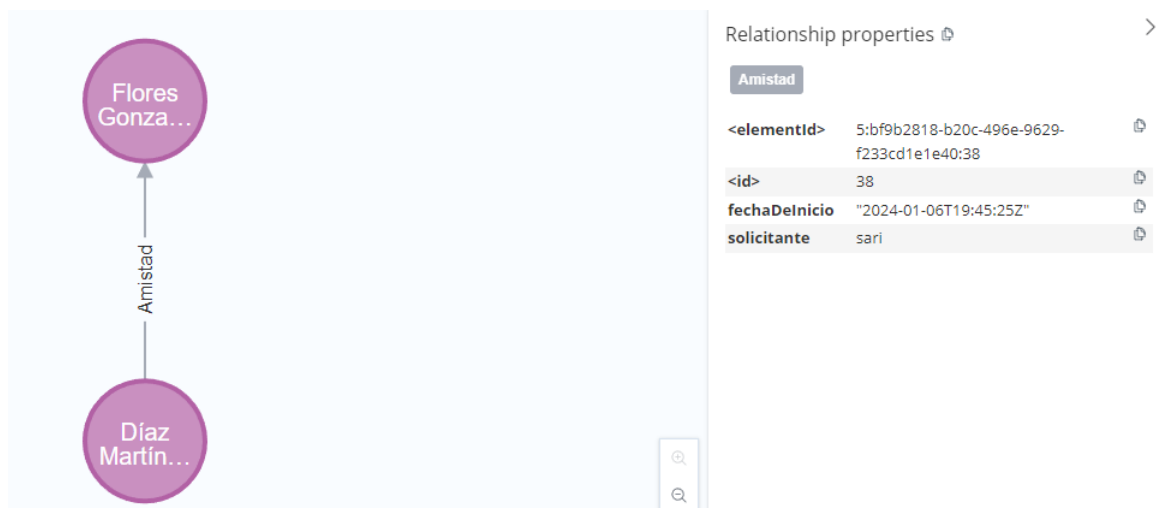


Imagen 6: Primera arista creada del grafo del tipo 'Amistad'

Ahora crearemos aristas aleatorias, entre los vértices de forma que, tengamos vértices con más o menos amistades, tampoco queremos saturarlo todo, así que lo limitaremos a 50, que será suficiente para luego hacer consultas interesantes:

```

1 MATCH (p1:Persona), (p2:Persona)
2 WHERE p1 <> p2 AND rand() < 0.2
3 WITH p1, p2
4 LIMIT 50
5
6 CREATE (p1)-[:Amistad{
7     fechaDeInicio: datetime({year: 2022 + toInteger(rand()*3), month: toInteger(rand()*12)+1, day: toInteger(rand()*28)+1, hour: toInteger(rand()*
8     24), minute: toInteger(rand()*60), second: toInteger(rand()*60)}), solicitante: CASE WHEN rand() < 0.5 THEN p1.nickname ELSE p2.nickname END
9     }->(p2)
10

```

Set 100 properties, created 50 relationships, completed after 13 ms.

Imagen 7: Generación de aristas de forma aleatoria para que las personas sean amigas entre ellas

Comandos:

MATCH (p1:Persona), (p2:Persona) WHERE p1 <> p2 AND rand() < 0.2 WITH p1, p2 LIMIT 50

CREATE (p1)-[:Amistad{fechaDeInicio:datetime({year:2022+toInteger(rand()*3),month:toInteger(rand()*12)+1,day:toInteger(rand()*28)+1,hour:toInteger(rand()*24),minute:toInteger(rand()*60),second:toInteger(rand()*60)}),solicitante:CASE WHEN rand()<0.5 THEN p1.nickname ELSE p2.nickname END}->(p2) [3]

Como resultado nos proporciona lo siguiente:



Imagen 8: Todas las personas y sus amistades en el grafo

Ahora crearemos de la misma manera, si alguien ha chateado con otra persona:

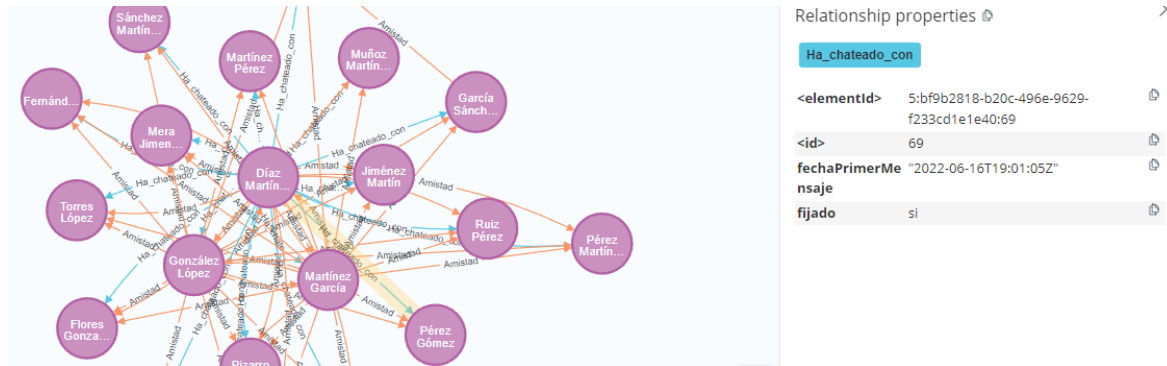


Imagen 9: Generación de las aristas de personas que se hayan hablado

Le hemos cambiado los colores a las aristas para poder diferenciarlas

Hemos creado los vértices de tipo 'Grupo', que hace referencia a grupos de mensajes entre varias personas:

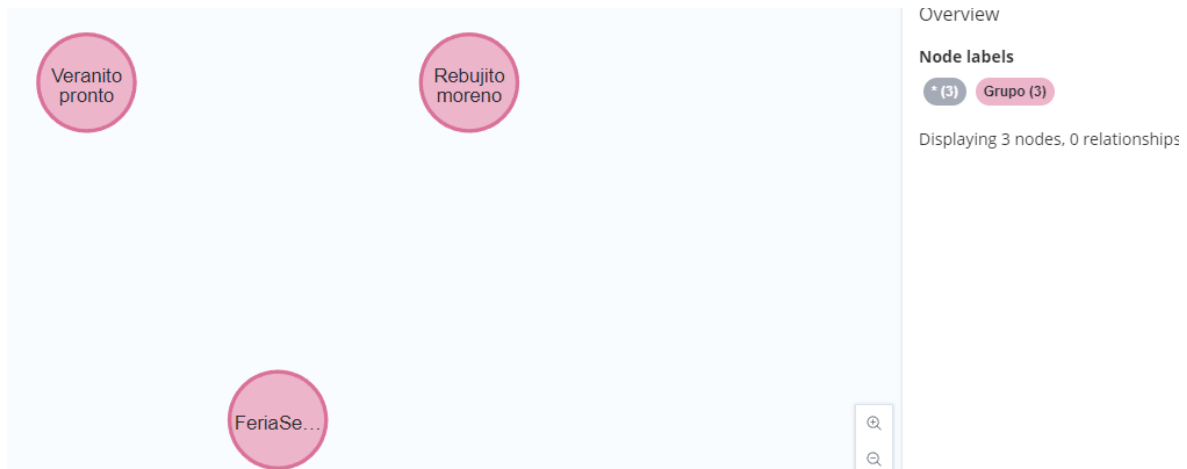


Imagen 10: Creación de los vértices del tipo 'Grupo'

Ahora crearemos aristas a 3, 4 y 7 personas para cada grupo de forma que el atributo antes creado de fijado ahora le pertenezca a esta arista:

```

1 MATCH (grupo1:Grupo {nombreGrupo: 'FeriaSevillaaa 2024'}), (persona:Persona)
2 WITH grupo1, persona
3 LIMIT 4
4 CREATE (grupo1)-[:Pertenece_a {fechaUnion: datetime({year: 2024, month: 4, day: 23})}]->(persona)

```

Imagen 11: Comando para que uno de los grupos, tenga como participantes 4 personas

Ese mismo código se usa con el resto de los grupos donde haremos que 4 personas (en los siguientes casos 3 y 7) pertenezcan a ese grupo y después le añadimos el atributo fijado, que tiene más sentido ya que así relacionamos a una 'Persona' y al 'Grupo' y no solo al 'Grupo':

```

1 MATCH (g:Grupo) REMOVE g.fijado
2

```

Set 3 properties, completed after 6 ms.

```

MATCH (grupo1:Grupo {nombreGrupo: 'FeriaSevillaaa 2024'}), (persona:Persona)
WITH grupo1, persona
LIMIT 4
CREATE (grupo1)-[:Pertenece_a {fechaUnion: datetime({year: 2024, month: 4, day: 23}), fijado: 'si'}]->(persona)

```

Imagen 12: Se elimina el atributo fijado de los grupos, y se le añade a las aristas de pertenencia

Ahora ya podemos mostrar lo que tendríamos por ahora:

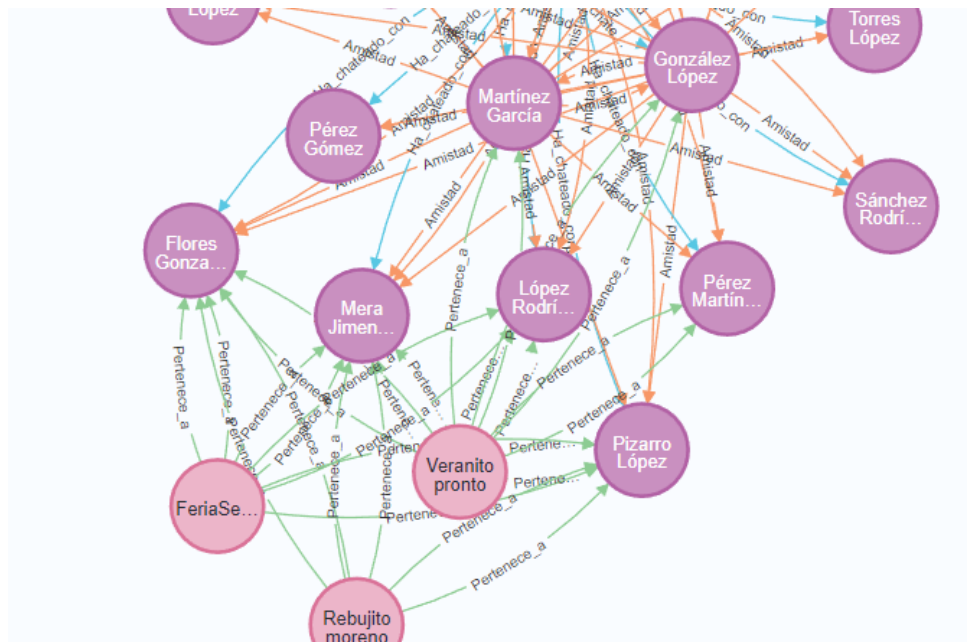


Imagen 13: Grafo resultante hasta ahora, con 'Grupo' y 'Persona' y sus aristas

Creamos los vértices de tipo 'Publicación', en este caso para no generar más caos, crearemos dos, y los enlazaremos con aristas del tipo 'Han publicado' a un solo 'Usuario', en este caso de 'nickname' 'luisi':

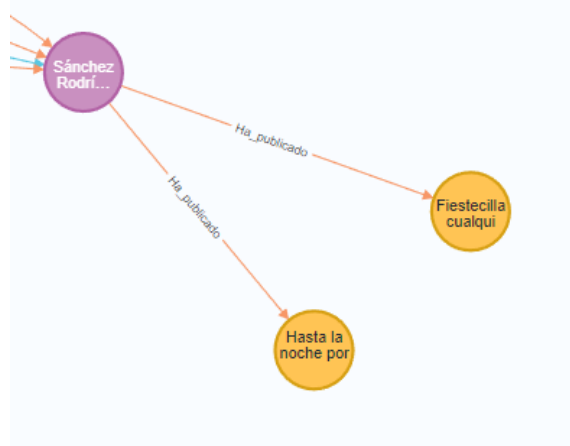


Imagen 14: Creación de las publicaciones, y las aristas que lo relacionan con persona

Comandos:

La creación de publicación es como hemos visto hasta ahora, con los atributos, 'descripción'(lo visible en el grafo), 'fechaPublicación', 'imagen', 'nombrePublicación'.

```
MATCH (p:Publicacion {descripcion: 'Fiestecilla cualquiera con la que poder bailar',  
fechaPublicacion: datetime('2024-04-23T00:00:00Z'))},
```

```
(persona:Persona {nickname: 'luisi'})
```

```
CREATE (persona)-[:Ha_publicado]->(p)
```

```
MATCH (p:Publicacion {descripcion: 'Hasta la noche por desgracia',  
fechaPublicacion: datetime('2024-04-23T00:00:00Z'))},
```

```
(persona:Persona {nickname: 'luisi'})
```

```
CREATE (persona)-[:Ha_publicado]->(p)
```

Vale ahora crearemos los vértices de tipo Video, le añadiremos aristas de 'Ha publicado':

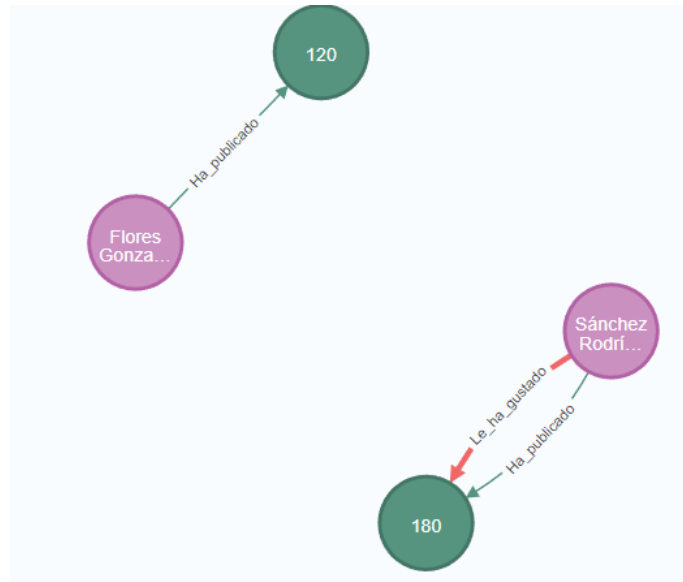


Imagen 15: Creación de los videos, las aristas para dar me gusta, y las personas que publicaron el video

Comandos:

```
CREATE (:Video {nombreVideo: 'Concierto Coldplay 2012', descripcionVideo: 'Me encanta como sale Chris Martin', fechaPublicacionVideo: datetime({year: 2024, month: 4, day: 23, hour: 12, minute: 0, second: 0}), duracionVideo: 120}),
```

```
(:Video {nombreVideo: 'Salida de la procesion del silencio', descripcionVideo: 'Como siempre otro año mas de hermosura', fechaPublicacionVideo: datetime({year: 2024, month: 4, day: 24, hour: 14, minute: 30, second: 0}), duracionVideo: 180})
```

```
MATCH (luisi:Persona {nickname: 'luisi'}),
      (video1:Video {nombreVideo: 'Salida de la procesion del silencio'}),
      (lolita3:Persona {nickname: 'lolita3'}),
      (video2:Video {nombreVideo: 'Concierto Coldplay 2012'})
CREATE (luisi)-[:Ha_publicado]->(video1),
```

(lolita3)-[:Ha_publicado]->(video2)

Ahora crearemos las aristas de 'Ha comentado' y 'Le ha gustado', para representar los comentarios y los likes de los videos o las publicaciones, realizados por los usuarios:

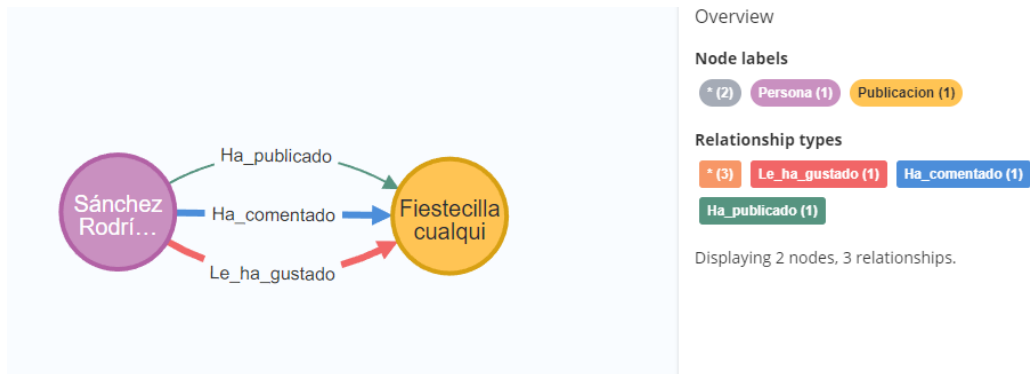


Imagen 16: Creación de la arista para comentar en las publicaciones o videos

En esta foto podemos ver como 'luisi', ha publicado, comentado y le ha gustado la publicación, de la 'fiestecilla'.

La generación de aristas ha sido similar a la de 'Ha publicado', con diferentes nombres de atributos, para las fechas, y que 'Ha comentado' tiene el atributo 'contenidoComentario'.

Por último, añadimos, una arista de tipo 'Ha guardado':



Imagen 17: La arista para guardar publicaciones o videos

Aquí podemos ver como hay una 'Persona' que tiene guardada un 'Video' (pero podria ser tambien una publicación), donde tendria tambien, como en las otras aristas, una fecha para saber cuando se guardo el 'Video'.

Para concluir, el grafo general quedaria asi:

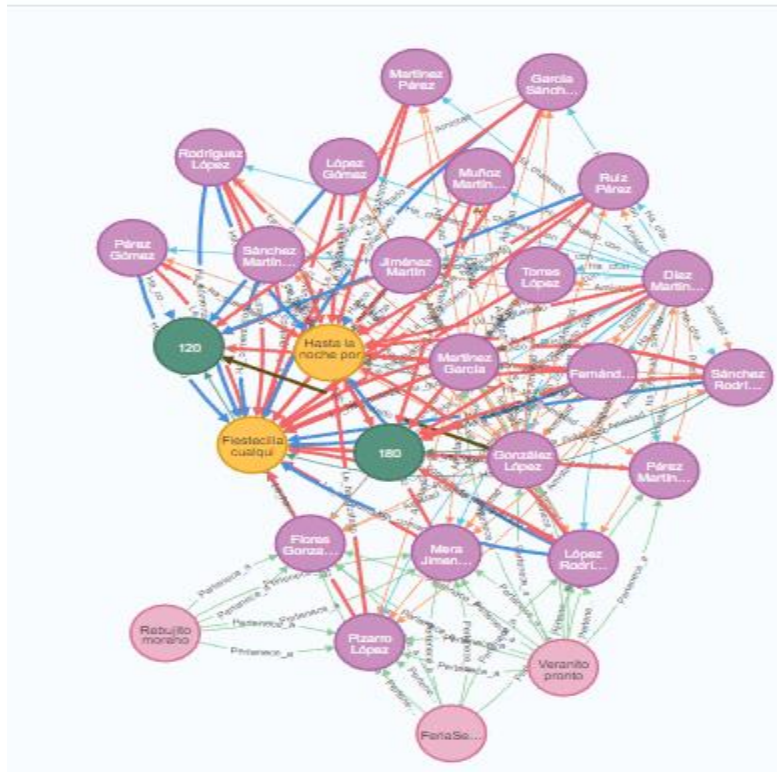


Imagen 18: Grafo resultante final

Obviamente esto, aunque sea una red social de solo 20 personas, ha generado un caos, que no nos genera ningun problema, por que mas adelante, procesaremos todo esto para sacar informacion interesante.

3. *Investigación:*

En esta sección voy a plasmar todo lo que he encontrado de forma teórica sobre las herramientas a utilizar.

Neo4Java: en las clases de CBD, en concreto las prácticas, hemos realizado ya una primera toma de contacto, donde dimos lo básico, páginas que usan los grafos como ayuda empresarial, como eBay, Walmart, HP, entre otros.

Lo principal realizado fue guardar información en grafos dirigidos, poder consultar sus nodos y conexiones.

Si nos vamos a Wikipedia [\[4\]](#), esta nos describe Neo4Java como un software libre de 2007, orientado a grafos y escrito en Java, guardando la información en grafos y eliminando el concepto de tablas.

En un artículo de mkdev [\[5\]](#), nos habla como las bases de datos SQL, a veces no son efectivas en ciertos casos de la realidad, y para ello usar grafos es la solución, ya que representan la relación natural entre cosas. Inclusive git usa grafos (usando el comando “git log --graph --abbrev-commit --decorate --date=relative –all”), incluso coloca el ejemplo de cómo se podrían usar grafos para relacionar, a los parientes en Star Wars.

Existen métodos como el de Dijkstra, que al igual que otros pueden acortarnos/facilitarnos el muestreo en este caso de encontrar un camino mínimo entre dos nodos. Aclara que hay diferentes tipos de grafos, y que uno simple puede ser representado incluso en tablas, como veremos en este proyecto también. Finalizando el artículo, habla de cómo usar Neo4Java, y su aplicación e integración con Spark, que no nos interesa para este proyecto, pero es destacable el hecho de que Neo4Java admite una API HTTP en formato hípster con JSON, el protocolo Bolt, un controlador de Java y un conector para Spark.

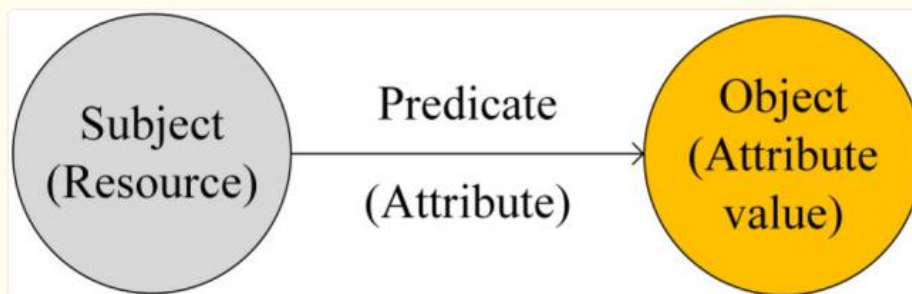
En esta página [\[6\]](#) se utiliza Neo4Java, esta vez para un proyecto de pavimento, usando gráficos de conocimiento y añadiendo un motor de búsqueda de acceso abierto. Destaco que en esta página hacen un comparativo de otras herramientas para construcción de bases de datos de grafos, donde la medida principal es la popularidad de estas en comparación a las otras:

El ranking de popularidad de los sistemas de gestión de bases de datos de gráficos en marzo de 2022.

Rango			Base de datos	Puntaje		
marzo de 2022	febrero de 2022	marzo de 2021		marzo de 2022	febrero de 2022	marzo de 2021
1	1	1	neo4j	59,67	+ 1,43	+ 7,35
2	2	2	Microsoft Azure Cosmos DB	40.90	+ 0,94	+ 8.49
3	3	3	ArangoDB	5.61	+ 0,21	+ 0,55
4	4	5	Virtuoso	5.57	+ 0,18	+ 2,70
5	5	4	OrientDB	4.92	-0,10	+ 0,22
6	7	7	GraphDB	2.84	-0,09	+ 0,57
7	6	8	Amazonas Neptuno	2.69	-0,30	+ 0,83
8	8	6	JanusGraph	2.47	+ 0,11	+ 0,04
9	9	11	TigreGraph	2.18	-0,06	+ 0,68
10	10	10	perro estrella	1,90	-0,08	+ 0,39

En esta tabla se puede visualizar diferentes herramientas de grafos y se comparan [\[6\]](#)

Y tambien nos sirve para ver el funcionamiento base de las bases de datos de grafos:



Como funcionan en lo basico dos nodos y una arista que los une [\[6\]](#)

Neo4Java, también tiene subido en YouTube una gran cantidad de tutoriales, videos explicativos, y yo voy a resaltar su introducción [\[7\]](#), también unos videos explicativos [\[8\]](#) [\[9\]](#) de como iniciar con fáciles pasos, aunque es verdad que en las prácticas de la asignatura proveen de una buena guía y finalmente una

recopilación de un curso en español [\[10\]](#) (la referencia es al primer video de estos).

Bonita Soft: en Wikipedia [\[1\]](#) nos habla de una plataforma también de software libre, para el trabajo de procesos de negocio y el flujo de este, creado en 2001. Se creo en Francia, como respuesta de las operaciones digitales y la modernización de los sistemas de información del software original de Bonita.

En YouTube la página oficial de Bonita Soft, tienen subido un curso/guía de 8 episodios [\[11\]](#) (link del primero de los 8 videos), donde empiezan explicándote todas las funcionalidades bonitas y todos sus componentes, hasta el despliegue. Introduce el concepto de BPMN, notación estándar para el modelado de procesos.

4. ***Funciones:***

Este apartado es para mostrar las diversas funciones que se pueden realizar desde Neo4Java, y el resultado en grafos que muestra, partiendo de los más sencillos, hasta los más complejos y siempre desde un contexto, es decir, nunca veremos un método no justificado, por ejemplo, el primero:

- *Un método que devuelva a la persona con más publicaciones, y otra con la que tiene más amistades, para ver si tiene relación:*

Comandos:

(La que mas amistades tiene tiene)

MATCH (p1:Persona)-[:Amistad]-(amigo)

WITH p1, count(amigo) AS numAmistades

ORDER BY numAmistades DESC

LIMIT 3

RETURN p1



Imagen 19: Martínez García es la persona con mas amistades

(La que mas publicaciones tiene)

```
MATCH (p:Persona)-[:Ha_publicado]->(publicacion)
```

```
WITH p, count(publicacion) AS numPublicaciones
```

```
ORDER BY numPublicaciones DESC
```

```
LIMIT 1
```

```
RETURN p
```

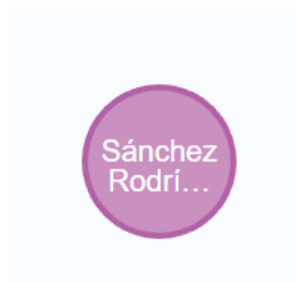


Imagen 20: Sanchez Rodriguez es la persona con mas publicaciones

La persona con más publicaciones, en esta red social, no es la que también tiene más amistades, por lo que no es proporcional

- Ahora un método que compruebe si la persona que más personas haya chateado es la que pertenece a más grupos:

Comandos:

(La que más ha chateado con alguien)

```

MATCH (p:Persona)-[:Ha_chateado_con]->(mensaje)
WITH p, count(mensaje) AS numMensajes
ORDER BY numMensajes DESC
LIMIT 1
RETURN p

```

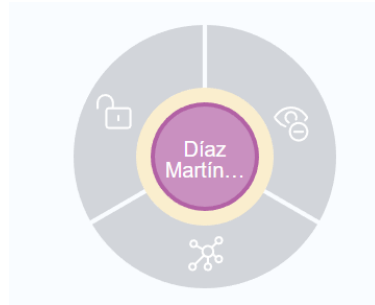


Imagen 21: Díaz Martínez es la persona que mas a hablado con otras

```

(La que pertenece a mas grupos)
MATCH (p:Persona)<[:Pertenece_a]-(grupo)
WITH p, count(grupo) AS numGrupos
ORDER BY numGrupos DESC
LIMIT 1
RETURN p

```

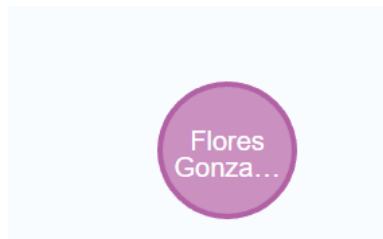


Imagen 22: Flores González es la persona que esta en mas grupos sociales

Vemos que tampoco esta relacionado, que la persona que mas chatea con otras personas, sea la que esta en mas grupos.

Cuanto mas personas, encontraremos ciertas relaciones, que pueden servir a la hora de, publicitar, ya sea por el contenido de los mensajes que se mandan en una publicación, por ejemplo:

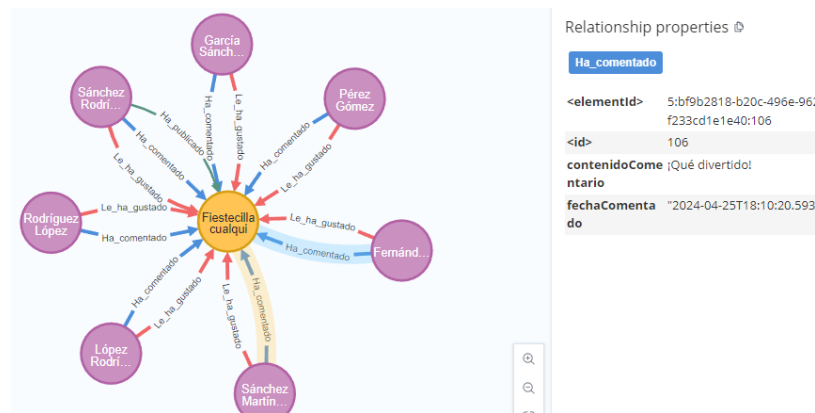


Imagen 23: Visualización que en la publicación de la Fiestecilla, muchas personas comentaron que es divertido

Ahí podemos visualizar que todo el mundo comenta en esa publicación que ha sido divertida, y por tanto podrian publicitar mas quien creo esa publicación.

Comandos:

```
MATCH (persona:Persona)-[:Ha_comentado]->(publicacion:Publicacion
{nombrePublicacion: 'fiesta de pijamas de elisaaa'})
```

```
RETURN persona, publicacion
```

Ahora vamos a ver unos métodos que proporciona Neo4Java, que son interesantes a usar, para tener mejor apreciación de todo:

- CALL db.schema.visualization() | Muestra un esquema general de todo lo relacionado, es interesante compararlo con el UML, planteado al inicio ya que aquí se muestra como varía entre una base de datos clásica, y una de grafos

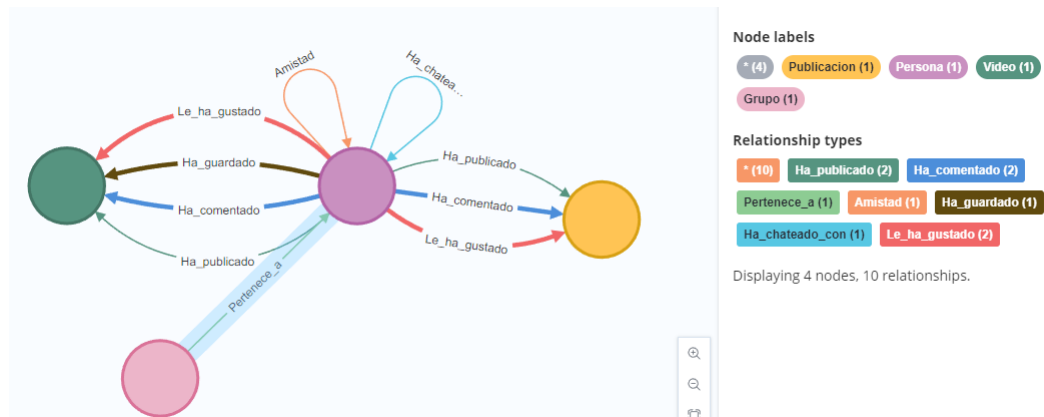


Imagen 24: Grafo que resume las relaciones y sus vertices

- CALL db.relationshipTypes() | Muestra las etiquetas de las relaciones

relationshipType
"Amistad"
2 "Ha_chateado_con"
3 "Pertenece_a"
4 "Ha_publicado"
5 "Ha_comentado"
6 "Le_ha_gustado"
7 "Ha_guardado"

Imagen 25: La etiqueta de todas las aristas

- CALL db.labels() | Muestra las etiquetas de los nodos

	label
1	"Persona"
2	"Grupo"
3	"Publicacion"
4	"Video"

Imagen 26: La etiqueta de todos los vertices

- `MATCH ()-->() RETURN count(*)` | Cuenta las relaciones

	count(*)
1	162

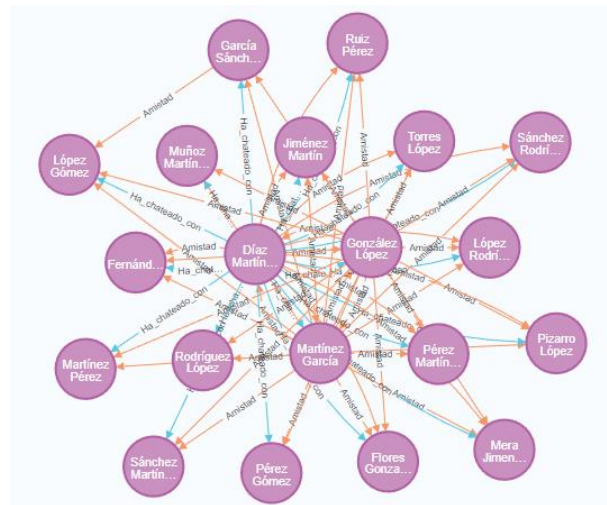
Imagen 27: Numero de relaciones totales

- `MATCH (n) RETURN count(n)` | Cuenta todos los nodos

	count(n)
1	27

Imagen 28: Numero de vertices

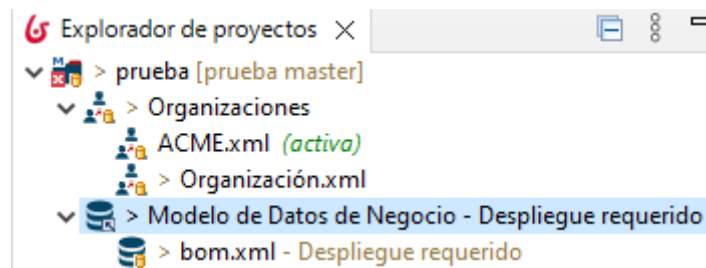
- `MATCH (n1)-[r]->(n2) RETURN r, n1, n2 LIMIT 25` | Si un Grafo es muy lioso y grande te muestra 25 nodos solo y sus relaciones



5. *Bonita*:

En este apartado definiremos todos los flujos BPMN, que ocurrirán en nuestra aplicación. Al ser una aplicación social, nos resultaran más familiares.

Lo primero de todo es que bonita proporciona un modelado de tu proyecto, pudiendo trasladarlo a este, para un control más fácil:



En este he creado la base de datos original que he declarado, añadiendo todos sus objetos:

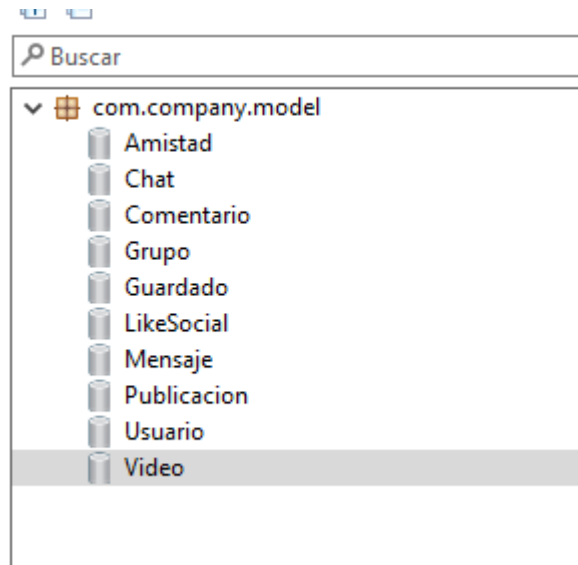


Imagen 31: Objetos del MDN

Ademas de declarar objetos, Bonita nos deja crear descripciones, añadir atributos y restricciones:

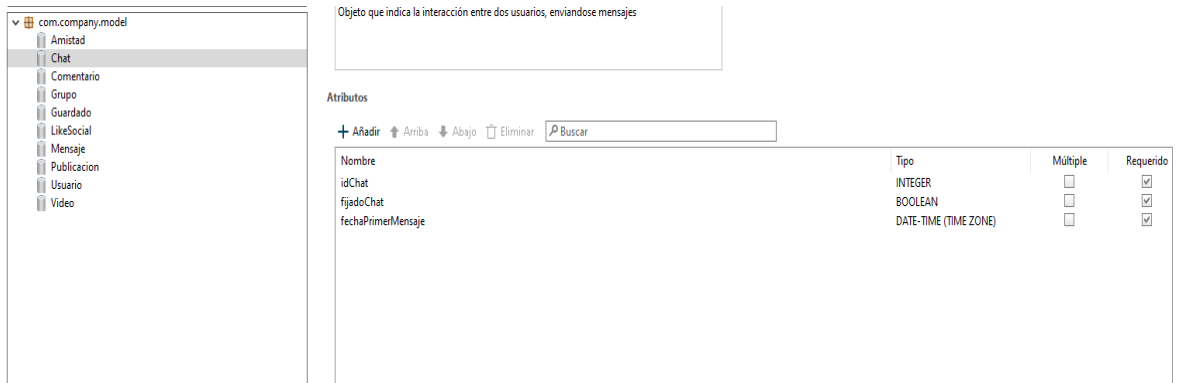


Imagen 32: Atributos y descripción del objeto

Descripción

Objeto que indica la interacción entre dos usuarios, enviándose mensajes

Atributos

+ Añadir ↑ Arriba ↓ Abajo ✖ Eliminar 🔍 Buscar

Nombre	Tipo	Múltiple	Requerido
idChat	INTEGER	<input type="checkbox"/>	<input checked="" type="checkbox"/>
fijadoChat	BOOLEAN	<input type="checkbox"/>	<input checked="" type="checkbox"/>
fechaPrimerMensaje	DATE-TIME (TIME ZONE)	<input type="checkbox"/>	<input checked="" type="checkbox"/>

▼ Detalles

Descripción de atributo

id del chat

Imagen 33: Detalles de los atributos y descripción de estos

Restricciones únicas para `Chat`

+ Añadir ✖ Eliminar

🔍 Buscar

Nombre	Atributos
resChat	[fijadoChat]

Descripción

No se pueden fijar mas de 3 chats

Atributos

Seleccione la combinación de atributos a los que se aplicará la restricción. No puede utilizar sus atributos de tipo TEXT por ser de solo lectura.

☐ idChat -- Integer

☒ fijadoChat -- Boolean

☐ fechaPrimerMensaje -- Offsetdatetime

Imagen 34: Restricciones de Chat

También te permite crear consultas propias, aunque Bonita tiene consultas por defecto:

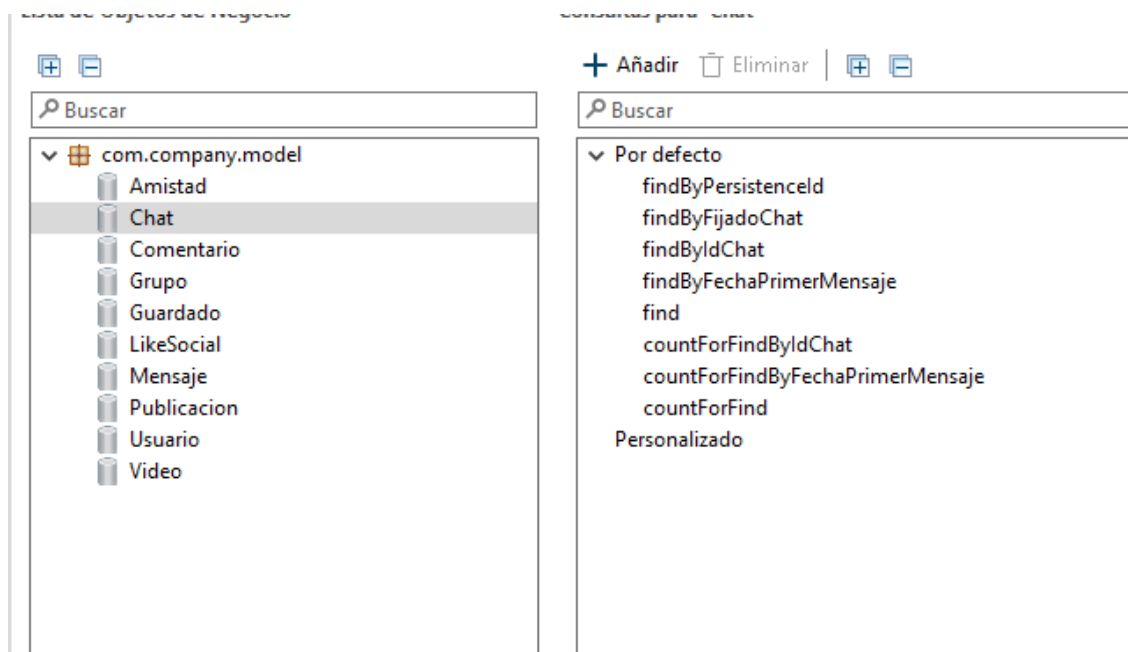


Imagen 35: Consultas predeterminadas de Chat

Por último, he generado 5 BPMN dentro del contexto de una red social:

1. El primero describe como se debería reclamar por parte de un usuario, si este cree que un post incumple las normas de la red social:

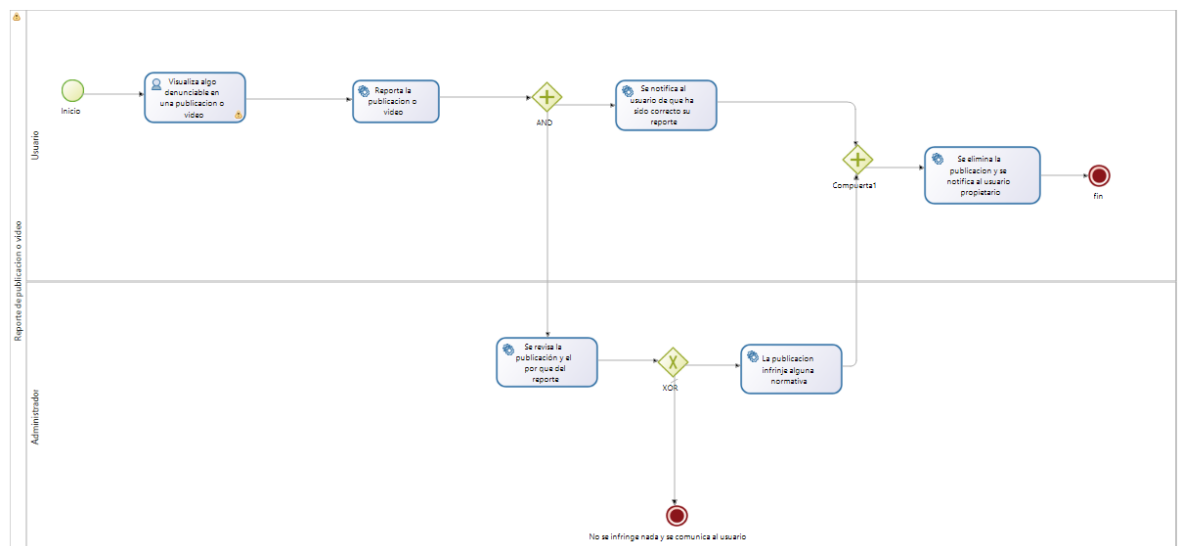


Imagen 36: BPMN de reclamacion de una publicacion o video

2. Para la creación de una publicación o video:



Imagen 37: BPMN para crear una publicación o video

3. Para crear un grupo de mensajes entre varios participantes:

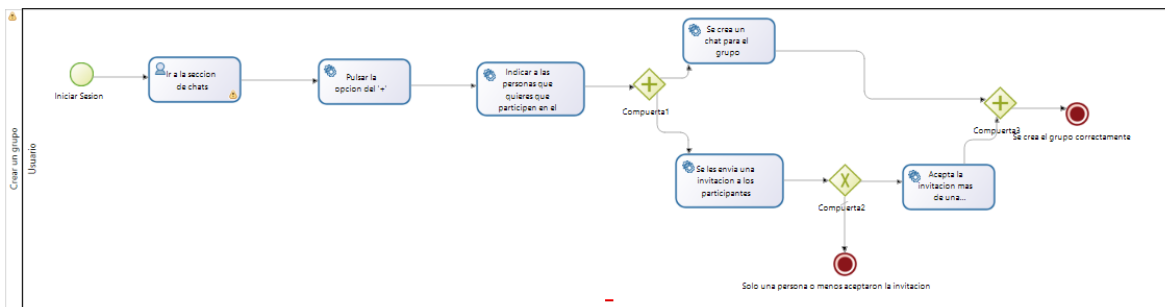


Imagen 38: BPMN para crear grupos de mensajes

4. Para enviar una solicitud de amistad:

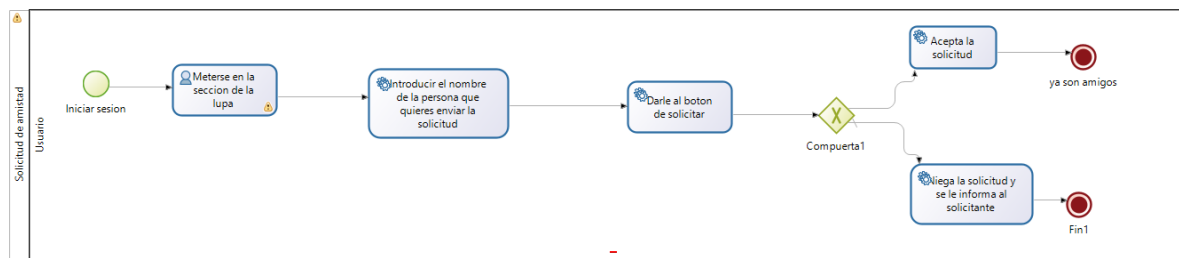


Imagen 39: BPMN para solicitar amistad

5. Para guardar publicaciones y videos:

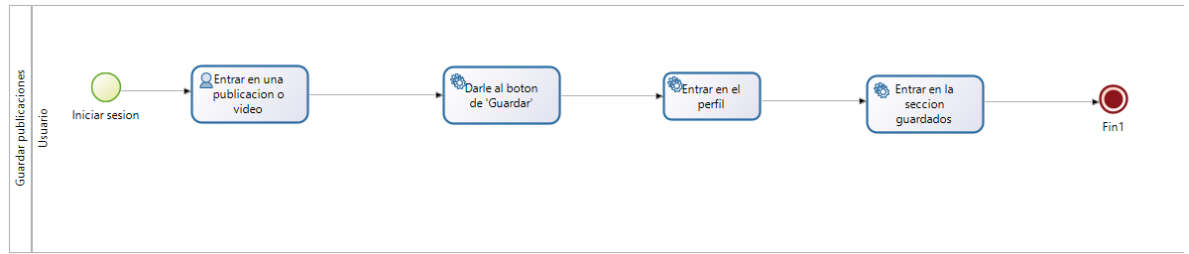


Imagen 40: BPMN para usar la funcion de guardar

Para solucionar el problema de las XOR:

La que de error, poner que sea flujo por defecto

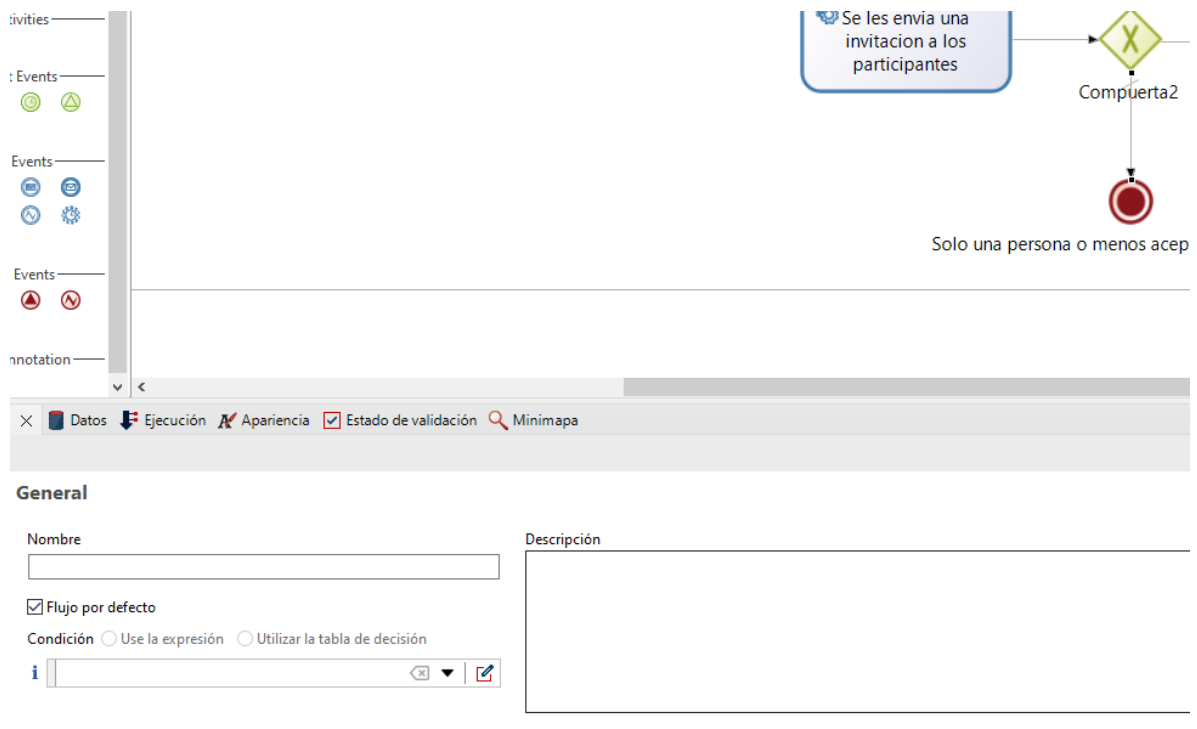


Imagen 41: Como solucionar en la primera arista del XOR

Y la otra arista, colocar que use la expresion 'true'

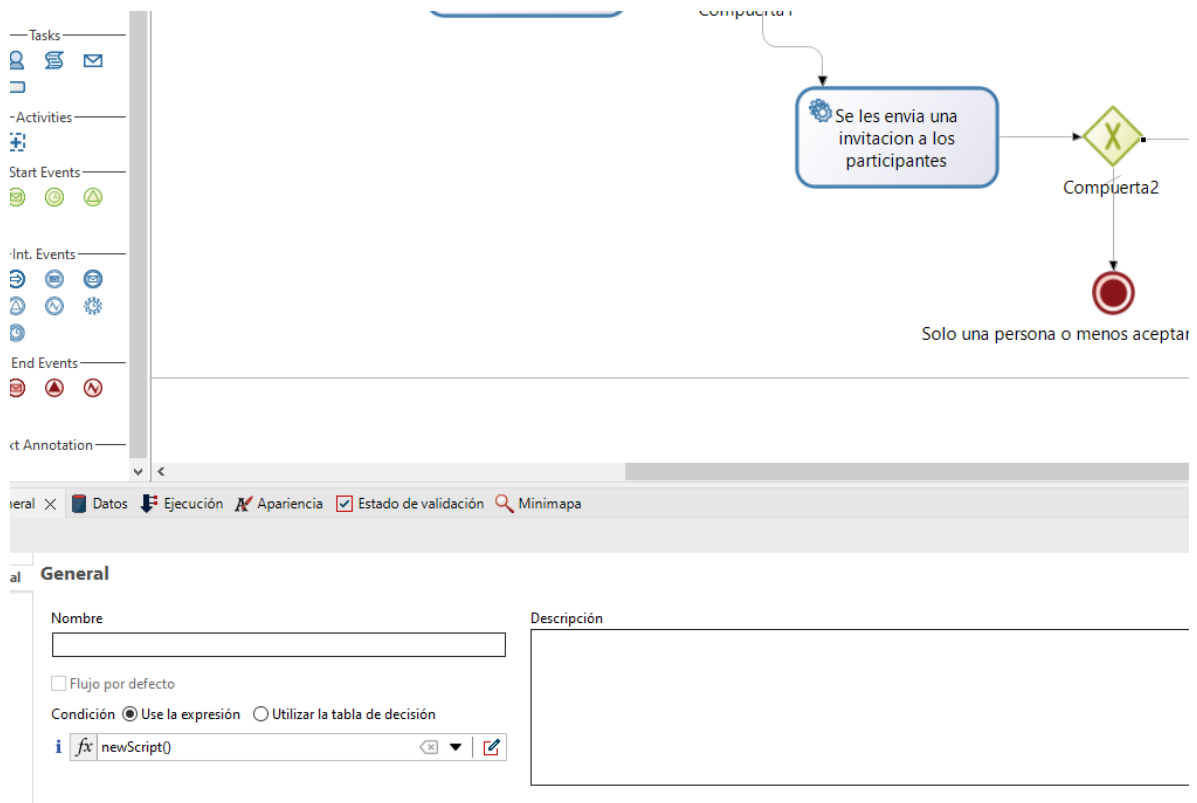


Imagen 42: Como solucionar la segunda arista del XOR

6. Manual de Instalación:

Neo4Java:

1. Primero descargar en el link, dándole a 'Download':
<https://neo4j.com/download/>

Experience Neo4j on Your Desktop

Free. Get Started Today.

Download 

Includes Neo4j Enterprise 5.19.0 for Developers

[Learn more](#) | [System Requirements](#)

2. Rellenar el formulario:

Please fill out this form to begin your download

First Name

Last Name

Email (Business Preferred)

Company Name

None

Student

Phone Number

Spain

By submitting this form, you confirm you are acting for your business/organisation.


☒ We would like to keep you updated on the latest news about graph database products, services & events. If you do not want to hear from us going forward, please tick this box.

Download Desktop

3. Saldrá un link Neo4j Desktop Activation Key, el cual es una clave de mucha longitud, cópialo:

Neo4j Desktop Activation Key

Use this key to activate your copy of Neo4j Desktop for use.

 Copy to clipboard

```
Y3ZDk5NSislm1peHBhbmVsUHJvamVjdElkjoinglmYjI0MTRhYjk3M2M3NDFlNmYwNjdiZjA  
ZZDU1NzUilCJvcmcilOiluKlslNB1Yil6lm5lbzRqLmNvbSlsInJZyl6liAilCJzdWiiOiuZW80ai1kZ  
XNrdG9wliwiZXhwIjoxNzA3ODk2Mzk3LCJ2ZXliOiliqiwiaXNzljoiibmVvNGouY29tliwibmJmljox  
Njc2MzYwMzk3LCJpYXQjOjE2NzYzNjAzOTcsIm0aSi6ijNaZDAzQTBKNyJ9.iiizqegKYANsjZGp  
R2iDhZuepuSRQvkrQWmNYuCYtT9XS4WexZm6lWhJX4ytOP-dTpsGRs2lC11MfEotEGSOW  
7T1d77...sqri6sY53oL4di515VwUilUilN_AUfnN6Gd-kOkv/Wa5BilI0ozsg6XJemC5XV4gKP140  
8-FUIG7A2fKd0DSqpmvD7fOpo3gGMpvnBEZvbbqDG0MZhjDa-9E3FgL1DMP7tj-zewKioy9  
Kso1Smsz81JczGm8zliBMuBerSBQVyb6yhHT1DS21DajT_gTBW8szEo_G1K3-2T6tR1ACNzu  
Ywj_dM7d6t1Y4O4s-4Xj1utlytLkPIEHoyGXWaKQ
```

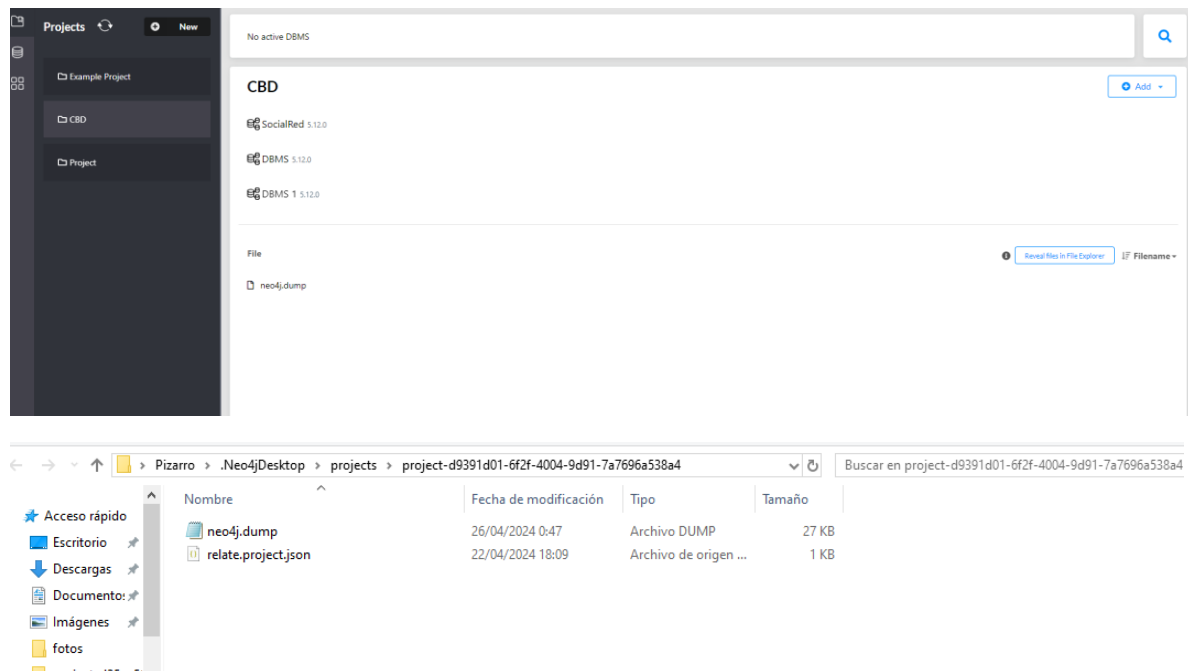
4. Ejecutar la aplicación que se ha descargado

5. Activar la clave al iniciar la aplicación:

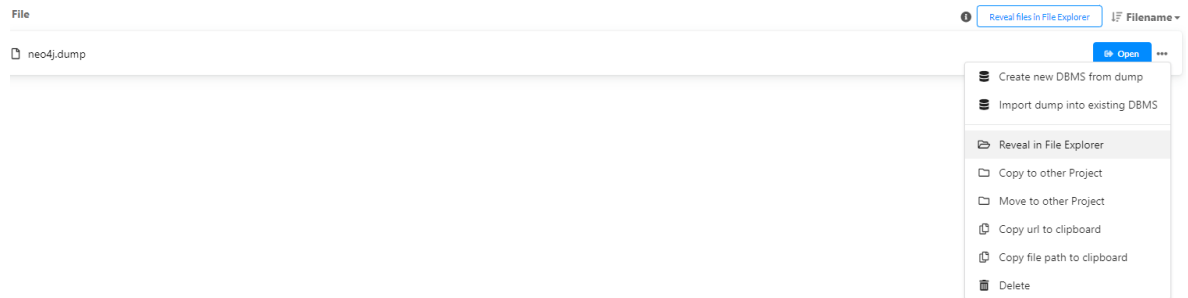
4. Una vez rellenado y enviado se te descargará un ejecutable, el cual, tras ejecutarlo, se te instalara BonitaStudio.

7. Manual de usuario:

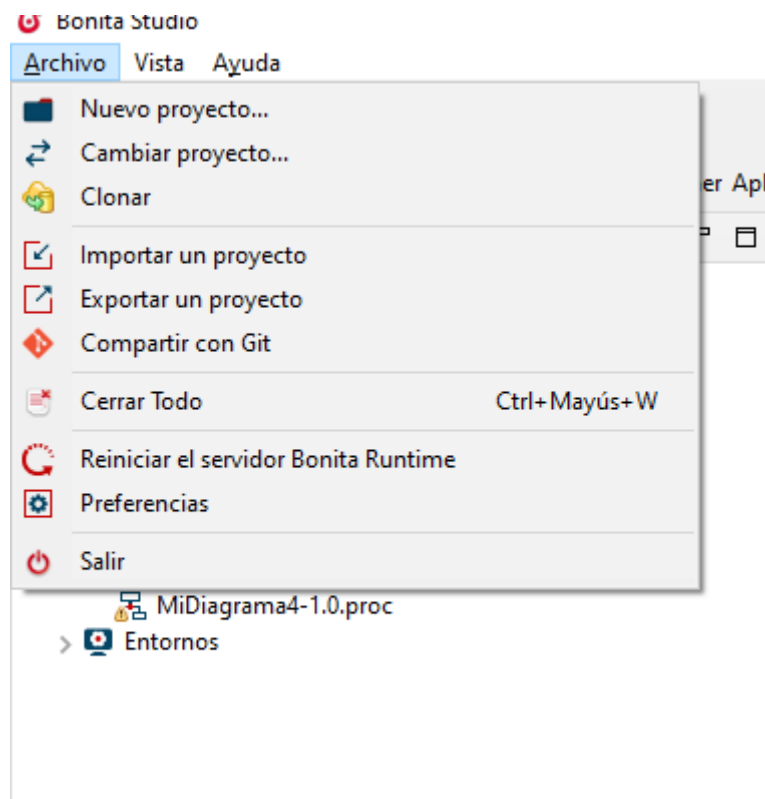
1. Descargar el archivo neo4j.dump (está en GitHub: <https://github.com/Domi-ATLAS/Neo4j-x-Bonita-CBD-edupizlop> y en la entrega)
2. Importarlo en Neo4Java, metiéndolo en la carpeta del proyecto, la cual se podrá visualizar en el botón 'Reveal files in File Explorer', en azul:



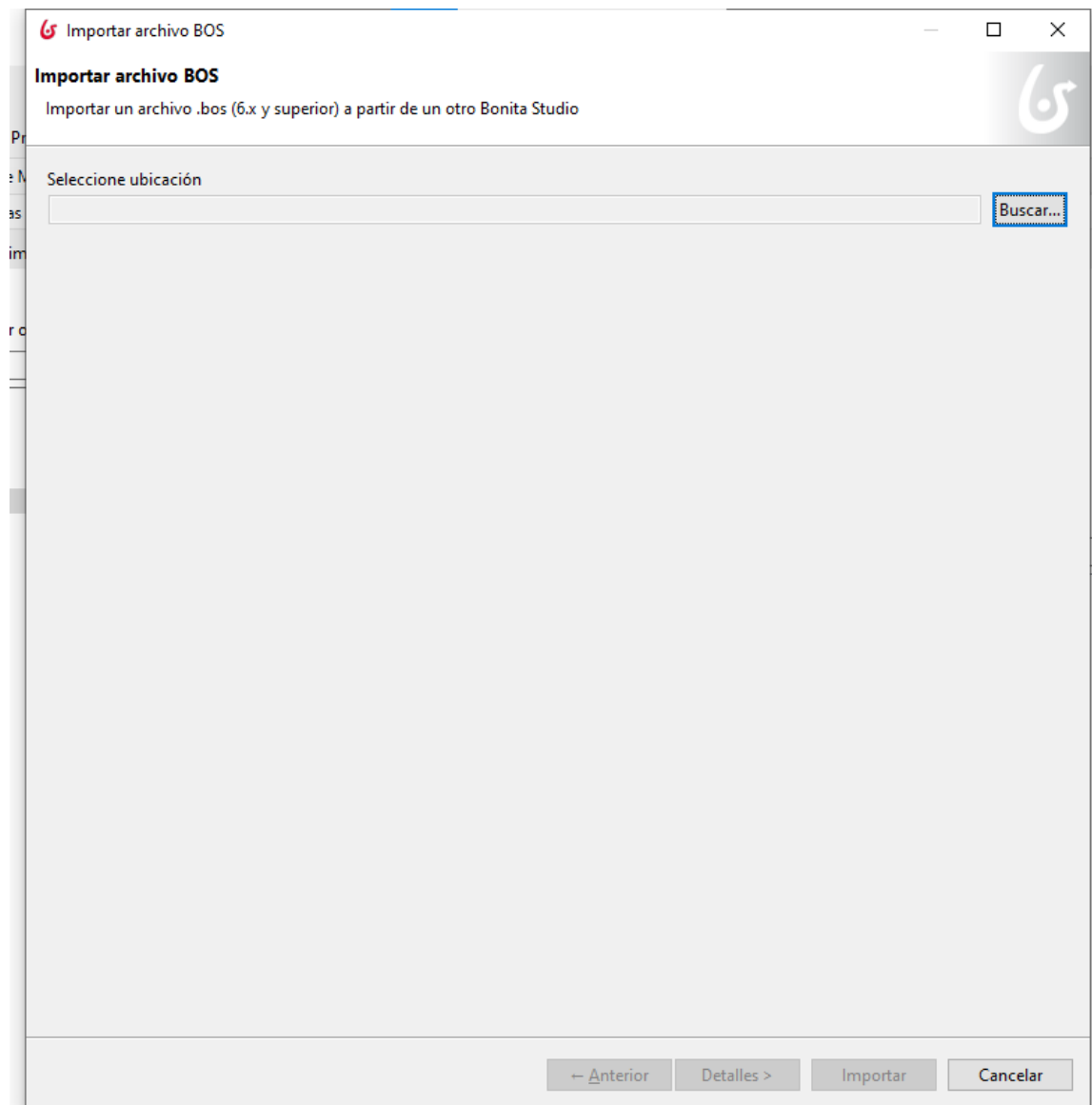
3. Pulsar a los tres puntitos del fichero dump, y seleccionar la opción 'Create a new DBMS from dump'



4. Abrir la base de datos con el botón 'Start' que nos aparecerá, al poner el cursor encima de la nueva BDMS, y después abrir. Como apreciación, los colores no se mantienen en esta versión, por lo que recomiendo cambiar el color de las aristas y vértices para una mejor visualización.
5. Para visualizar lo realizado en Bonita, se deberá descargar el archivo 'BonitaCBD.bos', e importarlo en la aplicación



6. Selecciona el archivo a importar y cuando termine se habrá completado con éxito



8. Conclusiones:

Para concluir este proyecto, hemos encontrado una forma rápida y automatizada de poder crear una base de datos basada en grafos, y no solo crearla, sino poder generar peticiones/queries, que nos permitan sacar información importante, para después, poder tomar decisiones de negocio, así enfocando las decisiones a lo que nos puede interesar más.

Además, hemos descubierto que diversas personas y entidades han usado Neo4Java para diversas situaciones, y Bonita Soft, no es tan popular como la otra herramienta, dispone de documentación y tutoriales, tanto en su página web como en YouTube.

En una red social más realista, seguramente hayamos podido sacar funciones mucho más interesantes, aun así, hemos podido ver ciertas correlaciones entre los propios nodos o con aristas.

Además, en Bonita, tampoco se ha profundizado tanto, debido a que en Neo4Java estaba el núcleo más importante del proyecto y que en Bonita, se puede apreciar que es muy útil para modelar bases de datos, con restricciones, relaciones, etc. Añadiendo que puedes crear jerarquías de usuarios, para darles permisos y lo más importante, una gran herramienta en la ayuda de BPMN.

9. Referencias, Bibliografía y Anexo:

USO DE IA:

[1'] <https://chat.openai.com/share/82fa2f9a-fa95-438d-a706-55a38582a56a>

[2'] <https://chat.openai.com/share/7a9f8501-dd03-475c-b0cf-85b8628657e2>

[3'] <https://chat.openai.com/share/ad9be784-da52-4a53-a830-193e813304f7>

[4'] <https://chat.openai.com/share/d2208436-42e9-48a7-830e-36c5e61639ab>

[5'] <https://chat.openai.com/share/ffdfd0fe-51ea-4af3-a551-626ea5205cb4>

INVESTIGACIÓN:

Referencias:

- [1] https://es.wikipedia.org/wiki/Bonita_Open_Solution
- [2] <https://www.inesem.es/revistadigital/informatica-y-tics/teoria-grafos/>
- [3] <https://drive.google.com/file/d/1pFnXDtfqG5fEUQ6VsCrsS1BcSCztdrv6/view?usp=sharing>
- [4] <https://es.wikipedia.org/wiki/Neo4j>
- [5] <https://mkdev.me/posts/introduction-to-graphs-and-neo4j-graph-processing-in-spark>
- [6] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9098876/>

Bibliografía:

- [7] <https://www.youtube.com/watch?v=LB2t08yaVPE>
- [8] <https://www.youtube.com/watch?v=kVkmuRVlaCA>
- [9] <https://www.youtube.com/watch?v=T6L9EoBy8Zk>
- [10] https://www.youtube.com/watch?v=aHxBMUq511c&list=PL9HI4pk2FsvU_RWkuVjEvwOXJuH9ibAF3
- [11] https://www.youtube.com/watch?v=eNspu_s5jNM&list=PLvvoQatxaHOOgWEMHZjk5rjc9qsCnh7bi&index=1

En el proyecto asignado se indica el uso de MongoDB, pero tras consultarlo con el profesorado, es un error, solo se debe usar Neo4Java y Bonita Soft.