Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Análisis de Neo4J como soporte al control de acceso en entornos sanitarios

Autor: Soledad Muñoz Jiménez
Tutor: Jorge Calvillo Arbizu

Dpto. Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Análisis de Neo4J como soporte al control de acceso en entornos sanitarios

Autor:
Soledad Muñoz Jiménez

Tutor:
Jorge Calvillo Arbizu
Profesor permanente laboral

Dpto. Ingenería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Análisis de Neo4J como soporte al control de acceso en entornos sanitarios

Autor: Soledad Muñoz Jiménez

Tutor: Jorge Calvillo Arbizu

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

No puedo creer que esté escribiendo estas líneas, ha sido un camino muy duro, intenso y largo, pero ha sido bonito gracias todas las personas que han formado parte de él. Tras este proyecto, pongo fin a mi carrera universitaria, y quiero aprovechar esta oportunidad para dedicar unas palabras de agradecimiento.

En primer lugar, quiero expresar mi más profundo agradecimiento a mi madre, Catalina, mi pilar fundamental. Gracias por apoyarme incondicionalmente y estar siempre, pero siempre, a mi lado. Por escucharme, por llorar conmigo, por celebrar mis logros y, sobre todo, por siempre saber cómo sacarme adelante. Sin ti, nada de esto habría sido posible. A mi hermano Juan Diego; gracias por estar siempre a mi lado tanto en los buenos momentos como en los malos. También quiero agradecer de corazón a toda mi familia materna, porque en los momentos difíciles, cuando sentía que me faltaban fuerzas siempre han estado para ayudarme.

Gracias a mi pareja, Antonio, por cuidarme y creer en mí. Por hacer siempre todo lo posible por verme bien en los malos momentos y por hacer aún mejores los buenos. Por celebrar conmigo cada triunfo y por escuchar día tras día mi rutina sin importar la monotonía de algunas etapas.

No podía no mencionar a mis compañeros de carrera: Claudia, Guadalupe y Marta. Cuando llegué a Sevilla, no pensé que pudiera conocer a alguien que en tan poco tiempo significara tanto para mí, pero me equivocaba, habéis sido una parte fundamental durante estos años de grado, y si hoy en día estoy aquí también es por vosotras.

No puedo olvidar a mis amigas de toda la vida, a pesar de que durante la carrera no me han visto tanto como me gustaría, han estado ahí cuando lo he necesitado y se que lo estarán siempre. Hay un dicho que dice que los amigos son la familia que se elige y creo que haberles escogido a ellos es una de las mejores decisiones que he tomado en mi vida.

Por último, pero no menos importante, agradecer a Jorge Calvillo, tutor de este trabajo. A lo largo de estos años he conocido a muchos profesores y he pasado más tiempo con varios de ellos que contigo, pero el apoyo, la paciencia y la confianza que me has brindado merecen ser reconocidos y agradecidos de todo corazón.

Gracias a todos los que han formado parte de este viaje. Os quiero con todo mi corazón.

Soledad Muñoz Jiménez

Sevilla, 2024

Resumen

En la actualidad, nuestros datos se utilizan para diversos propósitos, como estudios, análisis y administración. Este uso de datos contribuye a un funcionamiento más eficiente de los sistemas y organizaciones, pero también conlleva riesgos significativos. A menudo, no somos plenamente conscientes del destino final de nuestros datos.

Para abordar estos riesgos, se han desarrollado los consentimientos personales, regulados por el Reglamento General de Protección de Datos (GDPR). Estos consentimientos otorgan a los usuarios el control sobre el uso de su información, permitiéndoles aprobar o rechazar cómo se utilizarán sus datos para fines específicos, y garantizando que no se procesen sin el consentimiento adecuado.

Este principio es especialmente crucial en el ámbito sanitario, donde la información personal es extremadamente sensible. Frecuentemente, otorgamos nuestro consentimiento sin una comprensión total de los permisos que estamos concediendo. Dado que las tecnologías emergentes se integran cada vez más en nuestra vida diaria, es vital contar con sistemas que faciliten la gestión y consulta de esta información.

En respuesta a esta necesidad, se propone el desarrollo de un servicio que almacene tanto los consentimientos como la información sanitaria. Este sistema tiene como objetivo mejorar la transparencia y el control del usuario sobre su información personal en el ámbito sanitario, soportando el proceso de toma de decisiones de control de acceso a partir de la información almacenada.

Este sistema está diseñado para cumplir con los requisitos del GDPR, garantizando la protección de datos y facilitando la consulta de los consentimientos de los pacientes y la información relacionada en los mecanismos de control de acceso. El objetivo último es ofrecer una solución que soporte la toma de deciones de control de acceso incorporando los consentimientos otorgados por los pacientes para el control sobre su información sanitaria.

La solución propuesta incluye un servicio web REST que proporciona una interfaz accesible para que los usuarios u otros sistemas puedan realizar consultas sobre los consentimientos y la información sanitaria mediante peticiones HTTP, eliminando la necesidad de visitas físicas a los centros sanitarios y una base de datos Neo4j utilizada para almacenar y gestionar datos complejos relacionados con consentimientos, profesionales y pacientes, aprovechando las capacidades de Neo4J para manejar eficientemente las relaciones entre nodos.

Abstract

In today's world, our data is used for various purposes such as studies, analysis, and administration. While this use of data contributes to more efficient operations of systems and organizations, it also involves significant risks. Often, we are not fully aware of the final destination of our data.

To address these risks, personal consent has been developed, regulated by the General Data Protection Regulation (GDPR). This consent gives users control over how their information is used, allowing them to approve or reject how their data will be used for specific purposes, ensuring that it is not processed without the proper consent.

This principle is especially crucial in the healthcare sector, where personal information is extremely sensitive. Frequently, we give our consent without fully understanding the permissions we are granting. As emerging technologies become increasingly integrated into our daily lives, it is vital to have systems that facilitate the management and consultation of this information.

In response to this need, the development of a service that stores both consents and healthcare information is proposed. This system aims to improve user transparency and control over their personal information in the healthcare sector by supporting access control decision-making processes based on the stored information.

This system is designed to comply with GDPR requirements, ensuring data protection and facilitating the consultation of patient consent and related information in access control mechanisms. The ultimate goal is to provide a solution that supports access control decision-making by incorporating the consents given by patients to control their healthcare information.

The proposed solution includes a REST web service that provides an accessible interface for users or other systems to make inquiries about consents and healthcare information via HTTP requests, eliminating the need for physical visits to healthcare centers. It also includes a Neo4j database used to store and manage complex data related to consents, professionals, and patients, leveraging Neo4j's capabilities to efficiently handle relationships between nodes.

Índice

Agradecimientos **ix**

Resumen **xi**

Abstract **xiii**

Índice **xv**

Índice de Tablas **xvii**

Índice de Figuras **xix**

Notación **xxii**

1 Introducción **1**
1.1 Motivación y objetivos *1*
1.2 Metodología *3*
1.3 Plan de trabajo *3*

2 Marco teórico **5**
2.1 Norma GDPR *5*
2.2 Modelo de control de acceso basado en atributos (ABAC) *6*
2.2.1 Arquitectura 6
2.2.2 Atributos 6
2.2.3 Políticas 6
2.3 Base de datos basada en grafos *7*
2.3.1 Neo4j 8
2.3.2 Cypher 8
2.4 Herramientas software empleadas *14*
2.4.1 Spring Tool Suite 4 14
2.4.2 API Spring Data Neo4j 14
2.4.3 Neo4j Desktop 15
2.4.4 Maven 15
2.4.5 Interfaz REST 15
2.4.6 Postman 16
2.4.7 MagicDraw UML 16

3 Resultados **11**
3.1 Casos de uso *11*
3.1.1 CDU-01: Obtener información de un profesional 13
3.1.2 CDU-02: Obtener la especialidad de un profesional 14
3.1.3 CDU-03: Obtener el rol de un profesional 15
3.1.4 CDU-04: Obtener la organización de un profesional 16
3.1.5 CDU-05: Obtener información de un paciente 17
3.1.6 CDU-06: Obtener la información sanitaria de un paciente 18
3.1.7 CDU-07: Obtener el consentimiento de un paciente 19
3.1.8 CDU-08: Obtener información de un recurso Información 20
3.1.9 CDU-09: Obtener la especialidad de un recurso información 21
3.1.10 CDU-10: Obtener la organización de un recurso información 22

3.1.11 CDU-11: Obtener el consentimiento de un recurso información 23
3.1.12 CDU-12: Obtener un consentimiento 24
3.2 Base de datos Neo4j *25*
3.3 Servicio REST *36*
3.3.1 Esquema general 37
3.3.2 Profesional.java 38
3.3.3 ProfesionalController.java 39
3.3.4 ProfesionalImpl.java 41
3.3.5 ProfesionalDAO.java 41
3.3.6 ProfesionalDto.java 42
3.3.7 MensajeResponse.java 42
3.3.8 Paciente.java 43
3.3.9 PacienteController.java 43
3.3.10 PacienteImpl.java 46
3.3.11 PacienteDAO.java 46
3.3.12 PacienteDto.java 46
3.3.13 Información.java 46
3.3.14 InformaciónController.java 47
3.3.15 InformaciónImpl.java 48
3.3.16 InformaciónDAO.java 48
3.3.17 InformaciónDto.java 48
3.3.18 Consentimiento.java 48
3.3.19 ConsentimientoController.java 49
3.3.20 ConsentimientoImpl.java 52
3.3.21 ConsentimientoDAO.java 52

4 Conclusiones y líneas futuras **53**
4.1 Conclusiones *53*
4.1.1 Aprendizajes supuestos 53
4.1.2 Objetivos y resultados del proyecto 53
4.2 Líneas futuras *54*

Referencias **55**

ÍNDICE DE TABLAS

Tabla 1 Horas estimadas y reales para el plan de trabajo 4
Tabla 2 Elementos y Patrones de Cypher 9
Tabla 3 CDU-01: Obtener información de un profesional 13
Tabla 4 CDU-02: Obtener la especialidad de un profesional 14
Tabla 5 CDU-03: Obtener el rol de un profesional 15
Tabla 6 CDU-04: Obtener la organización de un profesional 16
Tabla 7 CDU-05: Obtener información de un paciente 17
Tabla 8 CDU-06: Obtener la información sanitaria de un paciente 18
Tabla 9 CDU-07: Obtener el consentimiento de un paciente 19
Tabla 10 CDU-08: Obtener información de un recurso información 20
Tabla 11 CDU-09: Obtener la especialidad de un recurso información 21
Tabla 12 CDU-10: Obtener la organización de un recurso información 22
Tabla 13 CDU-11: Obtener el consentimiento de un recurso información 23
Tabla 14 CDU-12: Obtener un consentimiento 24
Tabla 15 Estructura de la base de datos 25
Tabla 16 Ejemplos de consultas usando rangos 30
Tabla 17 Ejemplo de consulta usando filtro por cadenas parciales 31
Tabla 18 Consulta para obtener la información de un profesional 32
Tabla 19 Consulta para obtener el rol de un profesional 32
Tabla 20 Consulta para obtener el consentimiento relacionado con un recurso información 33
Tabla 21 Consulta para obtener la información sanitaria de un paciente 33
Tabla 22 Consulta para conocer la existencia de un consentimiento 34
Tabla 23 Consulta para conocer la existencia de un consentimeinto por rol del profesional 35
Tabla 24 Consulta para concocer la existencia de un consentimiento con rol del profesional 35
Tabla 25 Consulta para conocer la existencia de un consentimiento con organización del profesional y recurso información 36
Tabla 26 Método showById de ProfesionalController.java 39
Tabla 27 Método obtenerRolNombre de ProfesionalController.java 40
Tabla 28 Método findRolByProfesionalId de PorfesionalDAO.java 41
Tabla 29 Método showById de PacienteController.java 44
Tabla 30 Método obtenerInfo de PacienteController.java 45
Tabla 31 Método obtenerCons de PacienteController.java 46
Tabla 32 Método showById de InformacionController.java 47
Tabla 33 Método obtenerEspNombre de InformacionController.java 48

ÍNDICE DE FIGURAS

Figura 1-1 Representación del funcionamiento del sistema. 2
Figura 1-2 Representación de la metodogía incremental 3
Figura 2-1 Modelo ABAC 7
Figura 2-2 Gráfico social 8
Figura 2-3 Consulta 9
Figura 2-4 Ejemplo MERGE 10
Figura 2-5 Ejemplo CREATE 10
Figura 2-6 Ejemplo eliminación de nodo sin relaciones 11
Figura 2-7 Ejemplo de eliminación de nodo con relaciones 11
Figura 2-8 Ejemplo de creación de relación entre dos nodos 11
Figura 2-9 Ejemplo de creación de nodos y relación 11
Figura 2-10 Ejemplo de eliminación de una relación 12
Figura 2-11 Agregación de propiedades 12
Figura 2-12 Eliminación de una propiedad con REMOVE 12
Figura 2-13 Eliminación de una propiedad con SET 12
Figura 2-14 Eliminación total 13
Figura 2-15 Carga de archivo csv 13
Figura 2-16 IDE Spring Tool Suite 4 14
Figura 2-17 Logo Spring Data Neo4j 14
Figura 2-18 IDE Neo4j Desktop 15
Figura 2-19 Logo Maven 15
Figura 2-20 IDE Postman 16
Figura 2-21 Ejemplo diagrama UML 16
Figura 3-1 Diagrama general de los casos de uso 12
Figura 3-3 Diagrama de actividad CDU-01 13
Figura 3-4 Diagrama de actividad CDU-02 14
Figura 3-5 Diagrama de actividad CDU-03 15
Figura 3-6 Diagrama de actividad CDU-04 16
Figura 3-7 Diagrama de actividad CDU-05 17
Figura 3-8 Diagrama de actividad CDU-06 18
Figura 3-9 Diagrama de actividad CDU-07 19
Figura 3-10 Diagrama de actividad CDU-08 20
Figura 3-11 Diagrama de actividad CDU-09 21
Figura 3-12 Diagrama de actividad CDU-10 22

Figura 3-13 Diagrama de actividad CDU-11 23
Figura 3-14 Diagrama de actividad CDU-12 24
Figura 3-15 Ejemplo de un paciente que otorga dos consentimientos 27
Figura 3-16 Ejemplo de un nodo profesional y sus relaciones 27
Figura 3-17 Ejemplo de un nodo información y sus relaciones 28
Figura 3-18 Ejemplo de un nodo consentimiento y sus relaciones 28
Figura 3-19 Carga de datos 29
Figura 3-20 Creación de los nodos Información 29
Figura 3-21 Creación de una relación entre dos ndodos 29
Figura 3-22 Fichero de configuración application.properties 37
Figura 3-23 Esquema general del servicio REST 38
Figura 3-24 Clase Profesional.java 38
Figura 3-25 Clase ProfesionalController.java y objeto IProfesional instanciado. 39
Figura 3-26 Método showById al recibir una petición GET 39
Figura 3-27 Método obtenerRolNombre al recibir una petición GET 40
Figura 3-28 Clase ProfesionalImpl.java y objeto ProfesionalDao instanciado. 41
Figura 3-29 Clase ProfesionalDAO.java 41
Figura 3-30 Clase ProfesionalDto.java 42
Figura 3-31 Clase MensajeResponse.java 42
Figura 3-32 Clase Paciente.java 43
Figura 3-33 Clase PacienteController.java y objeto IPaciente instanciado 43
Figura 3-34 Método showById al recibir una petición GET 44
Figura 3-35 Método obtenerCons al recibir una petición GET 45
Figura 3-36 Clase InformacionController.java 47
Figura 3-37 Clase ConsentimientoController.java 49
Figura 3-38 Método obtenerConsentimiento al recibir una petición GET 49
Figura 3-39 Diagrama de flujo de cómo se procesa el método obtenerConsentimiento 51

Notación

GDPR	General Data Protection Regulation (Reglamento General de Protección de Datos)
ABAC	Attribute-based access control (Control de acceso basado en atributos)
PIP	Policy Information Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
REST	Representational State Transfer
UE	Unión Europea
BD	Base de datos
XML	Extensible Markup Language
UML	Unified Modeling Language
HTTP	Hypertext Transfer Protocol
API	Application Programming Interface
CASE	Computer Aided Software Engineering
CDU	Caso de Uso
JSON	JavaScript Object Notation
DAO	Data Access Object
URL	Uniform Resource Locator
DTO	Data Transfer Object

1 INTRODUCCIÓN

"El principio es la mitad del todo."
- Pitágoras de Samos -

En este primer capítulo se presenta la motivación que ha dado origen a este proyecto exponiendo el objetivo específico que se persigue con este trabajo. A continuación, se introduce la metodología empleada para su consecución. Finalmente, se describe el plan de trabajo seguido, incluyendo las tareas realizadas. Los capítulos posteriores profundizarán más en el contenido del proyecto.

1.1 Motivación y objetivos

En las últimas dos décadas, el uso de Internet ha crecido de manera exponencial. Para ofrecer un servicio de calidad, los sitios web, las plataformas de redes sociales, los programas y otros recursos en línea recopilan y almacenan información personal identificable de los usuarios, así como otros datos relevantes. Sin embargo, la mayoría de los ciudadanos no somos plenamente conscientes de la importancia crítica que tiene la privacidad y la seguridad de estos datos.

En respuesta a esta necesidad de protección, surge el Reglamento General de Protección de Datos (GDPR, por sus siglas en inglés), una normativa que impone a las empresas y organizaciones la obligación de proteger los datos y la privacidad de los ciudadanos de la Unión Europea en todas las transacciones realizadas dentro de los estados miembros [1]. Bajo esta regulación, el consentimiento para el tratamiento de datos se convierte en un elemento central y de vital importancia.

A menudo, compartimos información personal sin ser realmente conscientes de cómo se utilizará o quién tendrá acceso a ella. Este problema es especialmente delicado en el ámbito de la salud, donde la intimidad, la confidencialidad y la asistencia sanitaria están intrínsecamente conectadas en la relación entre los profesionales sanitarios y los pacientes. Sin embargo, la privacidad sigue siendo uno de los aspectos más frecuentemente descuidados en la atención sanitaria.

Para asegurar que solo el personal autorizado pueda acceder a datos sensibles, surge la necesidad de implementar sistemas de control de acceso robustos y flexibles. En este contexto, el modelo de Control de Acceso Basado en Atributos (ABAC) ofrece una solución que permite tomar decisiones de acceso basadas en atributos específicos asociados a usuarios, acciones y recursos [2]. Uno de los elementos clave del modelo ABAC, como veremos en el capítulo siguiente, es el Punto de Información de Políticas (PIP, *Policy Information Point*) que es quien se encarga de almacenar los atributos de usuarios, acciones y recursos, y proporcionarlos en el momento en que se tenga que tomar una decisión de acceso.

Dado que los datos en el ámbito sanitario están altamente interrelacionados y requieren un manejo seguro, este trabajo se centra en explorar cómo una base de datos basada en grafos, como Neo4j, puede ser utilizada para implementar el componente PIP dentro del modelo ABAC en entornos sanitarios. Neo4j, con su capacidad para gestionar relaciones complejas, proporciona un marco ideal para representar y consultar los atributos y relaciones que son críticos para las decisiones de acceso.

El objetivo de este proyecto es el diseño y despliegue de una base de datos Neo4j para el almacenamiento de los atributos para el control de acceso así como el desarrollo de una API REST que interactúe con Neo4j para verificar dinámicamente los atributos necesarios para la toma de decisiones de acceso, garantizando que solo el personal autorizado pueda acceder a la información de salud según las políticas establecidas. Esta solución no solo busca cumplir con las normativas de privacidad y seguridad, como el GDPR, sino también ofrecer un sistema eficiente y escalable para la gestión de accesos en el sector sanitario.

Dicho proyecto se basa en las siguientes tareas:

- Estudio y comprensión del GDPR.
- Estudio y análisis del modelo ABAC, concretamente, el componente PIP.
- Estudio y análisis de las bases de datos basadas en grafos.
- Estudio y comprensión del lenguaje CYPHER, lenguaje desarrollado por Neo4j.
- Diseño de los casos de uso que representan las funcionalidades del software.
- Creación y despliegue de una base de datos basada en grafos, específicamente, Neo4j para crear los atributos necesarios.
- Diseño, implementación y despliegue de un servicio que siga el modelo REST, que nos permita realizar las acciones deseadas.
- Pruebas y corrección del sistema completo.
- Estudio de nuevas posibilidades y horizontes futuros.

En la Figura 1-1 se ilustra el esquema general del proyecto; cómo la API REST actúa como intermediario entre la base de datos y los clientes, permitiendo el acceso y la gestión de datos almacenados en Neo4j. Los clientes de nuestra API serán elementos de decisión de control de acceso que necesitan atributos sobre las entidades implicadas en la solicitud de acceso para tomar la decisión.

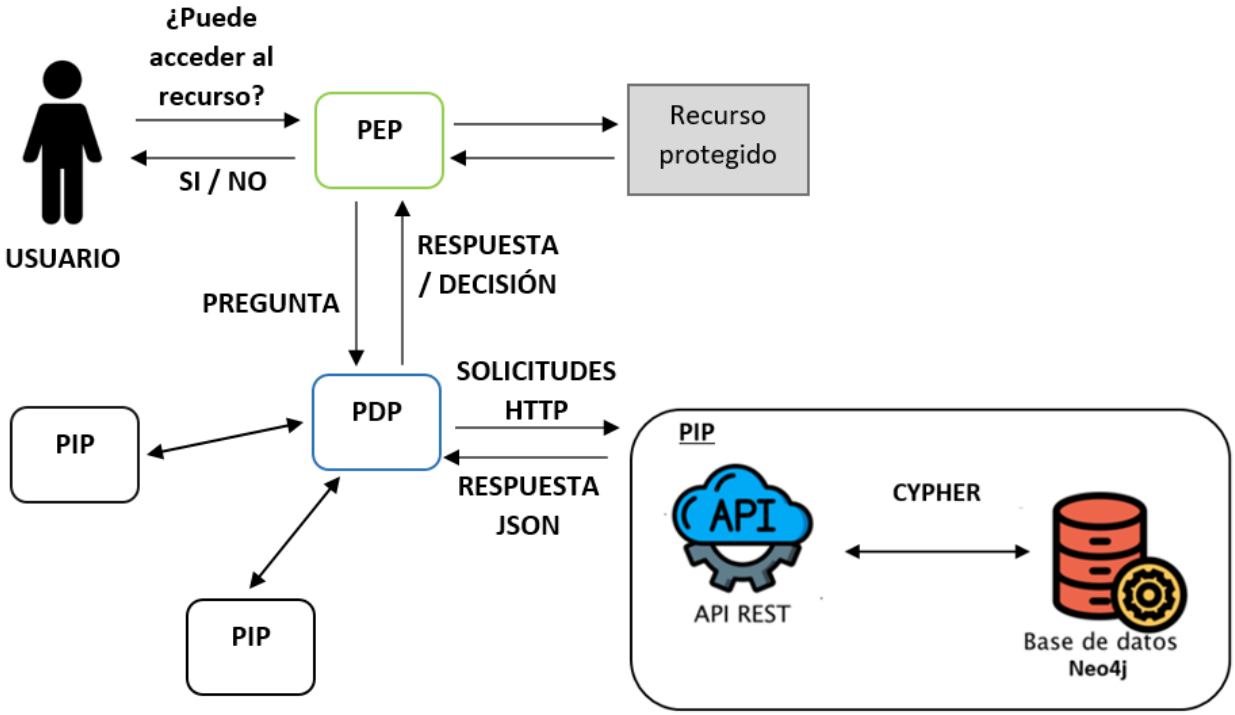


Figura 1-1 Representación del funcionamiento del sistema.

1.2 Metodología

Para el desarrollo de este proyecto, se ha optado por el uso de la metodología incremental [3]. Este modelo permite construir el sistema de manera progresiva, desarrollando, probando y mejorando el proyecto en incrementos o iteraciones hasta conseguir el propósito final. Cada incremento se compone de fases específicas, como análisis, diseño, codificación y prueba, con el objetivo de desarrollar y entregar una versión funcional del sistema. Es importante destacar que no se trata de iteraciones independientes, al contrario, están vinculadas de forma que cada una suponga un avance con respecto a la anterior permitiendo ajustes y refinamientos a lo largo del proceso.

Este modelo tiene las siguientes características esenciales:

- Los incrementos son pequeños.
- Permite una fácil administración de las tareas en cada iteración.
- La inversión se materializa a corto plazo.
- Es un modelo propicio a cambios o modificaciones.
- Se adapta a las necesidades que surjan.

A continuación, en la Figura 1-2 se puede observar cómo opera el modelo incremental, destacando las iteraciones y las fases involucradas en cada una de ellas.

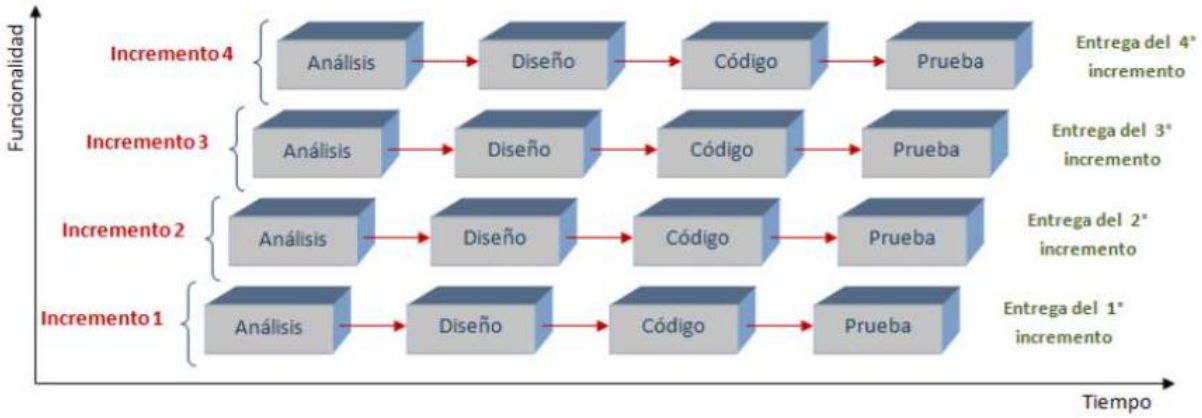


Figura 1-2 Representación de la metodogía incremental

1.3 Plan de trabajo

En primer lugar, se consideró esencial llevar a cabo un estudio del contexto teórico que rodea este proyecto. Esta etapa inicial de investigación fue fundamental para asegurar un desarrollo eficiente y alineado con los objetivos del trabajo. Por ello, se realizaron las siguientes tareas preliminares:

i. Estudio e interpretación del GRPD.

ii. Estudio e interpretación del modelo ABAC, concretamente, el componente PIP.

iii. Estudio y análisis de las bases de datos basadas en grafos, concretamente, Neo4j.

iv. Estudio y comprensión del lenguaje CYPHER, lenguaje desarrollado por Neo4j.

Una vez finalizada esta fase teórica, se avanzó hacia las etapas prácticas de diseño, implementación y pruebas, con las siguientes actividades:

v. Diseño de los casos de uso que representan las funcionalidades del software.

vi. Creación y despliegue de una base de datos Neo4j.

vii. Diseño y despliegue de una API REST, conforme a los casos de uso previamente definidos.

viii. Iteración entre el desarrollo de la API y la base de datos, realizando ajustes en la estructura de la base de datos Neo4j a medida que se identificaban necesidades y fallos durante la implementación de la API.

ix. Pruebas exhaustivas y correcciones continuas, verificando el correcto funcionamiento del sistema.

Para cada una de estas tareas, se planificó un número estimado de horas, y se compararon con las horas reales empleadas. Estos datos se recogen en la Tabla 1-1, proporcionando una visión clara de la planificación y la ejecución del proyecto.

Tabla 1 Horas estimadas y reales para el plan de trabajo

TAREAS	HORAS ESTIMADAS	HORAS REALES
Estudio e interpretación del GRPD	2	3
Estudio e interpretación del modelo ABAC, concretamente, el componente PIP	2	2
Estudio y análisis de las bases de datos basadas en grafos, concretamente, Neo4j	10	12
Estudio y comprensión del lenguaje CYPHER, lenguaje desarrollado por Neo4j	5	6
Diseño de los casos de uso que representan las funcionalidades del software	10	8
Creación y despliegue de una base de datos Neo4j	15	20
Diseño y despliegue de una API REST, conforme a los casos de uso previamente definidos	60	80
Iteración entre el desarrollo de la API y la base de datos, realizando ajustes en la estructura de la base de datos Neo4j a medida que se identificaban necesidades y fallos durante la implementación de la API.	7	9
Pruebas exhaustivas y correciones continuas, verificando el correcto funcionamiento del sistema.	50	60
TOTAL	161	200

2 MARCO TEÓRICO

Aprender no es prepararse para la vida. Aprender es la vida misma. John Dewey.

A lo largo de este capítulo, se explorará en detalle el marco teórico que sustenta el desarrollo del proyecto. Se comenzará analizando la normativa GDPR, fundamental para asegurar la protección de los datos. Posteriormente, se explicará el modelo ABAC (Control de Acceso Basado en Atributos), un componente esencial, facilitando un control de acceso dinámico y adaptable en el sistema. A continuación, se abordarán las bases de datos basadas en grafos, haciendo hincapié en Neo4j y su lenguaje de consulta Cypher. Finalmente, se presentarán las herramientas software empleadas en el proyecto, las cuales han sido cruciales para la implementación y despliegue del sistema.

2.1 Norma GDPR

El Reglamento General de Protección de Datos (GDPR) (Reglamento 2016/679) es un reglamento por el que el Parlamento Europeo, el Consejo de la Unión Europea y la Comisión Europea tienen la intención de reforzar y unificar la protección de datos para todos los individuos dentro de la Unión Europea (UE). El objetivo principal del GDPR es dar control a los ciudadanos y residentes sobre sus datos personales y simplificar el entorno regulador de los negocios internacionales unificando la regulación dentro de la UE.

Uno de los pilares fundamentales del GDPR es el concepto de consentimiento, que constituye uno de los fundamentos jurídicos en los que debe basarse el tratamiento de datos personales. Este concepto deberá cumplir las siguientes condiciones para que se considere válido [4]:

- Libre: El consentimiento debe ser otorgado de manera voluntaria, sin coerción ni desequilibrio de poder. El responsable del tratamiento debe demostrar que la persona tiene la opción de negarse o retirar su consentimiento sin enfrentar repercusiones negativas.
- Específico: El consentimiento debe ser otorgado para fines específicos, y el interesado debe poder elegir si autoriza el tratamiento para cada uno de esos fines. Esto implica que el consentimiento debe estar claramente delimitado en función de los propósitos del tratamiento.
- Informado: La persona debe recibir información clara y comprensible sobre lo que implica el tratamiento de sus datos. Esto incluye detalles sobre los propósitos del tratamiento, la identidad del responsable, y cualquier otra información relevante para que el interesado comprenda plenamente a qué está consintiendo.
- Inequívoco: El consentimiento requiere una declaración del interesado o una clara acción afirmativa, lo que significa que siempre debe darse el consentimiento mediante una acción o declaración.

Además, el GDPR introduce requisitos adicionales con el fin de garantizar la validez del consentimiento:

- Demostrar el consentimiento: Las organizaciones deben demostrar que han obtenido un consentimiento válido, de manera que puedan demostrar cómo y cuándo se obtuvo dicho consentimiento y también la información que se le facilitó a la persona que autoriza dicho consentimiento.
- Retirada del consentimiento: Las personas pueden retirar el consentimiento en cualquier momento sin que esto perjudique la legalidad del procedimiento. Este proceso debe ser tan fácil como fue dar el consentimiento.

El concepto de consentimiento es central en el proyecto, el cual se enfoca en gestionar y verificar los consentimientos para permitir el acceso a información sensible. A lo largo del trabajo, se analizará cómo se otorgan estos consentimientos y cómo se manejan conforme a los requisitos del GDPR.

2.2 Modelo de control de acceso basado en atributos (ABAC)

ABAC es un método de implementación de políticas de control de acceso que es altamente adaptable y se puede personalizar utilizando una amplia gama de atributos, lo que lo hace adecuado para su uso en entornos distribuidos o que cambian rápidamente [5].

2.2.1 Arquitectura

ABAC describe una arquitectura recomendada que se muestra en la Figura 2-1.

- El PEP o Punto de Cumplimiento de Política: es el encargado de proteger las aplicaciones y los datos a los que se desea aplicar ABAC. El PEP inspecciona la solicitud y genera una solicitud de autorización que envía al PDP. Cuando este le devuelva el resultado de la decision de acceso, el PEP es el encargado de procesarla y trasmitírsela al usuario solicitante de acceso.
- El PDP o Punto de Decisión de Políticas es el cerebro de la arquitectura. Es la parte que evalúa las solicitudes entrantes en función de las políticas con las que se ha configurado. El PDP devuelve una decisión de permitir o denegar al PEP. El PDP también puede consultar distintos elementos PIP para recuperar atributos necesarios para la toma de decisión.
- El PIP o Punto de Información de Políticas conecta el PDP con fuentes externas de atributos, por ejemplo, LDAP o bases de datos.

En este proyecto se realiza un PIP como un servicio REST conectado a una base de datos Neo4j y el PDP es quien solicita información necesaria a dicho servicio. Los atributos que gestiona el PIP estrán almacenados como nodos de Neo4j.

2.2.2 Atributos

Los atributos pueden referirse a cualquier cosa y a cualquier persona. Suelen clasificarse en cuatro categorías diferentes:

- Atributos del sujeto: describen al usuario que intenta acceder, por ejemplo, edad.
- Atributos de acción: describen la acción que se está intentando realizar, por ejemplo, eliminar.
- Atributos del objeto: caracterizan al objeto (o recurso) protegido al que se intenta acceder, por ejemplo, el tipo de objeto (historial médico).
- Atributos contextuales (de entorno): tratan sobre el tiempo, la ubicación o aspectos dinámicos del escenario de control de acceso.

2.2.3 Políticas

Las políticas son declaraciones que reúnen atributos para expresar quién puede acceder o no a los recursos protegidos. Las políticas en ABAC pueden ser políticas de concesión o denegación. Las políticas también pueden ser locales o globales y pueden redactarse de manera que anulen otras políticas.

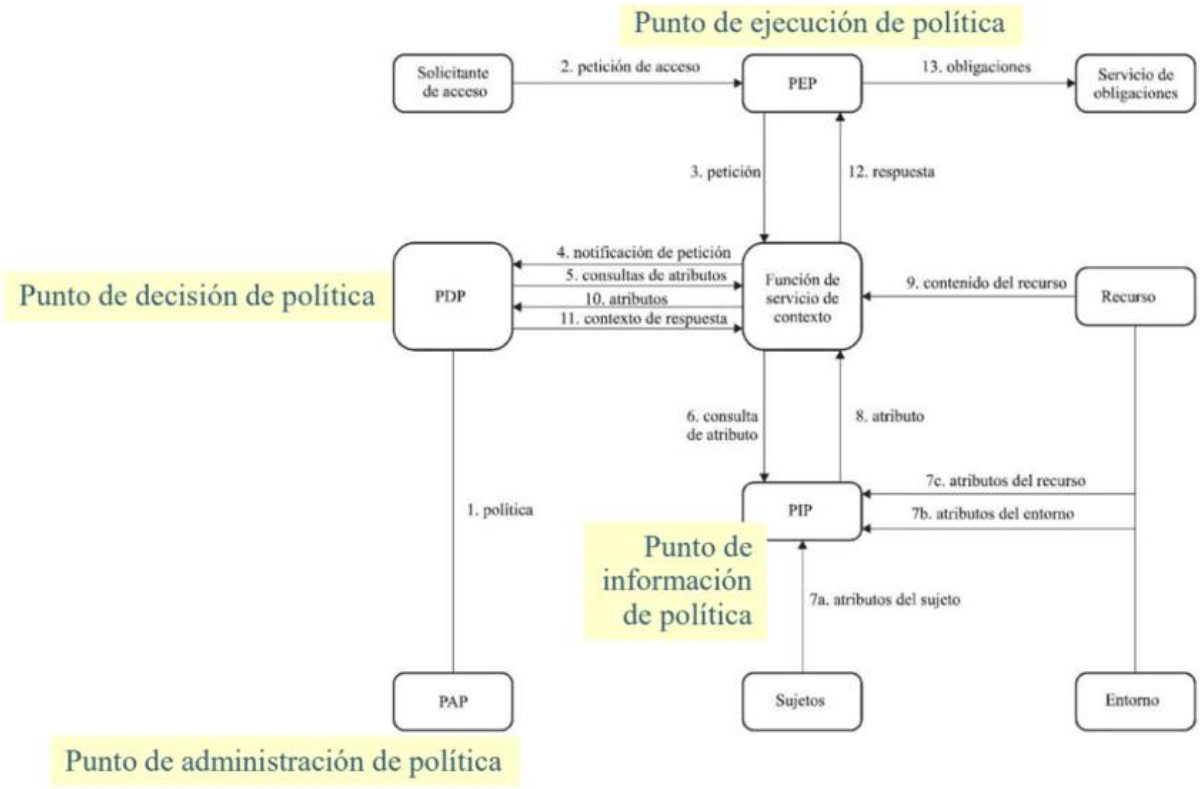


Figura 2-1 Modelo ABAC

2.3 Base de datos basada en grafos

Una base de datos orientada a grafos es una plataforma especializada diseñada para crear, gestionar y manipular grafos de manera eficiente [6]. Los grafos, originados en el campo de las matemáticas, se componen de nodos (también llamados vértices) y aristas, que representan las relaciones entre estos nodos. Estos elementos permiten representar y almacenar datos de forma clara, intuitiva y visual. Además, se pueden usan 2 elementos adicionales: etiqueta y propiedad.

- Nodos: representan objetos, entidades o cosas. Se ilustran mediante círculos
- Relaciones: deben tener una dirección y un tipo que permita elegir en el momento de la consulta qué parte del gráfico se está atravesando. Se muestran mediante líneas de intersección.
- Etiqueta: al agregar una etiqueta a un nodo se indica que el nodo pertenece a un subconjunto de nodos. El nodo puede tener cero, una o muchas etiquetas.
- Propiedad: agregar propiedades a un nodo o a una relación permite añadir información a estos elementos en el formato clave-valor. No es necesario que existan propiedades para un nodo con una etiqueta particular.

En la figura 2-2 se muestra un ejemplo de un gráfico social, en el cual las personas interactúan entre sí y forman relaciones de diferentes intensidades.

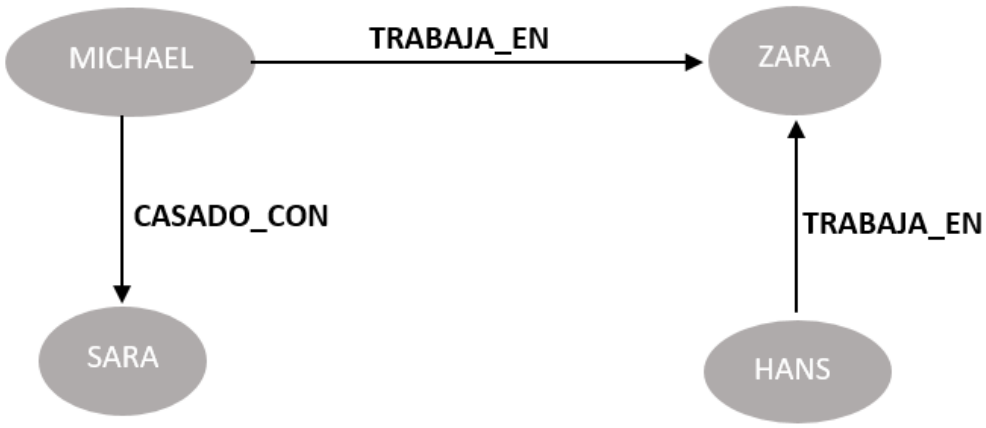


Figura 2-2 Gráfico social

Adicionalmente, las bases de datos de grafos poseen características técnicas que son consideradas directamente como ventajas frente a modelos de gestión de datos relacionales [7].

Estas son las tres ventajas principales de la analítica de grafos:

1. Flexibilidad.
   Este modelo cuenta con información actualizada y disponible en todo momento, permitiendo modificar las estructuras sin afectar a las funciones existentes.
2. Rendimiento.
   El rendimiento es constante y escalable en la medida en que el volumen de datos aumenta, sin perjudicar la velocidad de las consultas.
3. Eficiencia.
   Las consultas de grafos son más cortas y más eficientes, permitiendo realizar testeos o pruebas que ayuden a evolucionar los procesos, así como también permite realizar proyecciones fiables sobre el comportamiento de los datos.

2.3.1 Neo4j

Neo4j es la base de datos de grafos nativa por excelencia, lo que significa que desde el almacenamiento de datos hasta el lenguaje de consulta se ha diseñado teniendo en cuenta el recorrido [8]. Su principal característica es el concepto de adyacencia sin índice, es decir, cada nodo es consciente de cada relación entrante y saliente conectada a él. Por tanto, el motor gráfico subyacente buscará sólo punteros en la memoria.

2.3.2 Cypher

Cypher es un lenguaje de consulta diseñado para grafos. A continuación, se presentan algunos de los elementos y patrones básicos utilizados en Cypher [9]:

Tabla 2 Elementos y Patrones de Cypher

ELEMENTOS Y PATRONES	EJEMPLO
Los nodos están representados por **()**	(p)
Se usan **dos puntos** para indicar las etiquetas	(**:**Person)
Las relaciones entre nodos se escriben con **dos guiones**	(p:Person) - - (:Movie)
La dirección de una relación con un símbolo <>	(:Person) -- > (:Movie)
El tipo de relación se escribe utilizando los **corchetes** entre los dos guiones	[:ACTED_IN]
Las propiedades son pares clave/valor	{name:'Tom Hanks'} p.name='Tom Hanks'

Este lenguaje trabaja encontrando coincidencias en los patrones de los datos. Algunas de las cláusulas fundamentales incluyen:

- MATCH: se utiliza para recuperar datos.
- RETURN: devuelve los datos obtenidos de la consulta.
- WHERE: especifica qué nodo o relación se desea devolver según una condición.

En la Figura 2-3 se muestra un ejemplo de estas cláusulas, donde se realiza una consulta para devolver el nodo de una profesional cuyo nombre es Juan Jiménez y la especialidad a la que pertenece.

Figura 2-3 Consulta

2.3.2.1 Filtrado de Consultas

La **cláusula WHERE** se utiliza para filtrar los nodos y relaciones que se recuperan del grafo. Los métodos de filtrado más comunes incluyen:

I. Filtrado por Etiqueta de Nodo
Se hace referencia a las etiquetas de los nodos separadas por el conector AND.

II. Filtrado usando rangos
Se menciona un rango separado por <=,>= para limitar los resultados.

III. Filtrado por Existencia de una Propiedad
Se usa la cláusula IS NOT NULL para indicar si existe una propiedad de un determinado nodo.

IV. Filtrado por Cadenas Parciales
Se indica la cláusula STARTS WITH para saber por dónde comienza una propiedad de un nodo. Para las cadenas se debe usar toLower() o toUpper() para garantizar los resultados correctos.

V. Filtrado por Patrones en el Gráfico
Se usa la cláusula NOT EXISTS para especificar que la consulta debe devolver únicamente aquellos nodos o relaciones que no cumplen con un patrón específico.

VI. Filtrar usando Listas
Se utiliza la cláusula IN para comparar un valor con una lista existente en el gráfico o una lista que se indica.

2.3.2.2 Operaciones

A. Nodos
A.1 Creación

Hay dos formas de crear nodos:

1) MERGE: crea sólo un nodo sino existe dado que busca si existe la clave principal antes de crear el nodo. Si ya existe no crea nada. Hay que especificar la etiqueta del nodo y una propiedad que será la clave principal para cada nodo. Se pueden encadenar varias cláusulas MERGE dentro de un único bloque de código.

```
MERGE (p:Profesional {name:'Antonio Romero'})
MERGE (o:Organización {name:'Hospital Central'})
```

Figura 2-4 Ejemplo MERGE

2) CREATE: crea uno o más de un nodo. No busca la clave principal antes de agregar un nodo.

```
CREATE (p1:Paciente {name:'Cati Jimenez'}) ,
(p2:Paciente {name:'Inma Cid'})
```

Figura 2-5 Ejemplo CREATE

A.2 Eliminación

- Si el nodo no tiene relaciones:
    1) Se recupera el nodo con MATCH.
    2) Se elimina el nodo usando DELETE.

```
MATCH (p1:Paciente {name:'Inma Cid'})
DELETE p1
```

Figura 2-6 Ejemplo eliminación de nodo sin relaciones

- Si el nodo tiene relaciones:
    1) Se recupera el nodo con MATCH.
    2) Se elimina el nodo y sus relaciones con DETACH DELETE.

```
MATCH (p:Profesional {name:'Juan Jimenez'})
DETACH DELETE p
```

Figura 2-7 Ejemplo de eliminación de nodo con relaciones

B. Relaciones

B.1 Creación

Para crear relaciones se usa MERGE que comprueba si ya existen previamente, en cuyo caso no se hace nada. Deben cumplirse varios requisitos:

1) Debe tener referencias a los 2 nodos para los que se creará la relación.
2) Cuando se crea la relación, esta debe tener tipo y dirección. Si la dirección no se especifica, de forma predeterminada se usa la dirección de izquierda a derecha.

En la figura 2-8 se especifica un ejemplo donde se recuperan dos nodos y a continuación se crea la relación. Por el contrario, en la figura 2-9 donde se crean los dos nodos y la relación entre ellos en una misma cláusula.

```
MATCH (p1:Paciente {name:'Cati Jimenez'})
MATCH (c:Consentimiento {estado:'activo'})
MERGE (p1)-[:OTORGA]→(c)
```

Figura 2-8 Ejemplo de creación de relación entre dos nodos

```
MERGE (p:Profesional {name:'Juan Diego Muñoz'})-
[:TIENE]→(r:Rol {name:'Medico'})
```

Figura 2-9 Ejemplo de creación de nodos y relación

B.2 Eliminación

Para eliminar una relación primero se recupera una referencia a la relación y después se elimina la relación usando esa referencia con la cláusula DELETE.

```
MATCH (p:Profesional {name:'Juan Diego Muñoz'})-
[t:TIENE]→(r:Rol {name:'Medico'})
DELETE t
```

Figura 2-10 Ejemplo de eliminación de una relación

C. Propiedades

C.1 Agregación

Hay dos formas de agregar propiedades:

1) En línea como parte de la cláusula MERGE como en la figura 2-9.
2) Usar SET como referencia a un nodo o relación. En este caso, primero se debe recuperar el nodo con MATCH y seguidamente se agrega la propiedad con SET como ocurría más arriba en la creación de relaciones.

```
MATCH (p:Profesional {name:'Juan Diego Muñoz'})
SET p.genero = 'masculino'
```

Figura 2-11 Agregación de propiedades

C.2 Actualización

Para actualizar las propiedades se usa SET como en la figura 2-11 anterior.

C.3 Eliminación

Existen dos formas de eliminar propiedades:

1) Haciendo uso de la cláusula REMOVE.

```
MATCH (p:Profesional {name:'Juan Diego Muñoz'})
REMOVE p.genero
```

Figura 2-12 Eliminación de una propiedad con REMOVE

2) Utilizando SET con la propiedad igual a NULL.

```
MATCH (p:Profesional {name:'Juan Diego Muñoz'})
SET p.genero = NULL
```

Figura 2-13 Eliminación de una propiedad con SET

Para eliminar todos los nodos, todas las relaciones y las propiedades de ambos:

```
MATCH (n)
DETACH DELETE (n)
```

Figura 2-14 Eliminación total

2.3.2.3 Carga de datos con CSV

Los archivos csv almacenan datos en una estructura donde un carácter especial, en este proyecto se hace uso del punto y coma, aunque el carácter más habitual es la coma, dicho carácter separa cada valor en una fila, y cada registro es una nueva fila en el archivo. La primera fila de este archivo es una fila de encabezado que contiene los nombres de las columnas, esta fila no es obligatoria.

La cláusula **LOAD CSV** es la encargada de leer los datos de un archivo CSV y devolver las filas para su manipulación. En la figura 2-15 se puede observar un ejemplo de cómo cargar un archivo CSV. Es importante mencionar que siempre se debe incluir el elemento RETURN para obtener los resultados.

```
LOAD CSV [WITH HEADERS]
FROM 'url' [AS alias]
[FIELDTERMINATOR 'char']
RETURN alias
```

Figura 2-15 Carga de archivo csv

Cuando los datos se cargan desde un CSV, la creación de nodos y relaciones se realiza de manera masiva en lugar de individual. Por ejemplo, un solo comando LOAD CSV puede crear múltiples nodos en una sola ejecución. En lugar de especificar cada nodo y relación por separado, se recorre el archivo CSV para crear automáticamente nodos basados en las columnas del archivo y relaciones basadas en las asociaciones definidas por los datos.

2.4 Herramientas software empleadas

2.4.1 Spring Tool Suite 4

Spring Tool Suite (STS) es un entorno de desarrollo integrado (IDE) basado en Eclipse, diseñado específicamente para facilitar el desarrollo de aplicaciones utilizando el Spring Framework [10]. Eclipse es una plataforma de software que incluye un conjunto de herramientas de programación de código abierto y multiplataforma, ideal para diversos lenguajes y proyectos [11]. STS mejora la experiencia de desarrollo en Java al integrar características específicas de Spring, como la creación de proyectos Spring [12]. Con Spring, la programación en Java se vuelve más rápida, sencilla y segura.

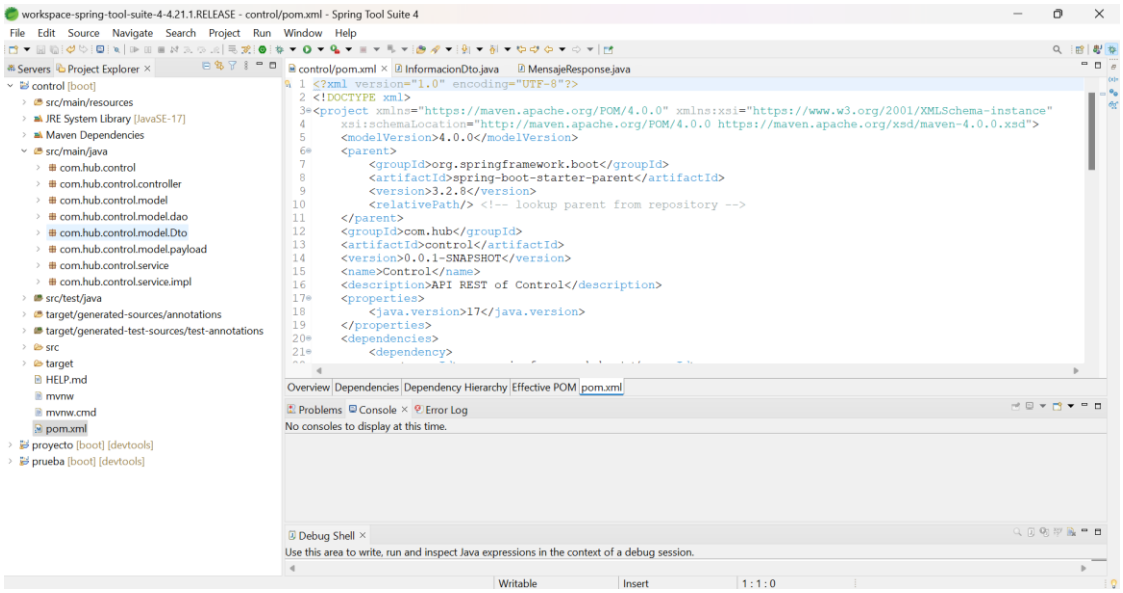


Figura 2-16 IDE Spring Tool Suite 4

2.4.2 API Spring Data Neo4j

Spring Data Neo4j, parte de la familia Spring Data más grande, proporciona una configuración y un acceso sencillos a las bases de datos de gráficos Neo4j desde las aplicaciones Spring. Ofrece tres niveles diferentes de abstracción para acceder a la tienda: el cliente Neo4j, la plantilla Neo4j y los repositorios Neo4j [13]. Spring Data Neo4j ofrece funciones avanzadas para mapear clases de entidad anotadas a la base de datos de gráficos Neo4j.

Figura 2-17 Logo Spring Data Neo4j

2.4.3 Neo4j Desktop

Neo4j Desktop es un entorno de desarrollo local para crear la base de datos Neo4j mediante instancia de base de dato local necesaria en el proyecto [14]. Con Neo4j Desktop, puede crear y administrar cualquier cantidad de bases de datos locales que su máquina admita. Desktop también proporciona y aloja una serie de aplicaciones gráficas, que son aplicaciones diseñadas específicamente para trabajar con sus bases de datos gráficas.

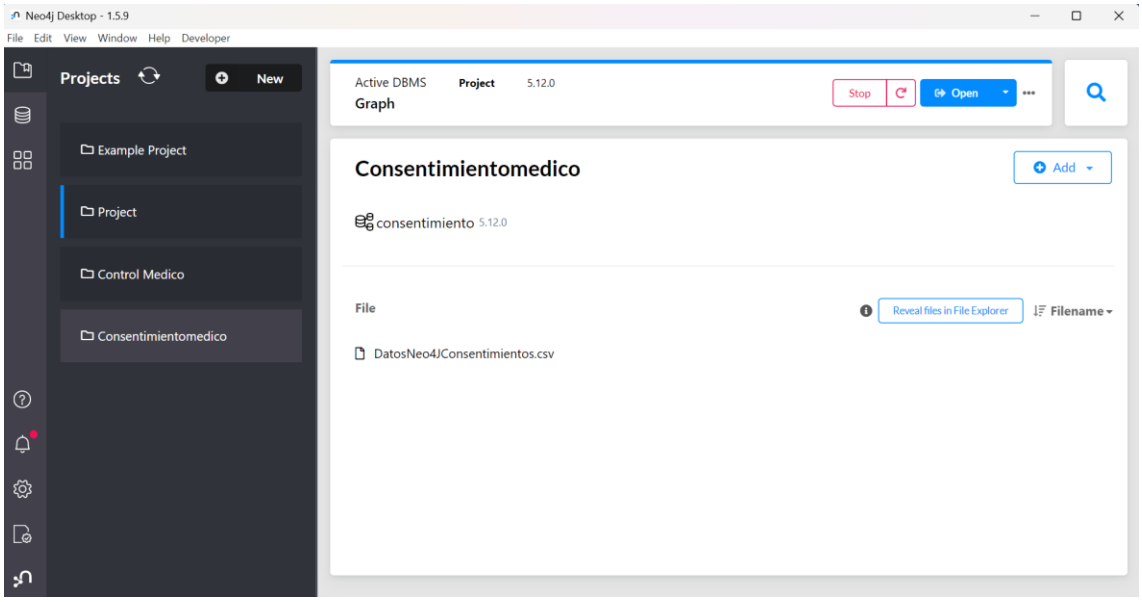


Figura 2-18 IDE Neo4j Desktop

2.4.4 Maven

Apache Maven es una herramienta de comprensión y gestión de proyectos de software. Permite construir proyectos de manera consistente y proporciona información detallada sobre el proyecto a través del archivo *pom.xml*. Gracias a este fichero, es posible gestionar dependencias y plugins de manera eficiente. Por ejemplo, se pueden añadir dependencias como *spring-boot-starter-data-neo4j, lombok, spring-boot* (framework que facilita la creación de aplicaciones independientes basadas en Spring y de nivel de producción que puedes "simplemente ejecutar") [15].

Figura 2-19 Logo Maven

2.4.5 Interfaz REST

REST (Representational State Transfer) es una interfaz que permite la comunicación entre sistemas utilizando el protocolo HTTP, además, facilita la obtención y manipulación de datos mediante operaciones como GET, POST, PUT y DELETE, devolviendo la información en formatos específicos, principalmente JSON y XML [16].

Este servicio destaca por su simplicidad en comparación con otras arquitecturas como SOAP, simplifica el desarrollo de APIs, facilitando la creación de servicios que gestionan la lógica de negocio y exponen los datos a través de URLs y recursos específicos.

2.4.6 Postman

Postman es una aplicación api que permite realizar solicitudes HTTP, en el proyecto se ha usado para llevar a cabo las pruebas necesarias sobre el servicio REST con peticiones GET [17].

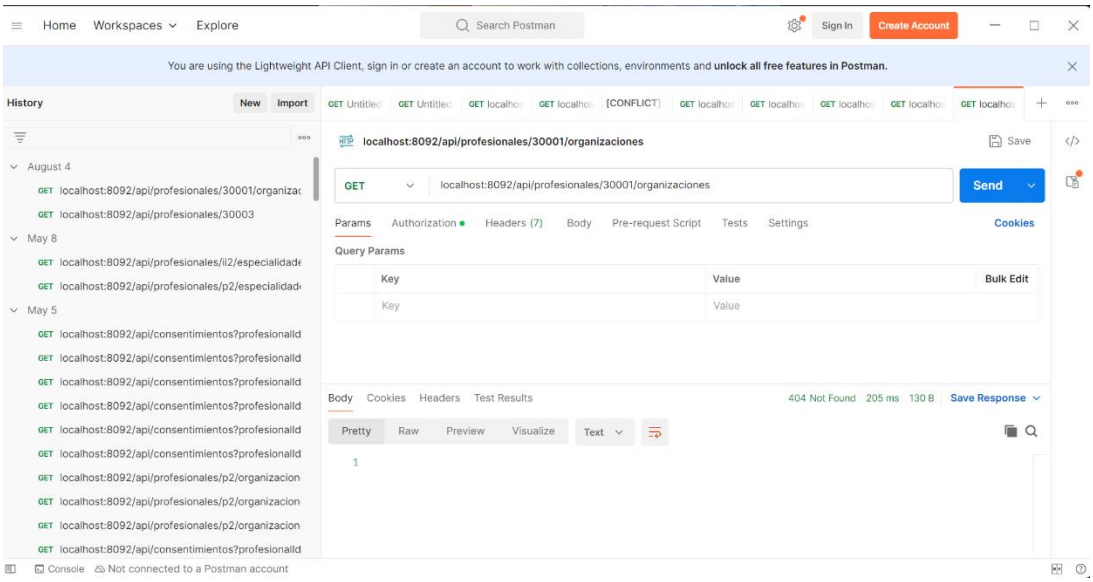


Figura 2-20 IDE Postman

2.4.7 MagicDraw UML

MagicDraw UML es una herramienta CASE compatible con el estándar UML usada en el proyecto para realizar diagramas UML [18].

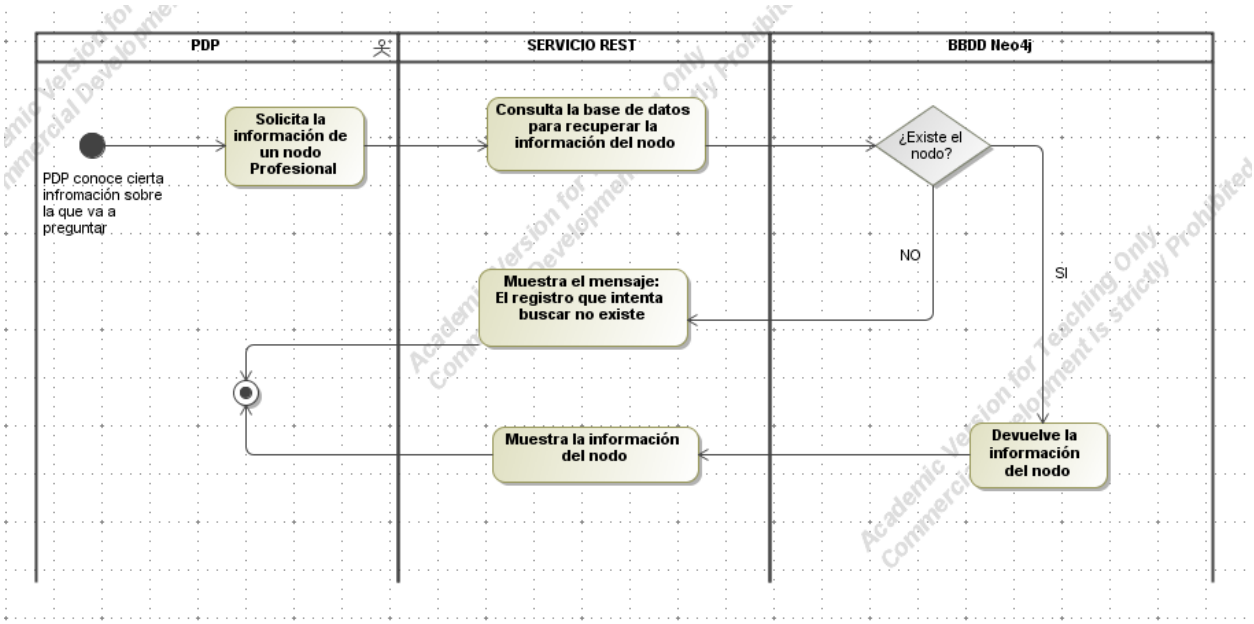


Figura 2-21 Ejemplo diagrama UML

3 RESULTADOS

"La función de un buen software es hacer que lo complejo aparente ser simple"

- Grady Booch -

El objetivo de este capítulo es detallar el desarrollo del proyecto realizado, describiendo los problemas encontrados, las soluciones implementadas y los resultados obtenidos. En primer lugar, se presentarán los casos de uso que definen el funcionamiento general del sistema. A continuación, se describirá en detalle el modelado e implementación de la base de datos en Neo4j, explicando cómo se han estructurado los nodos, relaciones y propiedades para representar de manera eficiente los datos y facilitar las consultas necesarias. Finalmente, se abordará el desarrollo del servicio REST y se realizarán pruebas para validar su funcionamiento.

3.1 Casos de uso

El proyecto se compone de una serie de casos de uso para los que el cliente principal es un PDP. No obstante, al desarrollar el PIP con una interfaz REST, otros tipos de clientes REST pueden beneficiarse de la información que proporciona nuestro PIP.

En la siguiente Figura 3-1 aparecen los posibles casos de uso que forman el sistema. A continuación se describe cada uno de ellos en detalle.

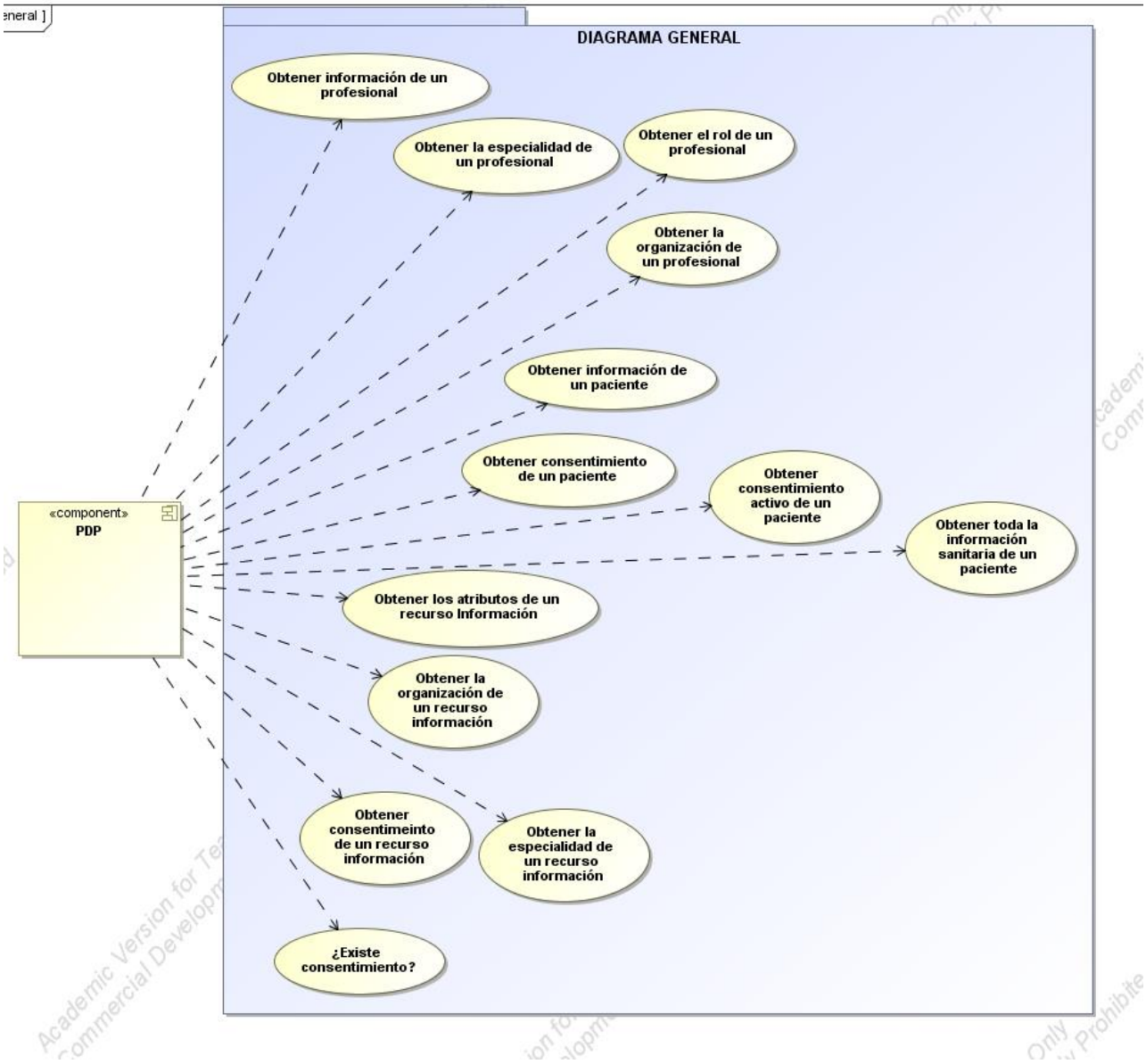


Figura 3-1 Diagrama general de los casos de uso

3.1.1 CDU-01: Obtener información de un profesional

En este caso de uso se pretende obtener la información completa sobre un profesional.

Tabla 3 CDU-01: Obtener información de un profesional

CDU-01	Obtener información de un profesional	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener toda la información relacionada con un profesional. El sistema recupera dicha información y, si la hubiera, la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la información completa de un "Profesional".
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo, el sistema devuelve la información solicitada.
	2.2	Si no existe el nodo, el sistema devuelve un mensaje comunicando que no hay información sobre ese "Profesional".

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

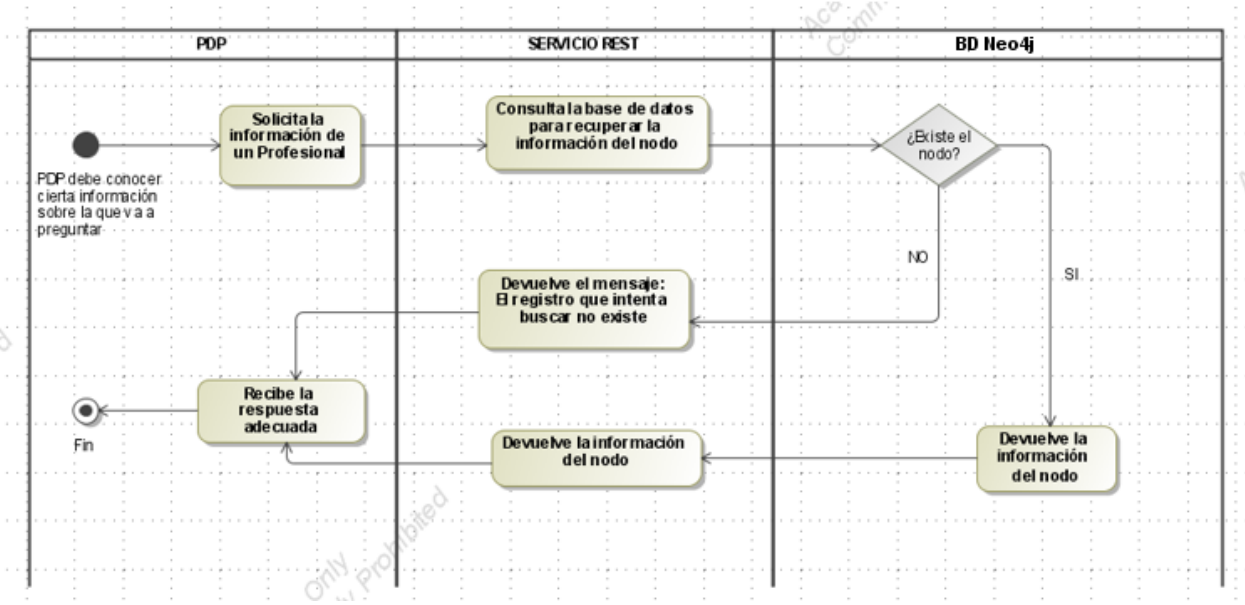


Figura 3-2 Diagrama de actividad CDU-01

3.1.2 CDU-02: Obtener la especialidad de un profesional

En este caso de uso se pretende obtener la especialidad a la que pertenece un profesional.

Tabla 4 CDU-02: Obtener la especialidad de un profesional

CDU-02	Obtener la especialidad de un profesional	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener la especialidad relacionada con un profesional. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la especialidad a la que pertenece un "Profesional".
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo profesional, se comprueba si dicho nodo pertenece a una especialidad.
	2.1.1	Si existe el nodo especialidad, el sistema devuelve la información solicitada.
	2.1.2	Si no existe el nodo especialidad, el sistema devuelve un mensaje indicando que el profesional no pertenece a ninguna especialidad.
	2.2	Si no existe el nodo profesional, el sistema devuelve un mensaje comunicando que el profesional no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

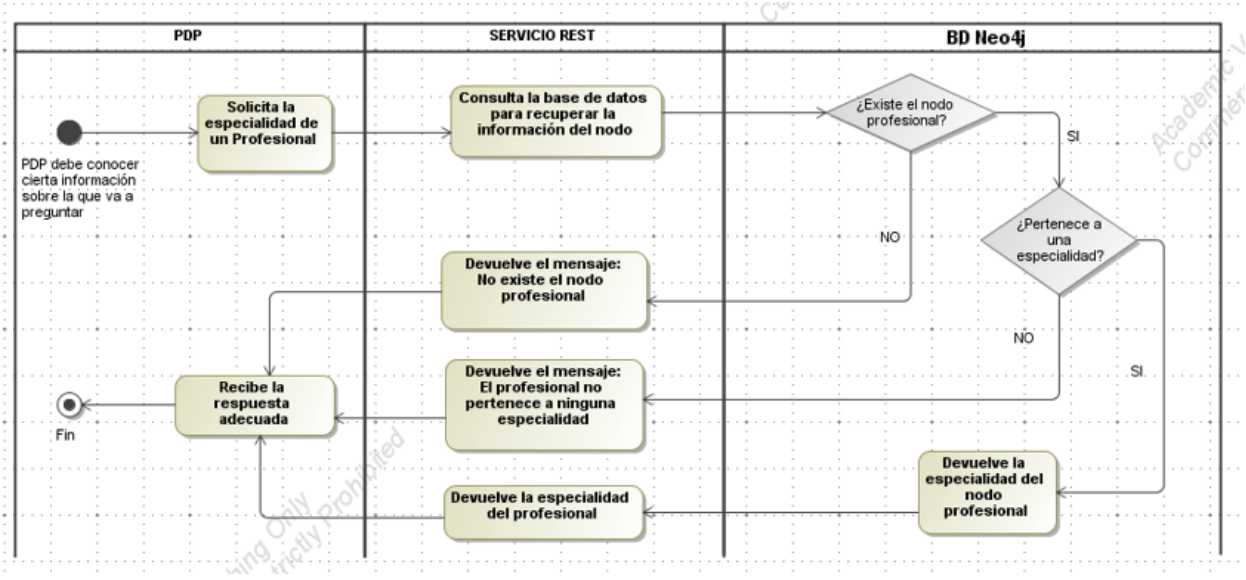


Figura 3-3 Diagrama de actividad CDU-02

3.1.3 CDU-03: Obtener el rol de un profesional

En este caso de uso se pretende obtener el rol que tiene un profesional.

Tabla 5 CDU-03: Obtener el rol de un profesional

CDU-03	Obtener el rol de un profesional	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener el rol relacionado con un profesional. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener el rol que tiene un "Profesional".
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo profesional, se comprueba si dicho nodo tiene un rol.
	2.1.1	Si existe el nodo rol, el sistema devuelve la información solicitada.
	2.1.2	Si no existe el nodo rol, el sistema devuelve un mensaje indicando que el profesional no tiene nigún rol.
	2.2	Si no existe el nodo profesional, el sistema devuelve un mensaje comunicando que el profesional no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

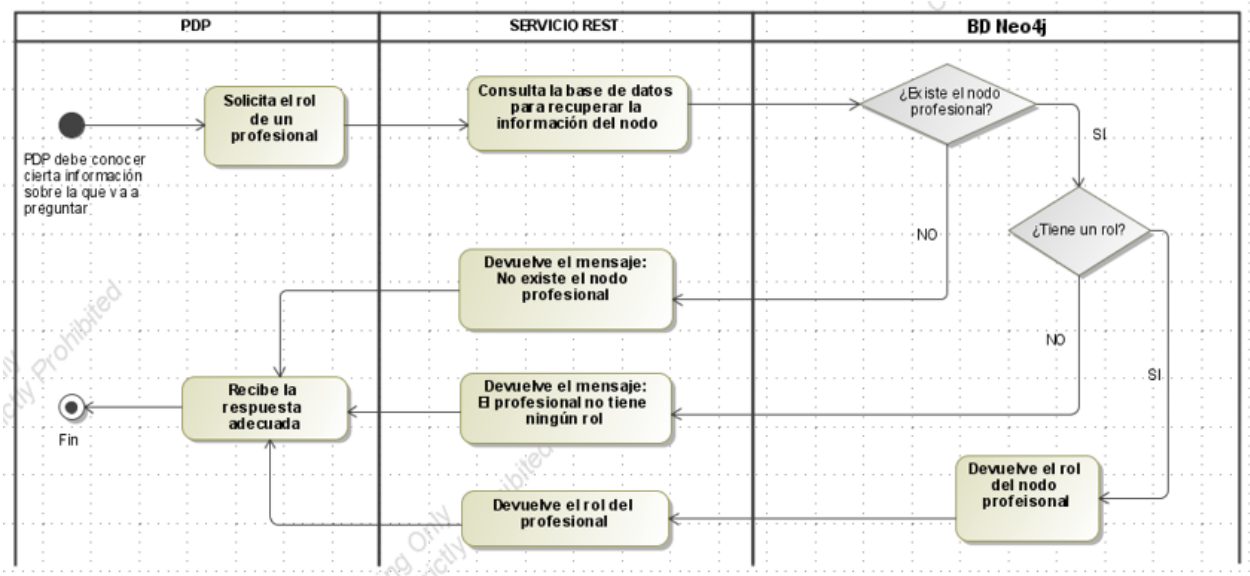


Figura 3-4 Diagrama de actividad CDU-03

3.1.4 CDU-04: Obtener la organización de un profesional

En este caso de uso se pretende obtener la organización en la que trabaja un profesional.

Tabla 6 CDU-04: Obtener la organización de un profesional

CDU-04	Obtener la organización de un profesional	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener la organización relacionada con un profesional. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la especialidad a la que pertenece un "Profesional".
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo profesional, se comprueba si dicho nodo está relacionado con una organización.
	2.1.1	Si existe el nodo organización, el sistema devuelve la información solicitada.
	2.1.2	Si no existe el nodo organización, el sistema devuelve un mensaje indicando que el profesional no trabaja en ninguna organización.
	2.2	Si no existe el nodo profesional, el sistema devuelve un mensaje comunicando que el profesional no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

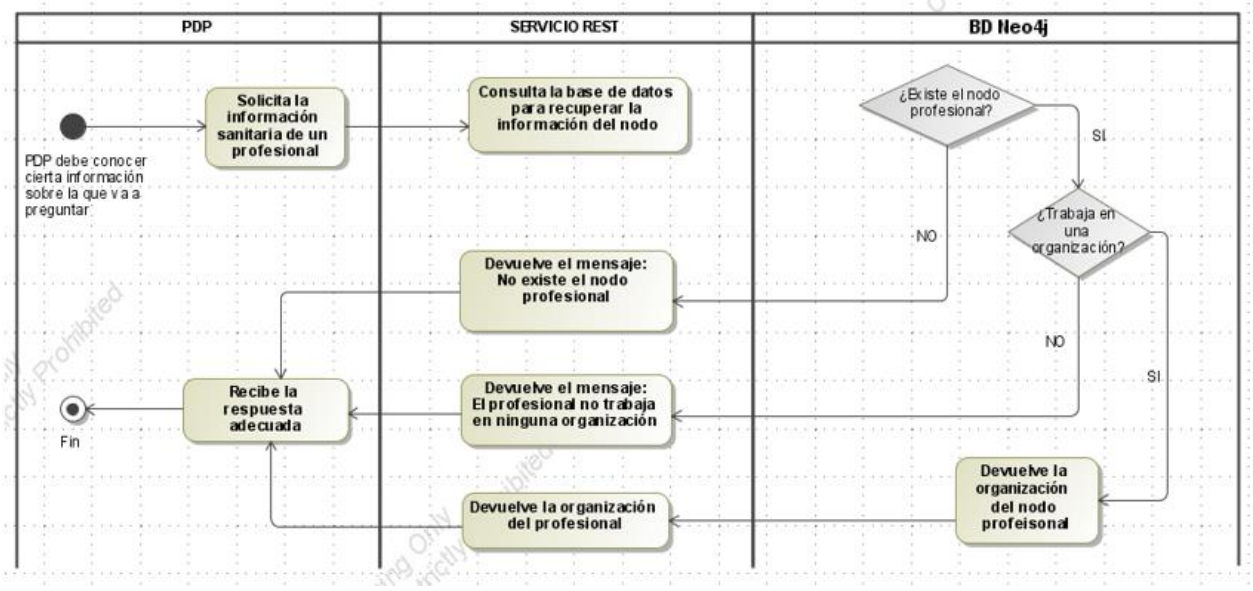


Figura 3-5 Diagrama de actividad CDU-04

3.1.5 CDU-05: Obtener información de un paciente

En este caso de uso se pretende obtener la información completa sobre un paciente.

Tabla 7 CDU-05: Obtener información de un paciente

CDU-05	Obtener información de un paciente	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener toda la información relacionada con un paciente. El sistema recupera dicha información y, si la hubiera, la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la información completa de un "Paciente".
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo, el sistema devuelve la información solicitada.
	2.2	Si no existe el nodo, el sistema devuelve un mensaje comunicando que no hay información sobre ese "Paciente".

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

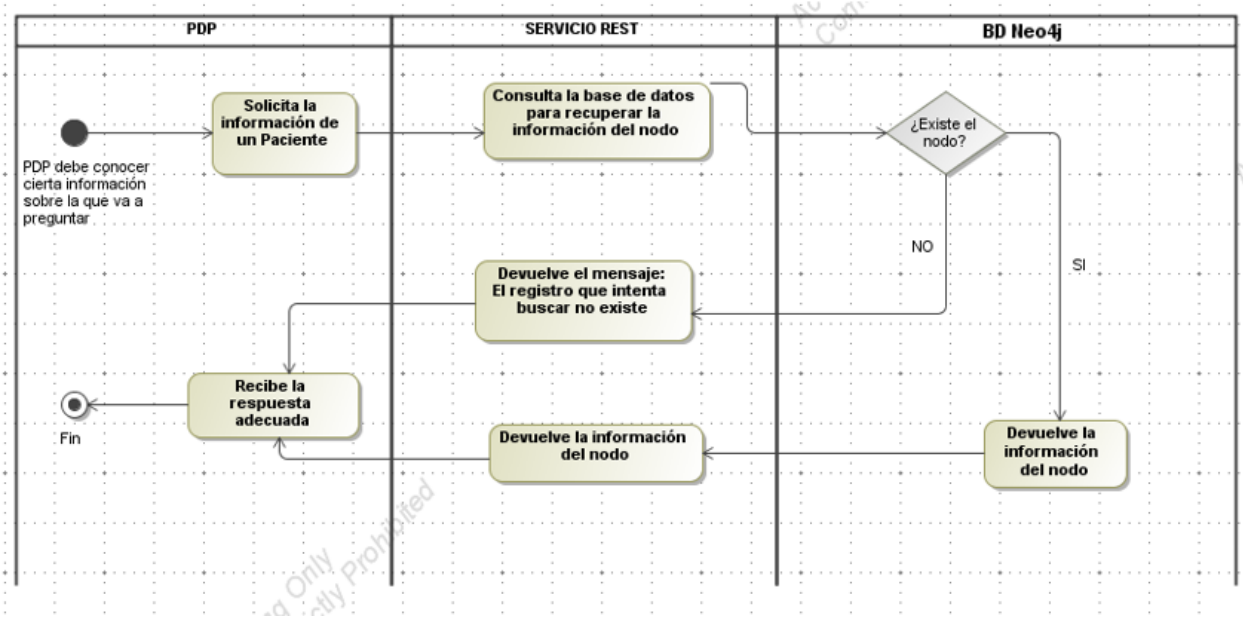


Figura 3-6 Diagrama de actividad CDU-05

3.1.6 CDU-06: Obtener la información sanitaria de un paciente

En este caso de uso se pretende obtener la información sanitaria que está asociada a un paciente.

Tabla 8 CDU-06: Obtener la información sanitaria de un paciente

CDU-06	Obtener la información sanitaria de un paciente	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener la información relacionada con un paciente. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la información sanitaria que está asociada a un paciente.
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo paciente, se comprueba si dicho nodo está asociado a una información sanitaria.
	2.1.1	Si existen nodos información, el sistema devuelve la información solicitada.
	2.1.2	Si no existen nodos información, el sistema devuelve un mensaje indicando que el paciente no tiene información sanitaria.
	2.2	Si no existe el nodo paciente, el sistema devuelve un mensaje comunicando que el paciente no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

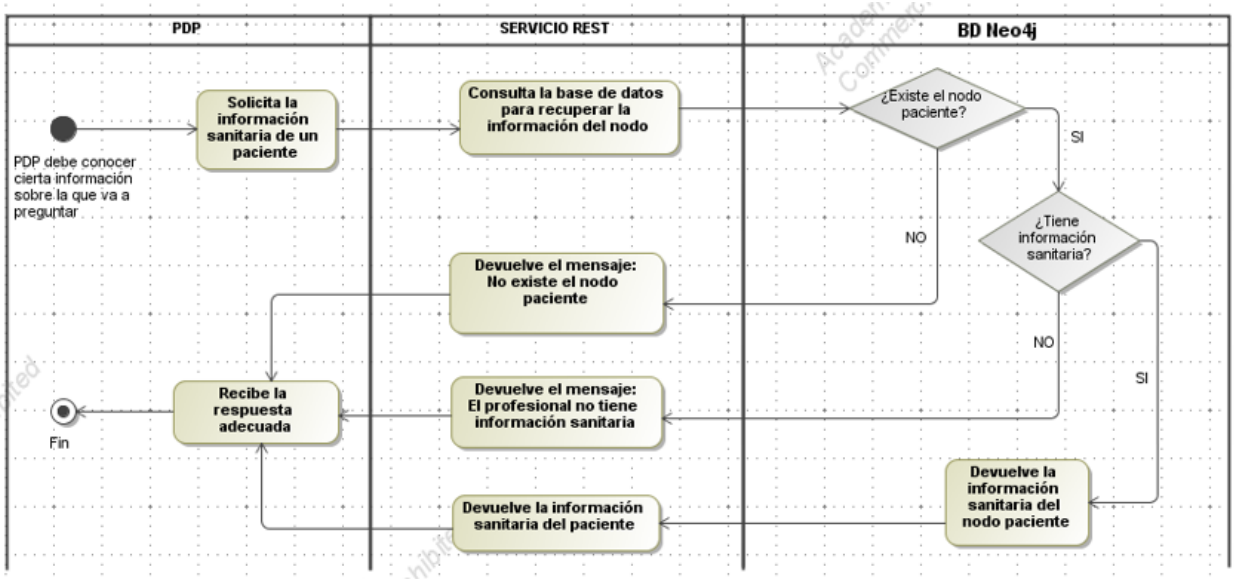


Figura 3-7 Diagrama de actividad CDU-06

3.1.7 CDU-07: Obtener el consentimiento de un paciente

En este caso de uso se pretende obtener el consentimiento que otorga un paciente.

Tabla 9 CDU-07: Obtener el consentimiento de un paciente

CDU-07	Obtener el consentimiento de un paciente	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener el consentimiento relacionado con un paciente. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la información sanitaria a la que está asociada un paciente.
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo paciente, se comprueba si dicho nodo está relacionado con uno o más nodos consentimiento.
	2.1.1	Si existe el nodo consentimiento, se verifica si se ha proporcionado el estado en la URL.
	2.1.1.a	Si se ha puesto estado, el sistema devuelve el nodo con ese estado.
	2.1.1.b	Si el estado no aparece en la URL, el sistema devuelve el nodo correspondiente.
	2.1.2	Si no existe el nodo consentimiento, el sistema devuelve un mensaje indicando que el paciente no otorga ningún consentimiento.
	2.2	Si no existe el nodo paciente, el sistema devuelve un mensaje comunicando que el paciente no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

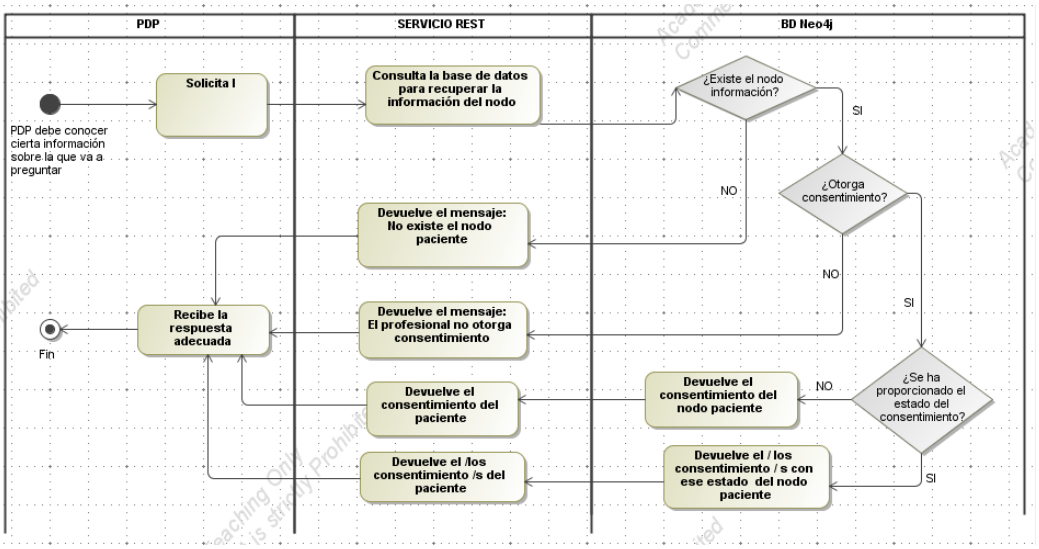


Figura 3-8 Diagrama de actividad CDU-07

3.1.8 CDU-08: Obtener información de un recurso Información

En este caso de uso se pretende obtener la información completa sobre un recurso información.

Tabla 10 CDU-08: Obtener información de un recurso información

CDU-08	Obtener información de un recurso información	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener toda la información relacionada con un recurso información. El sistema recupera dicha información y, si la hubiera, la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la información completa de un "recurso información".
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo, el sistema devuelve la información solicitada.
	2.2	Si no existe el nodo, el sistema devuelve un mensaje comunicando que no hay información sobre ese "recurso información".

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

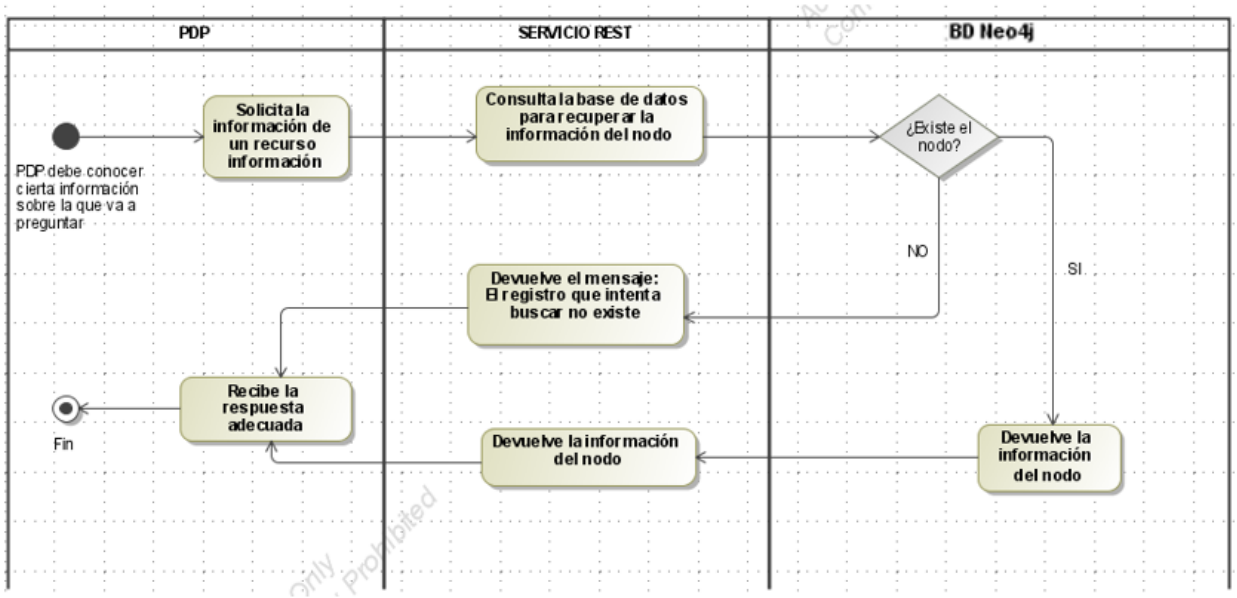


Figura 3-9 Diagrama de actividad CDU-08

3.1.9 CDU-09: Obtener la especialidad de un recurso información

En este caso de uso se pretende obtener la especialidad a la que pertenece un recurso información.

Tabla 11 CDU-09: Obtener la especialidad de un recurso información

CDU-09	Obtener la especialidad de un recurso información	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener la especialidad relacionada con un recurso información. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la especialidad a la que pertenece un recurso información.
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo información, se comprueba si dicho nodo pertenece a una especialidad.
	2.1.1	Si existe el nodo especialidad, el sistema devuelve la información solicitada.
	2.1.2	Si no existe el nodo especialidad, el sistema devuelve un mensaje indicando que el recurso información no pertenece a ninguna especialidad.
	2.2	Si no existe el nodo información, el sistema devuelve un mensaje comunicando que el recurso información no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

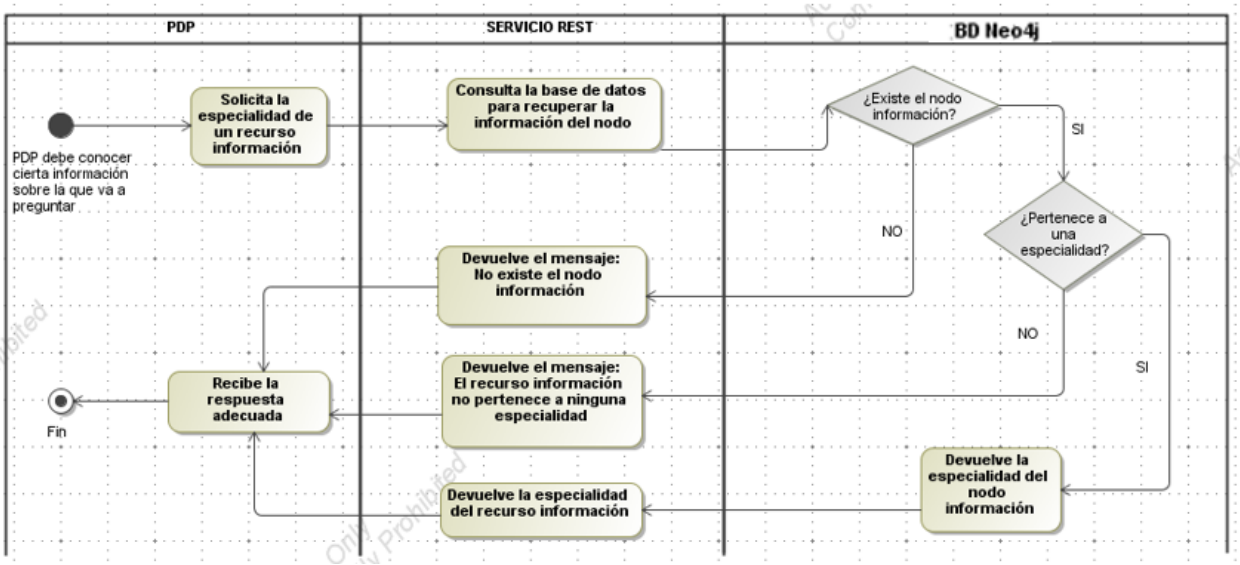


Figura 3-10 Diagrama de actividad CDU-09

3.1.10 CDU-10: Obtener la organización de un recurso información

En este caso de uso se pretende obtener la organización que custodia a un recurso información.

Tabla 12 CDU-10: Obtener la organización de un recurso información

CDU-10	Obtener la organización de un recurso información	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener la organización relacionada con un recurso información. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener la organización que custodia a un recurso información.
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo información, se comprueba si dicho nodo es custodiado por una organización.
	2.1.1	Si existe el nodo organización, el sistema devuelve la información solicitada.
	2.1.2	Si no existe el nodo organización, el sistema devuelve un mensaje indicando que el recurso información no está relacionado con una organización.
	2.2	Si no existe el nodo información, el sistema devuelve un mensaje comunicando que el recurso información no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

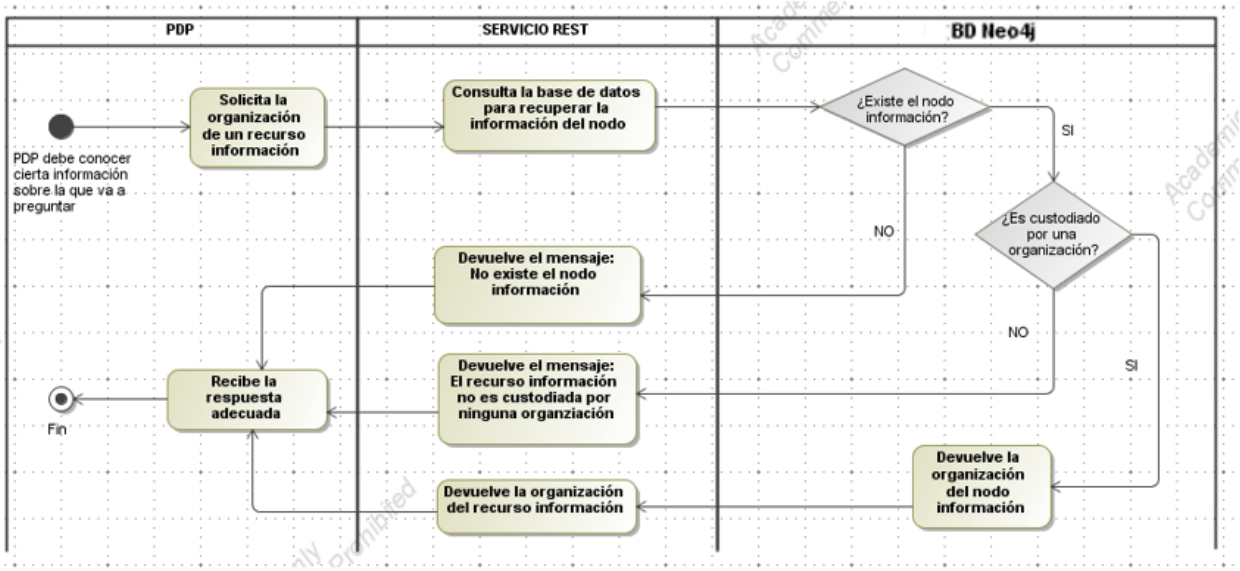


Figura 3-11 Diagrama de actividad CDU-10

3.1.11 CDU-11: Obtener el consentimiento de un recurso información

En este caso de uso se pretende obtener el consentimiento que tiene un recurso información.

Tabla 13 CDU-11: Obtener el consentimiento de un recurso información

CDU-11	Obtener el consentimiento de un recurso información	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener el consentimiento relacionado con un recurso información. El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener el consentimiento que tiene un recurso información.
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo información, se comprueba si dicho nodo tiene consentimiento.
	2.1.1	Si existe el nodo consentimiento, el sistema devuelve la información solicitada.
	2.1.2	Si no existe el nodo consentimiento, el sistema devuelve un mensaje indicando que el recurso información no está asociado a ningún consentimiento.
	2.2	Si no existe el nodo información, el sistema devuelve un mensaje comunicando que el recurso información no existe.

El diagrama de actividad correspondiente al caso de uso se representa a continuación.

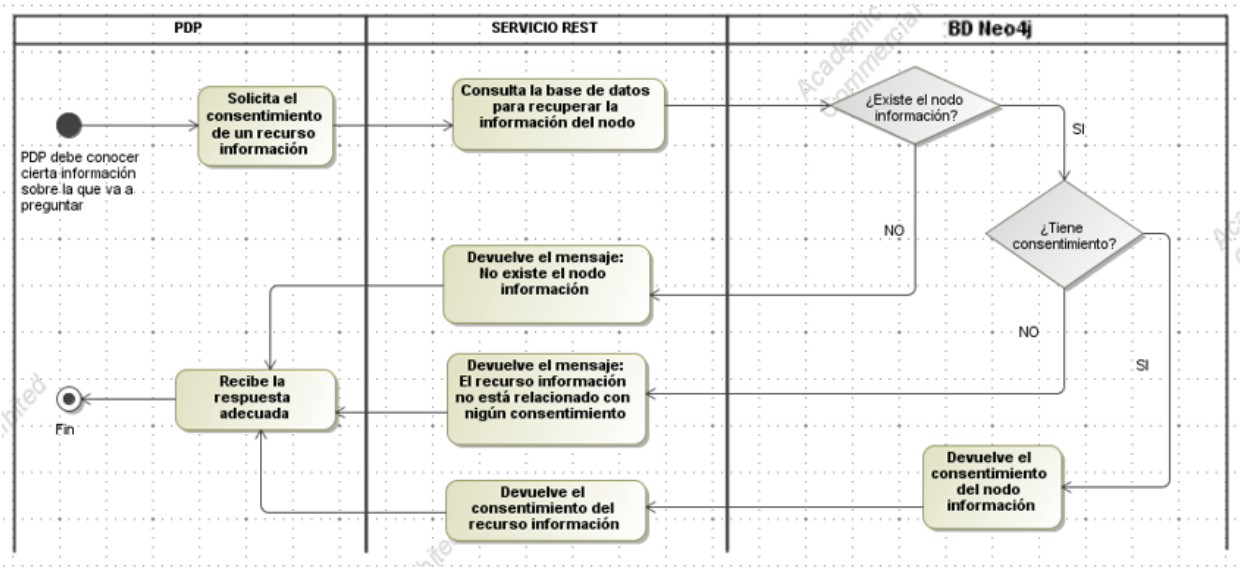


Figura 3-12 Diagrama de actividad CDU-11

3.1.12 CDU-12: Obtener un consentimiento

En este caso de uso se pretende obtener un consentimiento.

Tabla 14 CDU-12: Obtener un consentimiento

CDU-12	Obtener un consentimiento	
Precondición	El PDP debe conocer cierta información sobre la que va a preguntar	
Descripción	El PDP solicita obtener un consentimiento, El sistema recupera dicha información y la devuelve al PDP.	
Secuencia Normal	**Paso**	**Acción**
	1	El PDP realiza una solicitud para obtener un consentimiento.
	2	El sistema consulta la base de datos Neo4j para recuperar la información.
	2.1	Si existe el nodo consentimiento, el sistema devuelve la información solicitada.
	2.2	Si no existe el nodo consentimiento, el sistema devuelve un mensaje indicando que no existe consentimiento.

El diagrama de actividad correspondiente al caso de uso se representa a continuación:

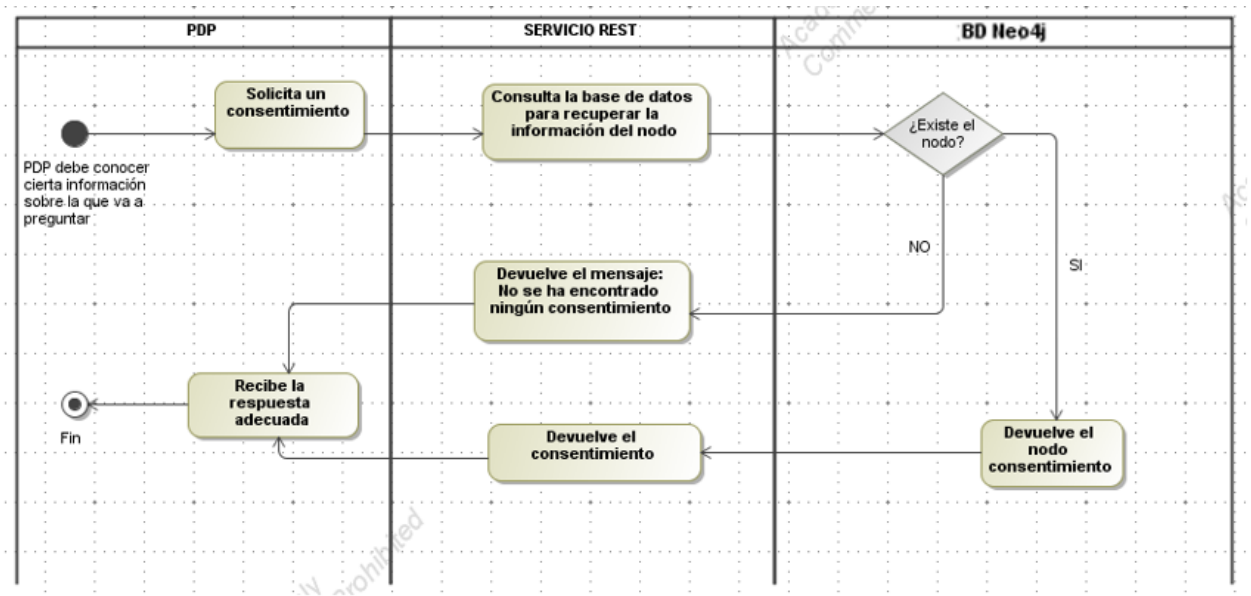


Figura 3-13 Diagrama de actividad CDU-12

Este caso de uso es el más importante en el proyecto, sin embargo, se reflejará más adelante un diagrama de flujo donde se explica detalladamente.

3.2 Base de datos Neo4j

En este proyecto, se ha creado una base de datos Neo4j para gestionar de manera eficiente los datos relacionados con el sistema de consentimiento y la política de acceso basada en atributos (ABAC). Los pasos seguidos para establecer y configurar la base de datos son los siguientes:

1. Descarga e instalación de Neo4j Desktop:

   - Se descarga e instala la aplicación Neo4j Desktop para el entorno de desarrollo. Esta herramienta facilita la gestión de bases de datos Neo4j de manera local.

2. Creación del proyecto y DBMS:

   - Se crea un nuevo proyecto llamado "TFG" dentro de Neo4j Desktop. A continuación, se añade un Sistema de Administración de Bases de Datos (DBMS) de nombre de usuario y contraseña "consentimiento".

3. Uso de la base de datos predeterminada:

   - Dentro del DBMS, la base de datos por defecto llamada neo4j se ha utilizado por simplicidad. En caso de querer usar una base de datos diferente, simplemente se crea una nueva base de datos con el nombre deseado y se configura el proyecto para usarla.

4. Inicialización y estructura de la base de datos Neo4j:

   - Después de iniciar la base de datos Neo4j, se procede a añadir la información que compone el grafo del sistema de consentimiento. La base de datos está formada por los siguientes nodos, cada uno con sus propiedades específicas y las relaciones que tienen con otros nodos. A continuación, se presenta una tabla de la estructura de la base de datos:

Tabla 15 Estructura de la base de datos

NODOS	PROPIEDADES	RELACIONES CON OTROS NODOS
Paciente	▪ ID ▪ Nombre ▪ Género	▪ OTORGA **Consentimiento**
Información	▪ ID ▪ Timestamp ▪ Tipo { Observación / Resultado / Medida } ▪ Valor ▪ Categoría	▪ ESTA_ASOCIADO **Paciente** ▪ PERTENECE **Especialidad**
Especialidad	▪ ID ▪ Nombre	
Organización	▪ ID ▪ Nombre	▪ CUSTODIA_A **Información**

Profesional	▪ ID ▪ Nombre	▪ TRABAJA_EN **Organización** ▪ TIENE **Rol** ▪ PERTENECE a **Especialidad**
Rol	▪ ID ▪ Nombre	
Consentimiento	▪ Estado { Borrador, Activo, Revocado, Anulado } ▪ Validez_desde ▪ Valez_hasta	▪ TIENE_RECURSO **Información** ▪ TIENE_ACCION **Acción** ▪ TIENE_PROPOSITO **Propósito** ▪ TIENE_SUJETO **Rol** ▪ TIENE_SUJETO **Profesional** ▪ PARA_ESPEC_PROF **Especialidad** ▪ PARA_ESPEC_INFO **Especialidad** ▪ PARA_ORG_PROF **Organización** ▪ PARA_ORG_INFO **Organización**
Propósito	▪ ID ▪ Nombre	
Acción	▪ ID ▪ Nombre	

Es importante destacar que la propiedad ID será la clave principal de cada nodo, es decir, el ID es único para cada uno de ellos.

Además, para comprender mejor algunos nodos y relaciones de la tabla, a continuación, se proporciona una descripción detallada:

- El nodo paciente puede otogar uno o varios consentimientos con el objetivo de que el paciente defina qué información suya puede ser utilizada, por quién y para qué.
- El nodo consentimiento tiene una serie de relaciones con el resto de nodos:
  i. TIENE_RECURSO Información: indica que el consentimiento otorga acceso a un recurso de tipo Información. Por ejemplo, el consentimiento puede conceder a un profesional el derecho a acceder a los registros médicos de un paciente.
  ii. PARA_ESPEC_PROF: indica que un consentimiento está relacionado con una especialidad específica del profesional. Por ejemplo, un consentimiento podría permitir el acceso a la información solo a cardiólogos, es decir, la especialidad de un profesional.
  iii. PARA_ESP_INFO: asocia un consentimiento con una especialidad específica de la información. Por ejemplo, un consentimiento podría estar limitado a información de la especialidad “Dermatología”.
  iv. PARA_ORG_PROF: el consentimiento está asociado con una organización específica a la que pertenece el profesional. Esto permite restringir el acceso a la información a profesionales de una organización específica, como un hospital.
  v. PARA_ORG_INFO: el consentimiento se relaciona con la organización que custodia o maneja la información. Por ejemplo, un consentimiento puede permitir el acceso a información que pertenece a una clínica particular.
  vi. TIENE_ACCION Acción: Especifica las acciones que están permitidas por el consentimiento, como "READ" (leer) y "WRITE" (escribir).
  vii. TIENE_PROPOSITO Propósito: Define el propósito para el cual se otorga el consentimiento, como "Asistencia sanitaria", "Investigación" y “Educación”.

A continuación, se muestran algunos ejemplos representativos de ciertos nodos y sus relaciones con el resto:

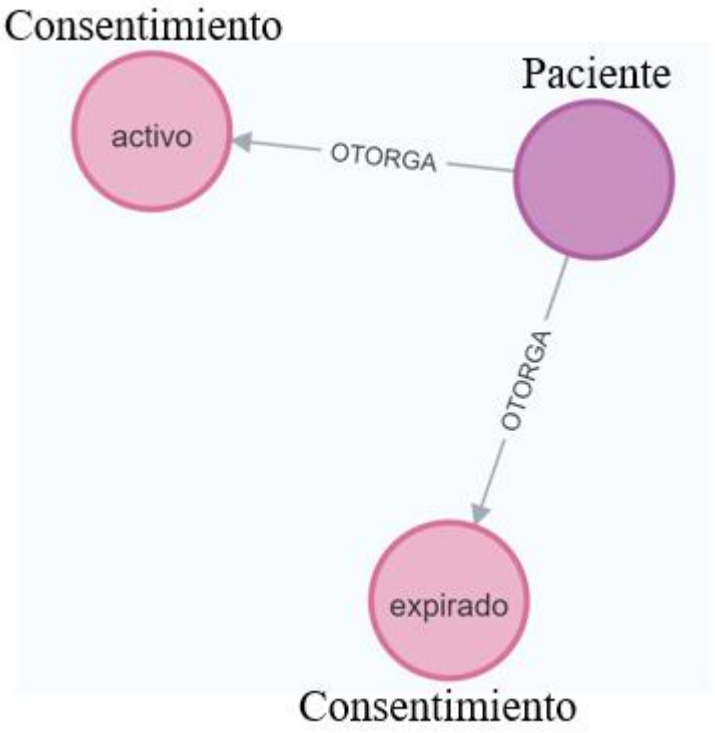


Figura 3-14 Ejemplo de un paciente que otorga dos consentimientos

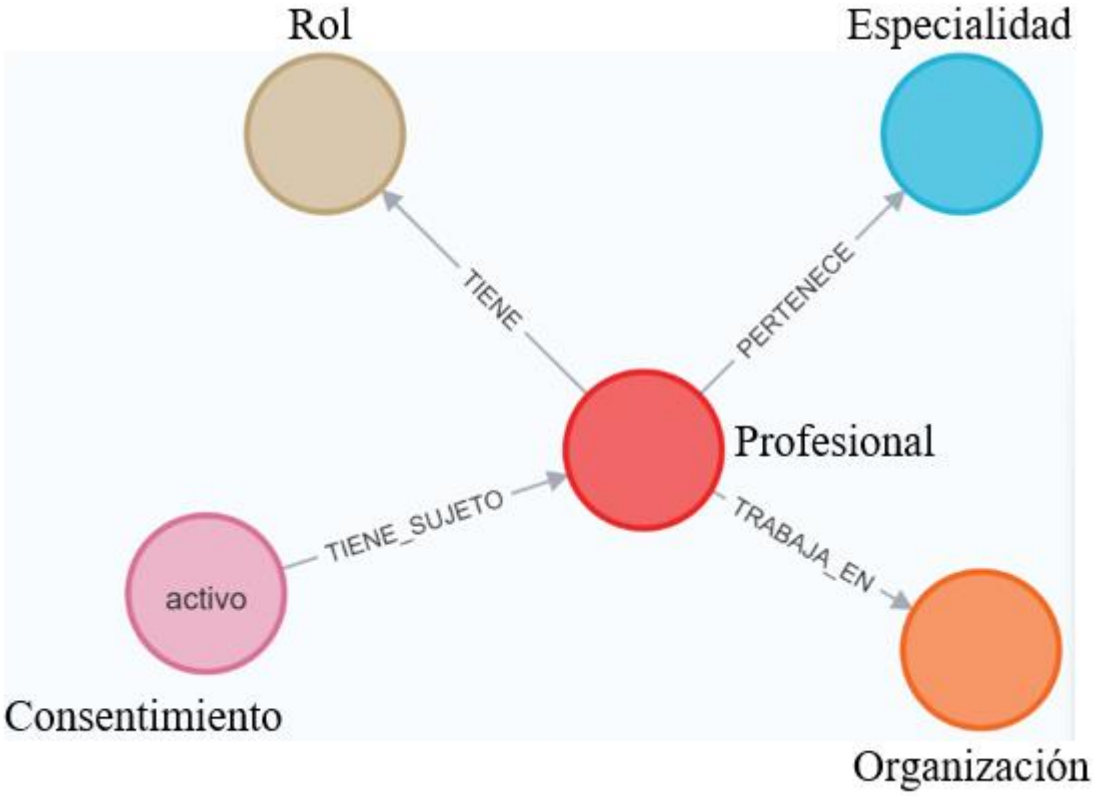


Figura 3-15 Ejemplo de un nodo profesional y sus relaciones

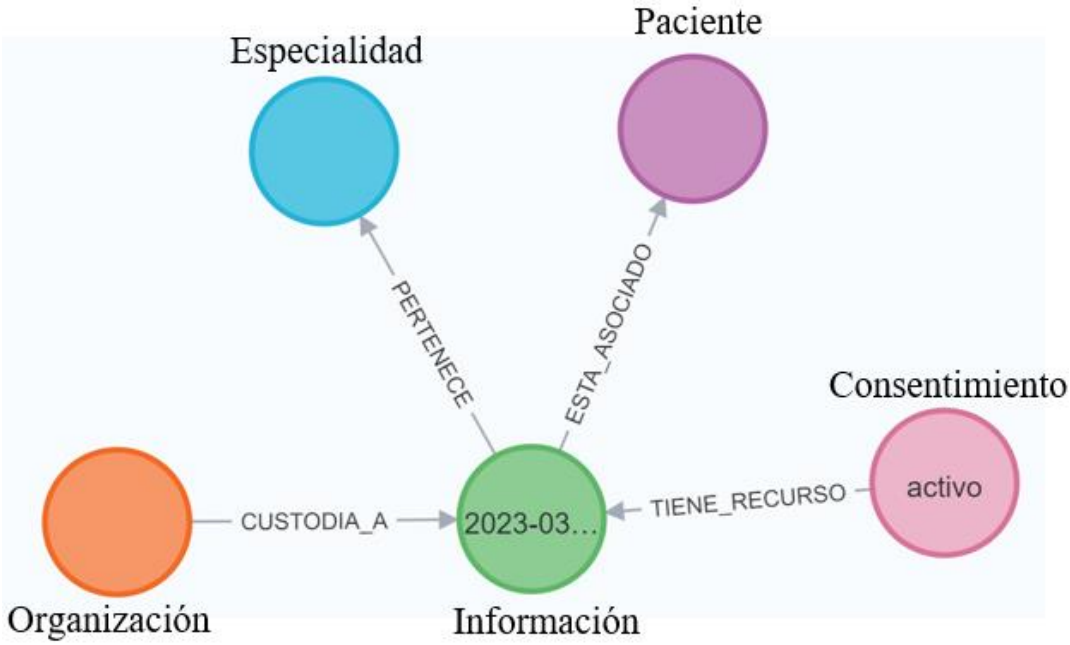


Figura 3-16 Ejemplo de un nodo información y sus relaciones

En la Figura 3-17 se puede observar un claro ejemplo de las relaciones del nodo consentimientos con el resto de nodos, concretamente, se comprueba la diferencia entre la relación PARA_ESPEC_INFO y PARA_ESPEC_PROF.

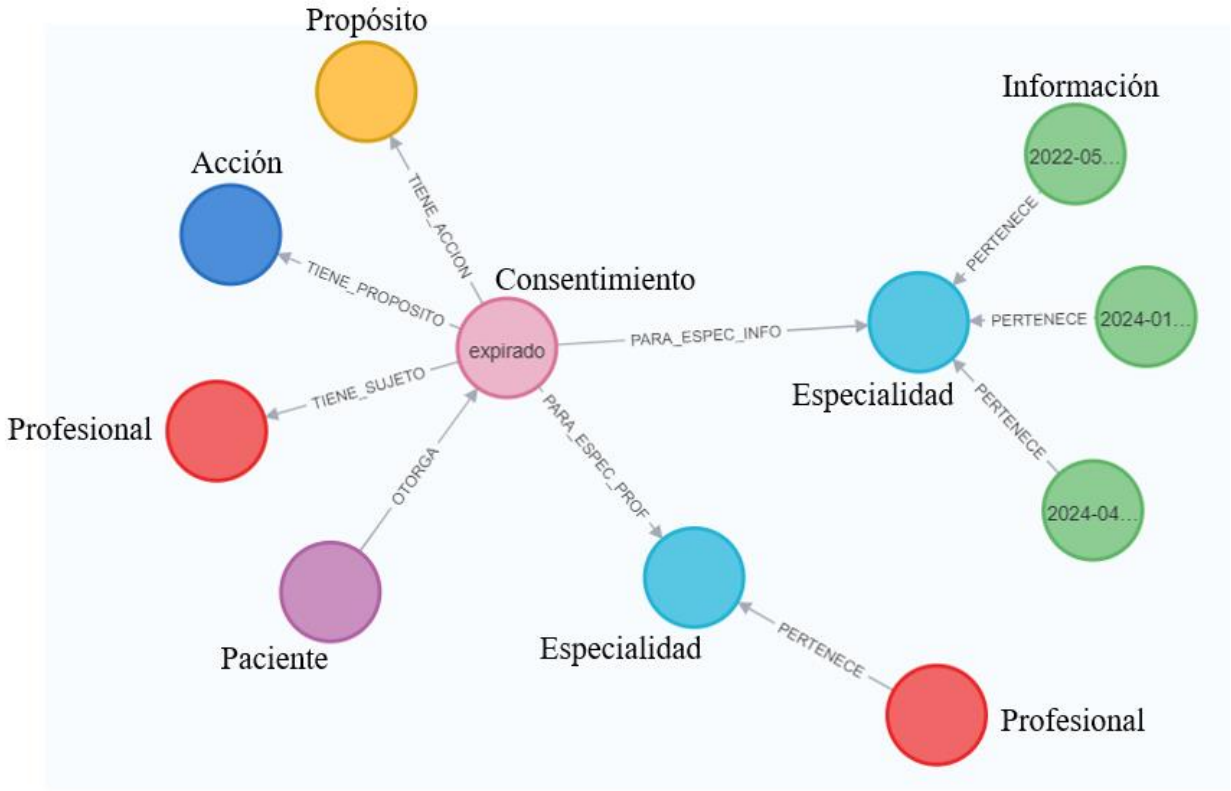


Figura 3-17 Ejemplo de un nodo consentimiento y sus relaciones

5. Carga masiva de datos con CSV:

   - Dada la cantidad de nodos y relaciones requeridas, se ha decidido crear un archivo CSV que contenga todos los datos del grafo. Este enfoque permite cargar datos de forma masiva, ahorrando tiempo y reduciendo errores manuales.

Para cargar los datos en Neo4j, se ha utilizado la cláusula LOAD CSV como se muestra en la próxima figura.

```
LOAD CSV WITH HEADERS
FROM 'file:///DatosNeo4JConsentimientos.csv' AS line
FIELDTERMINATOR ';'
RETURN line
```

Figura 3-18 Carga de datos

6. Creación de nodos:

   - Después de cargar el archivo, se crearon a partir de él los nodos necesarios con sus respectivas propiedades como se puede observar en la Figura 3-19. En dicha figura se crean todos los nodos Información necesarios en la base de datos usando MERGE y SET para agregar las propiedades. Además, se ha usado WHERE para asegurar que la propiedad ID no es nula.

```
LOAD CSV WITH HEADERS
FROM 'file:///DatosNeo4JConsentimientos.csv' AS line
FIELDTERMINATOR ';'
WITH line WHERE line.Info_ID IS NOT NULL
MERGE (i:Info {id: line.Info_ID})
SET   i.timestamp = line.Info_Timestamp, i.tipo = line.Info_Tipo,
i.valor=line.Info_Valor, i.categoria=line.Info_Categoria
```

Figura 3-19 Creación de los nodos Información

7. Creación de relaciones entre nodos:

   - Se han creado las relaciones necesarias entre los diferentes nodos como aparece en la Figura 3-20. En esta figura se recuperan los nodos Profesional y Especialidad con su clave principal utilizando MATCH y se crea la relación PERTENECE entre ellos con MERGE.

```
LOAD CSV WITH HEADERS
FROM 'file:///DatosNeo4JConsentimientos.csv' AS line
FIELDTERMINATOR ';'
MATCH ( p: Profesional {id:'30001'})
MATCH ( e:Especialidad {id:'40004'})
MERGE(p)-[:PERTENECE]→(e)
```

Figura 3-20 Creación de una relación entre dos ndodos

8. Filtrado de onsultas en la base de datos Neo4j:

   - Una vez creada la base de datos con todos sus nodos y relaciones, se implementaron consultas avanzadas para filtrar y obtener información específica. Estas consultas utilizan el lenguaje Cypher de Neo4j para seleccionar datos basados en atributos del modelo ABAC. Ejemplos de consultas se muestran en las Tablas (filtrado usando rangos) y 17 (filtrado por cadenas parciales).

Tabla 16 Ejemplos de consultas usando rangos

FILTRADO USANDO RANGOS
El filtrado por rangos es útil cuando se desea obtener datos dentro de un rango de fechas específicas. Este tipo de consulta es fundamental para encontrar los consentimientos que son válidos en un período específico. Por ejemplo, si necesitamos saber qué consentimientos están vigentes en el mes de enero de 2024, podemos hacer uso de este filtrado.

CONSULTA

```
MATCH (c:Consentimiento)
WHERE datetime(c.validez_desde) ⩾ datetime('2024-01-01T00:00:00')
AND datetime(c.validez_hasta) ⩽ datetime('2024-02-01T23:59:59')
RETURN c
```

RESULTADO

INFORMACIÓN DEL NODO

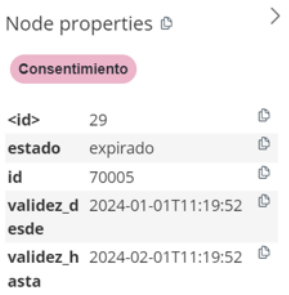


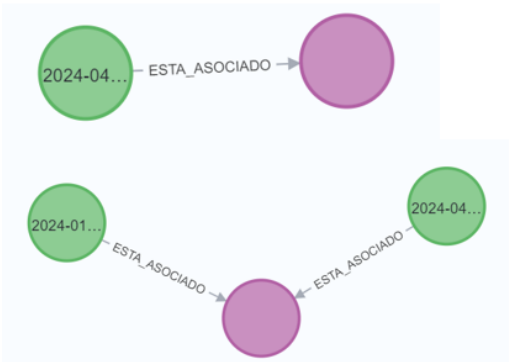
Tabla 17 Ejemplo de consulta usando filtro por cadenas parciales

FILTRADO POR CADENAS PARCIALES
El filtrado por cadenas parciales se puede utilizar en los casos en que se necesita buscar nodos que coincidan parcialmente con un criterio específico. En nuestro caso, es útil para encontrar nodos de tipo Informacion cuya propiedad Categoría comienza por una cadena dada, como "Adm". Este tipo de filtrado es útil cuando se necesita agrupar o identificar nodos con características similares sin conocer la cadena completa o exacta de búsqueda.

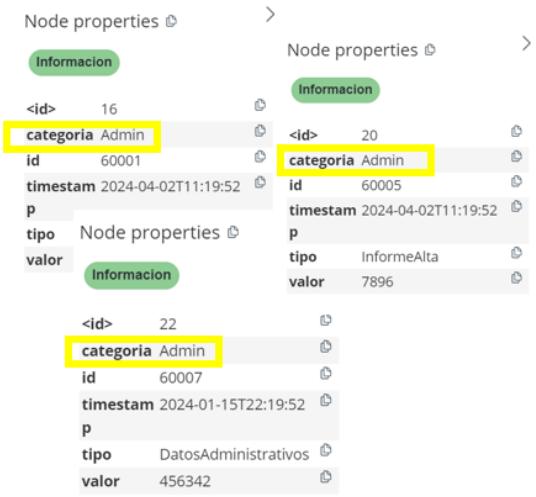
CONSULTA

```
MATCH (p:Paciente) <-[:ESTA_ASOCIADO]-(i:Informacion)
WHERE i.categoria STARTS WITH 'Adm'
RETURN p,i
```

RESULTADO



INFORMACIÓN DE LOS NODOS



A continuación, se muestran las consultas realizadas en el proyecto.

1. **Obtener información de un profesional:** La consulta devuelve toda la información del nodo profesional, esta misma, se puede usar también para obtener toda la información de un paciente o de un recurso de información. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestra en la Tabla 18.

Tabla 18 Consulta para obtener la información de un profesional

Sintaxis consulta Obtener información de un profesional

`MATCH (p:Profesional)`
`WHERE p.id = '30003'`
`RETURN p`

Resultado
`p` `(:Profesional {id: "30003",nombre: "Margarita Campos"})`

2. **Obtener el rol de un profesional:** En la consulta, si en lugar de usar rol, cambiamos la relación específica del nodo profesional con los nodos organización y especialidad, nos devolvería el nombre de la especialidad u organización del nodo que se defina. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 19.

Tabla 19 Consulta para obtener el rol de un profesional

Sintaxis consulta Obtener el rol de un profesional

`MATCH (p:Profesional)-[:TIENE]→(r:Rol)`
`WHERE p.id = '30001'`
`RETURN r.nombre`

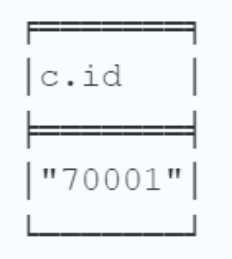
Resultado
`r.nombre` `"Jefe servicio"`

3. **Obtener el consentimiento de un recurso información:** La consulta devuelve el id del consentimiento que tiene un nodo información. Podemos hacer esto con cualquier otro nodo que esté relacionado un nodo información, por ejemplo, paciente, organización, etc. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 20.

Tabla 20 Consulta para obtener el consentimiento relacionado con un recurso información

Sintaxis consulta Obtener el consentimiento de un recurso información

`MATCH (c:Consentimiento)-[:TIENE_RECURSO]→(i:Informacion)`
`WHERE i.id = '60006'`
`RETURN c.id`

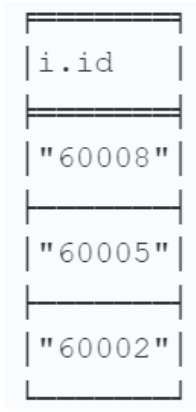
Resultado


4. **Obtener la información sanitaria de un paciente:** La consulta devuelve el id del recurso información de un paciente. Si se hace lo mismo pero con el consentimiento, se obtendrá el consentimiento (o consentimientos) que ha otorgado un paciente. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 21.

Tabla 21 Consulta para obtener la información sanitaria de un paciente

Sintaxis consulta Obtener la información sanitaria de un paciente

`MATCH (p:Paciente)←[:ESTA_ASOCIADO]-(i:Informacion)`
`WHERE p.id = '10002'`
`RETURN i.id`

La consulta devuelve el id del recurso información de un paciente. Si se hace lo mismo pero con el consentimiento, se obtendrá lo/s consentimiento/s que otorga un paciente.
Resultado


5. **Solicitar la confirmación de existencia de un consentimiento entre un profesional y un recurso información:** Este es el consentimiento más específico dado que está relacionado directamente con un recurso información y un profesiona, además, de tener una acción y un propósito. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 22.

Tabla 22 Consulta para conocer la existencia de un consentimiento

Sintaxis consulta ¿Existe consentimiento para un cierto profesional y recurso información?

`MATCH (p:Profesional)←[:TIENE_SUJETO]-(c:Consentimiento)-[:TIENE_RECURSO]→ (i:Informacion),(a:Accion)←[:TIENE_ACCION]-(c:Consentimiento)-[:TIENE_PROPOSITO]→ (pp:Proposito)`
`WHERE i.id = '60006' AND p.id = '30001'  AND a.nombre = 'WRITE' AND pp.nombre = 'Asistencia sanitaria'`
`RETURN c.id`

RESULTADO

6. **Solicitar información de existencia de un consentimiento entre un profesional con rol específico y un recurso información:** Este consentimiento no está relacionado con el nodo profesional directamente, sino con el rol de un cierto profesional. Se podría comprobar si existe consentimiento relacionado con la especialidad u organización de un profesional en lugar del rol. Por otro lado, también podría existir un consentimiento relacionado directamente con el profesional pero indirectamente con el nodo información, es decir, podría estar relacionado con la organización o especialidad de un recurso información. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 23.

Tabla 23 Consulta para conocer la existencia de un consentimeinto por rol del profesional

Sintaxis consulta ¿Existe consentimiento para un rol de un profesional y un recurso información?

`MATCH (p:Profesional)-[:TIENE]→(r:Rol)←[:TIENE_SUJETO]-(c:Consentimiento)-[:TIENE_RECURSO]→(i:Informacion),(a:Accion)←[:TIENE_ACCION]-(c:Consentimiento)-[:TIENE_PROPOSITO]→(pp:Proposito)`
`WHERE i.id = '60003' AND p.id = '30003'  AND a.nombre = 'READ' AND pp.nombre = 'Educación'`
`RETURN c.id`

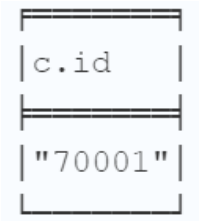
RESULTADO

7. **Solicitar información de existencia de un consentimiento entre un profesional con rol específico y un recurso información perteneciente a organización concreta:** Este consentimiento no está relacionado directamente ni con el profesional ni con el recurso información, está implícitamente relacionado. En este caso, esta relacionado con la organización de un nodo información y con el rol de un nodo profesional. Pueden existir otras consultas implícitas, por ejmplo, se puede variar la relación con la organización o especialidad del profesional dejando fija la relación con el nodo organización de un recurso información. Además, puede existir relación con la especialidad de la organización y con el rol, organización o especialiad de un profesional. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 24.

Tabla 24 Consulta para concocer la existencia de un consentimiento con rol del profesional

Sintaxis consulta ¿Existe consentimiento para un rol de un profesional y la organización de un recurso información?

`MATCH (i:Informacion)←[:CUSTODIA_A]-(o:Organizacion)←[:PARA_ORG_INFO]-(c:Consentimiento)-[:TIENE_SUJETO]→(r:Rol)←[:TIENE]-(p:Profesional),(a:Accion)←[:TIENE_ACCION]-(c:Consentimiento)-[:TIENE_PROPOSITO]→(pp:Proposito)`
`WHERE i.id = '60009' AND p.id = '30003'  AND a.nombre = 'WRITE' AND pp.nombre ='Asistencia sanitaria'`
`RETURN c.id`

RESULTADO


8. **Solicitar información de existencia de un consentimiento entre un profesional con organización específica y un recurso información perteneciente a organización concreta:** Este consentimiento está relacionado con la organización del profesional y con la organización del recurso información. Tanto un ejemplo de esta consulta codificado en Cypher como su resultado se muestran en la Tabla 24.

Tabla 25 Consulta para conocer la existencia de un consentimiento con organización del profesional y recurso información

Consulta 7: ¿Existe consentimiento para una organización de un profesional y la organización de un recurso información?

`MATCH (i:Informacion)←[:CUSTODIA_A]-(o:Organizacion)←[:PARA_ORG_INFO]-(c:Consentimiento)-[:PARA_ESP_PROF]→(e:Organizacion)←[:TRABAJA_EN]-(p:Profesional), (a:Accion)←[:TIENE_ACCION]-(c:Consentimiento)-[:TIENE_PROPOSITO]→(pp:Proposito)`
`WHERE i.id = '60004' AND p.id = '30002'  AND a.nombre = 'READ' AND pp.nombre ='Investigación'`
`RETURN c.id`

RESULTADO
Response ▾ []

3.3 Servicio REST

Para implementar la funcionalidad deseada en el proyecto, se ha decidido desarrollar una interfaz REST. Las principales ventajas de esta arquitectura son la sencillez de manipulación de datos y la gran escalabilidad que aporta. Las características de este servicio son:

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.
- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
- Los objetos en REST siempre se manipulan a partir de la URI. Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST.
- Uso de hipermedios para representar la información que suelen ser HTML, XML o JSON.

3.3.1 Esquema general

El servicio estará formado por las siguientes capas:

1) Modelo: representa las entidades del dominio del problema. La capa de modelo puede estar subdividida en varios componentes:
    - Entidades (Modelo): clases Java que representan los nodos en la base de datos Neo4j. Cada clase de dominio tiene el mismo nombre que el nodo y contiene las propiedades que definen el nodo en la base de datos.
    - DAO: se encarga de definir métodos que serán utilizados por la capa de servicio para realizar consultas específicas o para ejecutar operaciones sobre la base de datos Neo4j.
    - DTO: son objetos simples que se utilizan para transferir datos entre diferentes capas de la aplicación. Ayudan a encapsular los datos que se enviarán o recibirán a través de las solicitudes HTTP.
    - Payload: se utilizan para definir las respuestas que se devuelven al cliente desde la API. En el proyecto se tiene una clase de payload para enviar respuestas con un mensaje y un objeto.
2) Servicio: maneja la lógica para consultar la base de datos y recuperar los datos y proporcionarlos al controlador. En el proyecto, se requiere información del nodo profesional, paciente, información y consentimiento, esos son los distintos servicios.
3) Controlador: maneja las solicitudes HTTP y llama a los servicios correspondientes. Se ha creado un controlador de cada clase servicio.

Además, se tiene la clase Application.java que se usa para arrancar el servicio y el fichero de configuración *application.properties* donde se configura la base de datos y el puerto de ejecución, como se representa en la Figura 3-21. Como se mencionó en el apartado anterior, por simplicidad se ha usado la base de datos por defecto llamada neo4j pero si se desea usar otra solo hay que descomentar la línea *spring.data.neo4j.database=nombre_base_de_datos*.

*application.properties

```
1 server.port = 8092
2
3 # Configuración de conexión a la base de datos Neo4j
4 #spring.data.neo4j.database=bbdd-tfg
5 spring.neo4j.uri=bolt://localhost:7687
6 spring.neo4j.authentication.username=neo4j
7 spring.neo4j.authentication.password=consentimiento
8
```

Figura 3-21 Fichero de configuración application.properties

Así, un esquema general del funcionamiento del servicio sería el siguiente.

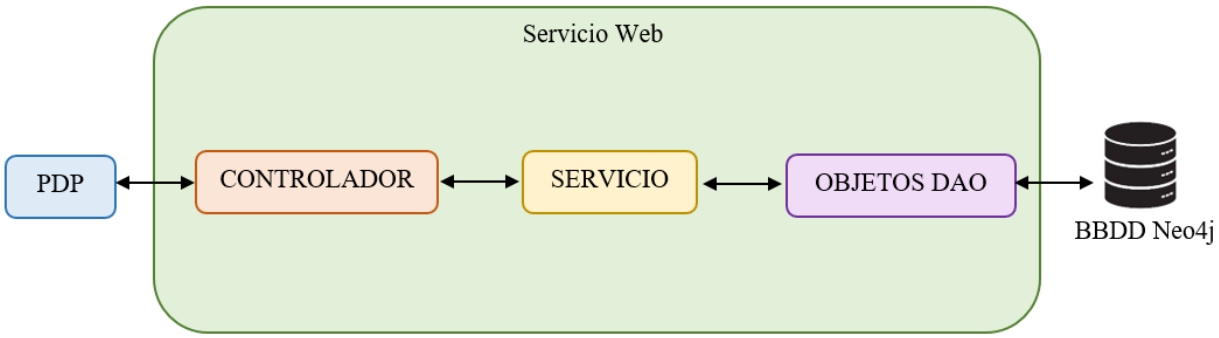


Figura 3-22 Esquema general del servicio REST

3.3.2 Profesional.java

Esta clase se ha utilizado para definer el recurso Profesional. Este está formado por los atributos id y nombre, además de sus correspondientes métodos Get y Set para obtener y establecer valores de los atributos y su constructor propio. Se han usado anotaciones de Spring Data Neo4j:

- @Node(): para indicar que esta clase es candidate para ser mapeada a la base de datos.
- @Id: para marcar el campo utilizado como identidad.
- @GeneratedValue: para especificar cómo debe generarse el identificador único.

```
@Node("Profesional")
public class Profesional {

    @Id @GeneratedValue(UUIDStringGenerator.class)
    private String id;
    private String nombre;


    public Profesional(String id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public String getId() {
        return id;
    }


    public void setId(String id) {
        this.id = id;
    }
```

Figura 3-23 Clase Profesional.java

Todos los demás modelos: Información.java, Propósito.java, Acción.java, Consentimiento.java, Especialidad.java, Organización.java, Paciente.java, Rol.java siguen la misma estructura que la figura Profesional.java pero cada uno con sus atributos correspondientes como se reflejó en las propiedades de los nodos de la base de datos Neo4j.

3.3.3 ProfesionalController.java

Esta clase actúa como controlador del servicio y es la encargada de procesar las peticiones relacionadas con el recurso profesional recibidas por el servicio. Esta cuenta con instancias al servicio IProfesional. La anotación *@RequestMapping("/api/profesionales")* define el punto de entrada base para todas las solicitudes relacionadas con los recursos de "profesionales".

```
@RestController
@RequestMapping("/api/profesionales")
public class ProfesionalController {

    @Autowired
    private IProfesional profesionalService;
```

Figura 3-24 Clase ProfesionalController.java y objeto IProfesional instanciado.

3.3.3.1 Obtener toda la información de un profesional

Cuando el servicio recibe una petición GET al punto de entrada base con los parámetros id en la URL, el controlador invoca al método showById.

```
@GetMapping("{id}")
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<?> showById(@PathVariable String id) {
    Profesional profesional = profesionalService.findById(id);
    if (profesional == null) {
        return new ResponseEntity<>(
                MensajeResponse.builder()
                .mensaje("El registro que intenta buscar no existe")
                .object(null)
                .build()
            , HttpStatus.INTERNAL_SERVER_ERROR);
    } else {
        return new ResponseEntity<>(
                MensajeResponse.builder()
                .mensaje("")
                .object(ProfesionalDto.builder()
                    .id(profesional.getId())
                    .nombre(profesional.getNombre())
                    .build())
                .build()
            , HttpStatus.OK);
    }
}
```

Figura 3-25 Método showById al recibir una petición GET

Tabla 26 Método showById de ProfesionalController.java

Método	**showById**
Descripción	Método que busca si existe un profesional con el id proporcionado en la URL llamando al método findById del servicio IProfesional.
Parámetros de entrada	Se usa la anotación @PathVariable para que el valor de id en la URL se vincule con el parámetro id del método.
Salida	Devuelve un ResponseEntity, si el profesional es igual a null se construye un objeto MensajeResponse con un mensaje y objeto concreto. Si el profesional no es null, devuelve un objeto MensajeResponse con los detalles del nodo Profesional, en este caso, se hace uso de la clase ProfesionalDto.java para devolver el objeto Profesional estructurado con varios atributos.

3.3.3.2 Obtener el rol de un professional

Cuando el servicio recibe una petición GET a /id/roles con el id del professional en la URL, el controlador invoca al método obtenerRolNombre.

```
@GetMapping("/{profesionalId}/roles")
public ResponseEntity<?> obtenerRolNombre(@PathVariable String profesionalId) {
    Profesional profesional = profesionalService.findById(profesionalId);
    if (profesional == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No existe el nodo profesional.");
    }else {

        List<String> rolNombre = profesionalService.findRolByProfesionalId(profesionalId);
        if (!rolNombre.isEmpty()) {
            return ResponseEntity.ok(rolNombre);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("El profesional no tiene ningún rol.");
        }
    }
}
```

Figura 3-26 Método obtenerRolNombre al recibir una petición GET

Tabla 27 Método obtenerRolNombre de ProfesionalController.java

Método	**obtenerRolNombre**
Descripción	Método que busca si existe el rol de un profesional con el id proporcionado en la URL. Primero llama al método findById del servicio IProfesional para comprobar que el id del profesional existe y, después, invoca al método findRolByProfesionalId.
Parámetros de entrada	Se usa la anotación @PathVariable para que el valor de id en la URL se vincule con el parámetro id del método.
Salida	Devuelve un ResponseEntity, si el profesional es igual a null devuelve “No existe el nodo profesional”. Si el id es válido y tiene un rol devuelve el nombre del rol; si no devuelve “El profesional no tiene nigún rol”. Este caso no se hace uso del DTO, porque la operación es mucho más sencilla por lo que es más rápido de volver los datos directamente haciendo uso de métodos de ResponseEntity.

En los casos de uso obtener una especialidad de un profesional (CDU-02) y obtener la organización de un profesional (CDU-04) se sigue la misma lógica que al obtener el rol, pero cambiando los métodos y lo que devuelven.

3.3.4 ProfesionalImpl.java

Esta clase actúa como servicio y es la encargada de recuperar los datos y proporcionarlos al controlador.

Utiliza un objeto profesionalDao para interactuar con la base de datos, haciendo uso de los métodos del objeto correspondiente a cada consulta de cada caso de uso.

```
@Service
public class ProfesionalImpl implements IProfesional{

    @Autowired
    private ProfesionalDao profesionalDao;


    @Override
    public Profesional findById(String id) {
        // TODO Auto-generated method stub
        return profesionalDao.findById(id).orElse(null); //Si no existe, devuelve null
    }

    public List<String> findRolByProfesionalId(String profesionalId) {
        return profesionalDao.findRolByProfesionalId(profesionalId);
    }
```

Figura 3-27 Clase ProfesionalImpl.java y objeto ProfesionalDao instanciado.

3.3.5 ProfesionalDAO.java

Esta clase se usa para acceder a la BD Neo4j y obtener todo el rescurso información, el rol, la organización o la especialidad de un profesional. Se extiende el repositorio Neo4jRepository, en este viene un método implementado que busca un nodo por su id. También, se ha usado la anotación @Query para crear la consulta que deseemos.

```
public interface ProfesionalDao extends Neo4jRepository<Profesional,String>{
    @Query("MATCH (p:Profesional)-[:TIENE]->(r:Rol) WHERE p.id = $profesionalId RETURN r.nombre")
    List<String> findRolByProfesionalId(String profesionalId);

    @Query("MATCH (p:Profesional)-[:TRABAJA_EN]->(o:Organizacion) WHERE p.id = $profesionalId RETURN o.nombre")
    List<String> findOrgByProfesionalId(String profesionalId);

    @Query("MATCH (p:Profesional)-[:PERTENECE]->(e:Especialidad) WHERE p.id = $profesionalId RETURN e.nombre")
    List<String> findEspByProfesionalId(String profesionalId);

}
```

Figura 3-28 Clase ProfesionalDAO.java

Tabla 28 Método findRolByProfesionalId de PorfesionalDAO.java

Método	findRolByProfesionalId
Descripción	Método que accede a la BD Neo4j y obtiene el rol del profesional correspondiente.
Parámetros de entrada	String profesionalId: id del profesional
Salida	El método devolverá una lista de String con los nombres del rol o roles si el profesional tiene un rol, si no devuelve una lista vacía.

Los demás métodos de la clase DAO son idénticos que el método findRolByProfesionalId solo cambia la query y el nombre del método.

3.3.6 ProfesionalDto.java

Esta clase actúa como un objeto de transferencia de datos (DTO) para encapsular la información relevante sobre un nodo de profesional. Para simplificar la construcción, generación de métodos, y hacer el código más limpio y conciso, se utiliza la librería **Lombok**, que permite evitar la escritura repetitiva de código.

```
import lombok.Builder;

@Data
@ToString
@Builder
public class ProfesionalDto {
    private String id, nombre;
}
```

Figura 3-29 Clase ProfesionalDto.java

Se han usado las siguientes anotaciones de Lombok:

- @Data: genera automáticamente los métodos getter, setter, equals(), hashCode(), y toString() para los campos id y nombre.
- @ToString: Genera el método toString() que devuelve una representación en forma de cadena de todos los campos de la clase.
- @Builder: permite construir instancias de la clase ConsentimientoDto de manera flexible y segura mediante el patrón Builder. Esto es especialmente útil para manejar objetos con muchos parámetros opcionales o cuando se desea una construcción más legible y fluida.

3.3.7 MensajeResponse.java

Es una clase simple para representar una respuesta diseñada. Esta clase implementa la interfaz Serializable para permitir que sus instancias puedan ser convertidas en un flujo de bytes, facilitando su transmisión o almacenamiento. Se utiliza cuando se devuelve toda la información de un cierto nodo como se comentó anteriormente. Se compone de un mensaje y un objeto que será el nodo devuelto o cuando no existe nodo estará vacío.

```
@Data
@ToString
@Builder
public class MensajeResponse implements Serializable{
    private static final long serialVersionUID = 1L;

    private String mensaje;
    private Object object;
}
```

Figura 3-30 Clase MensajeResponse.java

3.3.8 Paciente.java

Esta clase se ha utilizado para definir el recurso Paciente. Como se mencionó anteriormente sigue la misma lógica que el modelo Profesional.java.

```
@Node("Paciente")
public class Paciente {

    @Id @GeneratedValue(UUIDStringGenerator.class)
    private String id;
    private String nombre, genero;

    public Paciente(String id, String nombre, String genero) {
        this.id = id;
        this.nombre = nombre;
        this.genero = genero;
    }

    public String getId() {
        return id;
    }


    public void setId(String id) {
        this.id = id;
    }
```

Figura 3-31 Clase Paciente.java

3.3.9 PacienteController.java

Esta clase actúa como controlador del servicio y es la encargada de procesar las peticiones relacionadas con el recurso paciente recibidas por el servicio. Esta cuenta con instancias al servicio IPaciente. La anotación *@RequestMapping("/api/pacientes")* define el punto de entrada base para todas las solicitudes relacionadas con los recursos de "pacientes".

```
@RestController
@RequestMapping("/api/pacientes")
public class PacienteController {
    @Autowired
    private IPaciente pacienteService;
```

Figura 3-32 Clase PacienteController.java y objeto IPaciente instanciado

3.3.9.1 Obtener toda la información de un paciente

Cuando el servicio recibe una petición GET al punto de entrada base con los parámetros id en la URL, el controlador invoca al método showById.

```
@GetMapping("{id}")
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<?> showById(@PathVariable String id) {
    Paciente paciente = pacienteService.findById(id);
    if (paciente == null) {
        return new ResponseEntity<>(
                MensajeResponse.builder()
                .mensaje("El registro que intenta buscar no existe")
                .object(null)
                .build()
            , HttpStatus.INTERNAL_SERVER_ERROR);
    } else {
        return new ResponseEntity<>(
                MensajeResponse.builder()
                .mensaje("")
                .object(PacienteDto.builder()
                    .id(paciente.getId())
                    .nombre(paciente.getNombre())
                    .genero(paciente.getGenero())
                    .build())
                .build()
            , HttpStatus.OK);
    }
}
```

Figura 3-33 Método showById al recibir una petición GET

Tabla 29 Método showById de PacienteController.java

Método	**showById**
Descripción	Método que busca si existe un paciente con el id proporcionado en la URL llamando al método findById del servicio IPaciente.
Parámetros de entrada	Se usa la anotación @PathVariable para que el valor de id en la URL se vincule con el parámetro id del método.
Salida	Devuelve un ResponseEntity. Si el paciente es igual a null se construye un objeto MensajeResponse con un mensaje y objeto vacío. Si el paciente no es null, devuelve un objeto MensajeResponse con los detalles del nodo Paciente, en este caso, se hace uso de la clase PacienteDto.java para devolver el objeto Paciente más estructurado con varios atributos.

3.3.9.2 Obtener la información sanitaria de un paciente

Cuando el servicio recibe una petición GET a /id/informaciones con el id del paciente en la URL, el controlador invoca al método obtenerInfo.

Tabla 30 Método obtenerInfo de PacienteController.java

Método	**obtenerInfo**
Descripción	Método que busca si existe información sanitaria de un paciente con el id proporcionado en la URL. Primero llama al método findById del servicio IPaciente para comprobar que el id del paciente existe y, después, invoca al método findInfoByPaicenteId.
Parámetros de entrada	Se usa la anotación @PathVariable para que el valor de id en la URL se vincule con el parámetro id del método.
Salida	Devuelve un ResponseEntity. Si el paciente es igual a null devuelve "No existe el nodo paciente". Si el id es válido y tiene un recurso información devuelve el id del recurso información si no devuelve "No existe información sanitaria.". Este caso no se hace uso del DTO, como se comentó más atrás.

Este método está implementado de igual forma que obtenerRolNombre de ProfesionalController.java.

3.3.9.3 Obtener el consentimiento de un paciente

Cuando el servicio recibe una petición GET a /id/consentimientos con el id del paciente en la URL, el controlador invoca al método obtenerCons.

```
@GetMapping("{id}/consentimientos")
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<?> obtenerCons(@PathVariable String id, @RequestParam(value="estado", required=false) String estado) {
    Paciente paciente = pacienteService.findById(id);
    if (paciente ==null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No existe el nodo Paciente.");
    } else {
    List<String> list;
     if (estado != null) {
          list = pacienteService.findConsEstadoByPacienteId(id, estado);
        } else {
            list = pacienteService.findConsByPacienteId(id);
        }

        if (!list.isEmpty()) {
            return ResponseEntity.ok(list);
        } else {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body("El paciente no otorga consnetimiento.");
        }
```

Figura 3-34 Método obtenerCons al recibir una petición GET

Tabla 31 Método obtenerCons de PacienteController.java

Método	**obtenerCons**
Descripción	Método que busca si existe consentimiento de un paciente con el id proporcionado en la URL. Primero llama al método findById del servicio IPaciente para comprobar que el id del paciente existe y, después, dependiendo del estado del consentimiento, invoca un método u otro. Si se indica un estado, por ejemplo activo, llama al método findConsEstadoByPacienteId, si no se indica estado invoca el método findConsByPacienteId.
Parámetros de entrada	String id: id del paciente Se usa la anotación @RequestParam para indicar un parámetro de consulta en la URL, es decir, después de la URL se indica el estado con ?estado=activo, required=false indica que no es un parámetro obligatorio. Si no se indica se toma como null.
Salida	Devuelve un ResponseEntity. Si el paciente es igual a null devuelve "No existe el nodo paciente". Si el id es válido y tiene un consentimiento devuelve el id del consentimiento si no devuelve "El paciente no otorga consentimiento.". De este modo, si en el estado se indica "activo" devuelve los consentimientos activos, si no se indica nada devuelve todos los consentimientos y si se indica cualquier otro estado devuelve los consentimientos con dicho estado.

3.3.10 PacienteImpl.java

Esta clase actúa como servicio y es la encargada de recuperar los datos y proporcionarlos al controlador.

Utiliza un objeto pacienteDao para interactuar con la base de datos, haciendo uso de los métodos del objeto correspondiente a cada consulta de cada caso de uso.

3.3.11 PacienteDAO.java

Esta clase se usa para acceder a la BD Neo4j y obtener todo el paciente, la información sanitaria y los consentimientos de un paciente. Se implementa como ProfesionalDAO.java cambiando las consultas y el nombre de los métodos correspondientes a los casos de uso de un paciente.

3.3.12 PacienteDto.java

Esta clase actúa como un objeto de transferencia de datos (DTO) para encapsular la información relevante sobre un nodo de paciente. Sigue la misma implementación que la clase ProfesionalDto.java pero con sus campos correspondientes.

3.3.13 Información.java

Esta clase se ha utilizado para definer el recurso Información. Como se mencionó anteriormente sigue la misma lógica que el modelo Profesional.java.

3.3.14 InformaciónController.java

Esta clase actúa como controlador del servicio y es la encargada de procesar las peticiones relacionadas con el recurso Información recibidas por el servicio. Esta cuenta con instancias al servicio IInformacion. La anotación *@RequestMapping("/api/informaciones")* define el punto de entrada base para todas las solicitudes relacionadas con los recursos informaciones.

```
@RestController
@RequestMapping("/api/informaciones")
public class InformacionController {
    @Autowired
    private IInformacion infoService;
```

Figura 3-35 Clase InformacionController.java

3.3.14.1 Obtener toda la información de un recurso información

Cuando el servicio recibe una petición GET al punto de entrada base con los parámetros id en la URL, el controlador invoca al método showById.

Tabla 32 Método showById de InformacionController.java

Método	**showById**
Descripción	Método que busca si existe un recurso información con el id proporcionado en la URL llamando al método findById del servicio IInformacion.
Parámetros de entrada	Se usa la anotación @PathVariable para que el valor de id en la URL se vincule con el parámetro id del método.
Salida	Devuelve un ResponseEntity. Si la variable info es igual a null se construye un objeto MensajeResponse con un mensaje y objeto vacío. En caso contario, devuelve un objeto MensajeResponse con los detalles del nodo información, en este caso, se hace uso de la clase InformaciónDto.java para devolver el objeto Información más estructurado con varios atributos.

3.3.14.2 Obtener la especialidad de un recurso información

Cuando el servicio recibe una petición GET a /id/especialidades con el id del recurso información en la URL, el controlador invoca al método obtenerEspNombre. La lógica es la misma que cuando se quiere obtener el rol de un professional.

Tabla 33 Método obtenerEspNombre de InformacionController.java

Método	**obtenerEspNombre**
Descripción	Método que busca si existe la especialidad de un recurso información con el id proporcionado en la URL. Primero llama al método findById del servicio IInformacion para comprobar que el id del recurso información existe y, después, invoca al método findEspByInfoId.
Parámetros de entrada	Se usa la anotación @PathVariable para que el valor de id en la URL se vincule con el parámetro id del método.
Salida	Devuelve un ResponseEntity. Si el recurso información es igual a null devuelve "No existe el nodo información". Si el id es válido y tiene una especialidad devuelve el nombre de la misma si no devuelve "La información no pertenece a ninguna especialidad". Este caso no se hace uso del DTO.

Los métodos obtenerOrgNombre, obtenerConsId, están estructurados de la misma forma que obtenerEspNombre.

3.3.15 InformaciónImpl.java

Esta clase actúa como servicio y es la encargada de recuperar los datos y proporcionarlos al controlador.

Utiliza un objeto informaciónDao para interactuar con la base de datos, haciendo uso de los métodos del objeto correspondiente a cada consulta de cada caso de uso.

3.3.16 InformaciónDAO.java

Esta clase se usa para acceder a la BD Neo4j. Se implementa como ProfesionalDAO.java cambiando las consultas y el nombre de los métodos correspondientes a los casos de uso de un recurso información.

3.3.17 InformaciónDto.java

Esta clase actúa como un objeto de transferencia de datos (DTO) para encapsular la información relevante sobre un nodo de información. Sigue la misma implementación que la clase ProfesionalDto.java pero con sus campos correspondientes.

3.3.18 Consentimiento.java

Esta clase se ha utilizado para definir el recurso Consentimiento. Como se menciono anteriormente sigue la misma lógica que el modelo Profesional.java.

3.3.19 ConsentimientoController.java

Esta clase actúa como controlador del servicio y es la encargada de procesar las peticiones relacionados con el recurso Consentimiento recibidas por el servicio. Esta cuenta con instancias al servicio IConsentimiento, IInformación, IProfesional. La anotación *@RequestMapping("/api/consentimeintos")* define el punto de entrada base para todas las solicitudes relacionadas con los recursos consentimientos.

```
@RestController
@RequestMapping("/api/consentimientos")
public class ConsentimientoController {

    @Autowired
    private IConsentimiento consService;
    @Autowired
    private IInformacion infoService;
    @Autowired
    private IProfesional profService;
```

Figura 3-36 Clase ConsentimientoController.java

3.3.19.1 ¿Existe consentimiento?

Cuando el servicio recibe una petición GET al punto de entrada base, el controlador invoca al método obtenerConsentimiento.

```
@GetMapping("")
@ResponseStatus(HttpStatus.OK)
public ResponseEntity<?> obtenerConsentimiento(
        @RequestParam(required = false) String profesionalId,
        @RequestParam(required = false) String infoId,
        @RequestParam(required = false) String accNombre,
        @RequestParam(required = false) String ppNombre) {
```

Figura 3-37 Método obtenerConsentimiento al recibir una petición GET

Tabla 34 Método obtenerConsentimiento de ConsentimientoController.java

Método	**obtenerConsentimiento**
Descripción	Método que busca si existe consentimiento. Este método realiza doce consultas diferentes para ver si existe consentimiento. 1. Llama al método findConsByInfoIdAndProfesionalId del servicio IConsentimiento. 2. Método findConsByOrgInfo del servicio IConsentimiento. 3. Método findConsByEspInfo del servicio IConsentimiento. 4. Método findConsByOrgProf del servicio IConsentimiento. 5. Método findConsByEspProf del servicio IConsentimiento. 6. Método findConsByRolProf del servicio IConsentimiento. 7. Método findConsByOrgIEspP del servicio IConsentimiento. 8. Método findConsByOrgIOrgP del servicio IConsentimiento. 9. Método findConsByOrgIRolP del servicio IConsentimiento. 10. Método findConsByEspIEspP del servicio IConsentimiento. 11. Método findConsByEspIOrgP del servicio IConsentimiento. 12. Método findConsByEspIRolP del servicio IConsentimiento.
Parámetros de entrada	String id: id del paciente Se usa la anotación @RequestParam para indicar los cuatro parámetros de consulta en la URL: • Id del profesional • Id del recurso información • Nombre de la acción • Nombre del propósito
Salida	Devuelve un ResponseEntity. Si algún parámetro de consulta es nulo devuelve “Los parámetros 'profesionalId', 'infoId', 'accNombre' y 'ppNombre' son requeridos.", en el caso de que alguna consulta se cumpla devuelve el id del consentimiento, y si no se da ninguna consulta devuelve "No se ha encontrado ningún consentimiento.".

El diagrama de flujo correspondiente a este método se muestra en la Figura 3-38.

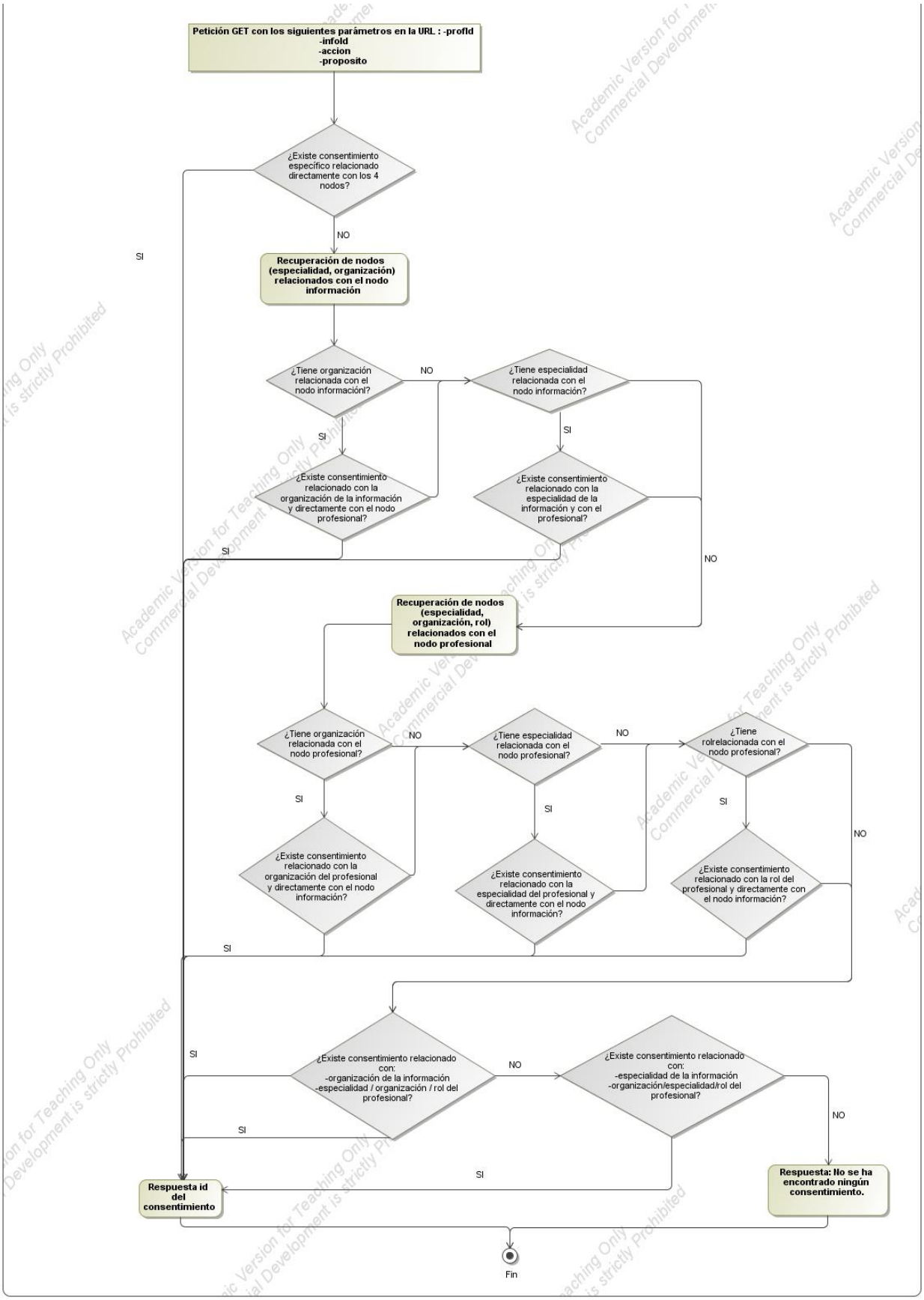


Figura 3-38 Diagrama de flujo de cómo se procesa el método obtenerConsentimiento

3.3.20 ConsentimientoImpl.java

Esta clase actúa como servicio y es la encargada de recuperar los datos y proporcionarlos al controlador.

Utiliza un objeto consentimientoDao para interactuar con la base de datos, haciendo uso de los métodos del objeto correspondiente a cada consulta de cada caso de uso.

3.3.21 ConsentimientoDAO.java

Esta clase se usa para acceder a la BD Neo4j y obtener si existe un consentimeinto. Se extiende e implementa como ProfesionalDAO.java.

4 CONCLUSIONES Y LÍNEAS FUTURAS

"Lo único constante es el cambio". - Heráclito.

Para este último capítulo se plantean las conclusiones tras la realización del proyecto, así como posibles mejoras y avances que se podrían realizar en el futuro con respecto a este.

4.1 Conclusiones

Tras finalizar el proyecto, se dividirán las conclusiones obtenidas de este en los siguientes puntos.

4.1.1 Aprendizajes supuestos

Llevar a cabo este proyecto ha supuesto tanto la aplicación de conocimientos adquiridos durante la carrera, como pueden ser:

- Desarrollo de proyectos software (asignatura de Ingeniería de Software, 3º GITT especialidad de Telemática)
- Servicios REST (asignatura de Sistemas Distribuidos y Servicios Web, 4º GITT especialidad de Telemática)

Me ha permitido aprender sobre el nuevo marco europeo de datos GDPR, el potencial modelo ABAC y la gran BD Neo4j.

A parte de los conocimientos teóricos adquiridos, también me ha permitido aprender a desarollar una base de datos que actualmente no sabía que existía, y me he dado cuenta de la gran utilidad que tiene dicha BD.

Por último, este proyecto me ha permitido poner a prueba mis capacidades, pues ha sido el primer proyecto de gran tamaño afrontado por mí mismo en solitario, con el único apoyo del tutor, suponiendo un gran reto en el que he podido actuar de forma proactiva y teniendo que enfrentarme a multitud de dificultades.

4.1.2 Objetivos y resultados del proyecto

Finalizado el proyecto, creo que los objetivos planteados al principio de este han sido cumplidos, pues se ha desarrollado un sistema PIP que permite incorporar los consentimientos de los pacientes en mecanismos de control de acceso almacenados en una base de datos basada en grafos. Además, se ha concluido que la base de datos de este tipo es de gran utilidad para el ámbito de la medicina porque permite manejar de manera eficiente y flexible grandes volúmenes de datos relacionados con pacientes, profesionales de la salud, consentimientos y otros elementos críticos, facilitando la integración y la relación entre diferentes tipos de información. Esta capacidad de conectar datos de manera dinámica mejora la toma de decisiones clínicas, la coordinación de la atención médica y garantiza un mayor control y seguridad sobre la información médica sensible.

4.2 Líneas futuras

Pese a que el proyecto se ha finalizado con resultado positivo, existen unas posibles líneas futuras que se podrían afrontar más adelante y que supondrían una mejora a este:

- **Optimización del rendimiento:** investigar y aplicar técnicas de optimización de consultas en Neo4j para mejorar el rendimiento, especialmente con grandes volúmenes de datos. Esto podría incluir el uso de índices, perfiles de consultas y estrategias de modelado más eficientes.
- **Ampliación de funcionalidades**: ampliar el servicio REST para incluir nuevas funcionalidades, como la gestión de permisos y roles, la integración con otros sistemas de salud, o la implementación de características avanzadas de análisis de datos y reportes.
- **Mejora de la seguridad**: implementar y reforzar medidas de seguridad para proteger tanto la base de datos como el servicio REST. Esto puede incluir la autenticación y autorización robusta, cifrado de datos en tránsito y en reposo, y la protección contra vulnerabilidades comunes.
- **Investigación y desarrollo**: continuar investigando nuevas tecnologías y enfoques en el ámbito de bases de datos gráficas y servicios REST. Mantenerse al día con las últimas tendencias y herramientas que puedan mejorar aún más el sistema y adaptarse a las necesidades emergentes en el ámbito de la salud.

REFERENCIAS

[1] Autor, «Este es el ejemplo de una cita,» *Tesis Doctoral,* vol. 2, nº 13, 2012.

[2] O. Autor, «Otra cita distinta,» *revista,* p. 12, 2001.

[3] Pérez, A.OBSBusiness School. "Características y fases del modelo incremental" https://www.obsbusiness.school/blog/caracteristicas-y-fases-del-modelo-incremental (consultado el 03-04-2023)

[4] European Data Protection Board, Directrices 5/2020 sobre el consentimiento en el sentido del Reglamento (UE) 2016/679. https://www.edpb.europa.eu/sites/default/files/files/file1/edpb_guidelines_202005_consent_es.pdf

[5] Wikipedia, Attribute-based access control. https://en.wikipedia.org/wiki/Attribute-based_access_control

[6] Oraclem ¿Qués es una base de datos orientada a grafos?. https://www.oracle.com/es/autonomous-database/what-is-graph-database/

[7] Amazon Web Services, ¿Qué es una base de datos de grafos?. https://aws.amazon.com/es/nosql/graph/

[8] GraphEverywhere, Las top 10 Bases de datos nativas de grafos. https://www.grapheverywhere.com/las-top-10-bases-de-datos-nativas-de-grafos/

[9] Neo4j, GraphAcademy - Neo4j. https://graphacademy.neo4j.com/

[10] Code 360, What is Spring Tool Suite (STS). https://www.naukri.com/code360/library/spring-tool-suite

[11] Wikipedia, Eclipse (software). https://es.wikipedia.org/wiki/Eclipse_(software)

[12] Spring by VMware Tanzu, Why Spring? https://spring.io/why-spring

[13] Spring by VMware Tanzu, Spring Data Neo4j. https://spring.io/projects/spring-data-neo4j

[14] Neo4j, Neo4j Desktop. https://neo4j.com/docs/desktop-manual/current/

[15] Apache Maven, Maven - Welcome to Apache Maven. https://maven.apache.org/

[16] OpenWebinars, Qué es REST: Conoce su potencia. https://openwebinars.net/blog/que-es-rest-conoce-su-potencia/

[17] Postman, What is Postman? https://www.postman.com/product/what-is-postman/

[18] Wikipedia, MagicDraw UM. https://es.wikipedia.org/wiki/MagicDraw_UML