ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

MIDDLEWARE UNIFICADO SQL PARA BASES DE DATOS NOSQL CLAVE-VALOR, GRAFOS Y GEOESPACIAL

DESARROLLO DE UN MIDDLEWARE PARA LA TRADUCCIÓN DE CONSULTAS SQL A UN SISTEMA DE BASE DATOS ORIENTADAS A GRAFOS

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO/A EN SOFTWARE

EDWIN ANDRES CANTUÑA GUANO

edwin.cantuna@epn.edu.ec

DIRECTOR: VICTOR VICENTE VELEPUCHA BONETT

victor.velepucha@epn.edu.ec

DMQ, diciembre 2025

CERTIFICACIÓN

Yo, EDWIN ANDRES CANTUÑA GUANO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

EDWIN ANDRES CANTUÑA GUANO

Certifico que el presente trabajo de integración curricular fue desarrollado por EDWIN ANDRES CANTUÑA GUANO, bajo mi supervisión.

PHD. VICTOR VICENTE VELEPUCHA BONETT

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

EDWIN ANDRES CANTUÑA GUANO
PHD. VICTOR VICENTE VELEPUCHA BONETT
SEBASTIAN ALEJANDRO SANCHEZ MALDONADO
JAIRO RENE SIMBAÑA CAIZA

DEDICATORIA

Este trabajo de titulación dedico a mis padres, Edwin Cantuña y Nancy Guano, quienes con su amor incondicional, su constante apoyo y esa incansable lucha que realizan día a día, guian cada paso que doy hacia las metas y objetivos que me he propuesto. A mis hermanas, Stefany y Damaris, quienes con sus ocurrencias y carismas me dan motivos y fuerzas para seguir adelante en este camino y no rendirme. También dedico a toda mi familia

AGRADECIMIENTOS

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.
Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Índice general

Índice de figuras

Índice de Tablas

RESUMEN

El presente Trabajo de Integración Curricular (TIC) se centra en el desarrollo de un middleware unificado SQL para bases de datos NoSQL orientadas a grafos. Este componente aborda el desafío de integrar aplicaciones tradicionales basadas en SQL con tecnologías NoSQL, reduciendo la curva de aprendizaje y mejorando la interoperabilidad. Para lograrlo, se diseñó e implementó un parser SQL que descompone consultas en un árbol de análisis sintáctico, junto con un motor de traducción basado en reglas de mapeo que convierte las consultas en un lenguaje específico de bases de datos orientadas a grafos, como Cypher de Neo4j. Además, se desarrolló una interfaz de usuario para validar el funcionamiento del conector. Las pruebas demostraron la capacidad del sistema para ejecutar consultas complejas en entornos NoSQL, manteniendo la familiaridad de la sintaxis SQL. Este enfoque contribuye a la migración eficiente de sistemas relacionales a arquitecturas modernas, facilitando su adopción en la industria.

Palabras Claves - Parser SQL, Traductor SQL-NoSQL, bases de datos orientadas a grafos, GQL

ABSTRACT

This Curricular Integration Project (TIC) focuses on the development of a unified SQL middleware for graph-oriented NoSQL databases. This component addresses the challenge of integrating traditional SQL-based applications with NoSQL technologies, reducing the learning curve and improving interoperability. To achieve this, a SQL parser that decomposes queries into a parse tree was designed and implemented, along with a translation engine based on mapping rules that converts queries into a specific graph-oriented database language, such as Neo4j's Cypher. In addition, a user interface was developed to validate the operation of the connector. Testing demonstrated the system's ability to execute complex queries in NoSQL environments while maintaining the familiarity of the SQL syntax. This approach contributes to the efficient migration of relational systems to modern architectures, facilitating their adoption in the industry.

Keywords - SQL Parser, SQL-NoSQL Translator, graph-oriented databases, GQL

Capítulo 1

INTRODUCCIÓN

El auge de los datos no estructurados y la necesidad de sistemas escalables horizontalmente han impulsado la creciente adopción de bases de datos NoSQL, las cuales ofrecen modelos de datos más flexibles y eficientes para grandes volúmenes de información en comparación con las bases de datos relacionales tradicionales. Sin embargo, la migración e integración de aplicaciones existentes, así como la curva de aprendizaje asociada a los diferentes lenguajes de consulta y APIs de cada base de datos NoSQL, representan un desafío significativo para los desarrolladores. Esta problemática dificulta una transición fluida desde entornos SQL bien establecidos hacia el paradigma NoSQL.

En este contexto, el componente desarrollado en este proyecto es un middleware SQL, diseñado para actuar como una capa de abstracción entre las aplicaciones que utilizan consultas SQL y los diversos sistemas de bases de datos NoSQL, específicamente aquellos de tipo orientados a grafos. La finalidad principal de este middleware es traducir las consultas SQL en operaciones compatibles con los lenguajes nativos de las bases de datos NoSQL seleccionadas, eliminando la necesidad de que los desarrolladores adquieran un conocimiento profundo en un lenguaje de consulta orientado a grafos como Cypher. Al proveer esta capa de traducción, el middleware simplifica la integración de aplicaciones existentes con entornos NoSQL y reduce significativamente la complejidad de la migración de datos y funcionalidades.

El desarrollo de este middleware implica la construcción de un parser SQL capaz de analizar y descomponer las consultas SQL en componentes manejables. Posteriormente, se implementará un conector para bases de datos orientadas a grafos utilizando el motor de Neo4j. Este conector será responsable de transformar la representación intermedia de las consultas SQL al lenguaje de consulta Cypher, propio de la base de datos orientada a

grafos Neo4j. Además, se desarrollará un conector de entrada para el middleware que recibirá las consultas SQL directamente, y una interfaz de usuario para que los desarrolladores puedan interactuar con el sistema, ingresar consultas SQL y visualizar los resultados traducidos y ejecutados en el sistema NoSQL apropiado. Este enfoque integral busca mejorar la interoperabilidad entre los paradigmas SQL y NoSQL, ofreciendo una solución práctica para gestionar la complejidad de los entornos de datos híbridos.

1.1. Objetivo general

Desarrollar un middleware que integre un parser SQL, un conector para base de datos orientada a grafos y un conector para bases de datos relacionales que permita traducir consultas SQL en operaciones compatibles con bases de datos orientadas a grafos a través de una interfaz de usuario que permita ingresar la consulta SQL e indique su equivalente en un lenguaje orientado a grafos.

1.2. Objetivos específicos

1. Desarrollar un parser SQL que analice y descomponga las consultas SQL en componentes manejables.

2. Implementar conectores para bases de datos orientadas a grafos, que traduzcan las consultas SQL a consultas GQL.

3. Desarrollar un conector que permita recibir consultas SQL y enviarlas al middleware. El middleware será responsable de traducir estas consultas SQL a una representación intermedia, facilitando su posterior conversión al lenguaje de consulta de grafos.

1.3. Alcance

El alcance del proyecto comprende el desarrollo de un middleware unificado SQL para bases de datos NoSQL, con capacidad para traducir consultas SQL en instrucciones compatibles con bases de datos orientadas a grafos, clave-valor y geoespaciales. Este componente incluye:

1. **Revisión de literatura y análisis de requisitos:** Se realizará una investigación exhaustiva sobre las técnicas y herramientas existentes para la traducción de consultas SQL

a NoSQL, así como un análisis de los requisitos necesarios para el desarrollo del middleware.

2. **Definición de la gramática SQL:** Utilizando la herramienta adecuada, se definirá y generará la gramática SQL necesaria para el análisis y descomposición de consultas SQL en componentes manejables.

3. **Desarrollo del parser SQL:** Implementación del parser SQL que analizará las consultas SQL y generará un árbol de análisis que servirá de base para la traducción a lenguajes de consulta NoSQL.

4. **Desarrollo de conectores NoSQL:** Implementación de un conector que traduzca consultas SQL al lenguaje de consulta Cypher. Este conector permitirá la ejecución de consultas traducidas, enfocándose en las relaciones y estructuras interconectadas.

5. **Desarrollo de conector:** Implementación de un conector que permita recibir consultas SQL y enviarlas al middleware. El middleware será responsable de traducir estas consultas SQL a una representación intermedia.

6. **Interfaz de usuario:** El middleware incluirá una interfaz de usuario que permitirá a los desarrolladores interactuar con el sistema de manera directa. A través de esta interfaz, los usuarios podrán ingresar consultas SQL y recibir los resultados traducidos y ejecutados en el sistema NoSQL correspondiente.

1.4. Marco teórico

1.4.1. Antecedentes

Dentro de los últimos años, el uso de bases de datos no relacionales en la implementación de sistemas de información ha crecido considerablemente motivo por el cual muchas empresas de desarrollo de software han optado su deseo de migrar las bases de datos SQL en sus soluciones a NoSQL, dado la flexibilidad, escalabilidad y variedad que brinda este tipo de bases de datos hacia sus intereses. Esta necesidad ha conllevado al desarrollo de soluciones que permitan realizar una eficiente traducción entre estos dos tipos de bases de datos. Para el presente trabajo, se ha realizado una búsqueda exhaustiva de anteriores trabajos referentes a la traducción de bases de datos relacionales a orientadas a grafos dentro de plataformas digitales como la Biblioteca Digital EPN, SCOPUS y Google Scholar.

Se utilizaron varias cadenas de búsqueda como “SQL a NOSQL”, “Traducción de SQL a NoSQL”, “Traducción SQL a grafos”, “Relacional a grafos” y “Relacional a NoSQL” con lo cual se encontraron algunos trabajos relevantes:

- Dai J [1], se centra en la transformación del esquema, la traducción y la optimización de consultas, especialmente en cargas de trabajo OLAP y bases de datos NoSQL orientadas a columnas. En este articulo indica que la desnormalización en NoSQL es clave para evitar costosas operaciones `JOIN` en entornos NoSQL y que técnicas como MapReduce son fundamentales en la traducción de consultas SQL a operaciones NoSQL.

- Namdeo y Suman [2], proponen un middleware para traducción de consultas SQL a NoSQL (SQL-No-QT) el cual actúa como capa intermedia entre aplicaciones heredadas y bases de datos MongoDB. Este modelo tiene la capacidad de traducir todas las consultas `CRUD` e incluso aquellas que contienen sentencias `JOIN` , de lo que carecía soluciones anteriores como UnityJDBC que se limitaban a traducir consultas `SELECT` sencillas.

- Đukic et al. [3], presenta un enfoque sistemático para poder convertir bases de datos relacionales a bases de datos de grafos, enfocándose especialmente en la migración a Neo4J. Esta metodología abarca tres fases: preparación, carga de datos y generación de relaciones, y optimizacion de la base de datos de grafos. A través de este enfoque se obtuvo una mayor eficiencia en el manejo de datos densamente relacionados así como tiempos de ejecución más cortos en comparación con las bases de datos relacionales.

1.4.2. Arquitectura de Software

La arquitectura de software es la forma de representar cómo se construye un sistema, dividiéndolo en componentes que se comunican entre sí. El propósito de diseñar una arquitectura eficaz es facilitar el entendimiento, desarrollo, despliegue y mantenimiento del producto por parte de los desarrolladores. De acuerdo con Martin [4], una buena arquitectura debe estar centrada en los casos de uso y ser plasmada sobre las funciones del sistema, en lugar de estar dominada por las tecnologías subyacentes.

Para el presente proyecto se definió una arquitectura modular para representar el middleware traductor de consultas SQL a su equivalente en Cypher, el lenguaje con el cual

trabaja las bases de datos de Neo4j. En la Figura 1.1, se puede visualizar la arquitectura adoptada para la realización de este proyecto. Aquí se identifican 4 componentes claves como son: Cliente, Frontend, Backend y las bases de datos, además se indica los protocoles de comunicación respectivos entre los componentes interrelacionados.

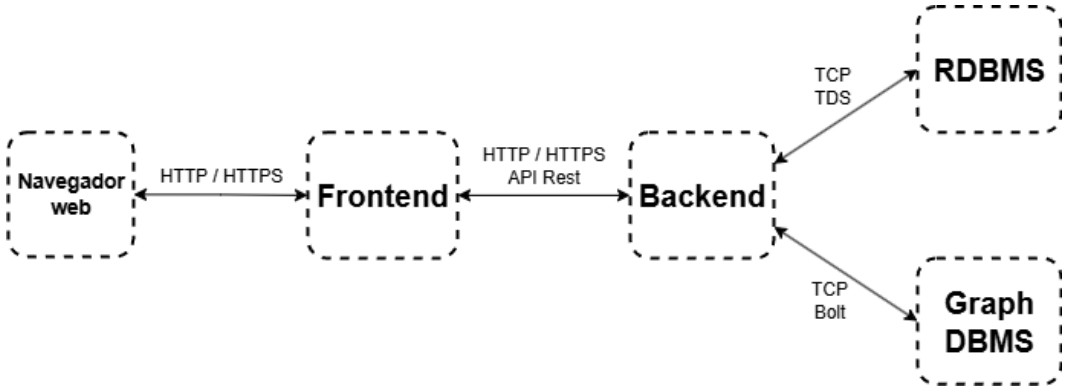

Figura 1.1: Arquitectura del middleware.

1.4.3. Metodología de Desarrollo de Software

Las metodologías de desarrollo de software son un conjunto de procesos, métodos y herramientas que permiten gestionar cada fase del ciclo de vida del desarrollo de un producto software. Su objetivo principal es organizar y estructurar el trabajo del equipo de desarrollo, garantizando que el proyecto se complete de manera eficiente [5]. A continuación se tratará acerca de dos enfoques muy conocidos como son las metodologías tradicionales y Agile.

Metodologías Tradicionales

De acuerdo con Pressman [5], este enfoque se caracteriza por seguir un proceso estricto y lineal, es decir, se requiere completar una fase para avanzar a la siguiente. Estos enfoques suelen aplicarse cuando los requerimientos se encuentran bien definidos y por tanto, los cambios serán mínimos, además de tener una documentación exhaustiva. Su única desventaja radica en la carencia de flexibilidad a los cambios.

Metodologías Ágiles

Por su parte, las metodologías ágiles se basan en un enfoque iterativo e incremental que prioriza la flexibilidad al cambio, el trabajo colaborativo y la entrega continua de valor. Se divide en ciclos cortos de trabajo con el propósito de que al finalizar cada ciclo se pueda

entregar una versión funcional del producto. Este enfoque fomenta la participación activa del cliente maximizando su satisfacción mediante la entrega rápida y constante de un producto de alta calidad [6].

Para cumplir de manera eficiente con los principios y valores de este tipo de metodologías, existen frameworks siendo el más conocido SCRUM, el cual será tratado a continuación.

Framework SCRUM

Es un marco de trabajo que proporciona un conjunto de roles, eventos y artefactos para gestionar y organizar proyectos de desarrollo de software. SCRUM permite una entrega de valor frecuente y una adaptación constante a las necesidades del negocio [7].

En las Tablas 1.1, 1.2 y 1.3, se podrá observar un resumen de los roles, eventos y artefactos proporcionados por este framework.

Tabla 1.1: Roles de Scrum

Rol	Descripción
Product Owner	Es el responsable de maximizar el valor del producto.
Scrum Master	Es el facilitador que guía al equipo en la correcta aplicación de Scrum.
Equipo de Desarrollo	Está encargado de entregar un incremento funcional del producto.

Tabla 1.2: Eventos de Scrum

Evento	Descripción	Duración
Sprint	Un ciclo de trabajo de duración fija, en el que se crea un incremento de producto potencialmente entregable.	De 2 a 4 semanas
Sprint Planning	La reunión donde el equipo planifica el trabajo a realizar durante el sprint y define su objetivo.	2 horas por semana de Sprint

Continúa en la siguiente página...

Evento	Descripción	Duración
Daily Scrum	Una reunión diaria de 15 minutos para que el equipo sincronice sus actividades y adapte el plan del día.	15 minutos
Sprint Review	El encuentro donde el equipo inspecciona el incremento de producto con los stakeholders y obtiene retroalimentación.	1 hora por semana de Sprint
Sprint Retrospective	Una reunión para que el equipo inspeccione su proceso de trabajo e identifique mejoras para el próximo sprint.	45 minutos por semana de Sprint.

Tabla 1.3: Artefactos de Scrum

Artefacto	Descripción
Product Backlog	La lista priorizada de todo el trabajo necesario para el producto.
Sprint Backlog	El conjunto de elementos del Product Backlog seleccionados para el sprint.
Incremento de Producto	Los elementos completados en un sprint y los anteriores, listo para producción.

1.4.4. Frontend

El frontend de una aplicación web es la parte con la que los usuarios interactúan directamente. Su propósito principal es presentar la interfaz de usuario (UI) y añadir interactividad. Las tres lenguajes fundamentales e indispensables para el desarrollo frontend son HTML (Hypertext Markup Language), que define la estructura y la semántica de una página web; CSS (Cascading Style Sheets), que se encarga del diseño y la presentación; y JavaScript, que hace que las páginas web sean interactivas y dinámicas, permitiendo funciones como la clasificación de datos o la validación de formularios en el lado del cliente. En aplicaciones web modernas, como las aplicaciones de una sola página (SPA) implementadas con librerías o frameworks como React, Angular o Vue, gran parte de la lógica de presentación se ejecuta en el cliente, lo que las convierte en çlientes gruesos". Además, el frontend utiliza APIs web, como el DOM API, para manipular elementos HTML, y el Fetch API o Ajax para

cargar datos de manera asíncrona desde el servidor, optimizando la experiencia del usuario [8].

TypeScript

Es un lenguaje de programación que contiene la misma sintaxis base de JavaScript razón por la que es conocido como el “superset de JavaScript”. Es un sistema caracterizado por cuatro elementos clave [9]:

- Extiende la sintaxis de JavaScript con características específicas para definir y usar tipos.
- Tiene un verificador de tipos que examina el código para detectar configuraciones incorrectas de variables y funciones.
- Su compilador ejecuta el verificador de tipos, reporta cualquier problema y luego genera el código JavaScript equivalente.
- Dispone de un servicio de lenguaje que ofrece herramientas útiles a los desarrolladores, garantizando la legibilidad y mantenibilidad del código.

Para Goldberg [9], la adopción de TypeScript aporta beneficios sustanciales al desarrollo web al mitigar las debilidades de JavaScript, como la falta de seguridad y la documentación imprecisa. Ofrece seguridad al permitirnos especificar explícitamente los tipos de valores para parámetros y variables, lo que ayuda a prevenir errores en tiempo de ejecución que de otro modo podrían causar fallos en JavaScript. Esta restricción controlada sobre el código garantiza que los cambios en una sección no afecten inesperadamente a otras. Además, TypeScript facilita una documentación precisa al integrar la sintaxis para describir la forma de los objetos, creando un sistema de tipado robusto y obligatorio. Las herramientas de desarrollo (developer tooling) se ven significativamente mejoradas, ya que la información de tipos permite a los editores proporcionar sugerencias inteligentes de autocompletado y facilitar refactorizaciones complejas. A pesar de ser un lenguaje en constante evolución, el código TypeScript se transpila a JavaScript y todas las anotaciones y alias de tipos se eliminan en el proceso, sin añadir sobrecarga al tiempo de ejecución.

React

Es una biblioteca de JavaScript declarativa que facilita construcción de interfaces de usuario. Se centra en la composición de componentes, el manejo del estado y un flujo de datos unidireccional. La arquitectura de React fomenta una jerarquía de componentes, donde las partes más grandes de la interfaz de usuario se dividen en subcomponentes más pequeños, siguiendo principios como el de responsabilidad única. Esta modularidad facilita la construcción de interfaces complejas y su mantenimiento [10]. La comunicación entre componentes en React se gestiona principalmente a través de dos conceptos:

- **Props:** Son la manera de enviar datos de un componente padre hacia sus componentes hijos. Se pasan a una función como argumentos. El uso de props hace que un componente padre pueda personalizar la apariencia y el comportamiento de sus componentes hijos, y una vez que se pasan, no pueden ser modificados por los componentes hijos.
- **State:** El estado representa el conjunto mínimo de datos cambiantes que su aplicación necesita recordar para que la interfaz de usuario sea interactiva. El estado permite que un componente realice un seguimiento de la información y la cambie en respuesta a las interacciones del usuario. El principio más importante para estructurar el estado es mantenerlo DRY, identificando la representación mínima absoluta del estado y calculando todo lo demás bajo demanda. Para determinar si algo debe ser estado, se considera si cambia con el tiempo, si se pasa como prop desde un padre o si se puede calcular a partir del estado o props existentes.

NextJS

NextJS es un framework robusto utilizado para la creación de aplicaciones web mediante el uso de componentes React y otras funciones adicionales, logrando que estas sean dinámicas e interactivas. Entres sus principales características están el soporte de renderizado del lado del servidor, la generación de sitios estáticos y la fomentación de buenas prácticas dentro del desarrollo web, como rendimiento, SEO y seguridad [11].

Tailwind CSS

Es un framework CSS que proporciona una vasta colección de clases de utilidad de bajo nivel. Estas clases tienen un propósito único el cual es construir interfaces de usuario completas sin necesidad de escribir directamente reglas CSS en archivos separados. Simplemente se aplican las clases de utilidad a los elementos HTML a través de su atributo class. Además, facilita la implementación de diseños responsivos mediante prefijos de puntos de interrupción y permite la reutilización de estilos y la adhesión a la estrategia DRY [12].

Este framework soporta un alto nivel de personalización y permite manejar eventos y estados directamente a través de clases de utilidad.

1.4.5. Backend

El backend es la parte de una aplicación web que se ejecuta en el servidor y maneja la lógica de la aplicación y la gestión de datos. A diferencia de las páginas estáticas que solo sirven archivos, las páginas web dinámicas requieren que el servidor genere contenido en tiempo real, a menudo recuperando información de una base de datos. Las tecnologías de backend incluyen lenguajes de programación como JavaScript, PHP, Java, Python o Ruby, servidores web como nginx o Apache HTTP Server, y bases de datos, que pueden ser relacionales o NoSQL [8].

Python

Es uno de los lenguajes de programación más populares en la actualidad, bastante utilizado en distintas áreas como en el desarrollo web, la ciencia de datos y el Machine Learning dada su eficiencia, la interoperabilidad y facilidad para aprender [13]. En Python se puede utilizar bibliotecas predefinidas para el trabajo con bases de datos y la validación de la información contenida. A continuación se presenta alguna bibliotecas útiles para el desarrollo del middleware:

- **Biblioteca Pydantic:** Esta biblioteca de Python permite validar datos mediante type hints para lo cual, se definen modelos para verificar y convertir datos automáticamente. Con esto, mejora la calidad de los datos y la robustez de aplicaciones Python.
- **Biblioteca Pyodbc:** Esta biblioteca ayuda a la conexión de aplicaciones Python con

bases de datos vía ODBC. Además, permite ejecutar consultas de forma eficiente desde distintos motores de bases de datos como SQL Server, PostgreSQL, MySQL entre otros.

- **Biblioteca sqlparse:** Es una biblioteca de Python utilizado para analizar, validar y generar consultas SQL.

Para el desarrollo de aplicaciones web existen frameworks conocidos que fueron creados en Python tales como Django, Flask o FastApi.

FastAPI

FastAPI es un framework web moderno y de alto rendimiento que se distingue por un conjunto de características que lo hacen altamente eficiente para el desarrollo de APIs [14]:

- Es uno de los frameworks web más rápidos en Python dado su rendimiento y por soportar la programación asíncrona permitiendo el manejo de grandes volúmenes de solicitudes con latencia mínima.
- Utiliza Python type hints y modelos Pydantic para la validación automática de datos, garantizando que los datos de entrada se validen con precisión según los tipos y restricciones especificados, lo que lleva a un código más seguro.
- FastAPI genera automáticamente documentación API interactiva a través del uso de interfaces como Swagger UI y ReDoc. Con esto, permite la prueba y exploración de endpoints en tiempo real desde el navegador, facilitando la comprensión de la API y aumentando la mantenibilidad del código.
- Utiliza el servidor ASGI Uvicorn, el cual se distingue por su ligereza y su alto rendimiento en aplicaciones web asíncronas.
- Reduce los errores y aumenta la velocidad de desarrollo a través de funciones de autocompletado y verificación de tipos en el editor.

1.4.6. Bases de Datos

De acuerdo con Garcia-Molina [15], una base de datos es una colección organizada de información que puede persistir durante mucho tiempo y es administrada por un Sistema

Gestor de Bases de Datos (DBMS, por sus siglas en inglés). Un DBMS tiene como objetivo almacenar, recuperar y gestionar esta información de manera eficiente [16].

Bases de Datos Relacionales

Las bases de datos relacionales conforman el tipo más común de almacenamiento de información que ha existido hasta la fecha. Son un conjunto de tablas con datos que comparten atributos similares y pueden establecer relaciones con otras tablas mediante algún atributo [17]. Para garantizar confiabilidad en las transacciones, estas bases de datos cumplen con principios ACID:

- **Atomicidad:** Una transacción es "todo o nada"; o se completa por completo, o no se realiza ninguna de sus partes.
- **Consistencia:** Cada transacción debe llevar la base de datos de un estado válido a otro estado válido.
- **Aislamiento:** Las transacciones concurrentes no deben interferir entre sí; para el usuario, parece que se ejecutan una tras otra.
- **Durabilidad:** Una vez que una transacción se confirma, sus cambios son permanentes y persisten incluso si hay fallas en el sistema.

SQL Server

SQL Server es un sistema de gestión de bases de datos relacionales desarrollado por Microsoft. Utiliza SQL como su lenguaje principal y soporta diversos tipos de datos incluyendo numéricos, caracteres, entre otros [17]. Para la administración de infraestructuras SQL como SQL Server y Azure SQL Database SQL, Microsoft dispone del entorno SQL Server Management Studio (SSMS). Esta herramienta permite acceder, configurar, administrar y desarrollar todos los componentes de servidores SQL además de poder administrar bases de datos, realizar consultas y auditorías mediante scripts.

Bases de Datos NoSQL

Las bases de datos no relacionales, también conocidas como NoSQL (Not only SQL) ofrecen una alternativa de almacenamiento y gestión de grandes volúmenes de datos ya

que no requiere una estructura formal para agregarlos [18]. Con esto proporciona mayor flexibilidad al tratar con datos no uniformes o campos personalizados [19].

Existen diferentes tipos de bases de datos NoSQL entre los que se encuentran [20]:

- **Clave-valor:** Se almacenan como pares, siendo la clave el identificador único del valor asociado.
- **Documentos:** Almacenan datos en documentos semiestructurados especialmente en formatos como JSON o XML.
- **Familias de Columnas:** Organizan los datos en filas que tienen muchas columnas asociadas con una clave de fila.

Bases de Datos Orientadas a Grafos

Las bases de datos orientadas a grafos representan una categoría importante de bases no relacionales al modelar los datos como una red de nodos y relaciones [21]. Este enfoque brinda alternativas para escenarios donde la complejidad de los datos reside en sus conexiones y no solo en el volumen de estos. Este modelo abarca cuatro componentes fundamentales [22]:

- **Nodos:** Almacenan información de las entidades.
- **Relaciones:** Conectan a los nodos de forma explícita, es decir, tiene un tipo, un nodo origen, un nodo destino y una dirección.
- **Propiedades:** Son pares clave-valor que permite agregar atributos tanto a nodos como a relaciones.
- **Etiquetas:** Permiten agrupar y categorizar nodos.

Neo4j

Es una base de datos de grafos diseñada para gestionar y recorrer grandes cantidades de datos conectados con facilidad. Su infraestructura y formato de almacenamiento optimizado para el manejo de grafos ofrece mejoras significativas de velocidad y rendimiento con respecto a otras bases de datos [23]. Tiene un lenguaje de consulta propio llamado Cypher. Este permite a los usuarios describir el patrón de datos que desean encontrar facilitando las optimizaciones de búsqueda y mejorando la legibilidad [24].

Neo4j Desktop y Neo4j Browser

Estas herramientas sirven para gestionar bases de datos Neo4j. Neo4j Desktop permite la exploración y uso de este tipo de bases de datos de forma local. Por otro lado, en Neo4j Browser se puede visualizar los resultados de consultas sobre una base de datos específica a través de scripts y manipular los datos que contiene, ya sea los nodos o relaciones [24].

1.4.7. Herramientas de desarrollo, gestión y pruebas

Figma

Es una herramienta de diseño colaborativa basada en el navegador, que permite a los usuarios trabajar juntos en tiempo real en la misma interfaz de usuario. Destaca por sus potentes componentes reutilizables y su sistema flexible de anulaciones. Ofrece entrega integrada para desarrolladores, permitiendo copiar fragmentos de código y activos directamente del diseño. Además, facilita el prototipado, ya que los diseños siempre están en la nube, eliminando la necesidad de cargas manuales [25].

Git

Git es un sistema de control de versiones distribuido que almacena datos como una serie de instantáneas del proyecto. Su modelo de ramificación es una característica fundamental, permitiendo crear y destruir ramas de forma rápida y sencilla para aislar el trabajo en temas específicos. Permite trabajar con múltiples repositorios remotos y operar sin conexión. Git se utiliza principalmente a través de la línea de comandos para acceder a toda su funcionalidad, ofreciendo además herramientas para reescribir la historia localmente antes de compartirla [26].

GitHub

GitHub es el mayor proveedor de alojamiento de repositorios Git y un punto central para la colaboración de desarrolladores en proyectos de código abierto. Su flujo de trabajo se centra en las Pull Requests (PRs), que facilitan la discusión, revisión y fusión de cambios de manera colaborativa directamente en la plataforma web. Permite a los usuarios bifurcar proyectos, trabajar en sus propias copias y luego solicitar la integración de sus cambios.

GitHub soporta Markdown con características especiales como listas de tareas y fragmentos de código, además de ofrecer un sistema de notificaciones y funcionalidades como la automatización de tareas [26].

Visual Studio Code

Visual Studio Code es un editor de código potente y completo que funciona en Mac, Windows y Linux. Una de sus principales ventajas es su comprensión de todo el proyecto, incluyendo la relación entre archivos y soporte para múltiples lenguajes como Python, JavaScript y CSS. Ofrece autocompletado de código, navegación y refactorización inteligentes, y facilita la gestión de entornos virtuales y dependencias. Visual Studio Code también integra control de versiones para sistemas como Git, y herramientas para el desarrollo web frontend y backend, razón por la cual ha sido la elegida para el desarrollo de este trabajo [27].

Postman

Esta herramienta diseñada para la prueba y el desarrollo de APIs, permite a los usuarios crear, organizar y enviar solicitudes API en colecciones. Ofrece amplias opciones de autorización para APIs, incluyendo Basic Auth, Bearer Tokens, y OAuth, lo que facilita la interacción con diversas seguridades. Permite la creación de scripts de validación de pruebas y soporta pruebas basadas en datos para escalar los tests. Además, Postman facilita el diseño de especificaciones API, como OpenAPI, y puede generar automáticamente mocks y pruebas a partir de estas especificaciones. La herramienta también soporta la documentación de APIs directamente en la aplicación y la gestión de variables [28].

1.4.8. Pruebas de Software

Son una actividad fundamental en la ingeniería de software, cuyo objetivo es demostrar que un programa hace lo que se pretende y descubrir defectos antes de que el sistema se use. Al probar el software, se ejecuta el programa con datos artificiales y se verifican los resultados para buscar errores, anomalías o información sobre los atributos no funcionales del programa. Es importante destacar que las pruebas solo pueden mostrar la presencia de errores, no su ausencia. Siempre existe la posibilidad de que una prueba pasada por alto revele más problemas con el sistema [29].

Pruebas Funcionales

Las pruebas funcionales se refieren directamente a la verificación de los servicios y funcionalidades específicas que el sistema debe proveer. Se basan en los requerimientos funcionales, que son enunciados que describen cómo el sistema debe reaccionar a entradas particulares y cómo debe comportarse en situaciones específicas. Pueden ser descripciones abstractas o muy detalladas de las funciones del sistema, sus entradas, salidas y excepciones. El objetivo de estas pruebas es demostrar que el software ha implementado adecuadamente sus requerimientos [29].

Pruebas No Funcionales

Las pruebas no funcionales se ocupan de las limitaciones o restricciones sobre los servicios o funciones que ofrece el sistema. Estas restricciones pueden relacionarse con propiedades emergentes del sistema como la fiabilidad, el rendimiento, la seguridad, la usabilidad y la protección. Son a menudo más críticas que los requerimientos funcionales individuales, ya que el fracaso en cumplir con un requerimiento no funcional puede hacer que todo el sistema sea inútil. Siempre que sea posible, los requerimientos no funcionales deben escribirse de manera cuantitativa y medible, para que puedan ser probados objetivamente [29].

Capítulo 2

METODOLOGÍA

En este capítulo se describe la construcción del *middleware* de traducción de sentencias SQL a un lenguaje de bases de datos orientado a grafos donde se detalla tanto el diseño como la implementación del mismo incluyendo una interfaz gráfica intuitiva y fácil de utilizar para el usuario final.

Para el desarrollo de este componente se eligió el marco de trabajo Scrum, caracterizado por ser iterativo e incremental, facilitando una entrega de valor de manera rápida y continua, además de su adaptabilidad a los requisitos cambiantes a lo largo del proyecto. Los roles, eventos y artefactos que ofrece este marco de trabajo permitirán una correcta organización y una comunicación eficiente entre los miembros del equipo Scrum para poder entregar un producto de alto valor.

Para ello, se definieron 3 fases importantes que serán tratadas a continuación.

2.1. Fase Exploratoria

En esta fase se definieron las partes interesadas del proyecto, los roles de Scrum que cada miembro del equipo cumplirá y además, se realizó la planificación para el desarrollo del *middleware*.

2.1.1. Stakeholders

Para poder cubrir las necesidades de este componente, en la Tabla 2.1 se identificaron los siguiente stakeholders y el papel que cumplirá cada uno de ellos:

Tabla 2.1: Partes interesadas del proyecto

ID	Stakeholder	Descripción
S1	Desarrollador (Usuario)	Persona que interactuará con el traductor de consultas SQL a un lenguaje de consulta de grafos.
S2	Docente	Persona encargada de supervisar el cumplimiento de los objetivos y necesidades del componente.
S3	Desarrollador	Persona responsable del diseño y construcción del *middleware* asegurando que se cumplan con todos los requisitos definidos.

2.1.2. Planificación del proyecto

Una vez identificados los stakeholders, se estableció la planificación del proyecto donde se contempló 358 horas de trabajo dentro de un periodo de 5 meses para el desarrollo del producto. Aquí se definieron actividades que cubren el análisis de requerimientos, diseño, implementación y pruebas que se realizarán al sistema.

En la Figura 2.1 se indica la lista de actividades consideradas con el número de horas asignadas a cada una de ellas.

	ACTIVIDADES ESPECÍFICAS	MES	MES 1				MES 2				MES 3				MES 4				MES 5				HORAS
		SEMANA	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
1	Revisión sistemática de la literatura sobre gramáticas, parsers y técnicas de traducción SQL a grafos.		20																				20
2	Elaboración de las historias de usuario.			10																			10
3	Diseño de la arquitectura del middleware.			10																			10
4	Diseño de la interfaz de usuario.				8																		8
5	Definición de la gramática SQL.					10	10																20
6	Desarrollo del parser SQL.					10	10	10	10	10													50
7	Diseño del conector para bases de datos SQL.								10	10													20
8	Diseño del conector para bases de datos orientadas a grafos.											20	20	10									50
9	Implementación de conectores para traducir consultas SQL a lenguaje de consulta de grafos.														20	20	10						50
10	Desarrollo de la interfaz de usuario.										10						10						20
11	Pruebas de integración y funcionales al sistema.																		10	10			20
12	Pruebas no funcionales al sistema.																			10			10
13	Documentación del sistema.																				5	5	15
	Daily Scrum		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1				17
	Sprint Planning		4			4			4			4			4			2					22
	Sprint Review							2			2			2			2		2				10
	Sprint Retrospective				1			1			1			1			1		1				6
		TOTAL HORAS																					358

Figura 2.1: Planificación del proyecto.

A lo largo del proyecto se llevaron a cabo reuniones semanales con el Ph.D. Víctor Velepucha, director del proyecto de integración curricular, con la finalidad de obtener una basta retroalimentación que permita la correcta construcción del *middleware*.

2.1.3. Roles de Scrum

Dado que se utilizó el marco de trabajo Scrum para una entrega de valor de manera ágil, es indispensable determinar los roles que cada miembro cumplirá dentro del equipo de trabajo. En la Tabla 2.2 se detalla la asignación de roles para el proyecto:

Tabla 2.2: Roles de Scrum

Rol	Persona(s) Asignada(s)
Product Owner	Víctor Velepucha
Scrum Master	Víctor Velepucha
Equipo de Desarrollo	Andrés Cantuña, Sebastián Sánchez y René Simbaña

2.2. Fase de Inicialización

Para esta fase se redactaron las historias de usuario que cubran las necesidades del negocio y se realizó la selección del stack tecnológico que permita la gestión y el desarrollo eficiente del sistema.

2.2.1. Herramientas

La Tabla 2.3 indica las herramientas elegidas para llevar a cabo el diseño y el desarrollo del sistema. Estas herramientas fueron elegidas en base a la versatilidad y la fácil integración entre sí.

Tabla 2.3: Stack tecnológico.

Herramienta	Descripción
Figma	Utilizado para el diseño de las interfaces de usuario del sistema.
Visual Studio Code	Editor de código utilizado para el desarrollo del sistema debido a su versatilidad y compatibilidad con múltiples lenguajes y frameworks.

Continúa en la siguiente página...

Herramienta	Descripción
Postman	Herramienta que permitirá evaluar los endpoints del *middleware* para asegurar su correcta integración con otros sistemas.
Git y GitHub	Se utilizarán tanto para el control de versiones como para el almacenamiento del código del proyecto.

2.2.2. Configuración del Proyecto

Historias de Usuario

Una vez precisado el desafío de negocio, se definieron las Historias de Usuario (HU) en base a los requisitos que el *middleware* debe cubrir. Estas HU fueron agrupadas en épicas de usuario tal y como se aprecia en la Tabla 2.4, que incluye el nombre de la épica junto a una descripción y las HU correspondientes.

Tabla 2.4: Épicas del proyecto.

N°	Épica	Descripción	Historias de Usuario
1	Traducción de consultas	Facilita el análisis y traducción de sentencias SQL a su equivalente en lenguaje Cypher	HU01, HU02
2	Conexion y ejecución de consultas	Permite al usuario conectarse a una base de datos de grafos específica y ejecutar consultas traducidas de forma directa.	HU04, HU05
3	Gestión de utilidades y retroalimentación	Permite al usuario el manejo de consultas anteriores y resultados según sea su necesidad, además de proporcionar retroalimentación que le brinde una mejor experiencia de uso.	HU03, HU06 y HU07

A continuación, la Tabla 2.5 presenta a través de la HU01, a manera de ejemplo, el formato elegido para las historias de usuario que se encuentran dentro del Anexo I. Este formato contempla campos importantes como la prioridad, estimación, historia de usuario como tal o los criterios de aceptación.

Tabla 2.5: Historia de Usuario 01

ID: HU01	**Usuario:** Desarrollador
Título: Traducción de consultas básicas	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Estimación: 8	**Iteración asignada:** 1
Historia de Usuario:	
Como desarrollador, **Quiero** traducir sentencias SELECT, CREATE, UPDATE o DELETE simples a lenguaje Cypher, **Para** interactuar con bases de datos orientadas a grafos sin tener que aprender un nuevo lenguaje de consulta.	
Criterios de aceptación:	
▪ El usuario debe agregar una sentencia SQL válida para ser traducida. ▪ Las sentencias soporta cláusulas, operadores y funciones básicas. ▪ Al ingresar una sentencia válida, el sistema devolverá su equivalente en lenguaje Cypher. ▪ Las consultas traducidas serán agregadas al historial de consultas.	
Observaciones:	
▪ Verificar que la consulta a ser traducida no contenga errores de sintaxis. ▪ La traducción a Cypher aparecerá junto a la sentencia SQL correspondiente.	

Product Backlog

Para la construcción del Product Backlog, cada una de las Historias de Usuario contempladas serán desglosadas en tareas que nos permitirá entregar valor un incremento de forma rápida y continua, tal y como lo indica el marco de trabajo Scrum. Para ello, se utilizará la técnica **Planning Poker** que ayudará a evaluar la prioridad y complejidad que conlleva cada tarea.

En la Tabla 2.6 y en la Tabla 2.7 se muestran las escalas de valoración de prioridad y

complejidad, establecidas mediante la serie de Fibonacci.

Tabla 2.6: Valoración de prioridad.

Valor	Prioridad
1	Baja
2	Media
3	Alta

Tabla 2.7: Valoración de complejidad.

Valor	Complejidad
1	Muy baja
2	Baja
3	Media
5	Alta
8	Muy alta

Con los criterios claros y en colaboración con el Scrum Team, se construyó el Product Backlog que se muestra en la Tabla 2.8. Aquí se indica la tarea a realizar, su complejidad y prioridad, así como la historia de usuario a la cual corresponde.

Tabla 2.8: Product backlog del proyecto.

HU	Código de tarea	Tarea	Complejidad	Prioridad
HU01	T1.1	Desarrollar la lógica de parseo para identificar el tipo de sentencia SQL y mapeo para Cypher.	8	3
	T1.2	Implementar un endpoint para la traducción de SQL a Cypher.	2	3
	T1.3	Diseñar la pantalla para traducción.	3	3
	T1.4	Desarrollar de la vista para traducción.	2	3

Continúa en la siguiente página...

HU	Código de tarea	Tarea	Complejidad	Prioridad
HU02	T2.1	Desarrollar la lógica para detección de JOINs y claves foráneas.	8	2
	T2.2	Implementar el mapeo de claves foráneas a relaciones de grafos.	5	2
	T2.3	Ajustar la visualización de consultas que incluyan JOINs.	1	1
HU03	T3.1	Diseñar el componente para visualización de errores.	1	2
	T3.2	Desarrollar el componente para visualización de errores.	1	2
	T3.3	Implementar el detalle de error y sugerencia de corrección.	3	2
HU04	T4.1	Desarrollar la lógica para conexión a una base de datos Neo4j.	5	3
	T4.2	Implementar un endpoint para conectarse a la base de datos de grafos.	3	3
	T4.3	Diseñar el formulario de conexión.	2	2
	T4.4	Desarrollar el formulario de conexión.	2	3
HU05	T5.1	Implementar la lógica para la ejecución de la traducción en la base de datos de grafos.	5	3
	T5.2	Implementar un endpoint para la ejecución de la traducción obtenida en la base de datos de Neo4j.	3	3
	T5.3	Diseñar los resultados de la ejecución.	1	2

Continúa en la siguiente página...

HU	Código de tarea	Tarea	Complejidad	Prioridad
	T5.4	Desarrollar del área de resultados con opciones de visualización en formato tabular y JSON.	3	3
HU06	T6.1	Implementar la lógica para el almacenamiento de consultas.	2	1
	T6.2	Desarrollar la lógica para rellenar los cuadros de texto de traducción al seleccionar una consulta guardada.	1	1
	T6.3	Diseñar el historial de consultas.	2	2
	T6.4	Desarrollar la vista del historial de consultas.	2	3
HU07	T7.1	Implementar la transformación de JSON a CSV.	3	1
	T7.2	Desarrollar la función de descarga de resultados.	1	1

Planificación de Sprints

Con la finalidad de cumplir con el Product Backlog establecido, se decidió dividir el trabajo en 5 sprints, incluyendo un sprint 0 dedicado a la configuración del entorno de desarrollo, el diseño de arquitectura del sistema y las interfaces de usuario respectivas.

En la Tabla 2.9, se detalla la planificación de cada sprint, donde se indica la duración en semanas, los objetivos a cumplir y las tareas asignadas a cada uno de ellos.

Tabla 2.9: Planificación de Sprints.

Sprint	Duración	Objetivos	Tareas
Sprint 0	2 semanas	▪ Configurar el entorno de trabajo. ▪ Diseñar la arquitectura del middleware. ▪ Prototipar las interfaces de usuario del sistema.	T1.3, T3.1, T4.3, T5.3, T6.3
Sprint 1	3 semanas	▪ Implementar la traducción de consultas SQL simples.	T1.1, T1.2, T1.4
Sprint 2	3 semanas	▪ Implementar la traducción de consultas SQL que incluyan relaciones.	T2.1, T2.2, T2.3
Sprint 3	3 semanas	▪ Implementar la retroalimentación de errores y sugerencias de corrección. ▪ Implementar la conexión a una base de datos de grafos Neo4j.	T3.2, T3.3, T4.1, T4.2, T4.4
Sprint 4	3 semanas	▪ Implementar la ejecución de consultas traducidas en una base de datos Neo4j. ▪ Implementar el historial de consultas.	T5.1, T5.2, T5.4, T6.1, T6.2, T6.4
Sprint 5	2 semanas	▪ Implementar la exportación de resultados de ejecución.	T7.1, T7.2

2.2.3. Sprint 0

Sprint 0 Planning

Dentro del sprint 0 se estableció la arquitectura del sistema y el diseño de las interfaces de usuario así también, se preparó el entorno de trabajo, el cual incluye la creación de los repositorios y la estructura del código tanto para el backend como para el frontend, de acuerdo al stack tecnológico elegido.

Sprint 0 Backlog

Este backlog incluye tareas relacionadas al diseño del middleware que nos ayudará como guía y base para los próximos sprints. La Tabla 2.10 detalla las tareas del Sprint 0.

Tabla 2.10: Sprint 0 Backlog.

Código	Tarea
T1.3	Diseñar la pantalla para traducción.
T3.1	Diseñar el componente para visualización de errores.
T4.3	Diseñar el formulario de conexión.
T5.3	Diseñar los resultados de la ejecución.
T6.3	Diseñar el historial de consultas.

Ejecución del Sprint 0

Durante la ejecución de este Sprint, el equipo se centró en establecer los cimientos del proyecto. Las actividades principales se dividieron en el diseño de la experiencia de usuario y la configuración técnica inicial.

En primer lugar, se abordó el diseño de las interfaces de usuario, asegurando que la interacción con el *middleware* fuera intuitiva y eficiente. Se diseñaron las pantallas clave para la traducción de sentencias (T1.3), la visualización de errores (T3.1), el formulario de conexión a la base de datos (T4.3), la presentación de resultados de ejecución (T5.3) y el historial de consultas (T6.3). Estos diseños sirvieron como guía visual para el desarrollo del frontend en los sprints subsiguientes.

Paralelamente, se definió la arquitectura del sistema y se configuró el entorno de desarrollo. Esto incluyó la creación de los repositorios de código, la configuración de las herra-

mientas de control de versiones y la preparación de la estructura base tanto para el *backend* como para el *frontend*, respetando el stack tecnológico seleccionado previamente.

Adicionalmente, se estableció el entorno de pruebas para las futuras APIs. Se creó una colección compartida en **Postman**, definiendo las variables de entorno globales (como la URL base del servidor de desarrollo) y la estructura de carpetas para organizar las peticiones por módulos, lo que facilitaría las pruebas de integración en los siguientes sprints.

Sprint 0 Review

Al finalizar el Sprint 0, se realizó la revisión con los interesados para validar los artefactos generados. Se presentaron los prototipos de alta fidelidad de las interfaces de usuario, obteniendo retroalimentación positiva sobre la disposición de los elementos y el flujo de navegación propuesto. Asimismo, se verificó la correcta configuración del entorno de desarrollo y la arquitectura base, confirmando que el equipo contaba con todas las herramientas necesarias para iniciar la codificación de las funcionalidades en el siguiente sprint.

Sprint 0 Retrospective

La retrospectiva del Sprint 0 permitió al equipo reflexionar sobre el inicio del proyecto.

- **¿Qué salió bien?**
  La definición temprana de la arquitectura y el stack tecnológico permitió configurar el entorno sin contratiempos mayores. La comunicación fluida durante la fase de diseño facilitó la creación de prototipos que cumplían con las expectativas de los interesados.
- **¿Qué se puede mejorar?**
  Se identificó que la estimación de tiempo para algunas tareas de diseño fue ajustada, por lo que se acordó refinar los criterios de estimación para futuras tareas creativas. Además, se propuso documentar con mayor detalle las decisiones arquitectónicas a medida que se toman.

2.3. Fase de Desarrollo

En esta fase se detalla todo el desarrollo del *middleware* para traducir sentencias SQL a sentencias GQL y ejecutarlas en una base de datos orientada a grafos utilizando el marco

de trabajo Scrum. Esta sección incluye todas las actividades desarrolladas y el incremento entregado al final de cada sprint.

2.3.1. Sprint 1

Sprint 1 Planning

El objetivo principal del Sprint 1 fue implementar la funcionalidad principal del *middleware*: la traducción básica de sentencias SQL a Cypher. Durante la planificación, se seleccionó la historia de usuario relacionada con el análisis sintáctico y la creación de la interfaz principal de traducción, priorizando la capacidad de procesar consultas simples antes de abordar estructuras más complejas.

Sprint 1 Backlog

La Tabla 2.11 detalla las tareas del Sprint 1.

Tabla 2.11: Sprint 1 Backlog.

Código	Tarea
T1.1	Desarrollar la lógica de parseo para identificar el tipo de sentencia SQL y mapeo para Cypher.
T1.2	Implementar un endpoint para la traducción de SQL a Cypher.
T1.4	Desarrollar la vista para traducción.

Ejecución del Sprint 1

La ejecución de este sprint se centró en el desarrollo del motor de traducción. Se implementó la lógica de parseo (T1.1) necesaria para identificar los componentes de una sentencia SQL básica (SELECT, FROM, WHERE) y mapearlos a sus equivalentes en Cypher (MATCH, RETURN, WHERE).

Simultáneamente, se desarrolló el endpoint en el backend (T1.2) que expone esta lógica de traducción. Se definió la ruta `POST /api/translate`, la cual recibe en el cuerpo de la petición un objeto JSON con la sentencia SQL y retorna la sentencia Cypher generada. En el frontend, se construyó la vista de traducción (T1.4), integrando el diseño aprobado en

el Sprint 0. Esta interfaz consume el servicio mencionado, permitiendo al usuario ingresar código SQL y visualizar la traducción resultante en tiempo real.

Pruebas con Postman

Para validar la funcionalidad del endpoint de traducción, se realizaron pruebas exhaustivas utilizando Postman. Se configuraron casos de prueba para verificar:

- **Traducción exitosa:** Envío de sentencias `SELECT` simples y verificación de que el código de estado HTTP sea 200 OK y el cuerpo de la respuesta contenga el Cypher esperado.
- **Manejo de errores:** Envío de cuerpos de petición vacíos o con formato JSON inválido para asegurar que la API responda con códigos 400 Bad Request y mensajes de error descriptivos.

Sprint 1 Review

En la revisión del sprint, se demostró la funcionalidad de traducción con sentencias SQL simples. Se verificó que el sistema recibiera una consulta SQL válida, la procesara correctamente en el backend y devolviera la sentencia Cypher correspondiente en la interfaz de usuario. Los interesados validaron que la sintaxis generada fuera correcta para una base de datos Neo4j estándar.

Sprint 1 Retrospective

- **¿Qué salió bien?**
  La integración entre el frontend y el backend fue fluida gracias a la definición clara de los contratos de la API. El uso de bibliotecas de parseo facilitó la identificación de tokens SQL.
- **¿Qué se puede mejorar?**
  Se subestimó la complejidad de manejar ciertas variaciones sintácticas de SQL. Para el próximo sprint, se acordó realizar un análisis más detallado de los casos borde antes de comenzar la implementación.

2.3.2. Sprint 2

Sprint 2 Planning

El Sprint 2 tuvo como meta extender la capacidad de traducción para soportar consultas relacionales complejas, específicamente aquellas que involucran uniones de tablas (JOINs). La planificación se enfocó en cómo interpretar las claves foráneas del modelo relacional y transformarlas en relaciones semánticas dentro del modelo de grafos.

Sprint 2 Backlog

La Tabla 2.12 detalla las tareas del Sprint 2.

Tabla 2.12: Sprint 2 Backlog.

Código	Tarea
T2.1	Desarrollar la lógica para detección de JOINs y claves foráneas.
T2.2	Implementar el mapeo de claves foráneas a relaciones de grafos.
T2.3	Ajustar la visualización de consultas que incluyan JOINs.

Ejecución del Sprint 2

Durante este periodo, el trabajo técnico fue de alta complejidad lógica. Se desarrolló el algoritmo para la detección de cláusulas JOIN y claves foráneas dentro de las sentencias SQL (T2.1). Este componente es crucial para entender cómo se relacionan las entidades en el esquema original.

Posteriormente, se actualizó el endpoint de traducción (`POST /api/translate`) para soportar estas nuevas estructuras. Se implementó el mapeo de claves foráneas a relaciones de grafos en Cypher (T2.2), traduciendo la estructura `TABLA_A JOIN TABLA_B ON A.id = B.a_id` a un patrón de grafo `(a:TablaA)-[:RELACION]->(b:TablaB)`. Finalmente, se ajustó la visualización en el frontend (T2.3) para manejar y presentar adecuadamente estas consultas más extensas y complejas.

Pruebas con Postman

Las pruebas de integración se enfocaron en validar la capacidad del endpoint para procesar relaciones:

- **Joins Implicitos y Explícitos:** Se enviaron peticiones con sentencias SQL conteniendo `INNER JOIN` y `LEFT JOIN`. Se verificó en la respuesta JSON que la cadena Cypher resultante incluyera la sintaxis de relaciones (flechas y etiquetas) correcta.
- **Validación de Sintaxis:** Se probaron casos con sintaxis de JOIN incompleta para confirmar que el parser devolviera los errores de sintaxis adecuados antes de intentar la traducción.

Sprint 2 Review

La revisión consistió en probar el sistema con consultas SQL que incluían 'INNER JOIN' y 'LEFT JOIN'. Se demostró cómo el *middleware* identificaba correctamente las tablas involucradas y generaba la relación explícita en Cypher. La precisión en la traducción de las relaciones fue el criterio de aceptación principal validado.

Sprint 2 Retrospective

- **¿Qué salió bien?**
  El algoritmo de detección de relaciones demostró ser robusto para los casos de uso estándar. La visualización de las consultas complejas se mantuvo limpia y legible.
- **¿Qué se puede mejorar?**
  La lógica para nombrar las relaciones en el grafo generó debates sobre la mejor convención a seguir. Se decidió estandarizar una nomenclatura para las relaciones generadas automáticamente para evitar ambigüedades en el futuro.

2.3.3. Sprint 3

Sprint 3 Planning

El Sprint 3 se dedicó a mejorar la robustez del sistema mediante la gestión de errores y a establecer la conectividad real con la base de datos de destino. El objetivo fue permitir que la aplicación no solo tradujera, sino que también se comunicara con una instancia de Neo4j y manejara adecuadamente las excepciones.

Sprint 3 Backlog

La Tabla 2.13 detalla las tareas del Sprint 3.

Tabla 2.13: Sprint 3 Backlog.

Código	Tarea
T3.2	Desarrollar el componente para visualización de errores.
T3.3	Implementar el detalle de error y sugerencia de corrección.
T4.1	Desarrollar la lógica para conexión a una base de datos Neo4j.
T4.2	Implementar el detalle de error y sugerencia de corrección.
T4.4	Desarrollar el formulario de conexión.

Ejecución del Sprint 3

En este sprint se desarrollaron dos frentes importantes. Por un lado, se implementó el módulo de gestión de errores. Se creó el componente visual para alertas (T3.2) y se programó la lógica para ofrecer detalles específicos sobre errores de sintaxis o traducción, incluyendo sugerencias de corrección para el usuario (T3.3).

Por otro lado, se abordó la conectividad. Se desarrolló el endpoint `POST /api/connection/test` (T4.1), diseñado para validar las credenciales de acceso a una base de datos Neo4j externa. Este servicio recibe la URI, usuario y contraseña, intenta establecer una sesión con el driver oficial y retorna el estado de la conexión. En el frontend, se integró este servicio en el formulario de conexión (T4.4), permitiendo al usuario recibir feedback inmediato sobre el estado de su conexión.

Pruebas con Postman

Se realizaron pruebas específicas para el nuevo endpoint de conectividad:

- **Conexión Exitosa:** Se enviaron credenciales válidas de una instancia activa de Neo4j, verificando una respuesta 200 OK y un mensaje de confirmación.
- **Fallos de Autenticación:** Se probaron credenciales incorrectas para asegurar que el sistema capturara la excepción del driver y devolviera un código 401 Unauthorized con un mensaje claro para el usuario.

- **Errores de Red:** Se simularon escenarios con URIs inalcanzables para validar el manejo de tiempos de espera (timeouts).

Sprint 3 Review

La demostración incluyó dos escenarios: uno fallido, donde se introdujo sintaxis SQL inválida para mostrar las nuevas alertas y sugerencias de corrección; y uno exitoso, donde se configuró la conexión a una instancia local de Neo4j, verificando que el sistema pudiera autenticarse correctamente.

Sprint 3 Retrospective

- **¿Qué salió bien?**
  La implementación de sugerencias de corrección aportó un gran valor a la usabilidad del sistema. El uso del driver oficial de Neo4j simplificó la gestión de la conexión.
- **¿Qué se puede mejorar?**
  La gestión de estados de conexión en el frontend resultó más compleja de lo anticipado, requiriendo refactorización para asegurar que la sesión se mantuviera activa o se cerrara adecuadamente según la interacción del usuario.

2.3.4. Sprint 4

Sprint 4 Planning

Con la traducción y la conexión resueltas, el Sprint 4 se enfocó en cerrar el ciclo funcional: ejecutar las consultas traducidas y gestionar el historial de uso. El objetivo fue permitir al usuario ver los resultados reales de sus consultas en la base de datos de grafos y acceder a sus operaciones previas.

Sprint 4 Backlog

La Tabla 2.14 detalla las tareas del Sprint 4.

Tabla 2.14: Sprint 4 Backlog.

Código	Tarea
T5.1	Implementar la lógica para la ejecución de la traducción en la base de datos de grafos.
T5.2	Implementar un endpoint para la ejecución de la traducción obtenida en la base de datos de Neo4j.
T5.4	Desarrollar del área de resultados con opciones de visualización en formato tabular y JSON.
T6.1	Implementar la lógica para el almacenamiento de consultas.
T6.2	Desarrollar la lógica para rellenar los cuadros de texto de traducción al seleccionar una consulta guardada.
T6.4	Desarrollar la vista del historial de consultas.

Ejecución del Sprint 4

Este sprint fue intensivo en integración. Se implementó el endpoint `POST /api/execute` (T5.2), el cual orquesta la ejecución de las consultas. Este servicio recibe la sentencia Cypher traducida y los parámetros de conexión, envía la instrucción a la base de datos Neo4j y procesa los registros retornados para devolverlos en una estructura JSON estandarizada.

Para visualizar estos datos, se desarrolló un área de resultados versátil que permite alternar entre una vista tabular y una vista en formato JSON (T5.4), facilitando la inspección de la respuesta. Adicionalmente, se construyó el módulo de historial. Se implementó el almacenamiento local de las consultas ejecutadas (T6.1) y se desarrolló la vista del historial (T6.4). Una funcionalidad clave añadida fue la capacidad de reactivar una consulta antigua: al seleccionarla del historial, los cuadros de texto de traducción se rellenan automáticamente con la información guardada (T6.2), agilizando el flujo de trabajo del usuario.

Pruebas con Postman

Las pruebas se centraron en la ejecución real de consultas contra la base de datos:

- **Ejecución de Consultas de Lectura:** Se enviaron sentencias `MATCH` válidas a través del endpoint de ejecución, verificando que la respuesta contuviera la estructura de nodos y relaciones esperada en formato JSON.

- **Manejo de Respuestas Vacías:** Se validó el comportamiento del endpoint cuando la consulta no retorna resultados, asegurando que se devuelva una lista vacía y no un error.
- **Integridad de Datos:** Se compararon los resultados obtenidos en Postman con los datos visualizados directamente en Neo4j Browser para garantizar la fidelidad de la información.

Sprint 4 Review

En la revisión se ejecutó un flujo completo: ingresar SQL, traducir, conectar a la BD, ejecutar la traducción y visualizar los nodos retornados. También se navegó por el historial, recuperando consultas de sesiones anteriores. La funcionalidad de ejecución real fue el hito más celebrado por los interesados.

Sprint 4 Retrospective

- **¿Qué salió bien?**
  La visualización dual (Tabla/JSON) fue muy bien recibida. El mecanismo de persistencia del historial funcionó correctamente sin afectar el rendimiento.
- **¿Qué se puede mejorar?**
  El volumen de tareas en este sprint fue alto. Aunque se completaron, la carga de trabajo fue considerable. Se aprendió la importancia de no sobrecargar los sprints finales con funcionalidades críticas de integración.

2.3.5. Sprint 5

Sprint 5 Planning

El último sprint, el Sprint 5, se destinó a funcionalidades de valor añadido y cierre del desarrollo. El objetivo principal fue dotar al sistema de capacidades de exportación de datos, permitiendo a los usuarios extraer la información obtenida para su uso en otras herramientas.

Sprint 5 Backlog

La Tabla 2.15 detalla las tareas del Sprint 5.

Tabla 2.15: Sprint 5 Backlog.

Código	Tarea
T7.1	Implementar la transformación de JSON a CSV.
T7.2	Desarrollar la función de descarga de resultados.

Ejecución del Sprint 5

Las tareas de este sprint se centraron en la manipulación de datos para exportación. Se desarrolló un servicio de transformación capaz de convertir la estructura jerárquica del JSON (resultado de Neo4j) a un formato plano CSV (T7.1). Aunque esta lógica se ejecuta principalmente del lado del cliente para optimizar el rendimiento, se implementaron validaciones para asegurar la integridad de los datos antes de la conversión.

Complementando esto, se implementó la función de descarga en el frontend (T7.2), integrando un botón que desencadena la conversión y genera el archivo descargable para el usuario. Además, se aprovechó este sprint para realizar pruebas finales de integración y pulir detalles estéticos de la interfaz antes de la entrega final.

Pruebas de Integración

Dado que la exportación depende de la correcta estructura de los datos recibidos, se realizaron pruebas de flujo completo:

- **Validación de Estructura JSON:** Se utilizó Postman para verificar que el endpoint de ejecución (`/api/execute`) retornara siempre un JSON válido y bien formado, requisito indispensable para el parser de CSV.
- **Pruebas de Descarga:** Se verificó manualmente que los archivos generados pudieran abrirse correctamente en software de hojas de cálculo y que los caracteres especiales (tildes, ñ) se codificaran correctamente.

Sprint 5 Review

La revisión final sirvió para presentar el producto completo. Se demostró la nueva funcionalidad de exportación descargando los resultados de una consulta compleja y abriéndolos en una hoja de cálculo externa. El sistema fue validado en su totalidad, cumpliendo con los requisitos de traducción, ejecución y gestión de datos definidos al inicio del proyecto.

Sprint 5 Retrospective

- **¿Qué salió bien?**

  El sprint fue menos cargado, lo que permitió dedicar tiempo a la estabilización del código y corrección de pequeños bugs visuales. La funcionalidad de exportación funcionó a la primera gracias a las pruebas unitarias previas.

- **¿Qué se puede mejorar?**

  Mirando hacia atrás en todo el proyecto, se podría haber integrado la funcionalidad de exportación antes, ya que resultó ser muy útil para las pruebas de los propios desarrolladores.

Capítulo 3

RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

3.1. Resultados

3.2. Resultados de las Pruebas

Para validar la efectividad y la calidad del *middleware* desarrollado, se llevó a cabo una fase exhaustiva de pruebas. Esta fase se dividió en dos categorías principales: pruebas funcionales, orientadas a verificar el correcto funcionamiento de la traducción y ejecución de sentencias; y pruebas de usabilidad, enfocadas en evaluar la experiencia del usuario al interactuar con la interfaz gráfica. A continuación, se detallan la metodología empleada y los resultados obtenidos en cada una de estas evaluaciones.

3.2.1. Pruebas Funcionales

El objetivo primordial de las pruebas funcionales fue asegurar que el sistema cumpla con los requisitos técnicos definidos en las historias de usuario. Se diseñó un plan de pruebas que abarcó desde la traducción de sentencias SQL simples hasta consultas complejas con múltiples uniones (*JOINs*), así como la gestión de errores y la conectividad con la base de datos Neo4j.

Metodología y Participantes

Para estas pruebas, se contó con la colaboración de un grupo de 5 participantes con perfil técnico, específicamente desarrolladores y estudiantes de ingeniería de software con conocimientos previos en SQL pero con experiencia limitada en bases de datos orientadas a grafos. Se les proporcionó un conjunto de casos de prueba que debían ejecutar utilizando el *middleware*.

Los escenarios evaluados incluyeron:

- Conexión exitosa y fallida a la base de datos.
- Traducción de sentencias `SELECT` con filtros `WHERE`.
- Traducción de sentencias con `INNER JOIN` y `LEFT JOIN`.
- Ejecución de las consultas traducidas y visualización de resultados.
- Identificación de errores de sintaxis SQL.

Resultados Obtenidos

Los resultados de las pruebas funcionales fueron altamente satisfactorios. El 100 % de las sentencias SQL válidas fueron traducidas correctamente a su equivalente en Cypher. En cuanto a la ejecución, el sistema logró recuperar los datos de la base de datos de grafos y presentarlos en los formatos tabular y JSON sin inconsistencias.

Se detectaron incidencias menores durante las primeras iteraciones, relacionadas principalmente con la sensibilidad a mayúsculas y minúsculas en ciertos identificadores, las cuales fueron corregidas inmediatamente. La Tabla **??** resume la tasa de éxito por categoría de prueba.

3.2.2. Pruebas de Usabilidad

Complementando la validación técnica, se realizaron pruebas de usabilidad para medir el grado de satisfacción del usuario y la facilidad de aprendizaje del sistema. Se utilizó la escala de usabilidad del sistema (SUS - *System Usability Scale*) como herramienta de medición estándar.

Metodología y Participantes

El grupo de prueba para la usabilidad estuvo conformado por 5 participantes distintos a los de las pruebas funcionales, seleccionados para representar a usuarios finales que podrían beneficiarse de la herramienta para la migración de consultas. Se les pidió realizar una tarea completa: configurar la conexión, traducir una consulta compleja, ejecutarla y exportar los resultados a CSV.

Al finalizar la interacción, cada participante completó el cuestionario SUS, que consta de 10 preguntas con una escala de Likert de 1 a 5. Además, se recogieron comentarios cualitativos sobre su experiencia.

Análisis de Resultados

El puntaje promedio obtenido en la escala SUS fue de 85 sobre 100, lo que sitúa al *middleware* en la categoría de .Excelente.en términos de usabilidad. Los participantes destacaron la claridad de la interfaz y la utilidad de las sugerencias de corrección de errores.

Entre los comentarios positivos, se resaltó la funcionalidad de autocompletado del historial y la visualización dual de resultados. Como oportunidad de mejora, dos participantes sugirieron aumentar el tamaño de la fuente en el editor de código para mejorar la legibilidad en pantallas pequeñas.

3.3. Conclusiones

El desarrollo del presente trabajo de integración curricular ha permitido alcanzar satisfactoriamente los objetivos planteados, resultando en la implementación de un *middleware* funcional capaz de traducir y ejecutar sentencias SQL en una base de datos orientada a grafos. A partir de los resultados obtenidos y el proceso de desarrollo, se derivan las siguientes conclusiones:

- Se logró construir un motor de traducción robusto que interpreta correctamente las sentencias SQL estándar, incluyendo cláusulas complejas como `INNER JOIN` y `LEFT JOIN`. La validación funcional demostró que la lógica de mapeo implementada traduce con precisión las relaciones de claves foráneas del modelo relacional a aristas semánticas en el modelo de grafos, facilitando la interoperabilidad entre ambos paradigmas.

- La aplicación de la metodología Scrum fue fundamental para el éxito del proyecto. La división del trabajo en sprints permitió una entrega incremental de valor, facilitando la detección temprana de errores y la adaptación ágil a los requisitos emergentes, como la necesidad de un módulo de exportación de datos, que no estaba contemplado inicialmente con tal nivel de detalle.

- Las pruebas de usabilidad, respaldadas por un puntaje de 85/100 en la escala SUS, confirman que la herramienta reduce significativamente la barrera de entrada para usuarios familiarizados con SQL que desean interactuar con bases de datos Neo4j. La interfaz intuitiva y las funcionalidades de asistencia, como el historial y las sugerencias de error, mejoran sustancialmente la experiencia del usuario en comparación con el uso directo de consolas de comandos.

- La arquitectura diseñada, basada en una separación clara entre el *frontend* y el *backend* a través de servicios REST, demostró ser escalable y mantenible. Esto permitió integrar librerías de terceros para el parseo y la conexión a bases de datos sin comprometer el rendimiento general del sistema, como se evidenció en las pruebas de ejecución.

3.4. Recomendaciones

Basado en la experiencia adquirida durante el desarrollo de este proyecto y los resultados obtenidos, se formulan las siguientes recomendaciones para trabajos futuros y la evolución de la herramienta:

- **Expansión de la Gramática SQL:** Se recomienda ampliar el motor de traducción para soportar características más avanzadas del lenguaje SQL, tales como subconsultas anidadas, funciones de agregación (`GROUP BY`, `HAVING`) y operaciones de modificación de datos (`INSERT`, `UPDATE`, `DELETE`). Esto permitiría una gestión completa del ciclo de vida de los datos desde la interfaz del *middleware*.

- **Visualización Gráfica de Resultados:** Aunque la visualización en formato JSON y tabular es funcional, se sugiere integrar una librería de visualización de grafos (como D3.js o Vis.js) en el *frontend*. Esto permitiría a los usuarios ver los nodos y relaciones de manera gráfica e interactiva, aprovechando al máximo la naturaleza visual de las bases de datos orientadas a grafos.

- **Soporte Multilenguaje y Multibase:** Para aumentar la versatilidad de la herramienta, sería beneficioso investigar la implementación de adaptadores para otros lenguajes de consulta de grafos, como Gremlin, y otras bases de datos más allá de Neo4j, como Amazon Neptune o JanusGraph.
- **Optimización de Rendimiento:** Se aconseja realizar pruebas de estrés con volúmenes masivos de datos para identificar posibles cuellos de botella en el proceso de traducción y renderizado de resultados. Implementar mecanismos de paginación en el servidor y carga diferida (*lazy loading*) en el cliente mejoraría la respuesta del sistema ante consultas que retornan miles de nodos.

Capítulo 4

REFERENCIAS BIBLIOGRÁFICAS

[1] J. Dai, «SQL to NoSQL: What to do and How,» en *IOP Conference Series: Earth and Environmental Science*, IOP Publishing, vol. 234, 2019, pág. 012 080.

[2] B. Namdeo y U. Suman, «A Middleware Model for SQL to NoSQL Query Translation,» *Indian Journal of Science and Technology*, vol. 15, n.º 16, págs. 718-728, 2022.

[3] M. Đukić, O. Pantelić, A. Pajić Simović, S. Krstović y O. Jejić, «A systematic approach for converting relational to graph databases,» 2024.

[4] R. C. Martin, *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall Press, 2017.

[5] R. S. Pressman, *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.

[6] A. Stellman y J. Greene, *Learning agile: Understanding scrum, XP, lean, and kanban*. .º'Reilly Media, Inc.", 2014.

[7] K. Schwaber y J. Sutherland, «The scrum guide,» *Scrum Alliance*, vol. 21, n.º 1, págs. 1-38, 2011.

[8] P. Ackermann, *Full Stack Web Development: The Comprehensive Guide*. Rheinwerk Publishing Incorporated, 2023.

[9] J. Goldberg, *Learning TypeScript*. .º'Reilly Media, Inc.", 2022.

[10] A. Banks y E. Porcello, *Learning React: modern patterns for developing React apps*. O'Reilly Media, 2020.

[11] Vercel, *Next.js docs*. dirección: `https://nextjs.org/docs`

[12] K. Bhat, *Ultimate Tailwind CSS Handbook: Build sleek and modern websites with immersive UIs using Tailwind CSS*. Orange Education Pvt Limited, 2023.

[13] M. Lutz, *Learning python: Powerful object-oriented programming*. .ºReilly Media, Inc.", 2013.

[14] B. Lubanovic, *FastAPI*. .ºReilly Media, Inc.", 2023.

[15] H. Garcia-Molina, *Database systems: the complete book*. Pearson Education India, 2008.

[16] A. Silberschatz, H. F. Korth y S. Sudarshan, *Database system concepts*. McGraw-Hill Education, 2011.

[17] C. M. Ricardo, *Bases de datos*. McGraw Hill Educación, 2009.

[18] D. Sullivan, *NoSQL for mere mortals*. Addison-Wesley Professional, 2015.

[19] T. Hills, *NoSQL and SQL data modeling: bringing together data, semantics, and software*. Technics Publications, LLC, 2016.

[20] P. J. Sadalage y M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[21] M. Lal, *Neo4j graph data modeling*. Packt Publishing Ltd, 2015.

[22] A. Vukotic, N. Watt, T. Abedrabbo, D. Fox y J. Partner, *Neo4j in action*. Manning Publications Co., 2014.

[23] R. Van Bruggen, *Learning Neo4j*. Packt Publishing Ltd, 2014.

[24] J. Baton y R. Van Bruggen, *Learning Neo4j 3. x: Effective data modeling, performance tuning and data visualization techniques in Neo4j*. Packt Publishing Ltd, 2017.

[25] F. Staiano, *Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop*. Packt Publishing Ltd, 2022.

[26] S. Chacon y B. Straub, *Pro git*. Springer Nature, 2014.

[27] M. Kennedy y M. Harrison, *Effective PyCharm: Learn the PyCharm IDE with a Hands-on Approach*. Effectivepycharm. com., 2019.

[28] D. Westerveld, *API Testing and Development with Postman*. Packt Publishing, 2021.

[29] I. Sommerville, «Software engineering 9th Edition,» *ISBN-10*, vol. 137035152, pág. 18, 2011.

[30] M. Jain, A. Khanchandani y C. Rodrigues, «Performance Comparison of Graph Database and Relational Database,» *Computer Science Department San Jose State University*, 2023.

[31] M. Singh y K. Kaur, «SQL2Neo: Moving health-care data from relational to graph databases,» en *2015 IEEE International Advance Computing Conference (IACC)*, IEEE, 2015, págs. 721-725.

ANEXO I

Historias de Usuario

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

ANEXO II

Formato de Entrevista

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

ANEXO III

Enlaces

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.