Una herramienta de migración de bases de datos basada en LLM

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Fabián Sola Durán
Tutor/es:
José Ramón Hoyos Barceló
María José Ortín Ibáñez

30 de Julio de 2025

Facultad
Informática
Universidad
Murcia

Una herramienta de migración de bases de datos basada en LLM

LLMigrator-DB: automatización de la migración de bases de datos haciendo uso de abstracciones de LLM

Autor
Fabián Sola Durán

Tutor/es
José Ramón Hoyos Barceló
Departamento de Informática y Sistemas
María José Ortín Ibáñez
Departamento de Informática y Sistemas

Facultad de Informática

Grado en Ingeniería Informática

DIS Departamento de Informática y Sistemas

UNIVERSIDAD DE MURCIA

Murcia, 30 de Julio de 2025

Preámbulo

Este Trabajo Fin de Grado (TFG) surge del interés por explorar el potencial de la Inteligencia Artificial Generativa (GenAI) en tareas complejas del ámbito del desarrollo software, concretamente en la automatización de procesos de migración de bases de datos. La elección del tema responde tanto a su relevancia práctica en entornos empresariales reales como al desafío técnico que plantea integrar modelos de lenguaje de propósito general en flujos de trabajo automatizados.

El objetivo principal de este estudio ha sido el diseño y desarrollo de la herramienta LLMigrator-DB la cual es capaz de asistir el proceso de migración de bases de datos, abarcando la migración del esquema, la migración los datos y la adaptación del código de acceso a datos. Para ello, se han aplicado técnicas de Ingeniería de *Prompts* sobre Modelos de Lenguaje de Gran Escala (LLM). El alcance del proyecto incluye desde la definición conceptual y arquitectónica de la solución hasta su validación práctica mediante casos de estudio significativos en este contexto.

Quiero expresar mi más profundo agradecimiento a los profesores que me han acompañado en esta travesía, Jesús Joaquín García Molina, María José Ortín Ibáñez y José Ramón Hoyos Barceló por sus valiosas orientaciones, por las discusiones técnicas mantenidas a lo largo del desarrollo del trabajo y por su apoyo continuo. Su experiencia y constante disposición han sido fundamentales para la realización de este proyecto.

De forma especial, quiero agradecer a Jesús Joaquín no solo por su acompañamiento académico, sino por su insaciable curiosidad, su hambre de conocimiento y, sobre todo, por contagiarme esa actitud. Su entusiasmo por explorar, comprender y cuestionarlo todo ha sido una influencia determinante en mi forma de enfrentar este trabajo y, en general, mi forma de aprender.

Agradecimientos. Este trabajo es parte del proyecto PID2020-117391GB-I00 financiado por el Ministerio Español de Ciencia, Innovación y Universidades y la Agencia Estatal de Investigación (MICIU/AEI/10.13039/501100011033) y ERDF, EU.

Agradecimientos

En un principio, no imaginaba que sería capaz de completar el grado en los cuatro años. Sin embargo, este logro ha sido posible gracias a la compañía constante, la paciencia inquebrantable y el apoyo generoso de las personas que han estado a mi lado durante todo este recorrido. A todas ellas, les debo mucho más que un agradecimiento formal; les debo la fuerza que me ha sostenido cuando flaqueaban las ganas y el ánimo.

Comenzar agradeciendo a Antonio Pérez Serrano, amigo y compañero de estudio durante este último año. Gracias por acompañarme en esta recta final, por aprender a lidiar conmigo y por compartir tantos momentos de estudio, desahogo y complicidad. Tu presencia ha hecho este camino mucho más llevadero.

Además, también quiero agradecer profundamente a mi padre. Durante estos años ha vivido con unos auriculares puestos, no por entretenimiento, sino como un gesto de generosidad y consideración, permitiéndome estudiar con la tranquilidad y el silencio que necesitaba. Ha soportado mis momentos de frustración, mis cambios de humor, y ha sabido regalarme también instantes de risa inolvidables, como aquella inolvidable anécdota de la *curva del Chirreta*.

A mi hermano, José María, gracias por abrir un camino que he tenido el honor de seguir, compartiendo contigo la profesión de ingeniero informático. Gracias por estar ahí cuando he necesitado ayuda, por escuchar mis quejas cuando alguna asignatura se volvía cuesta arriba, y por comprenderlas con empatía y experiencia. Extiendo este agradecimiento a su mujer, María Dolores, por su cariño y cercanía, y a sus hijos, quienes me recuerdan con su inocencia y alegría lo que de verdad importa.

A Luisa María, mi compañera, mi pareja, y, sobre todo, la persona que ha sostenido mi ánimo en los momentos más difíciles y, sin duda, la mujer de mi vida. Has estado ahí cuando más lo necesitaba, ofreciéndome tu hombro, tu tiempo y tu comprensión sin reservas. Gracias por las innumerables noches en las que dormiste a la luz de mi portátil mientras yo programaba, por todos los planes aplazados o cancelados en favor de este objetivo, y por no soltarme la mano en ningún momento del camino.

Finalmente, a mi madre, Ascensión, a quien la vida me arrebató demasiado pronto, pero cuya presencia me ha acompañado en cada paso durante estos últimos once años. Tus palabras, tus valores y tus consejos —tan vivos en mi memoria— han sido faro y sustento en los momentos de duda. Mamá, tú siempre me decías que lograría todo lo que me propusiera. Hoy, aunque no estés aquí físicamente, he cumplido ese sueño que también era tuyo. Espero que, estés donde estés, te sientas orgullosa.

Nuestra mayor gloria no consiste
en no caer nunca,
sino en levantarnos cada vez que caemos.

Confucio.

Declaración firmada sobre originalidad del trabajo

D./Dña. **Fabián Sola Durán**, con DNI **XXXXXXX**, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TF titulado "**Una herramienta de migración de bases de datos basada en LLM**".

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016, 28-09-2018 y 28-06-2022), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial.

Murcia, a 30 de Julio de 2025

Fdo.: Fabián Sola Durán
Autor del TF

Resumen

En el contexto actual del desarrollo software, la adaptación de los sistemas de persistencia de datos a nuevas tecnologías supone una tarea estratégica para garantizar la evolución y mantenibilidad de las aplicaciones. Esta problemática cobra especial relevancia en arquitecturas basadas en microservicios, donde cada uno de ellos puede hacer uso de distintos modelos de datos (relacional, documental, entre otros) y evolucionar de forma desacoplada. En este contexto, la migración de bases de datos no se limita a la transformación del esquema y adaptación de los datos en la base de datos destino, sino que también exige modificar el código fuente encargado de interactuar con dichos datos. Tradicionalmente, este proceso ha requerido una alta intervención manual, experiencia técnica especializada y validaciones exhaustivas, lo cual incrementaba de forma drástica tanto el coste como la complejidad del proceso, particularmente en escenarios de migración heterogénea.

Con el fin de abordar este problema, este Trabajo Fin de Grado (TFG) presenta el desarrollo de LLMigrator-DB, una herramienta basada en Modelos de Lenguaje Grande (LLM) diseñada para asistir la migración automática de bases de datos y del código fuente de la capa de persistencia. El objetivo principal de la herramienta es reducir la carga operativa del usuario experto, automatizando las tres fases críticas del proceso de migración: conversión del esquema, migración de los datos y adaptación del código fuente de la capa de acceso a dichos datos.

Desde el punto de vista técnico, LLMigrator-DB ha sido desarrollada como una aplicación web siguiendo una arquitectura REST modular compuesta por tres componentes principales: migrador de esquema, migrador de datos y migrador de código. Cada componente interactúa con el LLM a través de prompts que se han diseñado haciendo uso de técnicas avanzadas de Ingeniería de Prompts como aprendizaje *zero-shot*, el uso de mensajes de sistema, el aprendizaje en contexto, el encadenamiento de prompts y la generación de salidas estructuradas en JSON. Un aspecto novedoso de la implementación es el uso de un framework de abstracción de LLM como es Spring AI.

Un requisito esencial de la herramienta es la independencia de la plataforma, en particular, ser independiente de un SGBD o modelo de datos determinado y no estar ligado a un LLM concreto. Lo primero se obtiene a través del diseño de las plantillas de prompt, mientras que lo segundo se consigue utilizando la API unificada OpenRouter.

La validación de LLMigrator-DB se ha llevado a cabo mediante un caso de estudio realista basado en la aplicación distribuida *Eventalia*, un sistema de microservicios con persistencia heterogénea. En dicho estudio se han abordado dos migraciones: una migración homogénea (MySQL a PostgreSQL) para el microservicio de eventos, y otra

migración heterogénea (MongoDB a PostgreSQL) para el microservicio de reservas. En ambos casos, la herramienta ha sido capaz de generar esquemas relacionales completos, pautas de validación manual del esquema, scripts de migración de datos funcionales, procesos de validación de datos automáticos y una reestructuración precisa de las entidades, repositorios y servicios conforme a las convenciones del código origen.

Los resultados evidencian que la herramienta consigue automatizar en torno al 70% del trabajo necesario en una migración. Esto se traduce en una reducción drástica del tiempo requerido por un usuario experto para realizar estas tareas. Este grado de automatización es posible gracias a la combinación de una arquitectura modular con sus responsabilidades bien definidas, el uso efectivo de técnicas de Ingeniería de Prompts para la generación automática mediante LLMs, y la incorporación de estrategias de validación que proporcionan una base de coherencia, precisión y comprobación de los resultados generados.

A pesar de los prometedores resultados, este TFG pone de manifiesto varias consideraciones a tener en cuenta con el uso de LLMs, como su comportamiento no determinista y la necesidad de supervisión del proceso por un experto. En este sentido, se plantean diversas líneas de mejora para futuras versiones de la herramienta orientadas a aumentar la fiabilidad de las respuestas, reducir la necesidad de experiencia por parte del usuario final y ampliar las capacidades de LLMigrator-DB mediante el uso de arquitecturas más escalables.

En definitiva, este TFG ofrece una solución novedosa que demuestra el potencial de los LLMs en tareas complejas como una migración de base de datos. La herramienta LLMigrator-DB supone un paso relevante en la automatización de procesos, especialmente en aquellos entornos en los que la heterogeneidad tecnológica y la evolución continua son constantes.

Extended Abstract

In the contemporary landscape of software development, the strategic management and evolution of data persistence systems have become fundamental pillars to ensure the long-term viability, scalability, and maintainability of applications. This challenge is particularly pronounced in the context of microservice-based architectures, where individual services are granted the autonomy to choose their own data storage technologies. Such flexibility leads to increasingly heterogeneous persistence environments, introducing new levels of complexity when evolving or unifying data systems across services.

Within this setting, database migration has shifted from being an occasional, albeit technically demanding task, to becoming a recurrent necessity. This process, however, remains inherently complex, as it encompasses not only the structural transformation of the database schema and the physical transfer of data, but also the labor-intensive adaptation of the application's source code responsible for interacting with the persistence layer. Traditionally, this endeavor has relied heavily on manual intervention, requiring significant technical expertise, domain knowledge, and extensive validation cycles. Consequently, database migration is an expensive, time-consuming, and risk-laden process especially in heterogeneous contexts, such as moving from a document-based NoSQL database to a relational system.

The recent emergence of Generative Artificial Intelligence (GenAI), powered by Large Language Models (LLMs), offers a paradigm-shifting opportunity to tackle this long-standing challenge by introducing unprecedented levels of automation and intelligence into software engineering workflows. LLMs, with their ability to reason over unstructured information and generate code, have demonstrated their potential in a wide array of tasks, from code generation to test case synthesis and documentation. Their application to database migration represents a novel and largely unexplored frontier.

This Final Degree Project (TFG) proposes a new solution to the enduring problem of database migration: an automated, intelligent system named LLMigrator-DB, which harnesses the power of LLMs to perform and validate complete database migrations. LLMigrator-DB is designed to automate the three fundamental pillars of the process: schema conversion, data migration, and the adaptation of persistence-layer source code. The overarching objective is to reduce the operational burden placed on expert developers, minimize the potential for human error, and accelerate the migration lifecycle while ensuring correctness and consistency across systems.

In contrast to existing migration tools, many of which are tightly coupled to specific cloud providers or lack support for source code adaptation, LLMigrator-DB is built as

a vendor-agnostic and modular solution. It supports migrations between different types of databases (both homogeneous and heterogeneous) and is decoupled from any single LLM provider. To this end, the tool integrates Spring AI, a framework that abstracts the use of LLMs in Java applications, with OpenRouter, a proxy service that exposes a wide range of LLMs from different vendors through an API compatible with OpenAI's, enabling unified access via a single client implementation.

The methodological foundation of LLMigrator-DB rests on the strategic application of prompt engineering. Prompt engineering is a discipline that seeks to guide LLM behavior through carefully crafted instructions. This project explores and applies advanced techniques such as zero-shot learning, in-context learning, prompt chaining, and structured output parsing to extract accurate and reliable outputs from LLMs. These techniques form the backbone of the system's automation strategy and are tailored specifically to the challenges posed by database migration tasks.

To realize its objectives, LLMigrator-DB was conceived as a platform-agnostic, modular web-based tool that adheres to REST principles. Its design is grounded in Clean Architecture, ensuring a strict separation between business logic, communication layers, and infrastructure components. This architectural decision provides several benefits: it enhances the testability and maintainability of the system, facilitates future integration with new technologies or LLM providers, and aligns with best practices for building scalable, loosely coupled systems.

LLMigrator-DB exposes a well-defined API, described using OpenAPI specifications, through which users can initiate migrations. Each migration request is encapsulated in a Data Transfer Object (DTO) that carries the necessary parameters: source and target database technologies, the original schema, relevant code artifacts (when applicable), and any functional or performance requirements. This DTO structure ensures clarity, extensibility, and consistency across migration scenarios.

Internally, LLMigrator-DB is composed of three interdependent core components:

1. *Schema Migrator*: Responsible for transforming the database schema from the source model to the target model. It operates using a sequence of chained prompts that progressively (i) establish context for the LLM, (ii) generate the schema in the appropriate DDL syntax for the target database, and (iii) produce a validation guideline for manual review. This component supports homogeneous and heterogeneous migrations.

2. *Data Migrator*: Leverages the source and target schema information to generate executable scripts (typically in Python) that handle data extraction, transformation, and loading (ETL). The LLM is guided to produce code that preserves referential integrity, respects data types, and avoids duplication or loss during the migration.

3. *Code Migrator*: This component automates the translation of source code related to the persistence layer. It operates in two stages: first, it adapts the domain

model (including Java classes annotated with JPA or Spring Data—and the associated repository interfaces); second, it refactors the service-layer logic to align with the target database paradigm. This process ensures functional continuity of the application's business logic after the migration, while reducing the need for manual intervention.

Each of these components is built upon the disciplined use of prompt engineering techniques, tailored specifically for database engineering tasks. LLMigrator-DB leverages:

- System messages to define the LLM's role as a domain expert in database migration.
- Zero-shot learning to request tasks without prior examples, relying on the model's general training.
- In-context learning to inject relevant information into prompts, such as target DBMS characteristics or specific query requirements.
- Prompt chaining to split complex operations into manageable steps, ensuring higher accuracy and consistency.
- Structured output constraints, where outputs are required in JSON format, reducing hallucinations and enabling programmatic parsing of LLM responses.

This careful orchestration of prompt logic is facilitated by Spring AI, which offers high-level abstractions such as prompt templates and structured output converters. Additionally, LLMigrator-DB maintains full independence from an specific LLM by integrating OpenRouter. This integration enables the tool to interact with a wide variety of LLMs, such as OpenAI's GPT-4o-mini and Google's Gemini 2.0 Flash, without requiring changes to its internal implementation.

Prior to the development of LLMigrator-DB, a comprehensive review of the current landscape of database migration tools was conducted, encompassing both established commercial platforms and emerging AI-assisted solutions. This analysis revealed that while several mature tools exist, most are constrained by vendor lock-in, limited automation capabilities, or a narrow focus that fails to address the full lifecycle of a migration process.

Among the industry leaders, Azure Database Migration Service (DMS) offers robust support for both homogeneous and heterogeneous migrations. However, it still relies heavily on manual configuration and lacks integration with generative AI techniques. In contrast, Google Cloud DMS has begun to incorporate Gemini-powered features that automate schema transformations in heterogeneous scenarios, providing developers with significant time savings. AWS DMS also includes a range of utilities,

such as the Schema Conversion Tool, which help streamline conversion processes. More recently, it has introduced AI-based recommendations to assist in migration tasks. Despite these advancements, all three solutions remain tightly bound to their respective cloud ecosystems and typically do not address the adaptation of application source code at the persistence layer.

A particularly notable entry is Datafold, a third-party platform that introduces LLM-powered SQL translation and validation via feedback loops. While highly innovative, Datafold is primarily focused on data observability and query-level accuracy, rather than delivering end-to-end migration automation that includes schema generation, data transfer, and source code adaptation.

From this analysis, several key limitations in the current state of the art can be identified. First, most migration tools are tightly coupled to specific vendors, which hinders cross-platform portability and adoption. Second, existing solutions tend to overlook the application layer, particularly the transformation of source code that interacts with the persistence system, which is crucial for a complete migration. Third, many platforms still rely on manual supervision or custom scripting, preventing end-to-end automation. Finally, only a few approaches incorporate prompt engineering techniques in a systematic manner to guide LLMs toward producing structured and reliable outputs.

LLMigrator-DB was specifically designed to address these gaps, presenting a novel approach to the database migration problem. It offers full-scope automation across the three fundamental dimensions of migration: schema, data, and application code. Its architecture is LLM-agnostic, thanks to the integration of Spring AI and OpenRouter, which enables the dynamic use of different LLM without requiring changes to the system's internal logic. Additionally, it incorporates a robust prompt engineering framework, carefully designed and validated for each migration stage, including prompt chaining, structured output generation, and system-level guidance. The tool also adopts an open, modular, and RESTful design, allowing for seamless extensibility and integration into real-world distributed environments.

These contributions position LLMigrator-DB not merely as a functional migration utility but as a proof of concept system architecture for embedding LLM into complex software maintenance workflows. Its capacity to interface dynamically with multiple models while maintaining output quality and consistency sets it apart from existing solutions. Furthermore, its ability to automate the migration of source code, an aspect often overlooked in this domain, addresses a pressing need in enterprise systems where persistence logic is frequently embedded within application services.

To evaluate the effectiveness of LLMigrator-DB, a validation was conducted using a realistic production-inspired case study: Eventalia, a distributed event management platform composed of multiple microservices with heterogeneous persistence layers. Specifically, Eventalia includes two key services, eventos and reservas, which respectively rely on MySQL and MongoDB as their underlying databases. The migration objective was to unify the system's persistence by transitioning both services to PostgreSQL, thereby testing LLMigrator-DB in both homogeneous and heterogeneous mi-

gration scenarios.

The *first test case* involved the migration of the eventos microservice from MySQL to PostgreSQL using Gemini-2.0 Flash as the LLM.

The Schema Migrator generated a PostgreSQL compliant DDL script, correctly mapping MySQL specific types such as BIT(1) to BOOLEAN and preserving constraints and index structures. Importantly, indexes were intelligently added to columns used in filtering and ordering clauses based on the functional requirements provided, an optimization not explicitly requested by the user.

The Data Migrator produced a complete Python script for migrating data between the two systems. This script handled connections, data transformation, boolean normalization, and conflict resolution using constructs like `ON CONFLICT DO NOTHING` to avoid duplicates while maintaining referential integrity.

In the code migration phase, LLMigrator-DB demonstrated high precision. JPA entities were reannotated for PostgreSQL compatibility, repository interfaces were regenerated using the appropriate SQL dialects, and custom queries written in JPQL were successfully adapted. For instance, MySQL specific functions like `YEAR(...)` were translated into PostgreSQL equivalents such as `EXTRACT(YEAR FROM ...)`. The resulting codebase required minimal manual editing, and the service retained its original functionality.

The *second test case* focused on migrating the reservas microservice from MongoDB to PostgreSQL, using GPT-4o-mini as the LLM. This scenario posed a number of conceptual and structural challenges, primarily due to the differences between document-based and relational paradigms.

The Schema Migrator correctly identified embedded structures and referenced relationships. It generated a normalized relational schema with appropriate foreign key constraints. For example, the bidirectional one-to-many relationship between eventos and reservas was correctly translated into a relational model by introducing a `FOREIGN KEY` constraint in the reserva table referencing the evento table, with ON DELETE CASCADE semantics to preserve the original deletion behavior defined in MongoDB. This ensures that when an event is deleted, all associated reservations are automatically removed, maintaining referential integrity.

Once the target schema was established, the Data Migrator generated a Python-based ETL script capable of traversing the MongoDB collections, extracting documents, flattening nested structures, and inserting the transformed data into the PostgreSQL tables. Special attention was given to preserving relationships originally defined through `DBRef` fields in MongoDB, which were resolved and re-encoded as foreign key assignments in SQL. Boolean conversions, null handling, and identifier remapping were also performed seamlessly.

After completing the data migration and validating the integrity of the imported records, the Code Migrator proceeded to adapt the original persistence code written with Spring Data MongoDB. The annotated `@Document` classes were transformed into standard JPA `@Entity` classes, and references via `@DBRef` were replaced with `@ManyToOne`

or `@OneToMany` associations, depending on the directionality of each relationship. Repository interfaces were migrated from `MongoRepository` to `JpaRepository`, preserving both query semantics and method signatures. Furthermore, domain-specific queries were automatically rewritten in Java Persistence Query Language (JPQL). The service layer logic was also refactored to align with JPA conventions, incorporating elements such as the EntityManager, cascade operations, and transaction management.

The successful completion of this heterogeneous migration scenario not only validated LLMigrator-DB's ability to perform structural translation but also highlighted its capacity to semantically align application logic with the new data model, thereby enabling continuity in business functionality without the need for extensive manual rework.

In conclusion, this Final Degree Project has demonstrated that LLM can serve as effective engines for automating complex database migration tasks. The tool developed, LLMigrator-DB, successfully integrates schema transformation, data migration, and code adaptation into a cohesive system validated through both homogeneous and heterogeneous scenarios.

The architecture, based on Clean Architecture, Spring AI, and OpenRouter, enables independence from specific LLM providers and supports modular, scalable workflows. Experimental results using the Eventalia case study confirmed that LLMigrator-DB can automate approximately 70% of the overall migration effort, substantially reducing developer workload and the potential for error.

Despite its promising results, LLMigrator-DB still presents certain limitations. The inherent non-determinism of LLMs necessitates expert oversight to validate the correctness and consistency of outputs. Moreover, the cost associated with invoking an LLM, particularly when using chained prompts, can limit the scalability and continuous usage.

Future work aims to address these issues by integrating automated quality metrics to rank and select the best outputs, developing a natural language interface for refining migration scripts, and refactoring the system into an event-driven architecture to improve scalability and modularity.

Overall, LLMigrator-DB serves as a solid proof of concept for the application of generative AI in software evolution. With continued improvements, such tools could become essential for automating maintenance tasks in heterogeneous and dynamic data ecosystems.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

Contexto y motivación

La gestión de los datos es un aspecto clave para que cualquier organización cumpla con éxito sus objetivos, sea cual sea su tamaño. Por ello, los sistemas de gestión de bases de datos (SGBD) han jugado siempre un papel crucial en los sistemas de información de las empresas, sobre todo a partir de la adopción generalizada de los sistemas relacionales (SGBDR) en los años noventa del siglo pasado.

A finales de la primera década de este siglo, la irrupción de las aplicaciones móviles, las redes sociales y el Internet de las Cosas, entre otras innovaciones, trajeron consigo nuevas necesidades y nuevos desafíos para los SGBD, como son satisfacer la escalabilidad horizontal, flexibilidad para el cambio de esquema, y disponibilidad continua [1]. En este escenario, comenzaron a emerger los denominados *NoSQL stores* basados en modelos de datos diferentes al relacional, entre los que destacan los modelos de documentos, columnar, clave-valor, y de grafos [1]. En el ranking de popularidad de bases de datos (DB) *DB-Engines*, los sistemas MongoDB (documentos), Redis (clave-valor), Cassandra (columnar) y Neo4J (grafos) ocupan lugares destacados desde hace años (5, 7, 11 y 20 en la actualidad)[1], aunque los SGBDR continúan dominando el mercado claramente.

Ante las limitaciones de los SGBDR frente a las aplicaciones modernas, las arquitecturas relacionales han evolucionado para atender los nuevos requisitos y mejorar su eficiencia para las nuevas aplicaciones, surgiendo los denominados *sistemas NewSQL*, algunos ejemplos son CockroachDB y NuoDB [2]. Ante el interés de nuevos modelos de datos diferentes al relacional, la aparición de sistemas *multi-database* que permiten acceder a varios modelos de datos ha sido una tendencia creciente en los últimos años. En *DB-Engines* se puede observar que las bases de datos más populares son multi-modelo, como Oracle, MySQL o PostgreSQL.

La migración de bases de datos es una tarea que las organizaciones llevan a cabo cuando necesitan actualizar su SGBD para obtener mejores prestaciones en el almacenamiento y manipulación de sus datos. Como es lógico, el escenario que hemos descrito con nuevos sistemas que suponen una alternativa a los SGBDR dominantes puede considerarse propicio para un aumento del número de migraciones. Sin embargo, estas migraciones son procesos costosos que las empresas suelen postergar el mayor tiempo posible. Una migración es un proceso no trivial que conlleva una carga de trabajo considerable debido a que implica la migración del esquema, los datos y el código, y atajar

[1] `https://db-engines.com/en/ranking` del mes de mayo de 2025.

riesgos como pérdida de datos o inconsistencias entre ellos.

La emergencia de la Inteligencia Artificial Generativa (GenAI) basada en los Modelos de Lenguaje Grande (LLM) ha supuesto, probablemente, la mayor revolución provocada por una innovación en el campo de la informática. Desde la aparición de ChatGPT en noviembre de 2022, la GenAI está evolucionando a un ritmo acelerado y está permitiendo altos niveles de automatización de muchas actividades humanas, en especial, aquellas relacionadas con la Ingeniería del Software [3] [4], incluso, para muchos expertos, el futuro profesional de los desarrolladores está amenazado por los LLM.

Estos modelos han demostrado grandes capacidades para la generación automática de scripts, análisis sintáctico y semántico de código, transformación de estructuras de datos y procesamiento del lenguaje natural, lo que los califica como herramientas potenciales para asistir y optimizar los procesos de migración de bases de datos.

La integración de un LLM en flujos de trabajo de migración podría facilitar algunas tareas como la conversión automática de esquemas entre diferentes SGBD, la generación de scripts de transformación de datos, la identificación de incompatibilidades entre plataformas, o la documentación y validación del proceso [5]. Es importante tener en cuenta que los LLM no solo automatizan tareas técnicas, sino que mejoran la interacción con las herramientas mediante el uso del lenguaje natural, facilitando el trabajo a los usuarios sin experiencia técnica y reduciendo la barrera tecnológica.

En los últimos dos años, un gran número de artículos de investigación y herramientas basadas en LLMs han mostrado las capacidades de la GenAI en todas las áreas de la Ingeniería del Software y de las bases de datos. El Trabajo Fin de Grado (TFG) presentado en esta Memoria se enmarca en esa línea, en concreto, aplicar GenAI para la creación de una herramienta que automatice la migración de bases de datos.

En la actualidad, la migración de bases de datos se suele apoyar en herramientas ETL (*Extraction*, *Transformation*, *Load*) con las cuales es complicado resolver especificidades de la migración porque no están destinadas a ello, o bien con herramientas específicas como Qlik que son muy costosas [5]. Por otro lado, los grandes proveedores de la nube también proporcionan servicios propietarios para migrar bases de datos a su nube, como es el caso de Azure DMS, Google Cloud DMS y AWS DMS. Estos servicios de migración ofrecen distintos grados de automatización y, en algunos casos, soporte parcial para GenAI. No obstante, estas soluciones suelen estar estrechamente ligadas a sus respectivos ecosistemas y requieren configuraciones manuales significativas. En contraste a las soluciones actuales, este TFG propone una herramienta independiente del SGBD, basada en LLM de propósito general y sin dependencia de un proveedor específico de LLMs, capaz de automatizar tanto la migración del esquema y de los datos como la adaptación del código fuente de acceso a datos, integrando además mecanismos de validación estructurada.

A continuación, se exponen los objetivos de este trabajo.

Objetivos y Contribuciones

El principal objetivo del TFG es explorar cómo la GenAI permite automatizar las tres tareas básicas que constituyen una migración de bases de datos: migración del esquema, de los datos y del código, así como el proceso de validación requerido. Para ello, se ha construido una herramienta independiente de la plataforma, tanto de los SGBD origen y destino como del LLM utilizado. Para la construcción de la herramienta se ha utilizado una tecnología novedosa como son los *frameworks de abstracción de LLM*, esto es, plataformas que facilitan la construcción de aplicaciones que utilizan LLM para implementar alguna funcionalidad [6].

En la aplicación de GenAI, el diseño adecuado de los *prompts* es la principal tarea. Un prompt es una entrada o instrucción que se proporciona (programáticamente a través de una API o el usuario directamente a través de una UI) a un LLM para que produzca una respuesta, contenido o acción. Una característica destacada de los frameworks de abstracción de LLM es ofrecer un importante soporte para facilitar el complejo manejo de prompts [6].

El objetivo principal de nuestro trabajo se puede descomponer, por tanto, en los siguientes sub-objetivos.

1. Diseño, implementación y validación de los prompts adecuados para la migración del esquema.

2. Diseño e implementación de los prompts adecuados para la migración de los datos almacenados y del código para la validación, comprobando que no hay pérdida de datos y se mantiene la consistencia.

3. Diseño, implementación y validación de los prompts adecuados para la migración del código.

4. Integración de los tres migradores en una herramienta, denominada LLMigrator-DB y validación mediante un caso de estudio que implica dos diferentes migraciones y usa un LLM diferente para cada una de ellas.

La construcción de LLMigrator-DB se usará como objeto de estudio para evaluar si el uso de LLM para automatizar una migración completa de base de datos es factible y permite reducir el esfuerzo. Este estudio se llevará a cabo sobre las dos migraciones realizadas en el caso de estudio, en las que se cubrirán tanto *migraciones homogéneas* que implican un mismo modelo de datos origen y destino, como *migraciones heterogéneas*, donde los modelos de datos son diferentes.

Desde un punto de vista más técnico, LLMigrator-DB se ha desarrollado como una aplicación basada en microservicios, cuyo back-end incluye una Interfaz de Programación de Aplicaciones (API) de Transferencia de Estado Representacional (REST) que nos permite cambiar el LLM de forma dinámica. Esta independencia del LLM se consigue usando OpenRouter, una interfaz unificada que se ha lanzado recientemente [7].

El estudio se ha realizado para dos de los LLM más populares, como son GPT-4o-mini de OpenAI y Gemini-2.0 Flash de Google. En cuanto al framework de abstracción de LLM, se ha elegido Spring AI.

Respecto a la migración del código, se ha organizado siguiendo una *arquitectura limpia* [8], lo que ha permitido separar la lógica de la aplicación en distintas capas, con el objetivo de lograr un código independiente de tecnologías externas. Esto favorece una lógica de negocio duradera, comprobable y fácilmente mantenible.

En la Sección 2.4 se justifican las diferentes decisiones técnicas tomadas en el desarrollo de este TFG.

Es conveniente señalar que en el ámbito de un TFG no se puede llevar a cabo la validación de la herramienta para casos de estudio complejos, ni abordar el uso de un mayor número de LLM, por limitaciones de tiempo. En este sentido, el front-end de la herramienta se reduce a una interacción a través de la consola de Postman[9].

Las contribuciones del TFG son básicamente las mencionadas arriba al contrastar LLMigrator-DB con herramientas existentes de migración:

1. LLMigrator-DB es una herramienta de migración genérica, independiente del modelo de datos y SGBD.

2. LLMigrator-DB se apoya en GenAI para automatizar las principales tareas de una migración de bases de datos.

3. No sólo migra esquema y datos, sino también el código, a diferencia de las herramientas existentes.

4. Es independiente de LLMs concretos.

5. Integra un mecanismo de validación.

6. Utiliza una tecnología novedosa como son las abstracciones de LLM.

Organización del trabajo realizado

El trabajo realizado se puede dividir en cuatro fases bien diferenciadas. Una fase de estudio de los conceptos y tecnologías necesarios; la fase de diseño, implementación y pruebas para cada uno de los pasos de la migración; la fase de validación a través del caso de estudio, y finalmente la redacción de la presente Memoria. Como veremos más adelante, cada fase ha consistido en varias etapas. También cabe destacar que antes de escribir esta Memoria ya se realizó un proceso de difusión de resultados con la preparación de un artículo corto que fue enviado y aceptado en las Jornadas de Ingeniería del Software y Bases de Datos (JISBD) [10].

Fase de Estudio En una etapa inicial, primero se abordó el estudio de los fundamentos de la GenAI a través de algunos libros y artículos proporcionados por los autores, entre los que podemos destacar el libro "ChatGPT for Java: A Hands-on Developer's Guide to ChatGPT and OpenAI APIs" [11]. Entonces, se llevó a cabo un estudio en profundidad del API de OpenAI y cómo se podría integrar con una aplicación realizada en Java. Posteriormente, se indagó en qué características debería proporcionar Spring AI [12] para decantarnos por su uso.

En una tercera etapa se realizó un primer acercamiento a la *Ingeniería de Prompts* y a las distintas técnicas que se utilizan a la hora de interactuar con un Modelo de Lenguaje Grande (LLM). De nuevo, nos servimos de material de lectura proporcionado por los tutores y otros obtenidos por el alumno en la Web. Entre los libros cabe destacar: "Generative AI in Action" [3] y "LLMs in Production: From Language Models to Successful Product" [13].

A continuación, se realizó un estudio de cómo se podría integrar la independencia de LLM en dicha aplicación. Se descubrió OpenRouter y se procedió a su estudio.

Cabe destacar que durante la siguiente fase fue necesario profundizar en el estudio de algunos de los conceptos y tecnologías estudiados en esta primera fase.

Fase de Diseño, Implementación y Pruebas Esta fase consistió en tres etapas, una para cada artefacto a migrar: esquema, datos y código. Y cada una de estas etapas se planteó como un proceso iterativo en el que se sucedía el diseño del prompt, su implementación, el testing, y los cambios identificados para corregir errores o mejorar las plantillas de prompts creadas para ejecutar las migraciones. Para las pruebas se utilizó GPT-4o-mini de OpenAI como LLM. Además, con el fin de conseguir nuestro objetivo de la independencia de LLM, utilizamos la API unificada para LLMs OpenRouter. Esta API con un cliente REST de OpenAI, nos permite interactuar con cualquier LLM.

En este punto se preparó el artículo mencionado antes para las JISBD. En este artículo presentamos el nuevo enfoque de migración de bases de datos que se había ideado en el TFG y el diseño inicial de nuestra herramienta LLMigrator-DB.

Fase de Validación Una vez la herramienta satisfacía los requisitos previstos, abordamos la fase de validación a través de las dos migraciones del caso de estudio: una migración homogénea entre bases de datos relacionales (MySQL a PostgreSQL) y una migración heterogénea desde un modelo documental (MongoDB) hacia uno relacional (PostgreSQL), cubriendo tanto la transformación de esquemas y datos como la adaptación del código fuente.

Fase de Elaboración de la Memoria La última etapa se centró en la elaboración de la presente Memoria. Se ha utilizado *Overleaf* para facilitar las revisiones de los tutores.

Una explicación más detallada del flujo de trabajo seguido El trabajo se inició a principios de septiembre con una reunión presencial en la que participamos los dos alumnos con líneas de Trabajo Final de Grado (TFG) relacionadas con la migración de bases de datos y los tres tutores implicados: María José Ortín, José Ramón Hoyos y Jesús J. García Molina. En esta reunión se nos presentaron conceptualmente las bases de datos NoSQL, el modelo de datos genérico U-Schema [14] y algunas utilidades del API de OpenAI. En este caso fueron recomendadas algunas lecturas como "Generative AI in Action" [3] para la introducción a GenAI junto con el manejo de prompts y "ChatGPT for Java..." [11] para explorar el manejo de APIs de OpenAI con Java.

Después de esta reunión, el estudio de los conceptos y tecnologías indicados por los tutores nos llevó hasta comienzos del segundo cuatrimestre. Se revisaron diversas publicaciones sobre migración de bases de datos e Ingeniería de Prompts, y se comenzó a realizar ejercicios para el aprendizaje de Spring AI [15], esto es, todo lo necesario para comenzar la fase de desarrollo lo antes posible.

En la segunda reunión con los tutores, se definió el esquema de migración que se aplicaría junto con unos prompts iniciales. En una tercera reunión, se analizaron los primeros resultados obtenidos con GPT-4o-mini, y se hizo un refactoring de las plantillas de prompts utilizadas con el fin de ser más precisos. Tras probar las nuevas plantillas, se mantuvo una cuarta reunión para plantear cómo se podría implementar la independencia de LLM y fue en este momento cuando decidimos el uso de OpenRouter para facilitar la consecución de dicho objetivo. A partir de este punto, se mantuvieron varias reuniones de seguimiento hasta completar el trabajo, en las cuales se fueron refinando las plantillas de prompts y se definieron los dos casos de uso para validar la herramienta. Antes de comenzar la elaboración de la Memoria, una reunión con los tutores sirvió para acordar el índice de contenidos y recibir algunas indicaciones sobre la escritura.

Además de las reuniones presenciales, a lo largo de la realización del TFG, se ha mantenido contacto fluido con los tres tutores mediante un grupo de WhatsApp, videoconferencias Zoom, y un repositorio GitHub [16] que ha servido para aplicar un desarrollo iterativo del código y revisiones del mismo de forma asíncrona.

La Tabla 1.1 muestra una descripción corta de cada etapa y una estimación del esfuerzo dedicado por el alumno, incluyendo también el tiempo dedicado a reuniones con tutores.

Organización de la Memoria

En el capítulo 1 se ha motivado el trabajo, se han definido los objetivos y contribuciones, y se ha descrito el flujo de trabajo aplicado.

En el capítulo 2 se abordan los fundamentos teóricos que son requeridos para comprender el resto de la Memoria. En dicho capítulo, se introducen los siguientes conceptos: LLM, Ingeniería de Prompts, abstracciones de LLM, e independencia de LLM. Además, también se introducen las tecnologías utilizadas, en este caso, Spring AI,

Fase	Descripción	Duración
Estudio inicial I. Conceptos y tecnologías	Concepto de LLM y GenAI, Bases de datos NoSQL, migración, API de OpenAI y uso de Spring Boot con Java.	65 h
Estudio inicial II. Ingeniería de prompts	Investigación de técnicas de diseño de prompts, experimentación con ChatGPT, y diseño inicial de plantillas para la migración.	60 h
Estudio inicial III. Análisis de abstracciones LLM	Análisis de abstracciones LLM y elección de Spring AI.	35 h
Diseño de arquitectura, plantillas de prompts e implementación	Diseño de la arquitectura y plantillas de prompts, desarrollo iterativo con pruebas con GPT-4o-mini y refactorización de las plantillas.	75 h
Escritura del artículo para las JISBD	Elaboración de un artículo para el track de bases de datos de JISBD, presentando el estado actual de la herramienta.	70 h
Implementación de la independencia de LLM	Uso de OpenRouter para conseguir que la herramienta disponga de independencia de LLM sencilla y flexible.	60 h
Evaluación	Validación de la herramienta LLMigrator-DB con un caso de estudio que implicaba una migración homogénea y otra heterogénea. Evaluación de resultados. Refinamiento final de LLMigrator-DB.	40 h
Redacción de la Memoria	Escritura, estructuración y revisión del documento del TFG, incluyendo redacción final tras reuniones con los tutores.	50 h
Interacción con tutores durante el TFG	Reuniones presenciales y virtuales, soporte continuo mediante WhatsApp, Zoom y GitHub.	60 h
Total		**515 h**

Tabla 1.1: Resumen de fases y dedicación del trabajo fin de grado

OpenRouter y los LLM utilizados para llevar a cabo el TFG.

En el capítulo 3 se presentan algunos enfoques propuestos y herramientas existentes relacionadas con la automatización de la migración de bases de datos y se compara nuestro enfoque con ellos.

En el capítulo 4 se presenta una visión global de la solución propuesta. Se introduce la arquitectura general utilizada para implementar LLMigrator-DB, destacando sus componentes principales y su organización modular. A partir de esta arquitectura, se

describe de forma general el funcionamiento de la herramienta.

Tomando como referencia la arquitectura planteada en el capítulo 4, el capítulo 5 profundiza en cada uno de los tres componentes migradores de LLMigrator-DB: el migrador de esquemas, el de datos y el de código. Para cada uno de ellos, se muestran las *plantillas de prompt* diseñadas y se explican las técnicas de Ingeniería de *Prompts* empleadas para interactuar eficazmente con el LLM en el contexto de la migración de bases de datos. También se detallan los puntos de entrada de la aplicación y se describe cómo se integró Spring AI con OpenRouter.

En el capítulo 6 se presenta el caso de estudio que implica dos migraciones. Se describe el esquema inicial junto con los datos de entrada, además de los resultados obtenidos con cada LLM en cada caso.

Para finalizar, en el capítulo 7 se presentan las conclusiones del trabajo, destacando las principales ventajas derivadas del uso de GenAI para automatizar la migración de bases de datos, así como las limitaciones observadas durante el desarrollo. Además, se proponen diversas líneas de trabajo futuro orientadas al diseño de una versión más avanzada de la herramienta, capaz de refinar de forma autónoma los resultados generados mediante técnicas de evaluación automática, razonamiento más profundo y asistencia conversacional, con el objetivo de ofrecer salidas más precisas y adaptadas al usuario.

2. Fundamentos teóricos

En este capítulo se explicarán diversos conceptos que establecen la base teórica del TFG. Más concretamente, hablaremos de *migración de bases de datos, Inteligencia Artificial Generativa, Ingeniería de prompts, Abstracciones de modelos de lenguaje e Independencia de LLM* y, por último, introduciremos *Spring AI y OpenRouter*.

2.1. Migración de bases de datos

Como ya hemos comentado en la introducción, las organizaciones pueden abordar una migración de bases de datos por diversos motivos [17, 18]. Todo con el fin de mejorar la infraestructura de la que hacen uso, ya sea por motivos de seguridad, rendimiento o escalabilidad, entre otros.

Una migración de bases de datos [19, 20] consiste en el proceso de trasladar el contenido de una o varias bases de datos origen a una o varias bases de datos destino. Una vez finalizada la migración, los datos que se encontraban en las bases de datos origen también se encuentran en las bases de datos destino. Es muy posible que al hacer cualquier migración de bases de datos, los datos sean modificados a nivel estructural para que sean conformes al modelo de datos en el que están basadas las bases de datos destino. Habitualmente, durante este proceso de migración siempre se tratará de hacer una copia de seguridad de los datos de origen con el fin de actuar ante posibles pérdidas de datos en el traslado de la información.

Además, el resultado de la migración debe ser *coherente*. Este hecho implica que la migración ha de ser *ordenada*, *completa* y *sin duplicación de datos en las bases de datos destino*. Es decir, los datos han de ser insertados en el orden correspondiente, todo el conjunto de los datos de las bases de datos origen se debe encontrar en las bases de datos destino y no debe existir ningún dato duplicado.

Por añadidura, un aspecto que puede generar confusión es la diferencia entre *migración* y *replicación*. El primero implica mover de forma completa los datos desde una base de datos origen a una base de datos destino. Una vez garantizado que la nueva base de datos funciona correctamente, se elimina la original, y todos los usuarios pasan a operar exclusivamente sobre el nuevo sistema. Sin embargo, la *replicación* consiste en mantener una copia continua de los datos entre dos bases de datos que permanecen activas.

A continuación, se presentan algunos conceptos importantes que permiten caracterizar las migraciones de bases de datos.

- *Base/s de datos origen* son aquellas que contienen los datos a ser migrados.

- *Base/s de datos destino* son aquellas que reciben los datos migrados.

- *Cardinalidad de la migración.* Con este término se hace referencia al número de bases de datos origen y destino que están implicadas en el proceso de migración. Existen cuatro tipos de cardinalidades: *mapeo directo* (1:1), *consolidación* (n:1), *distribución* (1:n) y *redistribución* (n:m). En los casos de uso de este TFG, nos centraremos en migraciones de tipo *mapeo directo*, es decir, con cardinalidad 1:1.

- *Migración homogénea.* Es una migración donde tanto las bases de datos origen como las de destino comparten el mismo modelo de datos. Dicho de otra forma, Las bases de datos origen y destino pueden o no ser manejadas por el mismo SGBD del mismo proveedor.

- *Migración heterogénea.* Al contrario del caso anterior, en este caso las bases de datos origen y destino no están basadas en el mismo modelo de datos y, en consecuencia, no comparten sistema gestor de bases de datos.

- *Migración completa y migración parcial.* Diremos que hemos realizado una migración completa cuando se ha migrado el conjunto completo de los datos, sin embargo, diremos que hemos realizado una migración parcial cuando migremos un subconjunto de los datos de la base de datos origen a una base de datos destino.

Como ya hemos comentado, una migración de bases de datos es un *proceso*. Todo *proceso* se caracteriza por una identificación de pasos o etapas a llevar a cabo para conseguir un objetivo concreto. En el caso de las migraciones de bases de datos, estas etapas están estrechamente relacionadas con los elementos que constituyen una aplicación que hace uso de una base de datos. Estos elementos son el esquema (la estructura de los datos), los datos almacenados y el código (que accede y modifica los datos). Es decir, toda migración de bases de datos implicará una migración de esquema (exceptuando el caso de las bases de datos NoSQL sin esquema), una migración de datos y una migración de código, además de la correspondiente validación en cada etapa y la validación global al finalizar la migración completa.

Cada uno de estos pasos ha definido el diseño de nuestras plantillas de prompts y los datos de entrada que son necesarios para poder llevar a cabo la migración. En este TFG, la validación de LLMigrator-DB se llevará a cabo tanto con una migración homogénea como con otra heterogénea, ambas de *mapeo directo*, para analizar cómo nos puede ayudar la GenAI a mejorar este costoso proceso.

2.2. Modelos de Lenguaje Grande e IA Generativa

La inteligencia artificial generativa (Generative AI, GenAI) es un campo de la inteligencia artificial centrado en la creación automática de contenido original, como texto, imágenes, audio, vídeo o código, a partir de ejemplos o instrucciones. Esta área de la IA se ha vuelto especialmente relevante gracias a los avances recientes en Deep Learning que han conducido a la emergencia de los *modelos de lenguaje de gran escala* (*Large Language Models*, LLMs). Los LLM constituyen una de las tecnologías más influyentes en la actualidad. Desde la aparición de ChatGPT en noviembre de 2022, los avances se suceden a gran velocidad y se puede considerar la tecnología que mayor impacto social ha tenido en menos tiempo.

Los LLM son redes neuronales entrenadas con volúmenes masivos de datos. Este entrenamiento implica el ajuste de millones de parámetros que permiten al modelo aprender patrones lingüísticos, sintácticos y relaciones semánticas con el fin de generar textos coherentes con el contexto de la conversación que se mantiene con el usuario. Se basan en arquitecturas tipo *Transformer*, introducidas en 2017, que permiten capturar dependencias contextuales a distintas escalas de forma eficiente [21]. Aunque inicialmente diseñados para generar texto, los LLMs actuales son multimodales y son capaces de trabajar también con audio, imágenes, tablas o documentos complejos, además de texto.

Estos modelos, como GPT, Gemini o Claude, han demostrado una notable capacidad para comprender instrucciones expresadas en lenguaje natural (*prompts*) y generar respuestas coherentes, por lo que son muy útiles para un gran número de tareas como redacción asistida, resumir texto, el análisis semántico de documentos o la traducción entre lenguajes formales, y en especial han tenido un gran impacto en la construcción de software pudiendo ayudar en la mayoría de tareas involucradas en el ciclo de vida del software [22] [4]. Por tanto, los LLMs han abierto enormes posibilidades en el campo de la GenAI y la pregunta que plantean los expertos es qué trabajos actuales no se verán reemplazados parcial o totalmente por la GenAI. Cabe destacar que el uso de los LLM para GenAI ha sido posibilitado por las mejoras en la capacidad de cómputo de los procesadores y en la disponibilidad de grandes volúmenes de datos accesibles para el entrenamiento. De esta forma, se ha abierto la posibilidad de usar el lenguaje natural como interfaz para interactuar con sistemas complejos.

2.3. Ingeniería de Prompts

En el contexto de los LLMs, el término *prompt* se refiere a la entrada textual que se proporciona al modelo cuando se desea una respuesta que nos ayude en algún sentido. Puede consistir en una pregunta, una instrucción, un ejemplo o una descripción de una tarea, y su redacción influye de forma determinante en la calidad y pertinencia de la salida generada. Por ello, la Ingeniería de Prompts (Prompt Engineering) ha

surgido como un área de estudio dedicada a ofrecer buenas prácticas para la creación de prompts.

La Ingeniería de Prompts proporciona buenas prácticas para diseñar cuidadosamente las entradas al modelo para guiar su comportamiento y obtener resultados específicos, coherentes y útiles. Esta disciplina está adquiriendo una importancia creciente dentro del ámbito de la GenAI [Brousseau, 2025]. Dado que los LLMs no interpretan intenciones de forma directa, la formulación del prompt actúa como una interfaz de control indirecta, lo que convierte su diseño en un factor crítico para la efectividad del sistema [Bahree, 2024; Brousseau, 2025].

Desde el punto de vista técnico, el texto de un prompt se descompone en unidades mínimas denominadas *tokens*, que pueden corresponder a palabras completas, fragmentos de palabras o incluso caracteres individuales. A cada token se le asigna un identificador numérico que representa su posición en el vocabulario del modelo. Posteriormente, estos tokens se transforman en vectores de números reales —a través de técnicas de *embedding*—, lo que permite al modelo capturar relaciones contextuales, semánticas y sintácticas entre ellos. A partir de esta representación vectorial, el modelo de lenguaje genera una respuesta token a token, utilizando la información contenida en el prompt como punto de partida. Por ello, la calidad, claridad y estructura del prompt influyen directamente en la precisión y relevancia del resultado generado [21] [23]. A continuación, se enumeran seis elementos básicos que se considera deben incluir un buen prompt con el fin de maximizar la efectividad del modelo. Estos elementos han sido tomados de diversas recomendaciones publicadas recientemente como la guía "Best practices for prompt engineering with the OpenAI API" de OpenAI [24] y los trabajos de Liu et al. [25], y Sahoo et al. [26], y el libro "Prompt Engineering in Practice" de R. Davies [27].

1. *Contexto.* Proporciona al modelo la información necesaria para interpretar correctamente la tarea. Puede incluir una descripción general del dominio, el rol del modelo (por ejemplo, "actúa como un experto en bases de datos") o antecedentes relevantes.

2. *Instrucción.* Define con precisión lo que se espera que haga el modelo. Cuanto más específica sea la instrucción, menor será la ambigüedad en la respuesta. Ejemplo: "Convierte el siguiente esquema relacional en un modelo de documentos orientado a MongoDB."

3. *Entrada o dato de partida.* Es el contenido sobre el que se debe operar. Puede ser un esquema, una consulta, una descripción o cualquier otro dato sobre el que se deba generar una salida. Ejemplo: Una tabla SQL con sus columnas y tipos de datos.

4. *Formato de salida esperado.* Indica cómo debe estructurarse la respuesta. Esto puede implicar un lenguaje de salida concreto (e.g., JSON, YAML, Markdown)

o un patrón específico (e.g., una lista, una tabla, un bloque de código). Ejemplo: "Devuelve el resultado en formato JSON, con una clave por cada colección generada."

5. *Ejemplos (Opcional).* Los llamados *few-shot prompts* incluyen uno o más ejemplos de entrada-salida que ayudan al modelo a generalizar mejor el tipo de tarea requerida. Ejemplo: Mostrar una tabla relacional simple y su conversión equivalente en MongoDB antes de la entrada real.

6. *Condiciones o restricciones.* Especifican requisitos adicionales que debe cumplir la salida, como evitar repeticiones, usar cierta terminología, aplicar un patrón de normalización, etc. Ejemplo: "No utilices funciones específicas de un proveedor de base de datos."

La calidad de los resultados dependerá de la calidad de cada uno de los elementos del prompt. Además, también vendrá determinada por el orden en que aparece la información dentro del prompt, ya que en diversos modelos la arquitectura de los transformadores utilizada en las tecnologías subyacentes puede alterar la intención y el significado según cómo se dispongan las palabras. Este proceso de refinamiento de prompts es un proceso tradicional de prueba y error, es decir, una vez formulado un prompt, es necesario analizar la salida del modelo con el fin de, posteriormente, ajustar el prompt en función de los errores obtenidos.

2.3.1. Técnicas de Ingeniería de Prompts

Dentro de la Ingeniería de Prompts encontramos diversas técnicas para mejorar la calidad de los resultados del LLM de forma sistemática [28, 29]. A continuación se presentan algunas de las técnicas que han sido utilizadas en el desarrollo de este TFG. Entre ellas destacamos los mensajes de sistema, el aprendizaje *zero-shot, few-shot y many-shot*, el aprendizaje en contexto, la Cadena de Razonamiento (CoT), el Encadenamiento de Prompts y otras buenas prácticas generales en este ámbito, que son de utilidad para completar algunas de las partes de un prompt mencionadas arriba y para otras tareas relacionadas con el manejo de prompts.

Mensajes de sistema. Algunas APIs de proveedores de LLM proporcionan la capacidad de definir mensajes de sistema, que permiten establecer una serie de instrucciones precisas, restricciones temáticas y formatos de salida estructurados (como formato JSON, por ejemplo). El uso de esta herramienta es crítico para evitar desviaciones, puesto que permite acotar el dominio conversacional y evitar alucinaciones por parte del modelo.

Aprendizaje zero-shot, few-shot y many-shot. Permiten configurar el comportamiento del modelo en función del nivel de información previa que se le proporciona. El aprendizaje *zero-shot* es una variante en la que se le da una instrucción directa al LLM

sin proporcionarle ejemplos previos. Por otro lado, en el aprendizaje *few-shot* se proporciona al modelo uno o más ejemplos antes de la instrucción, con el fin de ilustrar el patrón deseado. Por último, en el aprendizaje *many-shot* se proporciona al modelo decenas, cientos o miles de ejemplos previos; este enfoque es menos utilizado debido a las limitaciones de tokens que presentan algunos modelos, aunque puede ser útil en tareas complejas.

Aprendizaje en contexto.. Esta técnica se encuentra estrechamente relacionada con el aprendizaje *few-shot* y consiste en ofrecer ejemplos o explicar el contexto del problema que se va a abordar. Se ha observado que la distribución y formato de los ejemplos proporcionados, junto con explicaciones del contexto que se aborda, implican una mejora en el rendimiento del modelo, ya que le permite aprender mejor la estructura de la tarea que va a realizar.

Cadena de Razonamiento (CoT). Considerando todas las técnicas ya mencionadas, esta es la que puede resultar de mayor interés para tareas complejas en las que se requiere un razonamiento lógico. Esta técnica está asociada al razonamiento complejo por parte del LLM y, como tal, consiste en forzar al modelo a pensar de forma secuencial y progresiva, es decir, pensar paso a paso, problema a problema, con el fin de mejorar su capacidad de razonamiento en tareas de lógica, inferencia o resolución de problemas, entre otras.

*Encadenamiento de Prompts (*Prompt Chaining*).* Es una técnica que consiste en dividir una tarea compleja en subtareas más concretas y sencillas, es decir, un "divide y vencerás" adaptado a la Ingeniería de Prompts. En esta técnica, la salida de un prompt se utiliza como entrada para otro prompt posterior de forma que una tarea compleja se divide en una serie de múltiples pasos encadenados, con prompts diferentes para cada etapa.

Buenas prácticas al redactar un prompt. Entre ellas destacan las siguientes: en todo momento se debe ser específico y claro al redactar cada prompt, de acuerdo a los resultados que esperamos por parte del LLM, ya que si no es así, aumentará la probabilidad de que el LLM alucine y proporcione respuestas ambiguas e inconsistentes; el diseño y desarrollo de prompts ha de ser iterativo y se deben ir refinando en función de los resultados obtenidos; además, el diseño de los prompts ha de ser modular, dividiendo instrucciones largas en bloques independientes muy específicos; y, por último, hay que indicar al LLM un patrón de salida uniforme, para que la salida esté siempre estructurada y sea fácilmente procesable.

2.4. Abstracciones de Modelos de Lenguaje Grande e Independencia de LLM

La aparición de los LLMs trajo consigo un interés creciente por utilizarlos para añadir capacidades inteligentes a nuevas o existentes aplicaciones. El desarrollo de estas aplicaciones suponía un esfuerzo considerable en todo lo relacionado con el manejo de

prompts así como en abordar las limitaciones de los LLMs, como la dificultad para recordar hechos o interacciones previas, imposibilidad de ejecutar cálculos o interactuar con entornos externos. Para superar estas limitaciones, surgieron algunos frameworks que son llamados "abstracciones de LLM" en el artículo de Peter Yong Zhong et al. que ofrece un estudio comparativo de ellos, como son LangChain, AutoGen y AutoGPT [6]. En ese artículo, los autores establecen una arquitectura de siete niveles o capas de abstracción como la deseable para este tipo de frameworks y los clasifican en cinco familias dependiendo de las capas que soportan. Los niveles van desde el 0 que es el "Nivel LLM" (la API del LLM) hasta el 6 que es el "Nivel de interfaz de usuario", y son los siguientes:

Capa 0: *LLM*. Acceso directo al LLM.

Capa 1: *Prompting*. Comunicación textual con el modelo a través de prompts usando la API.

Capa 2: *Restricción de Prompts*. Plantillas y validaciones para dirigir las salidas.

Capa 3: *Flujo de Control* . Incorporación de estructuras como bucles o condicionales.

Capa 4: *Optimización*. Mejora del rendimiento, precisión o eficiencia.

Capa 5: *Aplicación* . Librerías, Utilidades y código reutilizable para construir soluciones completas.

Capa 6: *Interfaz de Usuario* . Punto de contacto directo entre humanos y sistemas basados en LLM.

En la práctica, los frameworks para LLM ofrecen distintos niveles de soporte a lo largo de la jerarquía descrita. Aunque algunos, como Spring AI elegido en este trabajo, proporcionan utilidades también en las capas superiores (Interfaz de Usuario y Aplicación), lo más habitual es que estos frameworks centren su cobertura en las capas intermedias, especialmente en las relacionadas con el control, las restricciones de prompts y el prompting libre (capas 4 a 6). Esta tendencia responde a que dichas capas son las más comunes y reutilizables en múltiples contextos, lo que las convierte en un punto de apoyo clave para el desarrollo de aplicaciones basadas en LLM.

En la Tabla 2.1 se comparan cuatro de las abstracciones de LLMs más utilizadas: LangChain[30], LlamaIndex[31], Spring AI [32] y OpenAI API[33]. Para cada nivel, se expresa si es o no soportado por la abstracción de LLM y, en caso de serlo, si el grado de soporte es bajo, medio o alto. Los datos para LangChain, LlamaIndex y OpenAI API han sido tomados de la tabla que aparece en el artículo de Peter Yong Zhong et al. [6], y los de Spring AI han sido estimados en este TFG explicados en la Sección 2.6.

La tendencia general en el desarrollo de soluciones basadas en LLM es delegar las tareas más especializadas, como la gestión de modelos o el control del flujo conversacional, a frameworks de abstracciones LLM que abstraen la complejidad subyacente,

Nivel de Abstracción	Spring AI	LangChain	LlamaIndex	OpenAI API
1. Interfaz de Usuario	Alto	Sin soporte	Sin soporte	Sin soporte
2. Aplicación	Alto	Alto	Alto	Sin soporte
3. Optimización	Medio	Sin soporte	Sin soporte	Sin soporte
4. Control	Medio	Medio	Medio	Sin soporte
5. Restricción de Prompts	Medio	Alto	Alto	Alto
6. Prompting	Alto	Alto	Alto	Alto
7. LLM	Sin soporte	Sin soporte	Sin soporte	Bajo

Tabla 2.1: Comparativa de niveles de abstracción en distintos frameworks LLM

especialmente en lo relativo al funcionamiento interno LLM. Esta delegación permite que los desarrolladores se centren en las funcionalidades propias de la aplicación sin preocuparse por detalles técnicos de bajo nivel. Por ello, seleccionar un framework con una buena cobertura en los niveles de abstracción relevantes para la funcionalidad que se desea incorporar a una aplicación resulta fundamental en el diseño de sistemas basados en LLM.

2.5. LLMs utilizados: GPT-4o-mini y Gemini-2.0 Flash

En esta Sección nos centraremos en los modelos que hemos utilizado en los dos casos de estudio presentados en este TFG. Los modelos son GPT-4o-mini y Gemini-2.0 Flash, que pertenecen a dos de los principales proveedores, OpenAI y Google, respectivamente, y son dos de los LLM más utilizados en la actualidad.

2.5.1. GPT-4o-mini

GPT-4o-mini se presenta como un LLM perteneciente a la familia de modelos *GPT-4o* de OpenAI, que ha sido diseñado como una alternativa más eficiente y ligera, que mantiene las capacidades avanzadas de comprensión y generación sin incurrir en un excesivo coste computacional [34]. Desde un punto de vista técnico, este modelo hereda la arquitectura diseñada para la familia de modelos *GPT-4o*, es decir, presenta una arquitectura optimizada basada en transformadores auto-regresivos en la que se prioriza la eficiencia en todo el proceso de inferencia. Este rendimiento está justificado ya que presenta una mejora en los mecanismos de atención y en la estructura interna de los bloques de inferencia, permitiendo una ejecución eficiente y con un menor consumo de recursos [35, 36].

A nivel funcional, GPT-4o-mini es competente en tareas de análisis semántico, identificación de entidades y estructuración de respuestas en formatos estructurados, lo cual es esencial en la automatización de procesos. Además, en este modelo se mejoró su capacidad para mantener la coherencia en conversaciones largas o técnicas, ya que su diseño se encuentra centrado en el uso eficiente de la memoria de contexto. Este

último detalle nos permite manejar mensajes enlazados e instrucciones compuestas por múltiples pasos [37].

Además, al tratarse de un modelo entrenado sobre gran variedad de conocimientos técnicos, hace que sea capaz de reconocer terminología específica de cualquier sistema de bases de datos. Por tanto, este conocimiento generalista nos permite hacer que el modelo razone e infiera las distintas operaciones de evolución que pueden surgir en la migración, como renombramientos y cambios de tipo de datos.

A nivel comparativo, ofrece una relación coste-rendimiento favorable frente a otros modelos. Su rendimiento lo posiciona por encima de otros modelos, ofreciendo resultados favorables en generación técnica de nivel medio de complejidad [38]. Por último, en términos de integración, nos proporciona una gran flexibilidad ya que permite interactuar con el modelo desde múltiples puntos de entrada. Podemos interactuar utilizando la API de OpenAI [33] directamente. Por otro lado, también podemos hacer uso de una abstracción de LLM para interactuar con la API, por ejemplo, con Spring AI. Y otro enfoque interesante es utilizar OpenRouter [7] como intermediario para interactuar con el modelo. Esta variedad de opciones ha sido lo que nos ha permitido el desarrollo de una arquitectura flexible y desacoplada.

2.5.2. Gemini-2.0 Flash

El segundo modelo utilizado es Gemini 2.0 Flash, una variante optimizada y ligera del modelo multimodal *Gemini 2.0* desarrollado por Google DeepMind.

Desde el punto de vista funcional, este modelo es capaz de realizar tareas avanzadas de, entre otras, comprensión de instrucciones y generación de texto técnico y estructurado. Según el informe técnico de Google DeepMind [39], uno de los factores clave de este LLM es la utilización de una arquitectura de atención eficiente junto con una política de segmentación de contexto que optimiza la lectura y generación de tokens. Esta eficiencia favorece al usuario en aquellos casos donde se requiere una interacción iterativa dentro de un contexto conversacional. Es por ello que el modelo es idóneo para el contexto que se presenta en este TFG.

Por último, destacar que este modelo puede ser fácilmente integrado [40] haciendo uso de un cliente REST, o de una abstracción de modelo de lenguaje, o de OpenRouter, al igual que el modelo GPT-4o-mini de OpenAI.

2.6. Spring AI y OpenRouter

Spring AI [15] es una abstracción de LLM, extensión del framework Spring, que facilita la integración de LLMs dentro de una aplicación Java de forma desacoplada y modular.

LangChain es el framework de abstracción de LLM más popular y usado, pero en el momento de iniciar este TFG no ofrecía soporte para Java. Entonces analizamos Spring

AI y, al ver el grado de soporte de cada capa, mostrado en la Tabla 2.1 y explicado más adelante en esta Sección, nos decidimos en elegirlo para nuestro trabajo.

La asignación de los niveles de soporte a cada capa en Spring AI se ha realizado tras un análisis detallado de su documentación oficial [32]. El soporte alto en las capas superiores (Interfaz de Usuario y Aplicación) está justificado por la integración directa con el ecosistema Spring, especialmente Spring Boot [12], que facilita construir aplicaciones completas con interfaces REST y documentación integrada. El soporte en las capas de prompting y restricción de prompts también es alto, dado que Spring AI ofrece componentes específicos como `PromptTemplate` y `StructuredOutputConverter`. En cuanto al soporte medio a la capa de control (flujo conversacional) este se basa en elementos como `ChatMemory` y `SystemMessage`, que proporcionan un control moderado sobre el flujo conversacional, aunque sin alcanzar el control avanzado de otros frameworks como LangChain. Finalmente, se ha considerado que no ofrece soporte directo sobre la capa del LLM, ya que delega completamente en clientes específicos por cada proveedor. Destacar que todos los componentes mencionados para justificar las asignaciones de los niveles de soporte serán explicados más abajo en esta misma sección.

Spring AI nos permite construir aplicaciones que hagan uso de LLMs sin depender directamente de un proveedor concreto de modelos. Sin embargo, hay una importante limitación: se requiere conocer e implementar clases específicas para cada proveedor. Por ejemplo, para interactuar con modelos de OpenAI es necesario utilizar la clase *OpenAIChatClient*, mientras que para modelos como Gemini-2.0 Flash se debe emplear una implementación distinta, como es *VertexAIChatModel*. Este enfoque complica la extensibilidad y reutilización del código, y para evitarlo se ha utilizado una API uniforme de reciente aparición como es OpenRouter [7]. Esta interfaz es compatible con el cliente REST de OpenAI, lo que permite acceder a múltiples LLMs a través de un único punto de entrada, sin necesidad de adaptar el código a cada proveedor concreto.

Por tanto, Spring AI actúa como una capa intermedia, la cual nos ofrece clientes para distintos proveedores de LLM y define una serie de interfaces comunes para todos ellos. Estas interfaces nos permiten interactuar con diversas capacidades como, entre otras, la generación de texto y la generación de imágenes. En este TFG, hemos hecho uso de *OpenAIChatClient* de Spring AI para comunicarnos con los distintos modelos a través de OpenRouter.

La integración de Spring AI con OpenRouter es directa y se logra unificando el *endpoint* de OpenRouter como si se tratara de la API de OpenAI. Es decir, necesitamos un cliente REST OpenAI el cual nos servirá para comunicarnos y hacer peticiones a distintos proveedores de LLM utilizando una única implementación y un único cliente. Gracias a ello, conseguimos la independencia de LLM y, a su vez, conseguimos que el sistema en sí esté completamente modularizado para que sea fácilmente configurable, siendo lo propicio para experimentar y realizar un análisis comparativo futuro entre diferentes LLM.

Una vez justificado el porqué es útil el uso de OpenRouter, explicaremos las capacidades que nos ofrece el framework Spring AI que son de interés de cara al TFG.

Entre ellas destacamos las plantillas de prompts, el formato estructurado de salida, los mensajes de sistema y la memoria conversacional.

Spring AI soporta *plantillas de prompts*, un elemento esencial en el desarrollo de aplicaciones basadas en LLM. Las plantillas de prompts (*PromptTemplate*) nos permiten separar el diseño del prompt de los datos concretos que se insertan en él. Es decir, estas plantillas nos permiten la inserción en tiempo de ejecución de los datos de entrada para crear interacciones precisas, separando la lógica de negocio del contenido textual. Además, estas plantillas de *prompts* se pueden combinar con los *formatos estructurados de salida* (*StructuredOutputConverter*), los cuales nos permiten indicarle al LLM el formato de salida esperado en su respuesta y, en consecuencia, transformar dicha salida textual en un Objeto de Transferencia de Datos (DTO), es decir, un objeto que representa datos estructurados y que se utiliza para transferir información entre diferentes capas o componentes del sistema.

Spring AI ofrece también soporte para los *mensajes de sistema* (*SystemMessage*), esenciales para indicar al modelo cuál es su rol o tarea en cada interacción. Por último, el módulo de *memoria conversacional* (*ChatMemory*) resulta crucial para implementar flujos conversacionales en los que no solo se considera la petición actual, sino también el contexto establecido en interacciones anteriores.

En resumen, el uso combinado de estos componentes (*PromptTemplate*, *StructuredOutputConverter*, *SystemMessage* y *ChatMemory*), junto con la posibilidad de trabajar con múltiples proveedores de LLM sin necesidad de preocuparse por los detalles específicos de implementación de cada uno, ha facilitado significativamente el desarrollo de esta herramienta a lo largo del TFG.

3. Estado del arte

Como ya hemos comentado en la introducción, la migración de bases de datos es un proceso crítico y costoso dentro del mantenimiento y evolución de sistemas software con uso intensivo de datos. Normalmente, una migración es abordada con herramientas especializadas que permitían automatizar alguna parte del proceso con el fin de garantizar una migración segura hacia otros sistemas con el mismo o distinto modelo de datos. Como sucede con la mayoría de tareas relacionadas con la creación y evolución de software, se puede observar una tendencia clara a aprovechar el potencial de los LLMs en la migración de bases de datos, pero no hemos encontrado una herramienta alineada con nuestros objetivos.

En este capítulo introduciremos los servicios de migración de los tres principales proveedores cloud, como son Microsoft, Google y Amazon, y también consideraremos la herramienta Datafold, que incluye un agente de migración. Por otro lado, comentaremos el único trabajo de investigación que hemos encontrado y que fue publicado en noviembre de 2024 por M.K. Goyal et al. [5]. Acabaremos el capítulo presentado un estudio comparativo de las herramientas consideradas.

Azure Database Migration Service (Azure DMS) permite realizar migraciones homogéneas y heterogéneas desde múltiples orígenes hacia una instancia de base de datos alojada en los servicios de Azure [41]. A diferencia de sus competidores de Google y Amazon, Azure DMS no incorpora por el momento ninguna funcionalidad basada en GenAI, por lo que el proceso sigue dependiendo de configuraciones manuales.

Google Cloud Database Migration Service (Google Cloud DMS) permite migrar datos y metadatos desde una base de datos origen a una destino que, como requisito, debe estar alojada en la nube de Google. Una vez completada la migración, la base de datos destino pasa a ser la base de datos principal para las aplicaciones que dependían de la base de datos origen [42]. Este servicio admite tanto migraciones homogéneas (por ejemplo, mover datos entre MySQL y Cloud SQL para MySQL) como heterogéneas (por ejemplo, mover datos entre Oracle y Cloud SQL para PostgreSQL). En el caso de las migraciones heterogéneas, este servicio se ve potenciado por la incorporación de capacidades de conversión automatizadas potenciadas por *Gemini*. La incorporación de la inteligencia artificial nos permite transformar esquemas y estructuras del modelo de datos origen en estructuras del modelo de datos destino, reduciendo la necesidad de intervención humana. El servicio realiza una conversión inicial automática del esquema tan pronto como se crea en el servicio el entorno de trabajo de migración y, además, proporciona un editor SQL interactivo que facilita la revisión y personalización de la conversión, de modo que se pueden definir directivas específicas para ajustar la

conversión si el usuario lo desea.

AWS Database Migration Service (AWS DMS) también permite realizar migraciones homogéneas y heterogéneas, siempre y cuando la base de datos destino se encuentre alojada en el ecosistema de Amazon Web Services (AWS) [43]. Para realizar conversiones de esquema nos ofrecen dos herramientas, una de escritorio llamada *AWS Schema Conversion Tool (SCT)*, como una versión web llamada *DMS Schema Conversion* [44, 45]. Esta herramienta evalúa automáticamente la complejidad de la migración, transforma objetos de datos como tablas, vistas, procedimientos y funciones. Además, permite aplicar reglas de transformación para personalizar la conversión de dichos objetos y también permite editar el código previo a su despliegue. En las últimas actualizaciones de DMS Schema Conversion, ha comenzado a integrar capacidades de GenAI para hacer más predecible la conversión entre bases de datos heterogéneas, generar informes de evaluación y estimar el esfuerzo manual restante para completar la migración [46].

Datafold DMA es una herramienta que ofrece un framework para la automatización basada en AI de la migración de datos [47]. Su principal innovación es el *Agente de Migración de Datos* (*Data Migration Agent, DMA*) que está basado en un LLM [48]. Esta herramienta traduce de forma automática código SQL heredado a nuevos entornos, valida la equivalencia entre las bases de datos origen y destino y es capaz de corregir discrepancias mediante bucles de retroalimentación basados en el uso del LLM, es decir, cuando el DMA traduce código SQL o migra datos, no lo hace una sola vez, sino que compara los resultados obtenidos y, si detecta errores o diferencias, vuelve a ejecutar el proceso de traducción incorporando la información obtenida, hasta lograr una equivalencia entre origen y destino. Durante todo el proceso, el DMA genera inventarios automatizados de anomalías detectadas y, en función de estos, propone soluciones para eliminarlas y reconciliar los sistemas de origen y destino.

Cabe destacar que *Datafold* no actúa directamente como migrador de datos sino como una plataforma especializada en observabilidad de datos y control de calidad de los mismos, es decir, su labor se centra en detectar, prevenir y entender errores en los datos. Sin embargo, el enfoque basado en GenAI representa una propuesta muy avanzada en la automatización de la migración.

Antes de mostrar el análisis comparativo de estas herramientas y LLMigrator-DB, vamos a comentar un artículo de investigación publicado recientemente en noviembre de 2024 cuyo objetivo es el mismo que se plantea en este TFG [5]. Los tres autores plantean como objetivo: "(this paper) seeks to determine the extent to which generative AI particularly the Large Language Models can redefine Database Migration. Señalan que las herramientas ETL usadas normalmente para migrar datos y herramientas específicas de la migración como Qlick son costosas y no resuelven todas las especificidades de las migraciones de bases de datos. Primero, establecen algunas tareas del proceso de migración que pueden ser asistidas o automatizadas parcial o totalmente por los LLM como son: análisis de código para inferir el esquema, trasladar la lógica de aplicación desde triggers y procedimientos almacenados a código OO que usa ORM como Hiberante, la conversión del esquema, el mapeo de tipos de datos, y la optimización

de consultas entre otras. Para valorar eficientemente la viabilidad del uso del LLM en la migración de bases de datos, los autores realizaron varios experimentos con una base de datos del dominio del comercio electrónico con datos generados sintéticamente, siendo Oracle12c el SGBD de la base de datos origen y PostgreSQL 15 de la base de datos destino, y usaron Gemini como LLM. Las métricas que midieron fueron accuracy de la conversión de esquemas (95%), accuracy de la migración de código (90%), data integrity (100%) y Reducción del esfuerzo (60%). El trabajo no proporciona detalles sobre los experimentos realizados. Los resultados refuerzan el enfoque propuesto en este TFG, dónde sé ha construido una herramienta basada en LLM utilizando un framework de abstracción de LLM, se ha validado con dos casos de estudios que han implicado heterogeneidad en cuanto al modelo de datos, y se ha realizado un diseño e implementación de plantillas de prompts, así como prompts del sistema y de definición del formato de salida, entre otros aspectos diferenciadores.

Característica	**Azure DMS**	**Google Cloud DMS**	**AWS DMS**	**Datafold DMA**	**LLMigrator-DB**
Migraciones Homogéneas	Sí	Sí	Sí	No	Sí
Migraciones Heterogéneas	Sí	Sí	Sí	Sí	Sí
IA Generativa	No	Sí (Gemini)	Sí (DMS Schema Conversion)	Sí (LLM propio)	Sí (GenAI independiente via Spring AI + OpenRouter)
Conversión de Esquemas	Sí (manual)	Sí (automática con Gemini)	Sí (SCT y DMS Schema Conversion)	Sí (traducción SQL con feedback LLM)	Sí (automática mediante prompts específicos)
Validación Automática	No	Parcial	Sí	Sí (comparación continua)	Sí (generación de scripts de validación estructurada)
Interfaz Interactiva	Azure Data Studio	Editor SQL	Interfaz web y escritorio	Validación + revisión	CLI + OpenAPI (documentación interactiva)
Dependencia del Proveedor	Alta (Azure)	Alta (Google)	Alta (AWS)	Baja (independiente)	Nula (modelo desacoplado y multi-LLM)
Rol Principal	Migración gestionada en Azure	Migración con soporte IA en Google Cloud	Migración completa en AWS	Control de calidad y observabilidad con IA	Migración de esquema, datos y código con validación asistida por LLM

Tabla 3.1: Comparativa de herramientas de migración de bases de datos

4. Visión general de la solución

Con el fin de introducir el trabajo realizado, este capítulo establece el marco conceptual del proyecto, delineando tanto la arquitectura como el proceso seguido para construir LLMigrator-DB. El objetivo es proporcionar una visión estructural de alto nivel que sirva como base para comprender, en detalle, los procesos de implementación y validación que se abordan en los capítulos siguientes. En primer lugar, se presenta una explicación general de la arquitectura adoptada y, en segundo lugar, se describe cómo se han diseñado y estructurado los *prompts* con el fin de lograr una comunicación efectiva con el modelo de lenguaje. La Figura 4.1 ilustra un esquema general de la solución propuesta en este TFG.

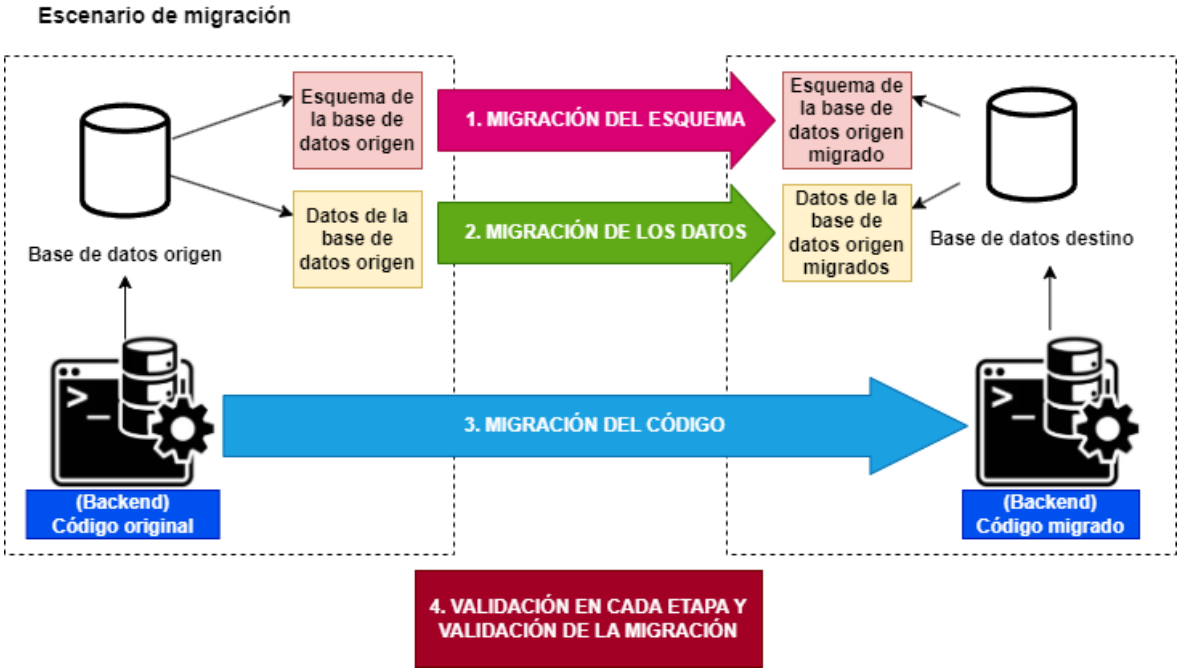

Figura 4.1: Escenario general de migración de bases de datos

Tal y como se explicó en el Capítulo 2.1, todo proceso de migración de bases de datos se estructura en tres fases principales, que deben ejecutarse de forma secuencial: migración del esquema, migración de los datos y migración del código asociado. La Figura 4.1 representa este escenario típico de migración. A partir de esta visión general, hemos diseñado una herramienta que, utilizando información del sistema origen, es

capaz de generar unos scripts con los que se pueda realizar cada una de estas fases de forma fiable y asistida por un LLM. Cabe aclarar que la migración de código se presenta como una fase opcional y separada del flujo principal que se describe a continuación. Además de llevar a cabo las migraciones de los 3 artefactos envueltos en una aplicación, esquema, datos y código, otro requisito principal en la construcción de LLMigrator-DB es la independencia de la plataforma, tanto de los SGBD y modelos de datos como de los LLMs usados.

De acuerdo con el esquema de la Figura 4.1, si vemos nuestro sistema como una caja negra, está formado por tres componentes principales que son: (1) migrador de esquema, (2) migrador de datos, y (3) migrador de código. Todos ellos utilizan la información proporcionada por el usuario (prompts) para realizar su respectiva labor.

Los tres módulos son interdependientes, en el sentido de que se deben ejecutar de forma secuencial y en todo momento se ha de tener en cuenta los resultados obtenidos por cada uno para ejecutar el siguiente. La Figura 4.2 muestra una visión general de la herramienta desarrollada. Los números indican la secuencia de pasos que deben llevarse a cabo en la migración, desde la introducción de la información necesaria por parte del usuario, pasando por la ejecución de los componentes antes mencionados, las comunicaciones externas de la aplicación, hasta las salidas producidas por parte de la herramienta de migración, descritas a un alto nivel de abstracción. Cabe destacar que, debido a la naturaleza no determinista de los LLM, cada ejecución se ha repetido tres veces para comprobar si los resultados no eran idénticos y, en ese caso, analizar las diferencias.

El migrador de esquema se encarga de producir un script del esquema destino o inferirlo y una explicación del script generado o de su respectiva inferencia. El migrador de datos se encarga de producir un script en *Python* y una explicación del mismo; dicho script ha de ser ejecutado por el usuario. Y, por último, el migrador de código se encarga de migrar el código de la aplicación inicial que hace uso de la base de datos, migrando el código del dominio, los repositorios y los servicios al lenguaje de programación y al framework de Mapeo Objeto-Relacional (ORM) de destino seleccionado por el usuario en la entrada. Este último paso proporciona varios scripts con el código requerido por el usuario, junto con sus respectivas explicaciones en texto plano.

Como podemos ver en la Figura 4.2, la herramienta presenta una interfaz REST denotando que una *arquitectura REST* ha sido establecida para implementar LLMigrator-DB. Este tipo de arquitectura ha sido escogida por su flexibilidad y su amplia adopción en los sistemas distribuidos modernos. La razón de la gran aceptación de las API REST[49], de forma simplificada, es el conjunto de principios arquitectónicos que permite una interacción cliente-servidor sin estado y, además, hace uso del protocolo HTTP como método de comunicación, que es el más utilizado en la actualidad.

Las Figuras 4.3 y 4.4 permiten observar el proceso completo de migración de una base de datos y su código asociado desde el punto de vista procedimental del usuario. Siguiendo la numeración utilizada en la Figura 4.3, el usuario se encarga de (1) proporcionar la información inicial; posteriormente, tras obtener los resultados de

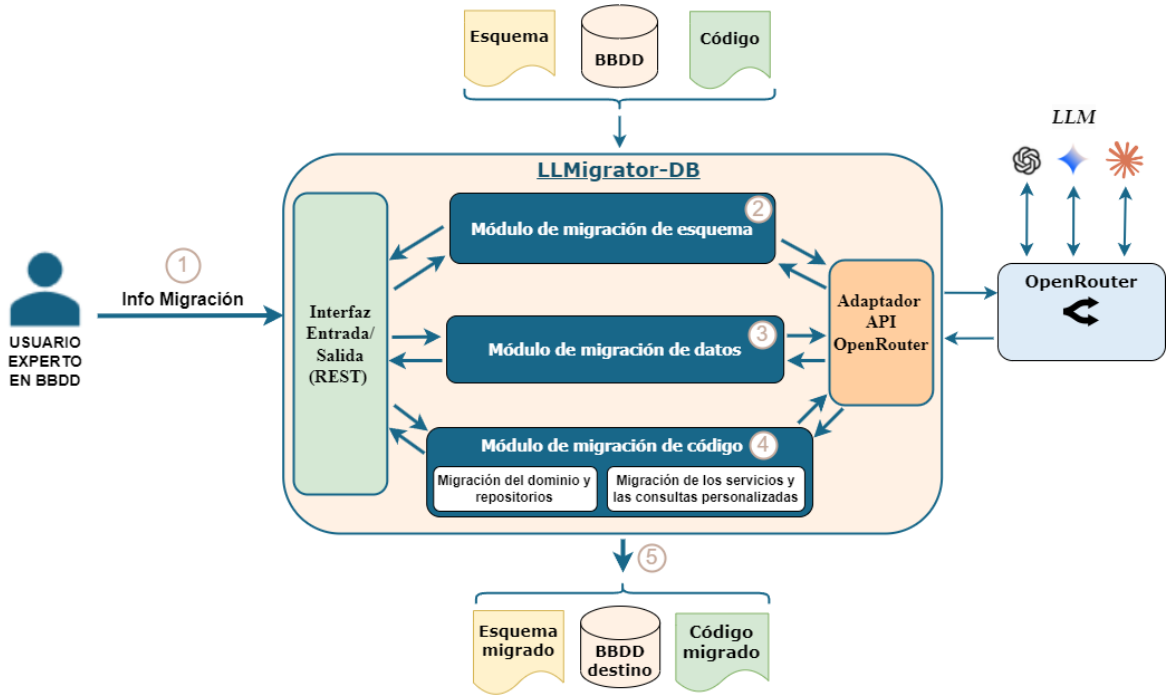

Figura 4.2: Visión general de la herramienta desarrollada

la migración (2, 3 y 4), deberá validar manualmente el esquema de la base de datos destino (5), ejecutar el script de generación de esquema proporcionado (6), ejecutar el script de migración de datos (7) y, finalmente, validar la migración ejecutando el correspondiente script de validación (8).

Por otro lado, en la Figura 4.4 se muestra que, tras haber llevado a cabo el proceso completo de migración de esquema y datos (pasos 1 a 8), el usuario deberá proporcionar la información relativa al código (9) y, tras obtener los resultados (pasos 10 a 12), realizará la inserción del código migrado de la capa de persistencia y las entidades del dominio (13) y la inserción del código migrado de la capa de aplicación con la lógica de negocio de la aplicación (14).

En el siguiente capítulo, se explicará en detalle cada uno de los tres componentes de LLMigrator-DB, comentando la información de entrada y salida. Uno de los principales retos abordados ha sido el refinamiento de los mensajes del sistema utilizados en los prompts, así como la integración entre Spring AI y la API de OpenRouter, debido a que el framework no ofrece soporte directo para dicha API.

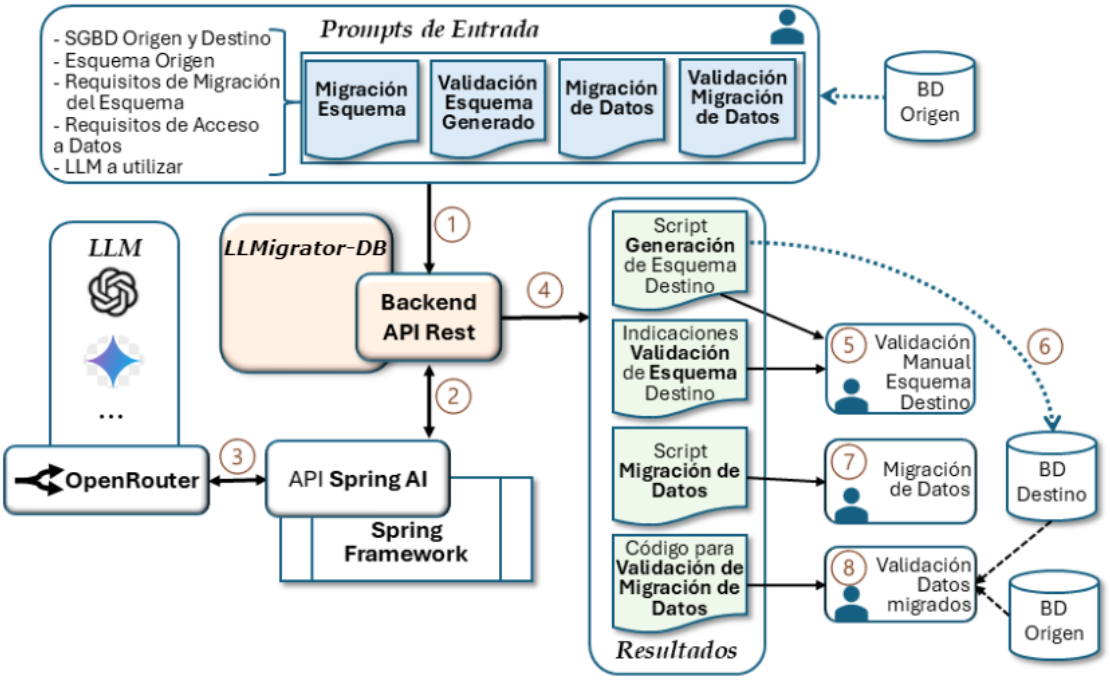

Figura 4.3: Punto de vista procedimental del usuario al migrar la base de datos

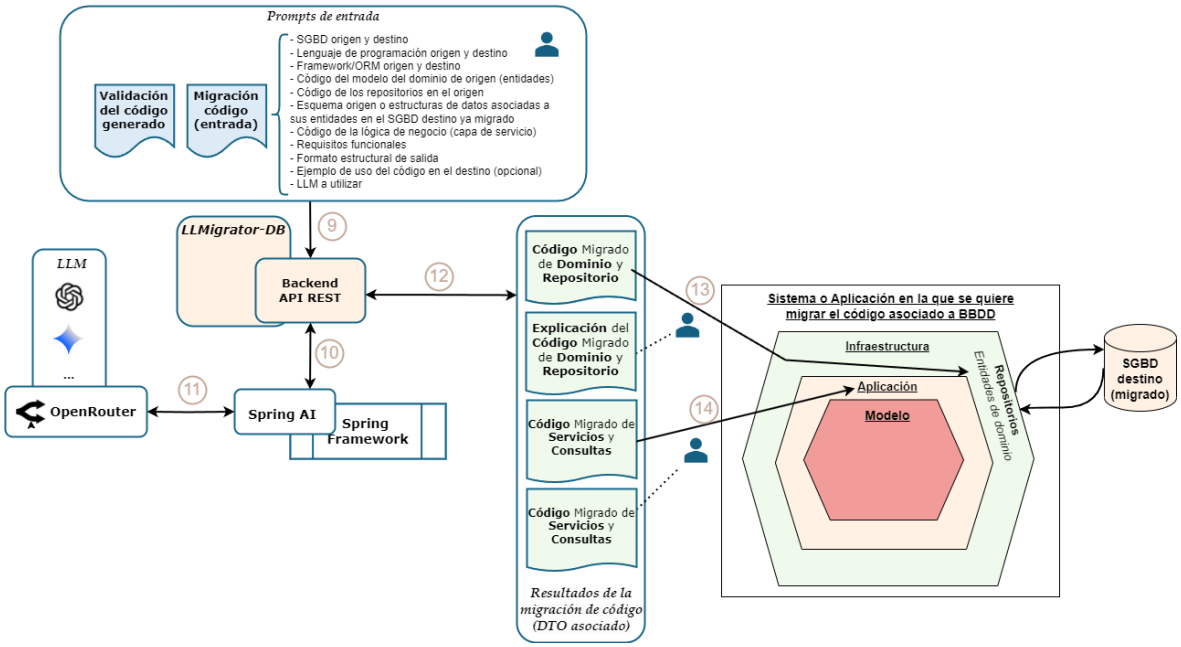

Figura 4.4: Punto de vista procedimental del usuario al migrar el código de la base de datos

5. Construcción de la herramienta LLMigrator-DB

En este capítulo se describe la construcción de LLMigrator-DB, la herramienta desarrollada en este TFG para automatizar el proceso de migración de bases de datos. En primer lugar, se explica cómo se han aplicado las técnicas de ingeniería de prompts, introducidas en 2.3, en cada uno de los componentes de LLMigrator-DB. A continuación, cómo los prompts se han estructurado en Spring AI para aplicar de forma efectiva dichas técnicas. Posteriormente, se explica el diseño de la arquitectura REST, mostrando el diseño de los puntos de entrada y la información que ha de recibir LLMigrator-DB por parte del usuario. Finalmente, se comenta la implementación de los tres módulos de migración, las dependencias existentes entre ellos y cómo se han resuelto las interdependencias entre las etapas mencionadas previamente en el capítulo 4.

5.1. Técnicas de ingeniería de prompts utilizadas

En la construcción de LLMigrator-DB, la ingeniería de prompts ha desempeñado un papel crucial para asegurar una interacción efectiva con los modelos de lenguaje utilizados. De hecho, la aplicación de diversas técnicas de ingeniería de prompts nos ha permitido estructurar al detalle la información introducida por el usuario con el objetivo de obtener respuestas precisas, consistentes y estructuradas, minimizando consecuencias negativas como las alucinaciones.

Las técnicas empleadas incluyen: mensajes de sistema, el aprendizaje en contexto, el encadenamiento de prompts, el uso de un formato de salida uniforme, haciendo uso de prompts estructurados, y un aprendizaje *zero-shot*, es decir, no se han incluido ejemplos. Estas técnicas se han utilizado de forma global en todos los prompts utilizados por la herramienta, gracias al uso de las plantillas de prompts que nos proporciona Spring AI (clase *PromptTemplate*) [50]), las cuales permiten separar el contenido fijo (como las instrucciones o la estructura del prompt) del contenido variable (información dinámica proporcionada por el usuario). Este enfoque a la hora de usar las plantillas de prompts ha facilitado la reutilización y el refinamiento de los prompts utilizados en las distintas fases del proceso de migración.

Los *mensajes de sistema* se han utilizado para establecer el rol que debe asumir el LLM para delimitar el contexto de migración en cada etapa durante la conversación que mantiene la herramienta con el LLM. Estos mensajes son fundamentales para guiar

al modelo en tareas específicas, como la migración del esquema o datos y la validación de dichas migraciones. En todos los casos se han incluido instrucciones claras y precisas relacionadas con cada etapa del proceso de migración, así como restricciones acerca del formato de salida (en concreto, hemos exigido el uso del formato JSON conforme al RFC 8259 [51]), con el fin de asegurar respuestas estructuradas y fácilmente procesables.

Para mejorar la compresión contextual del modelo, se ha hecho uso de la técnica de *aprendizaje en contexto*. El uso de esta técnica es apreciable en el enriquecimiento de los prompts enviados al modelo, incluyendo detalles adicionales sobre el entorno de ejecución como, entre otros, los sistemas gestores de bases de datos origen y destino, y los requisitos funcionales a seguir. Esta información complementaria permite al modelo ajustar su comportamiento, mejorando así la precisión de las respuestas generadas y disminuyendo la cantidad de alucinaciones producidas por el mismo. En la implementación de la herramienta, esta técnica se puede apreciar en el uso de *prompts contextuales iniciales*, que no solicitan una respuesta inmediata al modelo, sino que establecen un marco contextual completo antes de comenzar cada fase de la migración. Estos prompts tienen como finalidad preparar al modelo con todo el conocimiento necesario y garantizar respuestas coherentes y alineadas con lo esperado por el usuario.

El *encadenamiento de prompts* también ha jugado un papel importante para conseguir nuestros objetivos. Como ya hemos comentado en capítulos anteriores, el proceso de migración es un proceso que consiste de tres etapas principales: migración del esquema, de los datos y del código, y por supuesto la validación de la corrección de los artefactos migrados. Es por ello que, en lugar de realizar una única petición al modelo LLM con una instrucción global, el proceso se ha dividido en etapas interdependientes que se ejecutan de forma secuencial y donde la salida de uno es la entrada del siguiente. Por ejemplo, la migración de esquema y datos se compone de: contextualización, generación del esquema, validación manual del esquema, generación del script de migración de datos y validación del proceso de migración completo. Cada uno de estos pasos genera una salida que alimenta al siguiente prompt, haciendo que la tarea a realizar sea incremental y más controlada.

Se ha hecho uso de forma general de *prompts estructurados con formatos de salida uniformes*. En todos los casos, las respuestas esperadas por parte del LLM debían ajustarse a un formato específico definido en nuestro código (formato JSON) que nos permitía el procesamiento automático de los campos por parte de la herramienta. Este requisito facilita tanto la validación y control de errores en cada fase del proceso, como la estandarización de la comunicación entre los distintos módulos que conforman la herramienta. Esta consistencia estructural ha resultado de gran utilidad en el desarrollo de LLMigrator-DB ya que existía gran interdependencia entre las operaciones de migración llevadas a cabo.

Finalmente cabe señalar que se ha aplicado un *aprendizaje* zero-shot durante el proceso de interacción con el LLM. En cada etapa de la migración se proporcionan instrucciones específicas, junto con la información necesaria para llevar a cabo la tarea, sin necesidad de incluir ejemplos que establezcan cuál debe ser el resultado para una

determinada entrada. Esta aproximación resulta muy eficaz puesto que las tareas están bien definidas y no presentan ambigüedad y, por tanto, las instrucciones se pueden expresar de forma precisa.

5.2. Acceso a la herramienta mediante interfaz REST

La interacción entre el usuario y LLMigrator-DB se realiza a través de una API con arquitectura REST, como ya se mencionó en el capítulo 4. Este enfoque facilita una exposición estructurada y estandarizada del servicio, lo que permite su integración con otros sistemas y herramientas externas. Además, al desacoplar la lógica de negocio de la interfaz de acceso, se favorece la escalabilidad y la evolución futura de la herramienta. Está previsto una comunicación entre un cliente front-end que articule las peticiones al backend, pero en este TFG se ha usado un cliente REST Postman, dado que no es posible abordar el desarrollo del front-end en el marco de un TFG.

La herramienta ofrece dos endpoints principales: uno destinado a la migración de esquema y datos, y otro a la migración de código asociado a la capa de persistencia. Ambos endpoints aceptan peticiones `HTTP POST` cuyo cuerpo debe contener un fichero JSON que representa el DTO de entrada, con los parámetros necesarios para llevar a cabo el proceso de migración completo. La Figura 5.1 muestra la documentación interactiva generada automáticamente a partir de la especificación OpenAPI [52], donde se describen en detalle los distintos puntos de acceso del servicio. Por su parte, la Figura 5.2 presenta los campos que conforman los objetos DTOs, cuya correcta cumplimentación recae en el usuario experto que utiliza la herramienta.

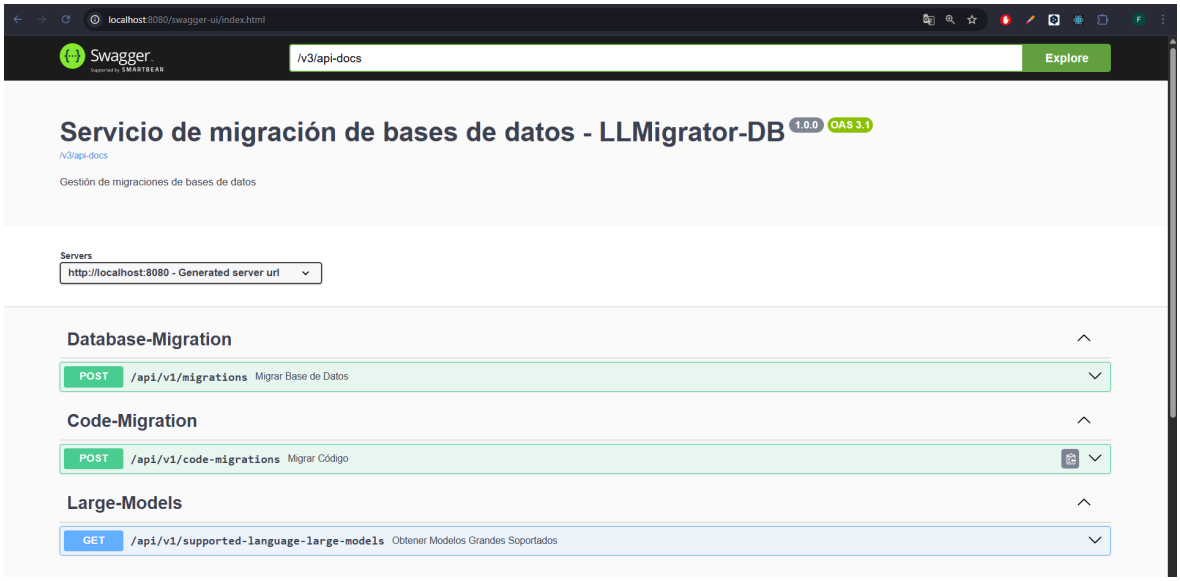

Figura 5.1: Documentación OpenAPI de LLMigrator-DB

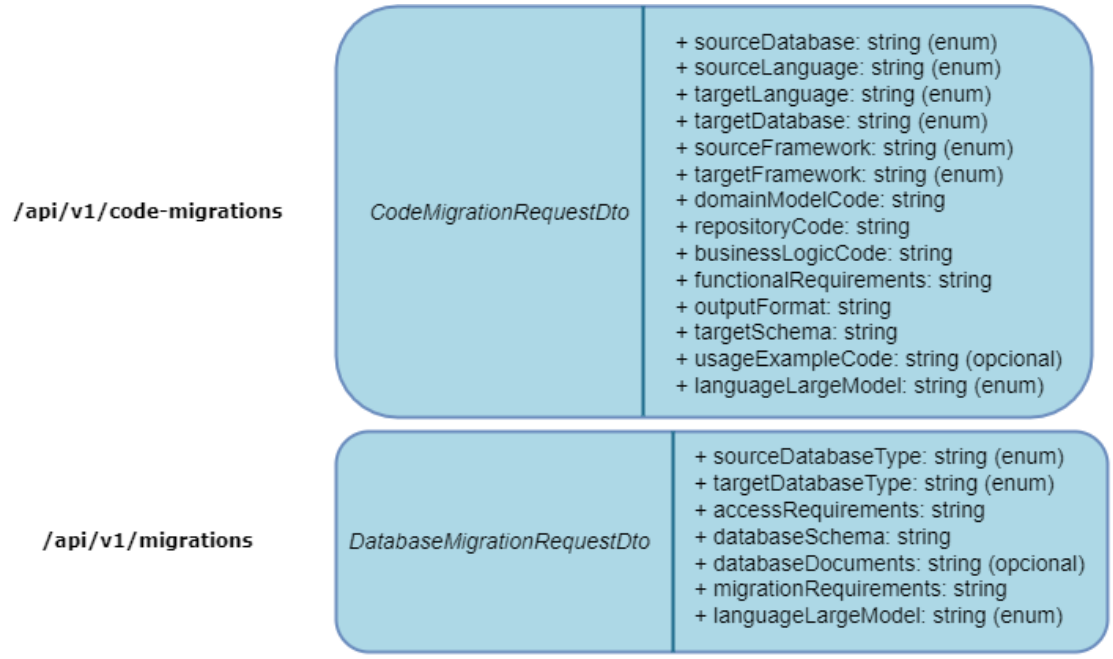

Figura 5.2: Objetos (DTO) de entrada a LLMigrator-DB con la información requerida del usuario

Los endpoints disponibles son: `/api/v1/migrations`, destinado a ejecutar la conversión del esquema y la migración de los datos almacenados, así como sus respectivas validaciones, y `/api/v1/code-migrations`, orientado específicamente a la migración del código de acceso a datos (modelo de dominio, repositorio y servicios). Este segundo punto de entrada permite, además, transformar dicho código a un nuevo lenguaje o framework especificado por el usuario. En ambos casos, el cuerpo de la petición debe ajustarse al DTO correspondiente, siendo su correcta definición un requisito fundamental para obtener una respuesta satisfactoria por parte de la herramienta.

La Figura 5.3 muestra cómo se establece la comunicación con el usuario dentro de la arquitectura general de LLMigrator-DB. El sistema ha sido implementado siguiendo una arquitectura limpia, con influencias de arquitectura hexagonal, al incorporar los conceptos de puertos y adaptadores para desacoplar la lógica de negocio de las interfaces externas [8]. En este esquema, los DTO recibidos por los controladores REST se transforman en modelos del dominio de la aplicación, los cuales encapsulan la información relevante para el proceso de migración.

Esta separación entre la capa de infraestructura y la lógica de negocio nos permite generar los prompts dinámicamente a partir de la información cumplimentada por el usuario, de forma que se garantiza la flexibilidad de la herramienta. Cualquier modificación futura en la información que introduce el usuario o en la estructura de los prompts al modelo puede ser gestionada localmente sin alterar el resto de los componentes del sistema.

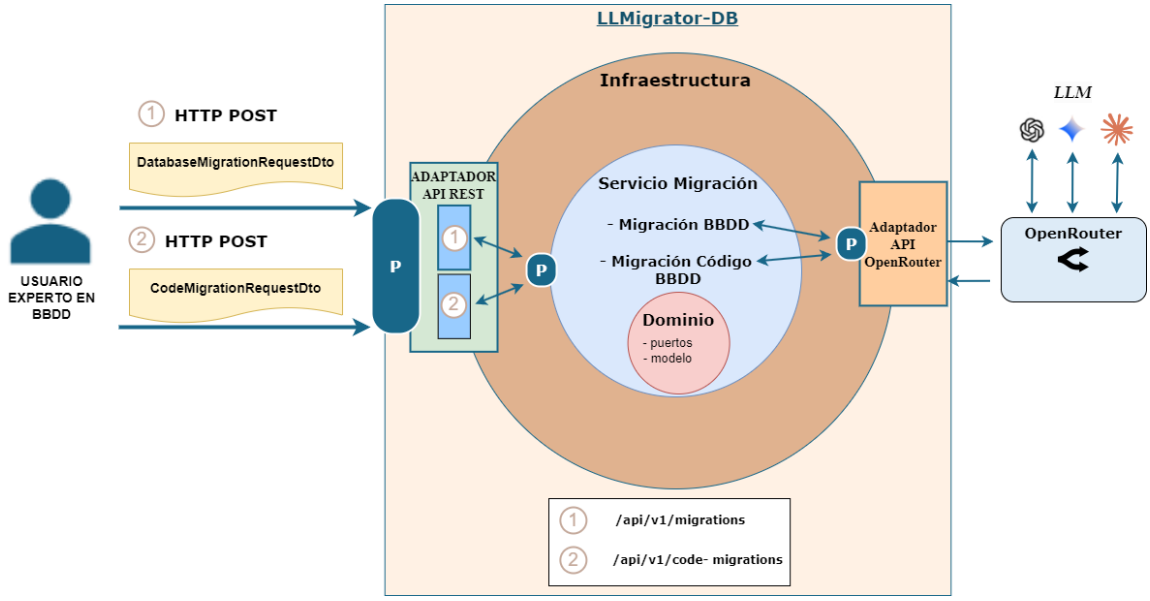

Figura 5.3: Arquitectura limpia aplicada en LLMigrator-DB y su integración con la comunicación del usuario y OpenRouter

Además de los dos endpoints ya mencionados, la herramienta expone un tercer endpoint en `/api/v1/supported-language-large-models`, que permite al usuario consultar los modelos de lenguaje actualmente soportados por el sistema. Esta funcionalidad proporciona un mayor grado de personalización de cara al usuario, ya que permite seleccionar el LLM más adecuado según el tipo de migración que se desea realizar, ya sea de bases de datos o de código.

5.3. Soporte de nuevos LLM; OpenRouter

OpenRouter proporciona un identificador único a cada uno de los LLM que soporta, lo cual permite integrarlos de forma muy sencilla en herramientas basadas en LLM. En el caso de LLMigrator-DB, este mecanismo permite añadir nuevos modelos siempre que estos admitan el uso de mensajes de sistema, con una modificación del código mínima y sin necesidad de modificar la lógica de negocio de la herramienta. Es por ello que uno de los aspectos más destacables de LLMigrator-DB es, precisamente, su capacidad para incorporar nuevos LLMs de cualquier proveedor mediante dos pasos muy simples:

- Añadir el identificador del nuevo LLM en OpenRouter al tipo enumerado que representa los modelos soportados por LLMigrator-DB. Este enumerado actúa como una lista de identificadores constantes que permite al sistema presentar al usuario las opciones disponibles y mantener el control de los modelos que pueden seleccionarse sin errores en tiempo de ejecución.

- Actualizar el fichero de configuración, añadiendo el identificador del nuevo LLM en OpenRouter. Esto permite a LLMigrator-DB crear dinámicamente una instancia del cliente de comunicación asociada a ese modelo.

Por ejemplo, si se desea utilizar el modelo *DeepSeek R1 0528*, primero debe añadirse su identificador (`deepseek/deepseek-r1-0528:free`) al enumerado del sistema, como se muestra en el Código 5.1. Posteriormente, dicho identificador debe añadirse también al fichero de configuración de Spring AI, como se ilustra en el Código 5.2.

Código 5.1: Enumerado asociado a los modelos de lenguaje soportados por la herramienta

```java
@RequiredArgsConstructor
@Getter
public enum SupportedLanguageLargeModels {
  DEEPSEEK_R1("deepseek/deepseek-r1-0528:free"), // <-- modelo añadido
  GEMINI_FLASH("google/gemini-2.0-flash-001"),
  GPT_4O_MINI("openai/gpt-4o-mini");

  private final String modelId;

  public String getModelId() {
    return modelId;
  }

  public static SupportedLanguageLargeModels fromModelId(String modelId) {
    for (SupportedLanguageLargeModels model : values()) {
      if (model.modelId.equalsIgnoreCase(modelId)) {
        return model;
      }
    }
    throw new IllegalArgumentException("Modelo no soportado: " + modelId);
  }
}
```

Código 5.2: Fichero de configuración de la herramienta

```yaml
spring:
  application:
    name: spring-migration-AI-api-\tool{}
  ai:
    openai:
      base_url: ${OPENAI_BASE_URL:https://openrouter.ai/api}
      api-key: ${OPENROUTER_API_KEY} # Variable de entorno
      chat:
        options:
          model: gpt-4o-mini # Modelo por defecto
    openrouter:
      chat:
        options:
          models:
            - deepseek/deepseek-r1-0528:free # <-- Modelo añadido
```

```
16            - google/gemini-2.0-flash-001
17            - openai/gpt-4o-mini
```

El lector interesado puede acudir al anexo A para encontrar una explicación detallada de cómo se ha integrado Spring AI con OpenRouter resultando muy simple la implementación y la adición de nuevos modelos a la herramienta. En las siguientes secciones, se introducirá el núcleo de la aplicación que es el código referente a los módulos de migración.

5.4. Migración de la base de datos

Cada fase del proceso de migración es implementada con una interacción concreta, utilizando plantillas de prompts adaptadas con el fin de proporcionar al LLM la información del usuario relativa al contexto de migración, siempre con el objetivo de obtener las salidas esperadas. Cada una de estas fases se encuentra orquestada de forma secuencial y está implementada haciendo uso de las técnicas de ingeniería de prompts previamente mencionadas para interactuar con el LLM de propósito general, siguiendo la arquitectura definida en el capítulo 4. En este sentido, todas las respuestas producidas por el LLM tienen un formato de salida estructurado, lo que permite a la herramienta extraer los campos relevantes para cada fase.

En esta sección, nos centraremos en los procesos relativos a la migración de una base de datos dentro de LLMigrator-DB, concretamente, la migración de esquema y datos, junto con la validación pertinente que requieren cada una de estas operaciones. Presentaremos la estructura de las plantillas de prompts diseñadas, pero no se han incluido los prompts completos los cuales el lector puede encontrar en el Anexo B.

5.4.1. Migración del esquema

Prompt contextual. La migración del esquema es iniciada con la emisión de un prompt contextual que establece la información del entorno origen y destino. Este prompt únicamente contiene un mensaje de sistema que incluye: el rol que va a tomar el LLM durante todo el proceso, los objetivos que se van a seguir en el proceso de migración los cuales se corresponden a las fases previamente planteadas, el contexto de la migración entendido como las tecnologías implicadas junto con los requisitos y restricciones del usuario que se han de tener en cuenta en la migración, una serie de instrucciones relacionadas con la interacción actual y las posteriores y, por último, el establecimiento del formato JSON de acuerdo al RFC 8259 [51]. Cabe destacar que este prompt no espera ningún tipo de respuesta por parte del LLM ya que es información contextual con la que se pretende establecer el contexto para la migración. Un extracto representativo de la plantilla del prompt contextual se muestra a continuación en el Código 5.3.

Código 5.3: Fragmento de la plantilla de prompt diseñada para la contextualización del modelo

```
String systemMessage =
"""
  Actúa como un experto en migración de bases de datos {sourceDatabaseType} a {
      ↪ targetDatabaseType}.

  **OBJETIVO**: Tu tarea es asistir en la migración, organizando el proceso en
      ↪ cuatro fases:
  - Migración del esquema...
  - Validación del esquema...
  - Migración de los datos...
  - Validación de la migración...

  **Contexto de la Migración**:
  - Sistema de base de datos origen: {sourceDatabaseType}
  - Sistema de base de datos destino: {targetDatabaseType}
  - Esquema original: {databaseSchema}
  - Requisitos y restricciones de la migración: {migrationRequirements}

  **Instrucciones**:
  - NO generes ninguna respuesta en este momento.
  - Usa esta información para adaptar el contexto en cada fase...
  - Asegúrate de tener en cuenta las decisiones y respuestas anteriores...
  - Sigue las instrucciones en cada fase para completar la migración con éxito.

  **IMPORTANTE**: Devuelve solo JSON puro (RFC8259) ...
""";
```

Prompt de migración del esquema. La plantilla de prompt asociada a la migración del esquema es la parte esencial del módulo de migración de esquema, ya que será instanciado para crear el prompt que se enviará al LLM. El propósito de este prompt es solicitar al LLM la conversión del esquema de una base de datos origen a un esquema válido para una base de datos destino, considerando tanto el sistema gestor de bases de datos utilizado como los requisitos funcionales que ya le fueron provistos en el prompt contextual. La construcción de este prompt se basa en un mensaje de sistema y un mensaje de usuario.

El mensaje de sistema nos sirve para indicar al LLM su actuación como experto en conversión de esquemas entre los dos tipos de SGBD implicados. Este prompt define una serie de criterios técnicos que son críticos en este proceso: el mapeo de entidades, las relaciones existentes entre ellas y restricciones (por ejemplo, el uso de un determinado tipo de datos para representar las claves primarias). Además, al LLM se le impone tanto una restricción de formato en la respuesta (con la finalidad de evitar cualquier error asociado al procesamiento de la respuesta y evitar ambigüedades semánticas) como que solo aborde la migración del esquema. Por otro lado, el mensaje de usuario complementa el contexto proporcionado en el mensaje de sistema, indicando la información específica del caso de uso del usuario, como el esquema de la base de datos de origen y los requisitos específicos del usuario para la migración. En definitiva, el diseño de este prompt nos

permite migrar el esquema de la base de datos asegurando aspectos como la trazabilidad del proceso de cara al usuario. A continuación, se presenta un extracto de la plantilla de prompt de migración de esquema en el Código 5.4.

Código 5.4: Extracto de la plantilla de prompt para la migración del esquema

```
// Mensaje de sistema: define el rol del modelo y delimita su tarea
String systemString =
    """
    Actúa como un experto en migración de esquemas de bases de datos {sourceDatabaseType} a {targetDatabaseType}.

    Tu tarea en esta fase es transformar el esquema de una base de datos {sourceDatabaseType} ...
    Para ello, debes tener en cuenta el mapping de entidades, relaciones...

    Información clave para la migración:
    - Base de datos de origen: {sourceDatabaseType}
    - Base de datos de destino: {targetDatabaseType}

    **IMPORTANTE**: La respuesta debe cumplir con el siguiente formato:
    {format}
    ...

    **IMPORTANTE**: Devuelve solo JSON puro (RFC8259)...

    **IMPORTANTE**: No abordes la migración de datos en esta fase, solo el esquema...
    """;

// Mensaje del usuario: se proporciona el caso concreto y los requisitos
String userString =
    """
    Debes migrar el esquema de una base de datos {sourceDatabaseType} ...

    - Esquema de la base de datos {sourceDatabaseType} original: {databaseSchema}
    - Requisitos y restricciones de la migración: {migrationRequirements}

    Crea el esquema de la base de datos {targetDatabaseType} en el lenguaje más apropiado para {targetDatabaseType}...
    """;
```

Validación de la migración del esquema

Prompt de validación del esquema migrado. Una vez que el modelo ha generado el nuevo esquema, se construye un prompt con la finalidad de proporcionar al usuario experto una guía detallada que le permita validar manualmente el esquema generado, es decir, un conjunto de directrices con las que comprobar la validez del nuevo esquema

respecto al original.

El prompt de validación se compone de un mensaje de sistema y un mensaje de usuario. El mensaje de sistema le indica al modelo que ha de actuar como un experto en validación de migraciones y se le solicita un conjunto de pautas y recomendaciones. Estas pautas han de considerar una serie de cuestiones, entre las que destacamos: los elementos del esquema de la base de datos se han migrado correctamente a la base de datos destino, la base de datos destino conserva las mismas restricciones que la base de datos origen, el esquema creado está optimizado para las consultas propuestas por el usuario y, por último, el LLM debe identificar posibles inconsistencias o riesgos en la migración. Además, de toda esta información también se le indican algunas restricciones en el formato de salida. Por otro lado, el mensaje de usuario introduce el script de migración generado en el paso anterior junto con la petición explícita de que genere acciones y sugerencias de validación que permitan discernir si el nuevo esquema implementa correctamente la lógica estructural del esquema origen.

A continuación, en el Código 5.5 se muestra un fragmento de la plantilla del prompt de validación, donde se pueden diferenciar los dos mensajes y cómo se complementan entre sí.

Código 5.5: Fragmento de la plantilla de prompt para la validación de la migración del esquema

```
//Mensaje de sistema
String systemString =
    """
    Actúa como un experto en validación de migración de esquemas de bases de datos.
    Tu tarea en esta fase es proporcionar pautas detalladas para que el usuario pueda comprobar...
    Ten en cuenta que estas pautas deben cumplir con los requisitos del usuario : {migrationRequirements}

    Para ello, te sugerimos algunas cuestiones a tener en cuenta:
    - Los elementos del esquema la base de datos {databaseSchema} se han migrado al esquema de la base de datos destino.
    - Las restricciones de la base de datos origen están reflejadas en la base de datos destino.
    - El esquema creado esté optimizado para consultas como las propuestas en {accessRequirements} y rendimiento en {targetDatabaseType}.
    - Posibles inconsistencias o riesgos en la migración.

    **IMPORTANTE**: Genera un listado de pasos con las recomendaciones para realizar pruebas manuales de validación.

    **IMPORTANTE**: No uses formato Markdown ni comillas triples. Solo devuelve texto plano...

    **IMPORTANTE**: Utiliza el script de migración de esquema proporcionado como base...
    {scriptForMigrateSchema}
```

```
20     """;
21 // Mensaje de usuario
22
23 String userString =
24     """
25     Se proporciona el siguiente script de migración de esquema:
26
27     {scriptForMigrateSchema}
28     Genera un conjunto de pautas y recomendaciones detalladas para que un ↩
           ↪ usuario pueda validar manualmente:
29     - La transformación del esquema desde {sourceDatabaseType} a {↩
           ↪ targetDatabaseType}.
30     - La correcta implementación y optimización del nuevo esquema en la base de↩
           ↪  datos de destino.
31
32     Incluye pasos de verificación, sugerencias de pruebas manuales y puntos de ↩
           ↪ control...
33     """;
```

5.4.2. Migración de los datos

La migración de los datos se inicia una vez que el usuario ha aplicado las pautas de validación manual del esquema destino y ha verificado que cumple con los requisitos esperados. Solo en caso de validación satisfactoria, el usuario ha de proceder con la ejecución del proceso de migración de datos.

Prompt de migración de datos. En esta etapa, se construye un nuevo prompt dirigido a generar el código necesario para trasladar los datos desde la base de datos origen hacia el nuevo sistema, respetando el esquema previamente generado. Este prompt, al igual que los anteriores, también está compuesto de un mensaje de sistema y un mensaje de usuario.

El mensaje de sistema especifica que el LLM ha de actuar como un experto en migración de datos, y se le indica que su respuesta debe consistir en un script funcional para realizar dicha migración y una explicación de la estrategia de migración utilizada para migrar los datos. En este mensaje se añaden algunas recomendaciones como: asegurar la integridad y consistencia de los datos migrados, garantizar un proceso de migración eficiente junto con una serie de restricciones para el formato de salida.

Por otro lado, el mensaje de usuario hace hincapié en los requisitos específicos definidos por el usuario (por ejemplo, transformaciones de tipos, compatibilidad con agregaciones, etc.) e incluye tanto el esquema original como el generado. El resultado es un script de código junto con su explicación técnica que, en caso de que el usuario aporte los parámetros de conexión, será directamente ejecutable. A continuación se muestra un extracto de dicho prompt en el Código 5.6.

Código 5.6: Fragmento de la plantilla de prompt para la migración de los datos

```
// Mensaje de sistema
String systemString =
  """
  Actúa como un experto en migración de datos de bases de datos {sourceDatabaseType} a {targetDatabaseType}.
  En esta fase, tu objetivo es migrar los datos de la base de datos...
  Para ello, debes, una vez, obtenido el esquema de destino, generar un código para migrar todos los datos...

  1. Asegurar la integridad referencial y consistencia de los datos migrados.
  2. Optimizar el proceso para garantizar la eficiencia en la migración.
  3. Producir un código que permita la migración de datos de la base de datos ...

  **Información clave para la migración:**
  - Esquema de la base de datos origen: {databaseSchema}
  - El esquema de la base de datos destino generado a partir del script: {scriptForTargetSchema}


  **IMPORTANTE:** La respuesta debe cumplir con el siguiente formato:
  {format}
  ...

  **IMPORTANTE:** Devuelve solo JSON puro (RFC8259), sin comillas triples, sin markdown...

  **IMPORTANTE:** Si la migración requerida se realiza desde una base de datos ...
  En cualquier otro caso, el script de migración de datos debe ser un script {language}.
  """;

// Mensaje de usuario
String userString =
  """
  Debes migrar los datos de una base de datos {sourceDatabaseType} a una base de datos {targetDatabaseType}.

  A continuación, se detallan los parámetros específicos...

  - Esquema de la base de datos {sourceDatabaseType} original:
  {databaseSchema}
  - Requisitos y restricciones de la migración:
  {migrationRequirements}

  Genera un script para migrar los datos a {targetDatabaseType}...
  """;
```

Validación de los datos y del proceso

La última etapa del proceso de migración de bases de datos es la validación de la migración completa. A diferencia de la validación manual del esquema realizada anteriormente, esta fase se centra en una comprobación automatizada que verifica que tanto el esquema como los datos han sido correctamente migrados al SGBD destino. Esta validación sirve para asegurar que se mantiene la coherencia y la consistencia de los datos en la base de datos destino.

Prompt de validación de datos. El cuarto y último prompt está dividido en dos mensajes. El mensaje de sistema le indica al modelo que ha de generar un conjunto de consultas comparativas ejecutables en ambos sistemas que permitan al usuario discernir que los datos son equivalentes y han sido correctamente migrados. Además, se solicita un análisis detallado para evaluar la integridad referencial, los posibles errores de mapeo y la idoneidad de las estructuras generadas. Además de estas directrices, también se le proporciona información de interés como los SGBD implicados, el esquema origen y los requisitos específicos del usuario para la migración. Por último, se le indica que la respuesta debe seguir un formato específico.

Por otro lado, el mensaje de usuario contiene, además de la información del usuario, el script de migración de datos previamente generado, lo cual permite al modelo analizar cómo se han migrado los datos. A continuación, se presenta la estructura de la plantilla de prompt definida en el Código 5.7.

Código 5.7: Fragmento del prompt para la validación de la migración de los datos

```
// Mensaje del sistema
String systemString =
    """
    Actúa como un experto en validación de migración de bases de datos {sourceDatabaseType} a {targetDatabaseType}.

    En esta fase, tu tarea es verificar la consistencia de los datos migrados ...

    1. Comparar los elementos de la base de datos...
    2. Verificar la integridad referencial y consistencia de los datos migrados
    3. Comprobar que las mismas consultas realizadas en la base de datos origen y destino devuelven los mismos resultados

    Información clave para la validación:
    - Base de datos de origen: {sourceDatabaseType}
    - Base de datos de destino: {targetDatabaseType}
    - Esquema de la base de datos original: {databaseSchema}
    - Requisitos y restricciones de la migración: {migrationRequirements}

    **IMPORTANTE**: Utiliza el script de migración de datos generado en la fase anterior para validar la migración: {scriptForTargetData}

     **IMPORTANTE**: Es crucial que la respuesta debe cumplir con el siguiente
```

```
        ↪ formato:
    {format}

     **IMPORTANTE**: Devuelve solo JSON puro (RFC8259)...
    """;

// Mensaje del usuario
String userString =
    """
    Debes validar la migración de datos y esquema de una base de datos {↩
        ↪ sourceDatabaseType} a {targetDatabaseType}.
    La validación de los datos debe comprobar...

    A continuación, se detallan los parámetros específicos...
    - Esquema {sourceDatabaseType} original: {databaseSchema}
    - Requisitos y restricciones de la migración: {migrationRequirements}
    - Requisitos de la aplicación (consultas más realizadas, índices...): {↩
        ↪ accessRequirements}

    - Requisitos de la aplicación (consultas más realizadas, índices...): {↩
        ↪ accessRequirements}
    ...
    Genera un script para validar la migración... junto con una explicación ↩
        ↪ detallada del proceso.
    """;
```

5.5. Migración del código

El proceso de migración de código ha sido diseñado en nuestra herramienta como una etapa opcional que se puede iniciar una vez finalizada la migración de la base de datos. Su funcionalidad consiste en migrar la capa de persistencia de una aplicación existente a una tecnología distinta, adaptando tanto el modelo del dominio como los repositorios y servicios asociados. La migración del código se aborda en dos fases principales: la conversión del modelo de dominio y sus respectivos repositorios, y la migración de la lógica de negocio. A continuación, se describe de forma resumida la estructura de las plantillas de prompts diseñadas para esta etapa, y se puede acudir al Anexo C para una descripción más detallada.

Prompt contextual de migración de codigo. La migración del código es iniciada con la emisión de un prompt contextual que proporciona toda la información necesaria sobre el entorno origen y destino. Este prompt no espera respuesta, sino que configura el contexto de la migración para las posteriores fases de generación de código. En este contexto se le indica al LLM su rol, el objetivo, el contexto asociado a la migración actual y algunas instrucciones adicionales. A continuación, se presenta el Código 5.8 con un fragmento de la plantilla prompt donde se reflejan los detalles más relevantes.

Código 5.8: Fragmento de la plantilla de prompt utilizada para la contextualización del LLM

```
// Mensaje del sistema
String systemString =
    """
    Actúa como un experto en migración de código. Tu objetivo es guiar la migración de una capa de persistencia completa...

    El proceso se dividirá en varias fases:
    1. **Migración del Modelo de Dominio y Repositorios:** ...
    2. **Migración de Lógica de Negocio de los servicios de la aplicación** ....

    Contexto General de la Migración:
    - **Lenguaje Origen:** {sourceLanguage}
    - **Lenguaje Destino:** {targetLanguage}
    - **Framework Origen:** {sourceFramework}
    - **Framework Destino:** {targetFramework}
    - **Base de Datos Origen:** {sourceDatabase}
    - **Base de Datos Destino:** {targetDatabase}
    - **Arquitectura de Salida del código Deseada:** {outputFormat}
    - **Esquema de Base de Datos Destino:** {targetSchema}
    - **Requisitos Funcionales Clave:** {functionalRequirements}

    Instrucciones Generales:
    - NO generes ninguna respuesta ahora. Asimila este contexto.
    - En los siguientes pasos, te proporcionaré el código fuente específico a migrar.
    - Asegúrate de que el código generado sea idiomático...
    - Presta especial atención a la gestión de transacciones, tipos de datos...
    """;
```

Prompt de migración del modelo del dominio y repositorios. Tras haber enviado el prompt contextual al LLM, se comienza con la migración del modelo del dominio (clases entidad) y los repositorios que acceden a la base de datos. Para ello, se construye un prompt compuesto por un mensaje de sistema y otro de usuario.

El mensaje de sistema indica al modelo su rol como experto en migración de capas de persistencia e incluye instrucciones sobre cómo adaptar anotaciones, tipos de datos y convenciones de nombrado en el sistema destino. Además, se le exige mantener la consistencia entre las entidades, estructurar el código de acuerdo al patrón arquitectónico e implementar el código de acuerdo a los requisitos funcionales definidos (por ejemplo, el uso de paginación). Por otro lado, el mensaje de usuario le proporciona el código fuente (modelo de dominio y repositorios), detallando la tecnología origen y solicitando la conversión a la tecnología destino.

Al igual que en las respuestas anteriormente presentadas, también se solicita al modelo una respuesta en el formato estructurado JSON que contenga el código migrado y una explicación de las decisiones tomadas en el mismo. A continuación, se presenta un extracto de la plantilla de prompt en el Código 5.9 el cual resulta relevante para

este caso de uso.

Código 5.9: Fragmento de la plantilla del prompt para migrar el modelo del dominio y los repositorios asociados

```
// Mensaje del sistema
String systemString =
    """
    Actúa como un experto en migración de capas de persistencia, especializado en Modelos de Dominio y Repositorios.
    Tu tarea es migrar el código del modelo de dominio y del repositorio...

    Instrucciones Clave:

    1. **Modelo de Dominio:** Convierte las clases de entidad de {sourceLanguage} a {targetLanguage}...
    2. **Repositorios/DAOs:** Migra las interfaces o clases de acceso a datos ...
    3. **Cohesión:** Asegúrate de que los repositorios migrados operen correctamente sobre los modelos de dominio migrados...
    4. **Arquitectura:** Organiza el código generado siguiendo el patrón de diseño `{outputFormat}`.
    5. **Requisitos Funcionales:** Implementa los siguientes requisitos en el código migrado: `{functionalRequirements}`. Esto puede incluir paginación, joins, etc.
    6. **IDs:** Asume que los identificadores (IDs) de las entidades deben mantenerse consistentes...

    **IMPORTANTE**: La respuesta debe ser un JSON puro (RFC8259)...
    {format}
    """;

// Mensaje del usuario
String userString =
    """
    Migra el siguiente código de {sourceFramework} a {targetFramework}.

    **Código del Modelo de Dominio Origen (en {sourceLanguage}):**
    ```
    {domainModelCode}
    ```
    **Código del Repositorio Origen (en {sourceLanguage}):**
    ```
    {repositoryCode}
    ```
    Genera el código equivalente para `{targetFramework}` en el lenguaje {targetLanguage}...
    """;
```

Tras haber migrado el código referente al modelo del dominio y sus repositorios, comienza la segunda fase, centrada en migrar la lógica de negocio implementada en los servicios. El punto de partida, y base contextual, es el código migrado en el paso

anterior, lo que permite al modelo razonar sobre cómo adaptar la capa de servicio al nuevo entorno de ejecución.

Prompt de migración de lógica del negocio. Esta plantilla de prompt también está compuesta por un mensaje de sistema y otro de usuario. El mensaje del sistema guía al modelo en la tarea de adaptación del código, detallando instrucciones como el cumplimiento de requisitos funcionales (por ejemplo, ordenación, filtrado, etc.). Además, se hace especial énfasis en aspectos como la gestión de transacciones y la integración adecuada del código de los servicios con los repositorios ya migrados. Por su parte, el mensaje de usuario proporciona al modelo el código correspondiente a la capa de servicio que hace uso de los repositorios en el sistema origen. La salida generada sigue el formato estructurado ya mencionado antes. A continuación, se muestra un extracto representativo de dicha plantilla en el Código 5.10.

Código 5.10: Fragmento de la plantilla de prompt para migrar la lógica de negocio

```
// Mensaje del sistema
String systemString =
    """
    Actúa como un experto en migración de lógica de negocio entre diferentes ↩
        ↪ lenguajes y frameworks.

    Tu tarea es migrar código de perteneciente a la capa de servicios...

    El código que utiliza estos repositorios y el modelo de dominio ya migrado ↩
        ↪ es el siguiente:
    {scriptForDomainAndRepo}

    1. **Adaptación de Servicios:** migra el código de los servicios...
    2. **Requisitos Funcionales:** Implementa los siguientes requisitos: {↩
        ↪ functionalRequirements}
    3. **Ejemplo de Uso:** Utiliza el ejemplo de uso ....
    IMPORTANTE: La respuesta debe ser un JSON puro (RFC8259)...
    {format}
    """;

// Mensaje del usuario
String userString =
    """
    Continuando con la migración a {targetFramework}, ahora migra el código ↩
        ↪ referente a la lógica de negocio...

    **Código de la capa de servicio en {sourceLanguage}/{sourceFramework}**
    ```
    {businessLogicCode}
    ```

    **Ejemplo de Uso en la Capa de Servicio Origen (opcional):**
    ```
    {usageExampleCode}
```
```

```
31      ```
32
33      Genera el código equivalente para `{targetFramework}` en {targetLanguage
            ↪ }...
34      """;
```

En conclusión, la migración del código se aborda de forma integral, proporcionando una solución que no solo transforma el modelo del dominio, sino que también adapta los repositorios asociados y la lógica de negocio que los utiliza. Este proceso tiene en cuenta los requisitos funcionales definidos por el usuario, asegurando que la nueva implementación mantenga la semántica original del sistema.

6. Validación

En este capítulo se presenta el caso de estudio Eventalia utilizado para validar el funcionamiento de la herramienta LLMigrator-DB en un escenario no trivial.

6.1. Descripción del Caso de estudio

El caso de estudio se basa en la aplicación *Eventalia* que el alumno había desarrollado en la asignatura *Arquitectura del Software* [53]. El objetivo de la aplicación es gestionar eventos, espacios y reservas. Había sido implementado siguiendo una arquitectura de microservicios, con servicios especializados (eventos, espacios, reservas), persistencia heterogénea (MySQL [54] y MongoDB [55]), y mecanismos de comunicación síncrona (Retrofit [56]) y asíncrona (RabbitMQ [57]) entre los microservicios. Cada microservicio había sido implementado con tecnologías distintas (Java EE [58] y Spring Boot [12]), y desplegado de forma orquestada con Docker Compose [59]. Antes de plantear el entorno de la migración que vamos a llevar a cabo, expliquemos el dominio de cada microservicio para contextualizar el caso de estudio.

El *microservicio de espacios* está dedicado a la administración de los espacios físicos donde se pueden llevar a cabo los eventos. La entidad principal son los espacios físicos y cada uno de los espacios puede tener un conjunto de puntos de interés asociados, los cuales se encuentran cerca del mismo. Para la persistencia de este microservicio se hace uso de Java Persistence API (JPA) [60], concretamente el que ofrece EclipseLink [61] para gestionar la base de datos MySQL.

El *microservicio de eventos* se encarga de la gestión y planificación de eventos. Su modelo del dominio está conformado por dos entidades principales: eventos y espacios físicos. Como se puede observar, se produce una duplicación parcial del modelo debido a la autonomía del microservicio de eventos, y se hace con el fin de operar de forma resiliente en caso de caídas. Esta duplicación se mantiene sincronizada mediante comunicación asíncrona entre el microservicio de eventos y el microservicio de espacios. Respecto a la persistencia en este microservicio, se hace uso de Spring Data JPA [62] el cual utiliza Hibernate [63] como proveedor de JPA para gestionar la persistencia a través de una base de datos MySQL.

Por último, el *microservicio de reservas* se encarga de gestionar las inscripciones de usuarios en eventos. Su modelo del dominio incluye las entidades reserva y evento. Al igual que en el microservicio de eventos, en este también se produce replicación de información, que también es gestionada por comunicación asíncrona. En cuanto a la

gestión de la persistencia, este microservicio hace uso de Spring Data MongoDB [64] y resulta de interés para llevar a cabo una migración heterogénea.

Esta aplicación proporciona un escenario representativo que nos permite evaluar el comportamiento de LLMigrator-DB en diferentes tipos de migraciones. Concretamente, en este capítulo se muestran las migraciones del microservicio de eventos y del microservicio de reservas, ya que son casos de estudio de interés al presentar una migración homogénea y heterogénea, respectivamente, ya que las dos bases de datos origen serán migradas a PostgreSQL [65]. Este caso de estudio nos ha permitido llevar a cabo un proceso de migración completo y evaluar el grado de automatización.

En el repositorio de GitHub [16] asociado al TFG se puede acceder a la información de entrada utilizada para la migración de cada uno de los microservicios involucrados en el caso de estudio, así como los resultados de las pruebas realizadas durante el proceso de validación de la herramienta.

6.2. Migración del microservicio de eventos

La migración del microservicio de eventos se ha realizado utilizando Gemini-2.0 Flash. Se trata de una migración homogénea entre dos bases de datos relacionales: de MySQL a PostgreSQL. A través de esta migración, se comprobará en qué medida la herramienta es capaz de trasladar correctamente el esquema de la base de datos, resolver diferencias sintácticas entre dialectos SQL, y conservar el comportamiento funcional del microservicio. La elección del microservicio de eventos es especialmente relevante porque presenta relaciones entre entidades, uso de tipos enumerados y estructuras embebidas, lo que permite validar la herramienta en un escenario representativo.

6.2.1. Migración de la base de datos

Se ha solicitado a cada uno de los LLMs que realice una migración automática del esquema y genere un script de migración de los datos a partir de la información de entrada que proporciona el usuario y sirve para rellenar las plantillas presentadas en el anterior capítulo. A continuación se describe de forma resumida la información de entrada, que está detallada al completo en la colección de pruebas del repositorio de GitHub [16].

La información de entrada se emplea para generar en tiempo de ejecución los prompts a partir de las plantillas previamente definidas, los cuales serán enviados al LLM. Para iniciar el proceso de migración, se proporciona a LLMigrator-DB información clave de los SGBDR implicados. A continuación, se describen los elementos que conforman la información de entrada proporcionada por el usuario a la herramienta.

Primero, se especifica el SGBD origen (MySQL) y el sistema gestor destino (PostgreSQL), lo cual permite que el modelo de lenguaje tenga en cuenta las diferencias sintácticas y semánticas de cada sistema gestor. Luego se incluyeron los requisitos funcionales derivados de las operaciones más frecuentes que ejecuta el microservicio de

eventos en su base de datos. Las consultas en cuestión permitían obtener los eventos activos de un mes o comprobar si un espacio físico tiene alguna ocupación de un evento, entre otras operaciones. Estas operaciones de acceso sirven para orientar al modelo para optimizar el nuevo esquema en función de su uso real. Por otro lado, también se le proporcionó a la herramienta una serie de requisitos para la migración, como los referentes a la conversión de tipos, con el fin de que el modelo realice una adaptación precisa a las convenciones propias de PostgreSQL. Por último, se indicó el esquema original de la base de datos.

La migración realizada por LLMigrator-DB fue completa. El modelo interpretó correctamente tanto la estructura relacional original como las particularidades del nuevo sistema gestor, generando un script SQL adaptado y dos scripts en Python para la migración y validación de datos, respectivamente.

El Código 6.1 presenta el script de migración de esquema generado. Se puede observar que (i) los tipos específicos de MySQL se convierten a sus equivalentes en PostgreSQL, por ejemplo, el tipo `bit(1)` de MySQL se transforma correctamente al tipo `BOOLEAN` de PostgreSQL, y (ii) que se crean índices en las columnas más utilizadas en las consultas que se le han adjuntado, lo cual contribuye a mejorar el rendimiento de las operaciones.

Código 6.1: Script de migración de esquema producido por LLMigrator-DB utilizando Gemini-2.0 Flash

```
-- Crear la tabla espacio_fisico
CREATE TABLE espacio_fisico (
    id UUID PRIMARY KEY,
    capacidad INTEGER,
    direccion VARCHAR(255),
    estado VARCHAR(255),
    nombre VARCHAR(255)
);

-- Crear la tabla evento
CREATE TABLE evento (
    id UUID PRIMARY KEY,
    cancelado BOOLEAN,
    categoria VARCHAR(255),
    descripcion TEXT,
    nombre VARCHAR(255),
    fecha_fin TIMESTAMP WITHOUT TIME ZONE,
    fecha_inicio TIMESTAMP WITHOUT TIME ZONE,
    organizador VARCHAR(255),
    plazas INTEGER,
    espacio_fisico_id UUID,
    CONSTRAINT fk_espacio_fisico
        FOREIGN KEY (espacio_fisico_id) REFERENCES espacio_fisico(id)
);

-- Crear índices para optimizar las consultas
CREATE INDEX idx_evento_fecha_inicio ON evento (fecha_inicio);
CREATE INDEX idx_evento_fecha_fin ON evento (fecha_fin);
```

```
CREATE INDEX idx_evento_espacio_fisico_id ON evento (espacio_fisico_id);
CREATE INDEX idx_evento_cancelado ON evento (cancelado);
```

El código 6.2 presenta un fragmento del script Python generado por la herramienta para la migración de los datos. Se observa que se utilizan las bibliotecas pertinentes para establecer la conexión con ambas bases de datos. El script también incluye el mapeo de tipos, conversión de identificadores, transformación de campos booleanos y un mecanismo de protección frente a duplicados mediante `ON CONFLICT DO NOTHING`.

Código 6.2: Código de migración de datos producido por la herramienta LLMigrator-DB usando Gemini-2.0-Flash

```python
# Configuración de la base de datos MySQL (origen)
...
# Configuración de la base de datos PostgreSQL (destino)
...
def migrate_data():
    try:
        # Conexión a MySQL...
        # Conexión a PostgreSQL...

        # Migración de la tabla espacio_fisico
        mysql_cursor.execute("SELECT id, capacidad, direccion, estado, nombre FROM espacio_fisico")
        espacios_fisicos = mysql_cursor.fetchall()

        for espacio_fisico in espacios_fisicos:
            postgres_cursor.execute(
                """
                INSERT INTO espacio_fisico (id, capacidad, direccion, estado, nombre)
                VALUES (%s, %s, %s, %s, %s)
                ON CONFLICT (id) DO NOTHING;
                """, # Evitar duplicados
                (
                    uuid.UUID(espacio_fisico['id']),
                    espacio_fisico['capacidad'],
                    espacio_fisico['direccion'],
                    espacio_fisico['estado'],
                    espacio_fisico['nombre']
                )
            )

        # Migración de la tabla evento
        mysql_cursor.execute("""
            SELECT id, cancelado, categoria, descripcion, nombre,
                  fecha_fin, fecha_inicio, organizador, plazas, espacio_fisico_id
            FROM evento
        """)
        eventos = mysql_cursor.fetchall()

```

```
        for evento in eventos:
            postgres_cursor.execute(
                """
                INSERT INTO evento (
                    id, cancelado, categoria, descripcion, nombre,
                    fecha_fin, fecha_inicio, organizador, plazas, ↩
                        ↪ espacio_fisico_id
                )
                VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
                ON CONFLICT (id) DO NOTHING;
                """, # Evitar duplicados
                (
                    uuid.UUID(evento['id']),
                    bool(evento['cancelado']),
                    evento['categoria'],
                    evento['descripcion'],
                    evento['nombre'],
                    evento['fecha_fin'],
                    evento['fecha_inicio'],
                    evento['organizador'],
                    evento['plazas'],
                    uuid.UUID(evento['espacio_fisico_id'])
                )
            )

        # Confirmar los cambios y cerrar las conexiones
        ...
        print("Migración de datos completada exitosamente.")
    except mysql.connector.Error as mysql_err:
        print(f"Error de MySQL: {mysql_err}")
        # gestión de errores...
    except psycopg2.Error as pg_err:
        # gestión de errores...
    except Exception as e:
        # gestión de errores...

if __name__ == "__main__":
    migrate_data()
```

En tercer y último lugar, se generan una serie de pautas para validar de forma manual la migración del esquema y además, se genera un script de validación en Python para verificar la correcta migración de los datos. Esta validación se utiliza con el fin de verificar la integridad referencial, además de comprobar que las consultas más utilizadas de la aplicación devuelven lo mismo. El Código 6.3 muestra un fragmento del script de validación generado.

Código 6.3: Fragmento del script de validación producido por la herramienta LLMigrator-DB con el modelo Gemini

```
# Importaciones de librerías necesarias
# Configuración de la base de datos MySQL (origen)
```

```
# Configuración de la base de datos PostgreSQL (destino)

def validate_migration():
    try:
        # Conexión a MySQL
        # Conexión a PostgreSQL
        # 1. Comparar el número de registros en cada tabla ...
        assert mysql_espacio_fisico_count == postgres_espacio_fisico_count, \
            "Error: El número de registros en espacio_fisico no coincide."
        ...
        assert mysql_evento_count == postgres_evento_count, \
            "Error: El número de registros en evento no coincide."

        # 2. Validar la integridad referencial en evento.espacio_fisico_id
        postgres_cursor.execute("""
            SELECT COUNT(*) AS count
            FROM evento
            WHERE espacio_fisico_id NOT IN (SELECT id FROM espacio_fisico)
        """)

        # 3. Validar las consultas más frecuentes
        def execute_and_compare_query(mysql_query, postgres_query, params=None)↩
            ↪ :

            # Obtención de los datos con cada query...

            assert len(mysql_result) == len(postgres_result), \
                "Error: Los resultados de la consulta no coinciden en número de ↩
                    ↪ filas."

            for i in range(len(mysql_result)):
                assert mysql_result[i].keys() == postgres_result[i].keys(), \
                    f"Error: Las columnas de la fila {i} no coinciden."
                for key in mysql_result[i].keys():
                    assert str(mysql_result[i][key]) == str(postgres_result[i][↩
                        ↪ key]), \
                        f"Error: El valor de la columna '{key}' en la fila {i} no ↩
                            ↪ coincide."

        # Consultas frecuentes
        # 3.1
        mysql_query_1 = """
            SELECT * FROM evento
            WHERE cancelado = 0 AND YEAR(fecha_inicio) = %s AND MONTH(↩
                ↪ fecha_inicio) = %s
        """
        postgres_query_1 = """
            SELECT * FROM evento
            WHERE cancelado = FALSE
              AND EXTRACT(YEAR FROM fecha_inicio) = %s
              AND EXTRACT(MONTH FROM fecha_inicio) = %s
```

```
        """
        execute_and_compare_query(mysql_query_1, postgres_query_1, params=(2023, 10))
        # resto de consultas...
        print("Validación de la migración completada exitosamente.")
    #Gestión de errores
    finally:
        # Cierre de conexiones

if __name__ == "__main__":
    validate_migration()
```

En resumen, podemos observar cómo la migración realizada por el modelo Gemini-2.0-Flash evidencia una comprensión sólida tanto del esquema relacional original como de las particularidades dialectales del sistema gestor destino. La conversión de tipos, la preservación de la integridad referencial y la inclusión de índices garantizan que podamos migrar la base de datos de forma correcta, e incluso puede llegar a aumentar el rendimiento de las consultas. En este sentido, podemos considerar que la migración de la base de datos es completa, automatizable y coherente de acuerdo con los requisitos especificados desde el punto de vista de un usuario experto.

6.2.2. Migración del código de la capa de acceso a datos

En este apartado describimos cómo, una vez realizada la migración de esquema y datos, la herramienta LLMigrator-DB es capaz de migrar el código fuente que compone la capa de acceso a datos de nuestro sistema. Esta capa incluye las entidades del dominio, los repositorios asociados a dichas entidades y las consultas personalizadas que se realicen en los mismos. Por añadidura, la herramienta también permite adaptar la lógica de negocio con el fin de que utilice los repositorios y entidades adaptadas.

La migración del código del microservicio de eventos que accedía a MySQL conserva el mismo lenguaje de programación (Java) y el mismo framework de persistencia (Spring Data JPA) para PostgreSQL, y se han adaptado las consultas de los repositorios. A continuación, se describe de forma resumida el contenido de la entrada que se puede consultar en la colección de pruebas del repositorio de GitHub [16].

Para instanciar las plantillas, se tuvo que indicar las tecnologías origen y destino: SGBDRs MySQL y PostgreSQL, Java y Spring Data JPA en ambos casos. También se proporcionó el código fuente de la capa de persistencia. Este incluía las entidades del dominio con sus anotaciones JPA, además de los repositorios con sus definiciones de consultas en el Lenguaje de Consulta de Persistencia de Java (JPQL). Asimismo, se incorporó el código referente a los servicios que hacían uso de dichas consultas. Además, también fue necesario especificar el formato de la salida. Se exigió que se respetaran los principios del patrón *Repository*, la inversión de dependencias y el mantenimiento de las buenas prácticas aplicadas en el sistema origen con el fin de que sea compatible con el resto de componentes de la arquitectura. Por último, fue necesario proporcionar

el esquema de base de datos ya migrado a PostgreSQL, lo cual permite guiar a la herramienta en la adaptación de las consultas y anotaciones.

LLMigrator-DB devolvió como resultado el código adaptado y una explicación asociada al mismo, en el formato correcto. En cuanto al código, incluía las entidades del dominio adaptadas a PostgreSQL, junto con sus correspondientes repositorios JPA y los servicios adaptados para usar los repositorios migrados. A continuación, detallamos algo más esta migración.

La herramienta generó las entidades JPA adaptadas para PostgreSQL. El extracto de Código 6.4 muestra la migración de las clases del dominio por parte de la herramienta. Estas clases incorporan anotaciones propias de JPA que aseguran un mapeo correcto al nuevo esquema relacional. Por ejemplo, se puede observar cómo se han conservado las entidades embebidas (`@Embedded`) y las relaciones muchos a uno (`@ManyToOne`).

Código 6.4: Modelo del dominio migrado para PostgreSQL usando Spring Data JPA

```
@Entity
@Table(name = "evento")
public class Evento {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    @Column(name = "id", columnDefinition = "UUID")
    private UUID id;

    @Column(name = "nombre", nullable = false)
    private String nombre;

    @Lob
    @Column(name = "descripcion", nullable = false)
    private String descripcion;

    @Column(name = "organizador", nullable = false)
    private String organizador;

    @Column(name = "plazas", nullable = false)
    private int plazas;

    @Column(name = "cancelado", nullable = false)
    private boolean cancelado;

    @Embedded
    private Ocupacion ocupacion;

    @Enumerated(EnumType.STRING)
    @Column(name = "categoria")
    private Categoria categoria;

    // Constructor por defecto, con todos los argumentos y personalizados...
    // Getters y Setters omitidos para brevedad (puedo incluirlos si los ↩
        ↪ necesitas)...
```

```
    // Método equals y hashcode...
}

@Embeddable
public class Ocupacion {
    @Column(name = "fecha_inicio")
    private LocalDateTime fechaInicio;

    @Column(name = "fecha_fin")
    private LocalDateTime fechaFin;

    @ManyToOne
    @JoinColumn(name = "espacio_fisico_id")
    private EspacioFisico espacioFisico;
    //Constructores por defecto y con todos los argumentos...
    // Getters y Setters omitidos para brevedad...
    //Método equals y hashcode...
}

@Entity
@Table(name = "espacio_fisico")
public class EspacioFisico {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    @Column(name = "id", columnDefinition = "UUID")
    private UUID id;

    @Column(name = "nombre", nullable = false)
    private String nombre;

    @Column(name = "capacidad", nullable = false)
    private int capacidad;

    @Enumerated(EnumType.STRING)
    @Column(name = "estado", nullable = false)
    private EstadoEspacioFisico estado;

    @Column(name = "direccion", nullable = false)
    private String direccion;
    // Constructores por defecto y con todos los argumentos
    // Getters y Setters omitidos para brevedad...
    // Métodos equals y hashcode
}

public enum Categoria {
    ACADEMICO,
    ...,
    OTROS
}

public enum EstadoEspacioFisico {
```

```
    ACTIVO,
    CERRADO_TEMPORALMENTE
}
```

En segundo lugar, se muestra la conversión del contenido de los repositorios en los que podemos observar cómo la herramienta ha mantenido la semántica de las consultas originales y las ha adaptado a la sintaxis JPQL compatible con PostgreSQL. Un ejemplo representativo es el uso de expresiones propias de PostgreSQL como `EXTRACT(YEAR FROM...)` en las consultas que utilizan fechas como datos. El Código 6.5 presenta un extracto del código migrado relativo a los repositorios.

Código 6.5: Repositorios migrados al uso de PostgreSQL

```
package eventos.infraestructura.repositorios.eventos;
@Repository
public interface RepositorioEventosJpa extends JpaRepository<Evento, UUID> {
    @Query("""
        SELECT e
        FROM Evento e
        WHERE e.ocupacion IS NOT NULL AND e.cancelado = FALSE
          AND EXTRACT(YEAR FROM e.ocupacion.fechaInicio) = :anio
          AND EXTRACT(MONTH FROM e.ocupacion.fechaInicio) = :mes
    """)
    Page<Evento> findEventosPorMesYAnio(
        @Param("mes") int mes,
        @Param("anio") int anio,
        Pageable pageable
    );
    @Query("""
        SELECT CASE WHEN (COUNT(e) > 0) THEN TRUE ELSE FALSE END
        FROM Evento e
        WHERE e.ocupacion.espacioFisico.id = :idEspacio
          AND e.ocupacion.fechaFin > CURRENT_TIMESTAMP
    """)
    boolean isOcupacionActiva(@Param("idEspacio") UUID idEspacio);
    // Otros métodos...
}

@Repository
public interface RepositorioEspaciosJpa extends JpaRepository<EspacioFisico, ↩
    ↪ UUID> {}
```

Por último, la herramienta también devuelve el código de los servicios adaptados, incluyendo la lógica de validación, el uso de transacciones (`@Transactional`) y la publicación de eventos mediante RabbitMQ. En este sentido, se evidencia el bajo acoplamiento existente entre la lógica de negocio y los repositorios adaptados, garantizando que la migración del código no afecte a la funcionalidad general del microservicio. A continuación, el Código 6.6 muestra un extracto del código generado, en el que se puede observar cómo los servicios hacen uso de los repositorios migrados.

Código 6.6: Servicios adaptados con Gemini-2.0 Flash

```
public interface ServicioEventos {
    UUID darAltaEvento(String nombre, String descripcion, String organizador, Categoria categoria, LocalDateTime fechaInicio, LocalDateTime fechaFin, int plazas, UUID idEspacioFisico) throws EntidadNoEncontrada;
    // Otros métodos ...
}

public interface ServicioDespachadorEspacios {
    void despacharEspacioFisicoCreado(UUID id, String nombre, String descripcion, int capacidad, String direccion);
    // Otros métodos ...
}

@Service
public class ServicioEventosImpl implements ServicioEventos {
    private final RepositorioEventosJpa repositorioEventos;
    private final RepositorioEspaciosJpa repositorioEspacios;
    private final PublicadorEventos publicadorEventos;
    private final ReservasAPI reservasAPI;

    // Constructor con @AutoWired para inyección por constructor

    @Override
    @Transactional(rollbackFor = Exception.class)
    public UUID darAltaEvento(
            String nombre,
            String descripcion,
            String organizador,
            Categoria categoria,
            LocalDateTime fechaInicio,
            LocalDateTime fechaFin,
            int plazas,
            UUID idEspacioFisico) throws EntidadNoEncontrada {
        // Implementación del método...
    }
    // Resto de métodos implementados...
}

@Service
public class ServicioDespachadorEspaciosImpl implements ServicioDespachadorEspacios {

    private final RepositorioEspaciosJpa repositorioEspacios;
    private final RepositorioEventosJpa repositorioEventos;
    private final PublicadorEventos publicadorEventos;

    // Constructor con @AutoWired para inyección por constructor

```

```
45    @Override
46    public void despacharEspacioFisicoCreado(
47            UUID id, String nombre, String descripcion, int capacidad, String ↩
                  ↪ direccion) {
48        // Implementación del método
49    }
50    // Resto de métodos implementados...
51}
```

Como se ha podido ver, la migración automatizada de la capa de acceso a datos en una migración homogénea ha demostrado ser coherente con la estructura del sistema original, preservar la lógica de negocio y adaptar las consultas a las particularidades del sistema destino.

6.3. Migración heterogénea del microservicio de reservas

En esta sección abordamos la migración completa del microservicio de reservas utilizando el modelo de lenguaje GPT-4o-mini. A diferencia del caso anterior, esta migración es heterogénea puesto que los modelos de datos de las bases de datos origen y destino son distintos. De hecho, consiste en migrar desde la base de datos NoSQL MongoDB hacia la base de datos relacional PostgreSQL, y resulta de especial relevancia ya que, como se verá más adelante, implica transformar información almacenada como colecciones de documentos en datos que serán almacenados en tablas normalizadas.

La finalidad de esta migración es evaluar el grado de automatización que puede alcanzar la herramienta a la hora de generar un esquema relacional coherente, preservar la semántica de las operaciones y realizar una adaptación precisa del código de acceso a datos en los repositorios, además de realizar correctamente la migración de los datos.

Los siguientes apartados seguirán la misma organización que para la migración del microservicio de eventos. Es decir, se presentará primero la migración de la base de datos, incluyendo la migración de esquema y datos junto con su validación, y luego la migración del código de la capa de acceso a datos.

6.3.1. Migración de la base de datos

La migración de base de datos del microservicio de reservas implica no solo la transformación completa del esquema, sino también la conversión semántica de los datos y, en último lugar, la validación completa del proceso para garantizar que la funcionalidad proporcionada por el microservicio no se ve afectada. Esta migración supone un reto significativo ya que implica la reinterpretación de estructuras tales como las referencias (`DBRef`) o los objetos embebidos en el modelo de datos origen, que no existen como tales en el modelo de datos relacional destino.

Como antes, la plantilla se instanció indicando (i) el SGBD origen, MongoDB en este caso, y el SGBD destino, PostgreSQL; (ii) el conjunto de consultas habituales del microservicio que cubrían operaciones como la obtención de las reservas de un evento, el filtrado de las reservas por usuario, la comprobación de cancelaciones de reservas y la disponibilidad de eventos activos; (iii) la definición del esquema MongoDB haciendo uso de los respectivos esquemas Mongoose [66] para las colecciones de *reservas* y *eventos*, (iv) ejemplos concretos (documentos) de cada colección, y (v) una serie de requisitos de migración relacionados con la conversión de referencias `DBRef` en claves ajenas, la normalización de las relaciones uno a muchos y la creación de índices para las consultas más comunes que han sido adjuntadas. El contenido del archivo con la entrada que acabamos de describir se puede encontrar en la colección de pruebas del repositorio de Github [16].

Una vez realizada la migración del esquema, LLMigrator-DB nos proporcionó el script con el esquema migrado a PostgreSQL junto con sus pautas de validación manual, el script de migración de datos y el script de validación del proceso. A continuación comentamos cada elemento. El esquema migrado contiene dos tablas principales (`EVENTO` y `RESERVA`). En ambas tablas, el identificador mantiene el tipo de datos que tenía en la base de datos origen y, además, la unicidad de estos identificadores queda garantizada al ser declarados como claves primarias. La herramienta realizó una gestión correcta de los campos numéricos al definir restricciones de integridad que replican las validaciones declaradas en los esquemas Mongoose proporcionados en la entrada. En cuanto a conversiones en el tipado de datos, también hay que destacar la conversión de las fechas ya que se han transformado en columnas de tipo `TIMESTAMP` cuando inicialmente eran de tipo `Date`.

Un aspecto relevante de la migración de esquema ha sido la correcta interpretación de las referencias entre colecciones. En el esquema original, la colección `reservas` contenía un campo `evento` definido como `DBRef` hacia la colección de `eventos`. Esta referencia ha sido migrada como una clave ajena en la tabla `RESERVA` hacia la tabla `EVENTO`. Además, se ha especificado una política de eliminación en cascada que asegura que al eliminar un evento, se eliminan automáticamente todas las reservas asociadas.

Es interesante indicar que esta clave ajena desde RESERVA a EVENTO, además de traducir la relación uno a uno entre cada reserva y su evento (campo `DBRef` antes mencionado), también refleja la relación uno a muchos entre cada evento y sus reservas, que en el esquema MongoDB original estaba implementada con un array de referencias a reservas dentro de cada documento `evento`.

Además de estos detalles, también se han generado índices para aquellos campos más consultados.

El Código 6.7 muestra el script SQL que permite crear el esquema completo migrado en PosgreSQL.

Código 6.7: Script del esquema migrado para el microservicio de reservas

```
-- Extensión para generación de UUIDs
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Crear la tabla EVENTO
CREATE TABLE EVENTO (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    plazas_disponibles INTEGER NOT NULL CHECK (plazas_disponibles >= 0),
    cancelado BOOLEAN NOT NULL,
    fecha_inicio TIMESTAMP,
    nombre_evento VARCHAR NOT NULL,
    _class VARCHAR NOT NULL DEFAULT 'reservas.dominio.Evento'
);

-- Crear la tabla RESERVA
CREATE TABLE RESERVA (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    id_usuario UUID NOT NULL,
    cancelado BOOLEAN NOT NULL,
    plazas_reservadas INTEGER NOT NULL CHECK (plazas_reservadas >= 1),
    evento UUID NOT NULL,
    _class VARCHAR NOT NULL DEFAULT 'reservas.dominio.Reserva',
    FOREIGN KEY (evento) REFERENCES EVENTO(id) ON DELETE CASCADE
);

-- Crear índices
CREATE INDEX idx_usuario ON RESERVA(id_usuario);
CREATE INDEX idx_evento ON RESERVA(evento);
CREATE INDEX idx_fecha_inicio ON EVENTO(fecha_inicio);
```

En definitiva, la conversión del esquema basado en un modelo de documentos a un esquema relacional ha sido realizada con éxito, preservando la semántica del dominio y garantizando la integridad referencial.

Tras la generación del esquema relacional fruto de su respectiva migración, la herramienta LLMigrator-DB nos proporcionó un script en Python que automatizaba la migración de los datos desde MongoDB a PostgreSQL.

La lógica del código incluido en el script se estructura en dos fases diferenciadas: la migración de los documentos de la colección `eventos` y la migración de los documentos de la colección `reservas`. En ambos casos se extraen los valores de los campos clave de cada documento y se insertan en las tablas correspondientes. Para los documentos `Evento` se mapean los campos directamente a las columnas del mismo nombre en la tabla `EVENTO` lo cual es una transformación directa. Sin embargo, en el caso de los documentos `Reserva`, como tienen un campo `evento` que es una referencia `DBRef`, entonces el script accede al identificador de dicho evento para mapear la relación uno a muchos. El Código 6.8 muestra un fragmento del código generado por LLMigrator-DB en el que se puede apreciar lo anteriormente descrito.

Código 6.8: Script para la migración de datos del microservicio de reservas

```
# Importaciones de librerías...
# Configuración de conexión a MongoDB...
# Configuración de conexión a PostgreSQL...

# Migrar datos de EVENTO
for evento in mongo_db.eventos.find():
    event_id = evento['_id']
    plazas_disponibles = evento['plazasDisponibles']
    cancelado = evento['cancelado']
    fecha_inicio = evento['fechaInicio']
    nombre_evento = evento['nombreEvento']
    _class = evento['_class']

    pg_cursor.execute(
        sql.SQL("""
            INSERT INTO EVENTO (id, plazas_disponibles, cancelado, fecha_inicio,↩
                ↪  nombre_evento, _class)
            VALUES (%s, %s, %s, %s, %s, %s)
        """),
        (event_id, plazas_disponibles, cancelado, fecha_inicio, nombre_evento, ↩
            ↪ _class)
    )

# Migrar datos de RESERVA
for reserva in mongo_db.reservas.find():
    reserva_id = reserva['_id']
    id_usuario = reserva['idUsuario']
    cancelado = reserva['cancelado']
    plazas_reservadas = reserva['plazasReservadas']
    evento_ref = reserva['evento']['$id'] # Obtener el ID del evento desde ↩
        ↪ DBRef
    _class = reserva['_class']

    pg_cursor.execute(
        sql.SQL("""
            INSERT INTO RESERVA (id, id_usuario, cancelado, plazas_reservadas, ↩
                ↪ evento, _class)
            VALUES (%s, %s, %s, %s, %s, %s)
        """),
        (reserva_id, id_usuario, cancelado, plazas_reservadas, evento_ref, ↩
            ↪ _class)
    )

# Confirmar los cambios y cerrar la conexión
```

En tercer lugar, LLMigrator-DB proporcionó un script para validar que la información había sido correctamente migrada, comparando los datos de ambas bases de datos a través de consultas equivalentes.

El código en cuestión se conecta a ambas bases de datos y ejecuta las consultas más

frecuentes que el usuario proporcionó en los requisitos funcionales de acceso a datos en la entrada. El Código 6.9 muestra un extracto del código encargado de realizar la validación.

Código 6.9: Script para validar la migración de base de datos del microservicio de reservas

```
# Importaciones de librerías necesarias...
# Configuración de conexión a MongoDB...
# Configuración de conexión a PostgreSQL...
# 1. Validar reservas asociadas a un evento concreto
uuid_evento = 'UUID_EVENTO' # Reemplaza con un UUID de evento real

mongo_reservas = list(
    mongo_db.reservas.find({ 'evento.$id': uuid_evento })
)
pg_cursor.execute(
    sql.SQL("SELECT * FROM RESERVA WHERE evento = %s"),
    (uuid_evento,)
)
pg_reservas = pg_cursor.fetchall()

assert len(mongo_reservas) == len(pg_reservas), \
    'Las reservas no coinciden para el evento.'

# 2. Comprobar si existen reservas asociadas a un usuario
uuid_usuario = 'UUID_USUARIO' # Reemplaza con un UUID de usuario real

mongo_reserva_existe = mongo_db.reservas.find_one({ 'idUsuario': uuid_usuario ↩
    ↪ })
pg_cursor.execute(
    sql.SQL("SELECT 1 FROM RESERVA WHERE id_usuario = %s LIMIT 1"),
    (uuid_usuario,)
)
pg_reserva_existe = pg_cursor.fetchone()

assert (mongo_reserva_existe is not None) == (pg_reserva_existe is not None), ↩
    ↪ \
    'La existencia de reservas no coincide.'

# 3. Obtener todas las reservas de un usuario...
# 4. Consultar eventos con disponibilidad...
# 5. Verificar estado de un evento...
# Cerrar conexiones...
```

6.3.2. Migración del código de la capa de acceso a datos

Tras haber completado la migración de la base de datos, el siguiente paso era adaptar y migrar el código que gestionaba el acceso a los datos dentro del microservicio. Al igual que en la sección referente al microservicio de eventos, se proporcionó a LLMigrator-DB

una descripción detallada del entorno de partida y el contexto funcional del sistema origen. A continuación, se describe de forma resumida la información de entrada, que aparece descrita con detalle en la colección de pruebas del repositorio de Github [16].

La entrada a LLMigrator-DB incluyó especificaciones explícitas sobre el sistema origen y destino, el dominio, los repositorios asociados a dichas entidades (los cuales utilizan `MongoRepostory` propio de Spring Data MongoDB), los servicios que contenían la lógica de negocio y hacían uso de dichos repositorios, el esquema destino generado en la migración de la base de datos y algunos requisitos funcionales y de estructuración de la salida. Se especificó Java como lenguaje de programación origen y destino, Spring Data MongoDB como framework de acceso a datos origen, Spring Data JPA como framework de acceso a datos destino, y las bases de datos origen, MongoDB, y destino, PostgreSQL.

La salida producida por LLMigrator-DB consistió en tres bloques de código: el código migrado del dominio, el código migrado de los repositorios y el código adaptado de los servicios. A continuación, se describe brevemente cada bloque de código y se muestra un fragmento del mismo.

Como se puede observar en el Código 6.10, el modelo del dominio migrado se adapta completamente a las convenciones JPA, pues se han sustituido las anotaciones específicas de Spring Data MongoDB por las correspondientes en JPA y se ha definido de forma explícita la estructura relacional en PostgreSQL del esquema migrado. Es decir, se mantiene la semántica de las clases `Evento` y `Reserva`, aunque se reorganizan aspectos estructurales para adaptarse al nuevo esquema.

La clase `Reserva` ha sido transformada a una entidad (`@Entity`) con su tabla asociada `reservas` (`@Table(name = "reservas")`). El aspecto más destacable es la definición de la relación con la entidad `Evento` mediante el uso de la anotación `@ManyToOne` junto con un `@JoinColumn` que especifica que la columna `evento_id` de la tabla `reservas` es una clave ajena que referencia al evento asociado a la reserva.

Por otro lado, la clase `Evento` ha sido traducida a una entidad con su tabla `eventos` asociada. Se establece la relación bidireccional con las reservas (`@OneToMany(mappedBy = "evento", cascade = CascadeType.ALL`) la cual era implícita en MongoDB.

En este caso, podemos ver cómo el LLM GPT-4o-mini ha introducido diversos errores con respecto al esquema presentado en el Código 6.7. En primer lugar, los nombres de las tablas en el esquema presentado son: `EVENTO` y `RESERVA`. Estos nombres son distintos a los indicados en las anotaciones `@Table` del modelo del dominio, lo cual produce un error de mapeo. Por otro lado, otro error es la clave ajena `evento` de la tabla `RESERVA` puesto que en la clase `Reserva` dicha columna se llama `evento_id`. Además, los atributos `plazasDisponibles` y `plazasReservadas` no hacen uso de la anotación correspondiente para alinear su nombrado con el utilizado en el esquema de la base de datos. Por último, también podemos ver cómo aquellos campos indicados en el esquema como `NOT NULL` no tienen en el modelo del dominio su anotación asociada `@Column(nullable=false, ...)` lo cual rompe la simetría semántica con respecto al modelo del dominio. Es por estos diversos errores que el uso de esta herramienta ha de

quedar en manos de usuarios expertos en bases de datos debido al no determinismo y los errores que puede incluir el LLM.

Código 6.10: Dominio del microservicio reservas migrado

```
@Entity
@Table(name = "reservas")
public class Reserva {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;
    @Column(name = "id_usuario")
    private UUID idUsuario
    private boolean cancelado;
    private int plazasReservadas;

    @ManyToOne
    @JoinColumn(name = "evento_id")
    private Evento evento;

    public Reserva(UUID idUsuario, int plazasReservadas, Evento evento) {
        this.id = UUID.randomUUID();
        ...
    }
    // Getters and Setters
}

@Entity
@Table(name = "eventos")
public class Evento {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private UUID id;
    private int plazasDisponibles;
    private boolean cancelado;
    private LocalDateTime fechaInicio;
    private String nombreEvento;
    @OneToMany(mappedBy = "evento", cascade = CascadeType.ALL)
    private List<Reserva> reservas;

    public Evento(int plazasDisponibles, boolean cancelado,
                LocalDateTime fechaInicio, String nombreEvento) {
        this.id = UUID.randomUUID();
        ...
    }
    // Getters and Setters
}
```

Con respecto a los repositorios migrados, han sido transformados desde interfaces

basadas en `MongoRepository` a interfaces que extienden `JpaRepository` haciendo uso de las convenciones de nombres de Spring Data JPA [67] para generar las consultas de forma implícita y automática. A continuación se adjunta el código 6.11.

Código 6.11: Repositorios migrados del microservicio reservas

```
@Repository
public interface RepositorioReservas extends JpaRepository<Reserva, UUID> {
    Page<Reserva> findAllByEventoId(UUID eventoId, Pageable pageable);
    boolean existsByIdUsuario(UUID idUsuario);
    List<Reserva> findAllByIdUsuario(UUID idUsuario);
}
@Repository
public interface RepositorioEventos extends JpaRepository<Evento, UUID> {}
```

Por último, destacar que el código de los servicios ha sido también adaptado para utilizar las respectivas clases de los repositorios migrados, tal y como se muestra en el fragmento de Código 6.12.

Código 6.12: Servicios adaptados para el microservicio de reservas

```
public interface ServicioReservas {

    UUID reservar(UUID idEvento, UUID idUsuario, int plazasReservadas) throws ↩
        ↪ Exception;
    // resto de métodos ...
}

@Service
public class ServicioReservasImpl implements ServicioReservas {

    private final RepositorioReservas repositorioReservas;
    private final RepositorioEventos repositorioEventos;
    private final PublicadorEventos publicadorEventos;

    public ServicioReservasImpl(RepositorioReservas repositorioReservas,
                                RepositorioEventos repositorioEventos,
                                PublicadorEventos publicadorEventos) {
        this.repositorioReservas = repositorioReservas;
        this.repositorioEventos = repositorioEventos;
        this.publicadorEventos = publicadorEventos;
    }

    // Métodos implementados de la interfaz...
}

public interface ServicioDespachadorEventos {
    // métodos a implementar...
}

@Service
```

```
30 public class ServicioDespachadorEventosImpl implements ↩
       ↪ ServicioDespachadorEventos {
31
32     private final RepositorioEventos repositorioEventos;
33     private final RepositorioReservas repositorioReservas;
34
35     public ServicioDespachadorEventosImpl(RepositorioEventos repositorioEventos↩
           ↪ ,
36                                           RepositorioReservas repositorioReservas) {
37         this.repositorioEventos = repositorioEventos;
38         this.repositorioReservas = repositorioReservas;
39     }
40
41     @Override
42     public void despacharCreacionEvento(UUID idEvento, int plazasDisponibles, ↩
           ↪ boolean cancelado, LocalDateTime fechaInicio, String nombreEvento) ↩
           ↪ throws Exception {
43         repositorioEventos.save(new Evento(idEvento, plazasDisponibles, ↩
               ↪ cancelado, fechaInicio, nombreEvento));
44     }
45     // Resto de métodos...
46 }
```

En definitiva, la migración realizada y descrita en esta sección ha demostrado la capacidad de la herramienta LLMigrator-DB para transformar de forma coherente y estructurada una base de datos basada en un modelo de datos de documentos a una base de datos basada en un modelo relacional convencional. Aunque los errores detectados han sido mínimos, su existencia impide una automatización completa del proceso, ya que sigue siendo necesaria la validación manual de las respuestas generadas por la herramienta.

6.4. Comparativa de los resultados obtenidos en las migraciones

En esta sección se presenta un resumen de los resultados obtenidos a lo largo de todo el capítulo. La Tabla 6.1 muestra, en términos porcentuales, la validez de los resultados alcanzados en cada una de las migraciones del caso de estudio.

Como podemos observar en la Tabla 6.1, se ha producido un éxito (100%) en todas las migraciones, excepto en los resultados relacionados con la migración de datos y código del microservicio de reservas. Esto evidencia la complejidad inherente de la migración heterogénea desde un modelo documental (MongoDB) hacia un modelo relacional (PostgreSQL). En el caso de esta migración, se han encontrado los problemas que se detallan a continuación.

A diferencia de Gemini-2.0 Flash en la migración homogénea del microservicio de eventos, GPT-4o-mini genera un script de validación de la migración de datos que no

Tareas/Microservicio	**Microservicio Eventos (Gemini-2.0 Flash)**	**Microservicio Reservas (GPT-4o-mini)**
Migración de esquema	100%	100%
Migración de los datos	100%	100%
Validación de los datos	100%	90%
Migración del dominio	100%	80%
Migración de las consultas de los repositorios	100%	90%
Adaptación de los servicios	100%	100%

Tabla 6.1: Resumen porcentual de validez de las tareas de migración

incluye un conteo de registros automatizado que nos sirva para conocer el número de entidades en cada base de datos.

Respecto a la migración del código, encontramos diversos errores en la definición de las anotaciones en las clases del dominio generadas, las cuales difieren del esquema generado inicialmente por LLMigrator-DB. Entre estos errores destacamos: los nombres de tablas incorrectos en las anotaciones `@Table`, la no alineación entre nombres de columnas y atributos (por ejemplo, `evento_id` en la definición de la clase y `evento` en el esquema generado), la ausencia de anotaciones `@Column(nullable = false)` en campos que en el esquema destino generado están definidos como `NOT NULL`. Estos errores pueden afectar al comportamiento del sistema si no son corregidos de forma manual, por lo que el grado de automatización no puede considerarse completo.

Aunque las consultas fueron correctamente transformadas en métodos derivados del nombre del método, según las convenciones de Spring Data JPA, no se incluyeron validaciones ni adaptaciones adicionales que aseguren que los repositorios implementan las restricciones adecuadas. Además, como no se utiliza la anotación `@Query` no se puede identificar ningún mecanismo para manejar casos límite de forma manual.

En resumen, estas limitaciones presentadas no suponen un fallo generalizado de LLMigrator-DB, sino que reflejan los retos propios de las migraciones heterogéneas, donde la transformación semántica va más allá de un simple mapeo estructural o la adaptación dialectal, como es en el caso de una migración homogénea. A pesar de estos inconvenientes, el proceso de migración puede ser completado en su totalidad, y las tareas automatizadas ofrecen un punto de partida sólido que reduce en gran medida el esfuerzo manual requerido. Cabe aclarar que, aunque las limitaciones no suponen un fallo perentorio, el hecho de su mera existencia y la necesidad de la intervención manual suponen que la herramienta ha de ser usada por usuarios con experiencia técnica.

7. Conclusiones y líneas futuras

7.1. Conclusiones generales

A lo largo de esta Memoria, se ha mostrado cómo los LLM y GenAI pueden ser muy útiles para automatizar las tareas involucradas en una migración de bases de datos, en particular, se han abordado las tres tareas principales: la conversión del esquema, la migración de datos y la adaptación del código de acceso a datos junto con la validación de cada proceso y de la migración completa. La construcción y validación de la herramienta LLMigrator-DB puede ser considerada un primer paso importante hacia disponer de una herramienta completa de migración de bases de datos. LLMigrator-DB ha demostrado ser capaz de automatizar gran parte del proceso de migración de bases de datos y de su código de acceso a datos en sistemas software, tanto en el caso de una migración homogénea como heterogénea. En relación con los objetivos planteados en el Capítulo 1 de introducción, puede afirmarse que todos ellos han sido alcanzados de forma satisfactoria.

1. Se han diseñado, implementado y validado las plantillas de prompts para la migración de esquema de forma satisfactoria.

2. Se han diseñado e implementado los prompts pertinentes para la migración de los datos almacenados y su respectiva validación, comprobando la integridad referencial y la inexistencia de pérdida de datos.

3. Se han diseñado, implementado y validado los prompts para la migración del código.

4. Se han integrado con éxito los tres migradores en LLMigrator-DB y se ha validado la herramienta de forma exitosa con el caso de estudio *Eventalia* que ha supuesto migrar dos microservicios y abordar una migración homogénea y otra heterogénea.

En cuanto a la construcción de LLMigrator-DB es destacable el uso de una tecnología novedosa como son los frameworks de abstracciones de LLM. Con su uso hemos podido comprobar que este tipo de soluciones facilita significativamente la integración de LLM en aplicaciones software, al proporcionar una interfaz unificada, desacoplar la lógica de negocio del LLM específico y simplificar la gestión de peticiones, formatos, entre otras funcionalidades. En concreto, el uso de Spring AI nos ha permitido definir flujos

de interacción estructurados con los LLM, incorporar validaciones sobre las salidas generadas y alternar entre distintos LLMs de forma transparente.

LLMigrator-DB ha sido desarrollado en torno a una arquitectura REST que permite un diseño modular, facilitando la separación de responsabilidades entre los distintos componentes encargados de la migración del esquema, los datos y el código fuente. Este enfoque favorece la escalabilidad, ya que permite extender o sustituir funcionalidades de forma independiente, y mejora la interoperabilidad al exponer una interfaz REST estandarizada que puede integrarse fácilmente con otros sistemas.

La herramienta consigue la independencia de plataforma, puesto que es independiente de un SGBD o de un modelo de datos determinado y no está vinculado a ningún LLM concreto. Lo primero se ha conseguido a través del diseño de las plantillas de prompt, y lo segundo utilizando la API unificada de acceso a LLM OpenRouter.

En cuanto a los resultados obtenidos, resumidos en la tabla 6.1, se puede ver que la migración homogénea se ha automatizado al 100%, mientras que en el caso de la migración heterogénea, la automatización de la conversión del esquema ha sido total y algo menor para la migración de datos (90%) y de código (80% en el peor de los casos).

De acuerdo a los resultados obtenidos y los errores subyacentes, se puede estimar que LLMigrator-DB permite automatizar alrededor del 70% del proceso completo de migración para un sistema basado en microservicios haciendo uso de los LLMs que hemos propuesto. Para un usuario experimentado en bases de datos, supone un ahorro sustancial en términos de tiempo, ya que se automatizan tareas repetitivas como la adaptación del código de la capa de acceso a datos, la generación de consultas estructuradas y la validación de la coherencia entre los sistemas origen y destino. Estos resultados avalan el potencial de la herramienta como asistente de migración, capaz de acelerar gran parte del proceso.

7.2. Consideraciones sobre el uso de la herramienta LLMigrator-DB y trabajo futuro

Como resultado de nuestra experiencia, en esta sección presentaremos algunas consideraciones que deben tenerse en cuenta para la adopción en un entorno productivo real de una herramienta de migración basada en GenAI como LLMigrator-DB.

En primer lugar, el uso de LLM introduce un grado de no determinismo inherente: una misma petición puede generar salidas distintas en ejecuciones sucesivas, lo que obliga al usuario a realizar múltiples intentos y comparar las respuestas obtenidas para seleccionar la más adecuada.

En segundo lugar, el proceso de validación de las salidas generadas por LLMigrator-DB requiere que el usuario tenga conocimientos avanzados en bases de datos, tanto de tipo relacional (SQL) como NoSQL. La revisión de esquemas, la evaluación de la integridad referencial o la comprobación semántica de las consultas generadas son tareas que actualmente no pueden ser automatizadas por completo y exigen un perfil

técnico especializado. Esta dependencia limita el acceso a la herramienta por parte de usuarios sin formación específica.

La tercera limitación de la herramienta es el coste económico asociado a cada solicitud enviada al modelo y la necesidad de repetir la misma consulta para obtener múltiples variantes, lo que puede suponer un aumento significativo del gasto asociado al uso de la herramienta, especialmente cuando se trata de modelos de alto rendimiento.

Con el fin de abordar y minimizar estas limitaciones en futuras versiones de esta herramienta, se proponen cuatro líneas de trabajo futuro principales:

1. *Estudio e integración de métricas de calidad*: se propone diseñar un sistema que, a partir de múltiples respuestas generadas por el modelo para una misma petición del usuario, sea capaz de evaluar automáticamente la calidad de cada una. Esta evaluación se basaría en criterios objetivos como la consistencia sintáctica, la preservación de la integridad referencial o el cumplimiento de los requisitos funcionales especificados.

2. *Desarrollo de una herramienta visual con agente integrado*: se contempla la creación de una interfaz visual interactiva que incorpore un agente conversacional basado en un modelo de lenguaje. Esta herramienta permitiría al usuario modificar, validar o refinar los scripts de migración generados utilizando instrucciones en lenguaje natural, actuando como una capa intermedia de edición asistida, especialmente útil para usuarios con menor formación técnica.

3. *Refactoring del código para usar una arquitectura basada en eventos*: se plantea refactorizar el sistema para implementar una arquitectura basada en eventos con el fin de modularizar más las responsabilidades del sistema, favoreciendo la escalabilidad horizontal.

4. *Uso de modelos de razonamiento avanzado*: si bien la herramienta se apoya en modelos de lenguaje de propósito general (como GPT-4o-mini o Gemini-2.0-Flash), estos no siempre garantizan un razonamiento avanzado. En consecuencia, ciertas transformaciones complejas o validaciones lógicas no son abordadas de forma óptima.

Bibliografía

[1] Pramod J. Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Addison-Wesley, 2013. ISBN 9780321826626 0321826620.

[2] Guy Harrison. *Next generation databases: NoSQL, NewSQL, and Big Data*. Apress, 2015. ISBN 9781484213292 1484213297.

[3] Amit Bahree. *Generative AI in Action*. Manning Publications, 2024. ISBN 9781633436947.

[4] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. From llms to llm-based agents for software engineering: A survey of current, challenges and future, 2025. URL `https://arxiv.org/abs/2408.02479`.

[5] Mahesh Kumar Goyal, Prasad Sundaramoorthy, and Harshini Gadam. Leveraging generative ai for database migration: A comprehensive approach for heterogeneous migrations. *Journal of Computational Analysis and Applications*, 33(8):3041–3052, November 2024. URL `https://www.researchgate.net/publication/391367834`. Publicado en noviembre de 2024. Accedido el 4 de junio de 2025.

[6] Peter Yong Zhong, Haoze He, Omar Khattab, Christopher Potts, Matei Zaharia, and Heather Miller. A guide to large language model abstractions, January 2024. URL `https://www.twosigma.com/articles/a-guide-to-large-language-model-abstractions/`. Accedido el 29 de mayo de 2025.

[7] OpenRouter. Openrouter, n.d. URL `https://openrouter.ai/`. Accedido el 30 de mayo de 2025.

[8] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, Boston, MA, 2017. ISBN 9780134494166.

[9] Postman. Postman learning center: Overview, 2025. URL `https://learning.postman.com/docs/introduction/overview/`.

[10] María José Ortín Ibáñez, Fabián Sola Durán, José Ramón Hoyos Barceló, and Jesús Joaquín García Molina. Automatización de la migración de bases de datos con abstracciones de llm. In *Actas de las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2025)*, 2025.

[11] Bruce Hopkins. *ChatGPT for Java: A Hands-on Developer's Guide to ChatGPT and OpenAI APIs*. Apress, Berkeley, CA, 1st edition, 2024.

[12] Spring. Spring boot documentation, n.d.. URL `https://docs.spring.io/spring-boot/index.html`. Accedido el 22 de junio de 2025.

[13] Christopher Brousseau and Matt Sharp. *LLMs in Production: From Language Models to Successful Products*. Manning, February 2025. ISBN 9781633437203.

[14] Carlos J. Fernández Candel, Diego Sevilla Ruiz, and Jesús J. García-Molina. A unified metamodel for nosql and relational databases. *Information Systems*, 104: 101898, 2022. doi: https://doi.org/10.1016/j.is.2021.101898.

[15] Amit Bahree. *Mastering Spring AI: The Java Developer's Guide for Large Language Models and Generative AI*. Apress, 2024. ISBN 9798868810008.

[16] Grupo Modelum. Llmigrator-db: Una herramienta de migración de bases de datos asistida por llm, 2025. URL `https://github.com/modelum/LLMigrator-DB`.

[17] Astera. Database migration: What it is and how it is done, July 2019. URL `https://astera1.medium.com/database-migration-what-it-is-and-how-it-is-done-5435290ee11b`. Accedido el 4 de mayo de 2025.

[18] Acronis. Qué es la migración de bases de datos y cómo funciona, March 2024. URL `https://www.acronis.com/es-es/blog/posts/what-is-database-migration/`. Accedido el 4 de mayo de 2025.

[19] Google Cloud. Database migration: Concepts and principles (part 1), April 2025. URL `https://cloud.google.com/architecture/database-migration-concepts-principles-part-1?hl=es-419`. Revisado el 29 de abril de 2025. Accedido el 2 de mayo de 2025.

[20] Prisma. What are database migrations?, December 2023. URL `https://www.prisma.io/dataguide/types/relational/what-are-database-migrations`. Modificado por última vez el 15 de diciembre de 2023. Accedido el 4 de mayo de 2025.

[21] Edward Raff, Drew Farris, and Stella Biderman. *How Large Language Models Work*. Manning Publications, 2025. ISBN 9781633437081.

[22] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. A survey on large language models for software engineering, 2024. URL `https://arxiv.org/abs/2312.15223`.

[23] Stephen Wolfram. What is chatgpt doing ... and why does it work?, 2023. URL `https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/`. Accedido: 1 de julio de 2025.

[24] OpenAI. Best practices for prompt engineering with the openai api. https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api, 2025. Accedido el 26 de junio de 2025.

[25] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM computing surveys*, 55(9):1–35, 2023.

[26] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, Vinija Jain, Samrat Mondal, and Aman Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*, 2024.

[27] Richard Davies. *Prompt Engineering in Practice*. Manning Publications, 2025. ISBN 9781633436305.

[28] Dhruv Gupta. Prompt engineering: Using intelligence to use artificial intelligence: A deep dive into prompt engineering, March 2024. URL https://medium.com/@researchgraph/prompt-engineering-21112dbfc789. Accedido el 30 de mayo de 2025.

[29] Addy Osmani. The prompt engineering playbook for programmers, May 2024. URL https://addyo.substack.com/p/the-prompt-engineering-playbook-for. Publicado en Elevate. Accedido el 1 de junio de 2025.

[30] LangChain. Langchain, n.d. URL https://www.langchain.com/. Accedido el 29 de mayo de 2025.

[31] LlamaIndex. Llamaindex, n.d. URL https://www.llamaindex.ai/. Accedido el 29 de mayo de 2025.

[32] Spring. Spring ai, n.d.. URL https://spring.io/projects/spring-ai. Accedido el 29 de mayo de 2025.

[33] OpenAI. Overview, n.d. URL https://platform.openai.com/docs/overview. Accedido el 29 de mayo de 2025.

[34] OpenAI. Gpt-4o mini: una apuesta por la inteligencia rentable, July 2024. URL https://openai.com/es-ES/index/gpt-4o-mini-advancing-cost-efficient-intelligence/. Accedido el 1 de junio de 2025.

[35] Amit Yadav. Introduction to openai's gpt models, October 2024. URL https://medium.com/biased-algorithms/introduction-to-openais-gpt-models-1de0d73fa5b5. Publicado en Medium. Accedido el 1 de junio de 2025.

[36] Ryan Ong. What is gpt-4o mini? how it works, use cases, api & more, July 2024. URL https://www.datacamp.com/blog/gpt-4o-mini. Publicado en DataCamp. Accedido el 1 de junio de 2025.

[37] Bernard Loki. Understanding gpt-4o mini. advancing cost-effective intelligence, July 2024. URL https://medium.com/@bernardloki/understanding-gpt-4o-mini-4926bdcac765. Publicado en Medium. Accedido el 1 de junio de 2025.

[38] Adel Basli. A performance showdown of low-cost llms (gpt-4o mini, gpt-4.1 nano, and beyond), May 2025. URL https://medium.com/@adelbasli/a-performance-showdown-of-low-cost-llms-gpt-4o-mini-gpt-4-1-nano-and-beyond-32f0d9e54f11. Publicado en Medium. Accedido el 1 de junio de 2025.

[39] Gemini Team, Google DeepMind & Google Research. Gemini: A family of highly capable multimodal models, May 2025. URL https://arxiv.org/pdf/2312.11805. Preprint publicado en arXiv, versión 5. Accedido el 1 de junio de 2025.

[40] E. Huizenga. Developer's guide to getting started with gemini 2.0 flash on vertex ai, December 2024. URL https://medium.com/google-cloud/developers-guide-to-getting-started-with-gemini-2-0-flash-on-vertex-ai-6b4fe3c6899f. Publicado en Medium. Accedido el 1 de junio de 2025.

[41] Microsoft. ¿qué es azure database migration service?, May 2025. URL https://learn.microsoft.com/es-es/azure/dms/dms-overview. Publicado en Microsoft Learn. Accedido el 4 de junio de 2025.

[42] Google Cloud. Descripción general de database migration service, June 2025. URL https://cloud.google.com/database-migration/docs/overview?hl=es-419. Accedido el 3 de junio de 2025.

[43] Amazon Web Services. What is aws database migration service?, n.d.. URL https://docs.aws.amazon.com/dms/latest/userguide/Welcome.html. Accedido el 4 de junio de 2025.

[44] Amazon Web Services. Getting started with aws schema conversion tool, n.d.. URL https://docs.aws.amazon.com/dms/latest/userguide/CHAP_GettingStarted.SCT.html. Accedido el 4 de junio de 2025.

[45] Amazon Web Services. Aws schema conversion tool, n.d.. URL https://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_Welcome.html. Accedido el 4 de junio de 2025.

[46] Amazon Web Services. Viewing your database migration assessment report for dms schema conversion, n.d.. URL https://docs.aws.amazon.com/dms/latest/userguide/assessment-reports-view.html. Accedido el 4 de junio de 2025.

[47] Datafold Team. A modern data migration framework for ai-powered success, January 2025. URL https://www.datafold.com/blog/modern-data-migration-framework. Publicado en Datafold. Accedido el 4 de junio de 2025.

[48] Datafold Team. What is the datafold migration agent, and how does it work?, February 2025. URL https://www.datafold.com/blog/what-is-the-datafold-migration-agent. Publicado en Datafold. Accedido el 4 de junio de 2025.

[49] IBM. Qué son las api rest, April 2025. URL https://www.ibm.com/es-es/think/topics/rest-apis. Accedido el 13 de junio de 2025.

[50] Spring. Prompttemplate. Documentación API de Spring AI, n.d.. URL https://docs.spring.io/spring-ai/reference/api/prompt.html#_prompttemplate. Accedido el 13 de junio de 2025.

[51] Tim (Ed.) Bray. The javascript object notation (json) data interchange format, December 2017. URL https://datatracker.ietf.org/doc/html/rfc8259.

[52] OpenAPI Initiative. Openapi, n.d. URL https://www.openapis.org/. Accedido el 13 de junio de 2025.

[53] Marcos Menarguez Tortosa. Caso de estudio: Arquitectura de software basada en microservicios para la organización de eventos. Trabajo desarrollado en la asignatura Arquitectura del Software, Universidad de Murcia, junio 2025.

[54] MySQL. Mysql documentation, n.d. URL https://dev.mysql.com/doc/. Accedido el 22 de junio de 2025.

[55] MongoDB, Inc. Mongodb documentation, n.d. URL https://www.mongodb.com/docs/. Accedido el 22 de junio de 2025.

[56] Square, Inc. Retrofit, n.d. URL https://square.github.io/retrofit/. Accedido el 22 de junio de 2025.

[57] VMware, Inc. Rabbitmq documentation, n.d. URL https://www.rabbitmq.com/docs. Accedido el 22 de junio de 2025.

[58] Oracle Corporation. Java platform, enterprise edition 6 documentation, n.d. URL https://docs.oracle.com/javaee/6/. Accedido el 22 de junio de 2025.

[59] Docker, Inc. Docker compose documentation, n.d. URL https://docs.docker.com/compose/. Accedido el 22 de junio de 2025.

[60] Sun Microsystems. Página principal de JPA. http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html. Último acceso: 28-06-2025.

[61] Eclipse Foundation. Eclipselink documentation, n.d. URL https://eclipse.dev/eclipselink/. Accedido el 22 de junio de 2025.

[62] Spring. Spring data jpa, n.d.. URL https://spring.io/projects/spring-data-jpa. Accedido el 22 de junio de 2025.

[63] Hibernate Team. Hibernate orm documentation, n.d. URL https://hibernate.org/orm/. Accedido el 22 de junio de 2025.

[64] Spring. Spring data mongodb, n.d.. URL https://spring.io/projects/spring-data-mongodb. Accedido el 22 de junio de 2025.

[65] PostgreSQL Global Development Group. Postgresql documentation, n.d. URL https://www.postgresql.org/docs/. Accedido el 22 de junio de 2025.

[66] Mongoose v8.16.1: Guide, 2025. URL https://mongoosejs.com/docs/guide.html.

[67] Spring data jpa - query methods, 2025. URL https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html#jpa.query-methods.query-creation.

[68] Chris Brousseau and Mike Sharp. *The Complete (and Obsolete?) Guide to Generative AI*. Manning Publications, 2025. ISBN 9781633436982.

[69] IBM. ¿qué son los grandes modelos de lenguaje (llm)?, November 2023. URL https://www.ibm.com/es-es/think/topics/large-language-models. Accedido el 29 de mayo de 2025.

[70] Red Hat. ¿qué son los modelos de lenguaje grandes (llm)?, September 2023. URL https://www.redhat.com/es/topics/ai/what-are-large-language-models. Accedido el 29 de mayo de 2025.

[71] Andreas Stöffelbauer. How large language models work, October 2023. URL https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f. Publicado en Medium – Data Science at Microsoft. Accedido el 29 de mayo de 2025.

[72] Google Cloud. Gemini 2.0 flash, May 2025. URL https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash. Actualizado el 30 de mayo de 2025. Accedido el 30 de mayo de 2025.

[73] Coursera. What is gpt? gpt-3, gpt-4, and more explained, October 2024. URL https://www.coursera.org/articles/what-is-gpt. Accedido el 1 de junio de 2025.

Lista de Acrónimos y Abreviaturas

API	Interfaz de Programación de Aplicaciones.
AWS	Amazon Web Services.
CoT	Cadena de Razonamiento.
DMA	Agente de Migración de Datos.
DMS	Servicio de Migración de Bases de Datos.
DTO	Objeto de Transferencia de Datos.
GenAI	Inteligencia Artificial Generativa.
HTTP	Protocolo de Transferencia de Hipertexto.
JISBD	Jornadas de Ingeniería del Software y Bases de Datos.
JPA	Java Persistence API.
JPQL	Lenguaje de Consulta de Persistencia de Java.
JSON	JavaScript Object Notation.
LLM	Modelo de Lenguaje Grande.
ML	Machine Learning.
NLP	Procesamiento del Lenguaje Natural.
ORM	Mapeo Objeto-Relacional.
REST	Transferencia de Estado Representacional.
SGBD	Sistema Gestor de Bases de Datos.
SGBDR	Sistemas de Gestión de Bases de Datos Relacionales.
TFG	Trabajo Final de Grado.

A. Integración completa de Spring AI con OpenRouter

En este anexo se documentará la integración completa de la herramienta LLMigrator-DB con modelos de lenguaje a través de la plataforma *OpenRouter*, utilizando el framework *Spring AI*. El objetivo de esta integración es permitir que la herramienta soporte múltiples LLM de forma flexible, centralizando la configuración y desacoplando el uso de modelos concretos del resto de la aplicación.

A.1. Configuración de propiedades

Para permitir una configuración dinámica de los modelos disponibles, se define una interfaz genérica con una clase asociada que contiene su implementación. El Código A.1 asociado a dicha configuración se presenta a continuación.

Código A.1: Interfaz de propiedades y su implementación

```
@Setter
@Getter
public class Options {
  private Set<String> models;
}

@Configuration
@ConfigurationProperties(prefix = "spring.ai.openrouter.chat")
@Getter
@Setter
public class OpenRouterAiChatProperties {
  private Options options = new Options();
}
```

La anotación `@ConfigurationProperties` nos permite cargar desde el fichero de configuración del proyecto, en nuestro caso denominado `application.yml`, una lista de modelos accesibles vía *OpenRouter*. Este fichero ya ha sido presentado en el contenido principal del TFG en la Sección 5.3.

La clase principal, `OpenRouterClientConfiguration`, contiene la configuración de modelos y se encarga de crear dinámicamente instancias de `ChatClient` por cada modelo especificado en el fichero de configuración antes mencionado. A continuación, se adjunta el fragmento de código encargado de la creación de instancias `ChatClient`.

Código A.2: Instanciación dinámica de los clientes que interactúan con OpenRouter

```
@Configuration
public class OpenRouterClientConfiguration {

  @Bean
  public Map<String, ChatClient> chatClients(...,
    OpenRouterAiChatProperties openRouterProperties) {
    //... Configuraciones previas
    return openRouterProperties.getOptions().getModels().stream()
        .collect(
            Collectors.toMap(
                model -> model,
                model -> {
                  OpenAiChatOptions chatOptions =
                      OpenAiChatOptions.fromOptions(openAiChatProperties.↩
                          ↪ getOptions());
                  chatOptions.setModel(model);
                  chatOptions.setStreamUsage(false);
                  OpenAiChatModel openAiChatModel =
                      new OpenAiChatModel(
                          openAiApi,
                          chatOptions,
                          DefaultToolCallingManager.builder()
                              .observationRegistry(ObservationRegistry.NOOP)
                              .build(),
                          retryTemplate,
                          ObservationRegistry.NOOP);
                  return ChatClient.builder(openAiChatModel)
                      .defaultAdvisors(
                          MessageChatMemoryAdvisor.builder(chatMemory(↩
                              ↪ chatMemoryRepository()))
                              .build())
                      .build();
                }));
  }
 // Resto de métodos de configuración...
}
```

A.2. Soporte para memoria conversacional

Como se puede ver en el Código A.2, cada `ChatClient` construido incorpora un componente de memoria conversacional, un `ChatMemory`, que permite mantener el contexto a lo largo del proceso de migración.

Cada uno de estos componentes de memoria conversacional han de ser configurados para cada uno de los `ChatClient` presentados anteriormente. A continuación, se presenta el Código A.3 donde se puede apreciar dicha configuración.

Código A.3: Código para la configuración del soporte de memoria conversacional

```
@Bean
public ChatMemoryRepository chatMemoryRepository() {
  return new InMemoryChatMemoryRepository();
}

@Bean
public ChatMemory chatMemory(ChatMemoryRepository chatMemoryRepository) {
  return MessageWindowChatMemory.builder()
      .maxMessages(20)
      .chatMemoryRepository(chatMemoryRepository)
      .build();
}
```

A.3. Cliente HTTP personalizado

Para garantizar un comportamiento robusto y evitar falsos errores debidos a la expiración del tiempo máximo previsto para el procesamiento de los prompts por parte del modelo, se define un componente que nos permite personalizar los clientes HTTP, concretamente la clase `RestClientCustomizer`, que se configurar con el fin de establecer tiempos de espera amplios para las peticiones HTTP realizadas a OpenRouter.

Código A.4: Configuración de clientes HTTP personalizados

```
@Bean
public RestClientCustomizer restClientCustomizer() {
  return restClientBuilder -> {
    SimpleClientHttpRequestFactory factory = new SimpleClientHttpRequestFactory
        ↪ ();
    factory.setConnectTimeout((int) Duration.ofMinutes(20).toMillis());
    factory.setReadTimeout((int) Duration.ofMinutes(20).toMillis());
    restClientBuilder.defaultHeader(HttpHeaders.ACCEPT, MediaType.
        ↪ APPLICATION_JSON_VALUE);
    restClientBuilder.requestFactory(factory);
  };
}
```

A.4. Integración de modelos de lenguaje mediante OpenRouter

Tras haber configurado los diferentes clientes en el contenedor de inyección de dependencias de Spring, la herramienta puede seleccionar de forma dinámica, en tiempo de ejecución, el LLM deseado por el usuario. Esta selección se realiza a partir de la información proporcionada por el usuario en la petición realizada, donde se indica de

forma explícita el LLM que se desea utilizar para la migración.

Durante la ejecución de un proceso de migración (ya sea de base de datos o del código asociado a la misma), se obtiene el cliente correspondiente mediante una simple consulta al mapa de instancias de clientes inyectado:

```
ChatClient chatClient = chatClients.get(request.getLanguageLargeModel().↩
    ↪ getModelId());
```

Esta simple línea nos permite recuperar de forma inmediata el cliente configurado para el LLM solicitado, sin necesidad de crear ninguna instancia de forma manual. Este enfoque nos permite abstraer por completo la lógica de negocio del LLM específico utilizado, lo cual facilita la reutilización de las plantillas de prompts y el soporte multimodelo.

Consideraciones de atribución y licencia

Parte del código presentado en este anexo ha sido adaptado a partir del repositorio *spring-ai-openrouter-example*, desarrollado por Chris Phillipson y está disponible públicamente en `https://github.com/pacphi/spring-ai-openrouter-example`, bajo la licencia Apache 2.0.

En concreto, las siguientes clases fueron inspiradas funcionalmente del código original:

- `OpenRouterClientConfiguration` (basada en `MultiChat`)
- `OpenRouterAiChatProperties` y `Options` (adaptadas de la configuración original de propiedades)

Los cambios realizados en esta adaptación incluyen:

- Sustitución del uso de la interfaz `MultiChatProperties` por una clase concreta configurada mediante `@ConfigurationProperties`.
- Renombrado de clases y separación modular para mejorar la legibilidad y el mantenimiento en el contexto del microservicio.
- Reemplazo del `SimpleLoggerAdvisor` original por `MessageChatMemoryAdvisor`, e incorporación del soporte para `ChatMemory` utilizando almacenamiento en memoria con una ventana de mensajes.
- Ajustes en el `RestClientCustomizer` para personalizar los tiempos de espera y cabeceras HTTP, incluyendo valores más adecuados al entorno de procesamiento de modelos LLM.

El código resultante mantiene la lógica general de construcción de múltiples `ChatClient` dinámicos en función de los modelos declarados, pero ha sido modificado sustancialmente para adaptarse a las necesidades del proyecto. La adaptación respeta todas las condiciones de la licencia Apache 2.0: se han mantenido los avisos de atribución en los archivos fuente, se ha incluido el archivo de licencia original y se deja constancia expresa de ello en esta memoria y en el código correspondiente del repositorio de GitHub [16].

B. Plantillas de prompts diseñadas para la migración de bases de datos

En este anexo se presentan al detalle las plantillas de prompts utilizadas por LLMigrator-DB para realizar la migración completa de una base de datos. Estas plantillas han sido diseñadas utilizando las técnicas de Ingeniería de Prompts expuestas en la Sección 5.1 para interactuar de forma eficaz con el LLM que el usuario requiera. De esta forma, se garantiza que las tareas de migración de esquema, datos y validación de resultados se generen de forma coherente con la petición del usuario.

Cada plantilla de prompt está compuesta por uno o varios mensajes estructurados, los cuales combinan instrucciones específicas, contenido contextual y restricciones en el formato de salida.

Las plantillas de prompts han sido introducidas de forma fragmentada en la Sección 5.4. La clase Java principal es `DbMigrationPromptBuilder`, que se encarga de la construcción de los prompts a partir de las plantillas que define. Aclarar que la construcción de todos estos prompts se hace cada vez que el usuario accede al punto de entrada `/api/v1/migrations`.

En las secciones que siguen a continuación, se describen las diferentes plantillas de prompts definidas para su uso en cada fase del proceso de migración.

B.1. Contextualización del modelo

A continuación, se presenta el método Java en el Código B.1 que contiene la plantilla de prompt utilizada para la construcción del prompt contextual que se envía al modelo.

Código B.1: Plantilla de prompt para la contextualización del modelo

```
1  /** prompt contextual * */
2  public Prompt buildContextualPrompt(OpenrouterMigrateDatabaseRequestDto ↩
       ↪ promptInfo) {
3    // Mensaje del sistema: Solo establece el contexto, sin requerir una ↩
         ↪ respuesta inmediata
4    String stringSystemMessage =
5        """
6        Actúa como un experto en migración de bases de datos {↩
             ↪ sourceDatabaseType} a {targetDatabaseType}.
7
8        **OBJETIVO**: Tu tarea es asistir en la migración, organizando el ↩
             ↪ proceso en cuatro fases:
```

```
9
10      - Migración del esquema: Convertir el esquema de la base de datos {sourceDatabaseType} en un esquema de la base de datos {targetDatabaseType}.
11      - Validación del esquema: Proporcionar pautas para validar manualmente la migración del esquema.
12      - Migración de los datos: Una vez obtenido el esquema de destino, generar un código {language} para migrar todos los datos de la base de datos original
13      a la base de datos destino.
14      - Validación de la migración: Genera un conjunto de queries para ambas bases de datos para validar que la migración de datos es correcta. Para ello, dichas consultas han de abordar\s
15      todas las relaciones del esquema origen y destino.
16
17      **Contexto de la Migración**:
18      - Sistema de base de datos origen: {sourceDatabaseType}
19      - Sistema de base de datos destino: {targetDatabaseType}
20      - Esquema de la base de datos original: {databaseSchema}
21      - Requisitos y restricciones de la migración: {migrationRequirements}
22
23      **Instrucciones**:
24      - NO generes ninguna respuesta en este momento.
25      - Usa esta información para recordar y adaptar el contexto en cada fase de la migración.
26      - Asegúrate de tener en cuenta las decisiones y respuestas anteriores al proporcionar nuevas recomendaciones.
27      - Sigue las instrucciones en cada fase para completar la migración con éxito.
28
29      **IMPORTANTE**: Devuelve solo JSON puro (RFC8259), sin comillas triples, sin markdown, sin texto adicional antes o después del JSON. TODOS los saltos de línea y caracteres de control deben estar escapados como `\\\\n`, `\\\\t`, etc. No uses saltos de línea reales dentro de las cadenas.
30      """;
31
32  PromptTemplate systemPromptTemplate = new SystemPromptTemplate(stringSystemMessage);
33
34  SystemMessage contextualMessage =
35      (SystemMessage)
36          systemPromptTemplate.createMessage(
37              Map.of(
38                  "sourceDatabaseType",
39                  promptInfo.getSourceDatabaseType(),
40                  "targetDatabaseType",
41                  promptInfo.getTargetDatabaseType(),
42                  "migrationRequirements",
43                  promptInfo.getMigrationRequirements(),
44                  "databaseSchema",
```

```
                promptInfo.getDatabaseSchema(),
                "language",
                "python"));
  return new Prompt(List.of(contextualMessage));
}
```

B.2. Migración del esquema

A continuación se presenta el Código B.2 el cual contiene el método de la clase `DbMigrationPromptBuilder` encargado de crear el prompt para migrar el esquema a partir de la información proporcionada por el usuario en el objeto de entrada. Este prompt nos devuelve como resultado un script con el esquema de la base de datos destino y una explicación asociada al mismo, con la cual el usuario puede entender qué se expresa .

Código B.2: Plantilla de prompt para la migración del esquema

```
/*** Construcción del primer prompt ***/
public Prompt buildFirstPromptToMigrateSchema(↩
    ↪ OpenrouterMigrateDatabaseRequestDto promptInfo) {

  String systemString =
      """
      Actúa como un experto en migración de esquemas de bases de datos {↩
          ↪ sourceDatabaseType} a {targetDatabaseType}.

      Tu tarea en esta fase es transformar el esquema de una base de datos {↩
          ↪ sourceDatabaseType} en el esquema especificado {↩
          ↪ targetDatabaseType}.
      Para ello, debes tener en cuenta el mapping de entidades, relaciones y ↩
          ↪ restricciones de la base de datos de origen a la base de datos ↩
          ↪ de destino.

      Información clave para la migración:
      - Base de datos de origen: {sourceDatabaseType}
      - Base de datos de destino: {targetDatabaseType}

      **IMPORTANTE**: La respuesta debe cumplir con el siguiente formato:

      {format}

      donde en la parte de "script" quiero que adjuntes el script y en la ↩
          ↪ parte de "scriptExplication"
      quiero que adjuntes una explicación detallada del script que has ↩
          ↪ generado.

      **IMPORTANTE**: Devuelve solo JSON puro (RFC8259), sin comillas triples↩
          ↪ , sin markdown, sin texto adicional antes o después del JSON. ↩
```

```
            ↪ TODOS los saltos de línea y caracteres de control deben estar ↩
            ↪ escapados como `\\\\n`, `\\\\t`, etc. No uses saltos de línea ↩
            ↪ reales dentro de las cadenas.

        **IMPORTANTE**: No abordes la migración de datos en esta fase, solo el ↩
            ↪ esquema teniendo en cuenta la integridad referencial y agregados↩
            ↪  propios de la base de datos destino.
        """;

    SystemMessage systemMessage =
        (SystemMessage)
            new SystemPromptTemplate(systemString)
                .createMessage(
                    Map.of(
                        "sourceDatabaseType",
                        promptInfo.getSourceDatabaseType(),
                        "targetDatabaseType",
                        promptInfo.getTargetDatabaseType(),
                        "format",
                        getFormatConverter().getFormat()));

    // Prompt del usuario
    String userString =
        """
        Debes migrar el esquema de una base de datos {sourceDatabaseType} a una↩
            ↪  base de datos {targetDatabaseType}.

        A continuación, se detallan los parámetros específicos para la migració↩
            ↪ n del esquema:

        - Esquema de la base de datos {sourceDatabaseType} original: {↩
            ↪ databaseSchema}
        - Requisitos y restricciones de la migración: {migrationRequirements}

        Crea el esquema de la base de datos {targetDatabaseType} en el lenguaje↩
            ↪  más apropiado para {targetDatabaseType} siguiendo las ↩
            ↪ instrucciones proporcionadas anteriormente.
        """;

    UserMessage userMessage =
        (UserMessage)
            new PromptTemplate(userString)
                .createMessage(
                    Map.of(
                        "sourceDatabaseType",
                        promptInfo.getSourceDatabaseType(),
                        "targetDatabaseType",
                        promptInfo.getTargetDatabaseType(),
                        "databaseSchema",
                        promptInfo.getDatabaseSchema(),
                        "migrationRequirements",
```

```
                promptInfo.getMigrationRequirements()));

  return new Prompt(List.of(systemMessage, userMessage));
}
```

B.3. Validación de la migración del esquema

En este apartado, se presenta el método Java en el Código B.3 que se encarga de crear los prompts para ofrecer al usuario un conjunto de pautas con las que poder validar la correcta migración del esquema en el sistema gestor destino, teniendo en cuenta el esquema original de referencia.

Código B.3: Plantilla de prompt para la validación manual de la migración del esquema

```
public Prompt buildPromptToValidateSchema(
    OpenrouterMigrateDatabaseRequestDto promptInfo, String
        ↪ scriptForMigrateSchema) {
  String systemString =
      """
      Actúa como un experto en validación de migración de esquemas de bases
          ↪ de datos.

      Tu tarea en esta fase es proporcionar pautas detalladas para que el
          ↪ usuario pueda comprobar que el esquema destino ha sido creado
          ↪ correctamente a partir del esquema {databaseSchema} de la base de
          ↪  datos de origen.
      Ten en cuenta que estas pautas deben cumplir con los requisitos del
          ↪ usuario: {migrationRequirements}

      Para ello, te sugerimos algunas cuestiones a tener en cuenta:
      - Los elementos del esquema la base de datos {databaseSchema} se han
          ↪ migrado al esquema de la base de datos destino.
      - Las restricciones de la base de datos origen están reflejadas en la
          ↪ base de datos destino.
      - El esquema creado esté optimizado para consultas como las propuestas
          ↪ en {accessRequirements} y rendimiento en {targetDatabaseType}.
      - Posibles inconsistencias o riesgos en la migración.

      **IMPORTANTE**: Genera un listado de pasos con las recomendaciones para
          ↪  realizar pruebas manuales de validación.

      **IMPORTANTE**: No uses formato Markdown ni comillas triples. Solo
          ↪ devuelve texto plano conformado por explicaciones e instrucciones
          ↪  propias de la base de datos destino para que el usuario pueda
          ↪ validar manualmente la migración del esquema generado en el paso
          ↪ anterior.

      **IMPORTANTE**: Utiliza el script de migración de esquema proporcionado
          ↪  como base para tus recomendaciones.
```

```
21      {scriptForMigrateSchema}
22      """;
23  // Crear el mensaje del sistema reemplazando los placeholders necesarios
24  SystemMessage systemMessage =
25      (SystemMessage)
26          new SystemPromptTemplate(systemString)
27              .createMessage(
28                  Map.of(
29                      "sourceDatabaseType",
30                      promptInfo.getSourceDatabaseType(),
31                      "targetDatabaseType",
32                      promptInfo.getTargetDatabaseType(),
33                      "accessRequirements",
34                      promptInfo.getAccessRequirements(),
35                      "databaseSchema",
36                      promptInfo.getDatabaseSchema(),
37                      "migrationRequirements",
38                      promptInfo.getMigrationRequirements(),
39                      "scriptForMigrateSchema",
40                      scriptForMigrateSchema));
41  // Mensaje del usuario: Se suministra el script generado para validar
42  String userString =
43      """
44      Se proporciona el siguiente script de migración de esquema:
45
46      {scriptForMigrateSchema}
47
48      Genera un conjunto de pautas y recomendaciones detalladas para que un ↩
            ↪ usuario pueda validar manualmente:
49      - La transformación del esquema desde {sourceDatabaseType} a {↩
            ↪ targetDatabaseType}.
50      - La correcta implementación y optimización del nuevo esquema en la ↩
            ↪ base de datos de destino.
51
52      Incluye pasos de verificación, sugerencias de pruebas manuales y puntos↩
            ↪  de control para asegurar la integridad del proceso.
53      """;
54
55  // Crear el mensaje del usuario con el script de migración
56  UserMessage userMessage =
57      (UserMessage)
58          new PromptTemplate(userString)
59              .createMessage(
60                  Map.of(
61                      "scriptForMigrateSchema",
62                      scriptForMigrateSchema,
63                      "sourceDatabaseType",
64                      promptInfo.getSourceDatabaseType(),
65                      "targetDatabaseType",
66                      promptInfo.getTargetDatabaseType()));
67
```

```
68    // Combinar ambos mensajes en un solo prompt
69    return new Prompt(List.of(systemMessage, userMessage));
70  }
```

B.4. Migración de los datos

En el siguiente código que se presenta, se puede observar el método Java en el Código B.4 el cual muestra la plantilla de prompt que permite construir los prompts encargados de proponer al modelo la obtención de un script Python con el que migrar los datos de una base de datos origen a una destino.

Código B.4: Plantilla para la migración de los datos

```
1  /*** Construcción del segundo prompt ***/
2   public Prompt buildSecondPromptToMigrateData(
3       OpenrouterMigrateDatabaseRequestDto promptInfo, String ↩
            ↪ scriptForTargetSchema) {
4     String systemString =
5         """
6         Actúa como un experto en migración de datos de bases de datos {↩
              ↪ sourceDatabaseType} a {targetDatabaseType}.
7
8         En esta fase, tu objetivo es migrar los datos de la base de datos {↩
              ↪ sourceDatabaseType} a la base de datos {targetDatabaseType}.
9         Para ello, debes, una vez, obtenido el esquema de destino, generar un c↩
              ↪ ódigo para migrar todos los datos de la base de datos original a ↩
              ↪ la base de datos destino.
10
11        1. Asegurar la integridad referencial y consistencia de los datos ↩
              ↪ migrados.
12        2. Optimizar el proceso para garantizar la eficiencia en la migración.
13        3. Producir un código que permita la migración de datos de la base de ↩
              ↪ datos {sourceDatabaseType} a la base de datos {targetDatabaseType↩
              ↪ }.
14
15        **Información clave para la migración:**
16        - Esquema de la base de datos origen: {databaseSchema}
17        - El esquema de la base de datos destino generado a partir del script: ↩
              ↪ {scriptForTargetSchema}
18
19        **IMPORTANTE:** La respuesta debe cumplir con el siguiente formato:
20        {format}
21        donde en la parte de "script" quiero que adjuntes el script y en la ↩
              ↪ parte de "scriptExplication"
22        quiero que incluyas una explicación detallada del script que has ↩
              ↪ generado.
23
24        **IMPORTANTE:** Devuelve solo JSON puro (RFC8259), sin comillas triples↩
              ↪ , sin markdown, sin texto adicional antes o después del JSON. ↩
```

```
            ↪ TODOS los saltos de línea y caracteres de control deben estar ↩
            ↪ escapados como `\\\\n`, `\\\\t`, etc. No uses saltos de línea ↩
            ↪ reales dentro de las cadenas.

        **IMPORTANTE:** Si la migración requerida se realiza desde una base de ↩
            ↪ datos origen SQL hacia base de datos destino SQL con el MISMO ↩
            ↪ sistema gestor de bases de datos, el script de migración de datos↩
            ↪  debe ser un script SQL adaptado a la {targetDatabaseType}.
        En cualquier otro caso, el script de migración de datos debe ser un ↩
            ↪ script {language}.
        """;

    SystemMessage systemMessage =
        (SystemMessage)
            new SystemPromptTemplate(systemString)
                .createMessage(
                    Map.of(
                        "sourceDatabaseType",
                        promptInfo.getSourceDatabaseType(),
                        "targetDatabaseType",
                        promptInfo.getTargetDatabaseType(),
                        "scriptForTargetSchema",
                        scriptForTargetSchema,
                        "databaseSchema",
                        promptInfo.getDatabaseSchema(),
                        "format",
                        getFormatConverter().getFormat(),
                        "language",
                        "python"));

    String userString =
        """
        Debes migrar los datos de una base de datos {sourceDatabaseType} a una ↩
            ↪ base de datos {targetDatabaseType}.

        A continuación, se detallan los parámetros específicos para la migració↩
            ↪ n de datos:

        - Esquema de la base de datos {sourceDatabaseType} original: {↩
            ↪ databaseSchema}
        - Requisitos y restricciones de la migración: {migrationRequirements}

        Genera un script para migrar los datos a {targetDatabaseType} siguiendo↩
            ↪  las especificaciones proporcionadas anteriormente.
        """;

    UserMessage userMessage =
        (UserMessage)
            new PromptTemplate(userString)
                .createMessage(
                    Map.of(
```

```
                    "sourceDatabaseType",
                    promptInfo.getSourceDatabaseType(),
                    "targetDatabaseType",
                    promptInfo.getTargetDatabaseType(),
                    "databaseSchema",
                    promptInfo.getDatabaseSchema(),
                    "migrationRequirements",
                    promptInfo.getMigrationRequirements()));

  return new Prompt(List.of(systemMessage, userMessage));
}
```

B.5. Validación de la migración de base de datos

Esta última plantilla de prompt se caracteriza por una alta especialización en la generación de scripts de validación de migraciones de base de datos. La plantilla está diseñada con el fin de validar la migración, haciendo hincapié en la consistencia del esquema y los datos, además de comprobar la equivalencia de resultados entre consultas en origen y destino. A continuación, el Código B.5 muestra el método Java que permite generar prompts de este tipo.

Código B.5: Plantilla para la validación de la migración del esquema de forma manual

```
/*** Construcción del tercer prompt ***/
public Prompt buildThirdPromptToValidateMigration(
    OpenrouterMigrateDatabaseRequestDto promptInfo, String ↩
        ↪ scriptForTargetData) {
  String systemString =
      """
      Actúa como un experto en validación de migración de bases de datos {↩
          ↪ sourceDatabaseType} a {targetDatabaseType}.

      En esta fase, tu tarea es verificar la consistencia de los datos ↩
          ↪ migrados y el esquema generado en los pasos anteriores. Para ello↩
          ↪ , debes generar un código que permita validar la migración de ↩
          ↪ datos y esquema, asegurando que la información se haya migrado ↩
          ↪ correctamente. Algunas tareas clave son:

      1. Comparar los elementos de la base de datos {sourceDatabaseType} con ↩
          ↪ los elementos de la base de datos {targetDatabaseType}.
      2. Verificar la integridad referencial y consistencia de los datos ↩
          ↪ migrados.
      3. Comprobar que las mismas consultas realizadas en la base de datos ↩
          ↪ origen y destino devuelven los mismos resultados
      Información clave para la validación:
      - Base de datos de origen: {sourceDatabaseType}
      - Base de datos de destino: {targetDatabaseType}
      - Esquema de la base de datos original: {databaseSchema}
```

```
17      - Requisitos y restricciones de la migración: {migrationRequirements}
18
19      **IMPORTANTE**: Utiliza el script de migración de datos generado en la ↩
            ↪ fase anterior para validar la migración. {scriptForTargetData}
20
21      **IMPORTANTE**: Es crucial que la respuesta debe cumplir con el ↩
            ↪ siguiente formato:
22      {format}
23      donde en la parte de "script" quiero que adjuntes el script y en la ↩
            ↪ parte de "scriptExplication"
24      quiero que incluyas un informe detallado sobre la migración y validació↩
            ↪ n de datos.
25
26      **IMPORTANTE**: Devuelve solo JSON puro (RFC8259), sin comillas triples↩
            ↪ , sin markdown, sin texto adicional antes o después del JSON. ↩
            ↪ TODOS los saltos de línea y caracteres de control deben estar ↩
            ↪ escapados como `\\\\n`, `\\\\t`, etc. No uses saltos de línea ↩
            ↪ reales dentro de las cadenas.
27      """;
28
29  SystemMessage systemMessage =
30      (SystemMessage)
31          new SystemPromptTemplate(systemString)
32              .createMessage(
33                  Map.of(
34                      "sourceDatabaseType",
35                      promptInfo.getSourceDatabaseType(),
36                      "targetDatabaseType",
37                      promptInfo.getTargetDatabaseType(),
38                      "scriptForTargetData",
39                      scriptForTargetData,
40                      "format",
41                      getFormatConverter().getFormat(),
42                      "databaseSchema",
43                      promptInfo.getDatabaseSchema(),
44                      "migrationRequirements",
45                      promptInfo.getMigrationRequirements()));
46
47  String userString =
48      """
49      Debes validar la migración de datos y esquema de una base de datos {↩
            ↪ sourceDatabaseType} a una base de datos {targetDatabaseType}.
50      La validación de los datos debe comprobar no sólo que los datos se ↩
            ↪ hayan migrado bien, sino también que las consultas a la base de ↩
            ↪ datos que haya podido especificar el usuario en {↩
            ↪ accessRequirements} se pueden ejecutar correctamente en la base ↩
            ↪ de datos destino.
51
52      A continuación, se detallan los parámetros específicos para la validaci↩
            ↪ ón de la migración:
53
```

```
54          - Esquema {sourceDatabaseType} original: {databaseSchema}
55          - Requisitos y restricciones de la migración: {migrationRequirements}
56          - Requisitos de la aplicación (consultas más realizadas, índices...): {↩
                ↪ accessRequirements}
57
58          Genera un script para validar la migración de datos y esquema a la base↩
                ↪  de datos {targetDatabaseType} siguiendo las especificaciones ↩
                ↪ proporcionadas anteriormente.
59          """;
60
61      UserMessage userMessage =
62          (UserMessage)
63              new PromptTemplate(userString)
64                  .createMessage(
65                      Map.of(
66                          "sourceDatabaseType",
67                          promptInfo.getSourceDatabaseType(),
68                          "targetDatabaseType",
69                          promptInfo.getTargetDatabaseType(),
70                          "databaseSchema",
71                          promptInfo.getDatabaseSchema(),
72                          "migrationRequirements",
73                          promptInfo.getMigrationRequirements(),
74                          "accessRequirements",
75                          promptInfo.getAccessRequirements()));
76
77      return new Prompt(List.of(systemMessage, userMessage));
78  }
```

C. Plantillas de prompts diseñadas para la migración del código de acceso a bases de datos

En el siguiente anexo presentamos de forma detallada las plantillas de prompts definidas en LLMigrator-DB para automatizar la migración del código de acceso a bases de datos. Dicha migración requiere como información de entrada las entidades del modelo de dominio, los repositorios asociados a dichas entidades y la lógica de negocio de la capa de aplicación que hace uso de dichos repositorios.

Las plantillas de prompt han sido diseñadas utilizando las técnicas presentadas en la Sección 5.1. Concretamente, este anexo está estructurado de acuerdo a las tres plantillas de prompts que hemos diseñado: contexto inicial de la migración de código, migración del código referente al modelo del dominio y sus respectivos repositorios, y la migración de la lógica de negocio de los servicios.

Cada plantilla de prompt se define con uno o varios mensajes, los cuales combinan instrucciones concretas, el contexto asociado y una serie de restricciones. Para cada fase del proceso de migración de código se presenta la plantilla utilizada.

Las plantillas de prompts han sido introducidas de forma fragmentada en la Sección 5.4.2. La clase Java principal es `DbCodeMigrationPromptBuilder`, que se encarga de la construcción de los prompts a partir de las plantillas que define. La construcción de estos prompts se hace cada vez que el usuario ha hecho una petición al punto de entrada `/api/v1/code-migrations`.

C.1. Contextualización del modelo

A continuación, se presenta el Código C.1 que contiene la plantilla de prompt utilizada para la construcción del prompt contextual que se envía al modelo.

Código C.1: Plantilla para la contextualización del modelo

```
1 public Prompt buildContextualPrompt(OpenrouterMigrateCodeDatabaseRequestDto ↩
      ↪ request) {
2    String systemString =
3        """
4         Actúa como un experto en migración de código. Tu objetivo es guiar la ↩
              ↪ migración de una capa de persistencia completa desde una pila ↩
```

```
            ↪ tecnológica a otra, asegurando que el código resultante sea ↩
            ↪ idiomático y eficiente en el nuevo entorno.

        El proceso se dividirá en varias fases:
        1. **Migración del Modelo de Dominio y Repositorios:** Migrar las ↩
            ↪ clases de entidad y los patrones de acceso a datos (repositorios↩
            ↪ /DAO).
        2. **Migración de Lógica de Negocio de los servicios de la aplicación**↩
            ↪  Adaptar los servicios que consumen los repositorios.

        Contexto General de la Migración:
        - **Lenguaje Origen:** {sourceLanguage}
        - **Lenguaje Destino:** {targetLanguage}
        - **Framework Origen:** {sourceFramework}
        - **Framework Destino:** {targetFramework}
        - **Base de Datos Origen:** {sourceDatabase}
        - **Base de Datos Destino:** {targetDatabase}
        - **Arquitectura de Salida del código Deseada:** {outputFormat}
        - **Esquema de Base de Datos Destino:** {targetSchema}
        - **Requisitos Funcionales Clave:** {functionalRequirements}

        Instrucciones Generales:
        - NO generes ninguna respuesta ahora. Asimila este contexto.
        - En los siguientes pasos, te proporcionaré el código fuente específico↩
            ↪  a migrar.
        - Asegúrate de que el código generado sea idiomático para el lenguaje y↩
            ↪  framework de destino.
        - Presta especial atención a la gestión de transacciones, tipos de ↩
            ↪ datos y convenciones de nomenclatura del ecosistema de destino.
        """;

    SystemMessage contextualMessage =
        (SystemMessage)
            new SystemPromptTemplate(systemString)
                .createMessage(
                    Map.of(
                        "sourceLanguage",
                        request.getSourceLanguage().name(),
                        "targetLanguage",
                        request.getTargetLanguage().name(),
                        "sourceFramework",
                        request.getSourceFramework().name(),
                        "targetFramework",
                        request.getTargetFramework().name(),
                        "sourceDatabase",
                        request.getSourceDatabase().name(),
                        "targetDatabase",
                        request.getTargetDatabase().name(),
                        "outputFormat",
                        request.getOutputFormat(),
                        "functionalRequirements",
```

```
                    request.getFunctionalRequirements(),
                    "targetSchema",
                    request.getTargetSchema()));

    // Este prompt no necesita UserMessage, solo establece el sistema.
    return new Prompt(List.of(contextualMessage));
}
```

C.2. Migración de código

En esta sección se presentan los métodos encargados de realizar la migración y adaptación del código de la capa de acceso a datos.

C.2.1. Migración del código del dominio y los repositorios asociados

En este apartado se muestra el Código C.2 que contiene el método Java que incluye la plantilla de prompt que permite la construcción de los prompts para adaptar el modelo del dominio y los repositorios asociados.

Código C.2: Plantilla para la migración de clases del dominio y los repositorios asociados

```
/** Primer paso: Migra las clases del modelo de dominio (entidades) y los repositorios/DAOs. */
public Prompt buildFirstPromptToMigrateDomainModelAndRepository(
    OpenrouterMigrateCodeDatabaseRequestDto request) {

  // Mensaje del sistema: Define el rol y las reglas para esta tarea específica.
  String systemString =
      """
      Actúa como un experto en migración de capas de persistencia, especializado en Modelos de Dominio y Repositorios.
      Tu tarea es migrar el código del modelo de dominio y del repositorio desde un entorno {sourceLanguage}/{sourceFramework}/{sourceDatabase} a {targetLanguage}/{targetFramework}/{targetDatabase}

      Instrucciones Clave:

      1. **Modelo de Dominio:** Convierte las clases de entidad de {sourceLanguage} a {targetLanguage}. Presta atención a las anotaciones de mapeo (Ej: `@Entity`, `@Id` de JPA) y tradúcelas a sus equivalentes en `{targetFramework}`. Si el destino es NoSQL, adapta la estructura para reflejar documentos anidados o referencias.
```

```
14      2. **Repositorios/DAOs:** Migra las interfaces o clases de acceso a datos. Implementa los métodos CRUD estándar utilizando las abstracciones de `{targetFramework}` (Ej: `CrudRepository` de Spring Data, `DbSet` de EF Core).
15      3. **Cohesión:** Asegúrate de que los repositorios migrados operen correctamente sobre los modelos de dominio migrados. Los tipos de datos y nombres deben ser consistentes.
16      4. **Arquitectura:** Organiza el código generado siguiendo el patrón de diseño `{outputFormat}`.
17      5. **Requisitos Funcionales:** Implementa los siguientes requisitos en el código migrado: `{functionalRequirements}`. Esto puede incluir paginación, joins, etc.
18      6. **IDs:** Asume que los identificadores (IDs) de las entidades deben mantenerse consistentes entre el origen y el destino.
19
20      **IMPORTANTE**: La respuesta debe ser un JSON puro (RFC8259) que cumpla con el siguiente formato. No incluyas ` ```json ` ni ningún texto fuera del JSON.
21      {format}
22      """;
23
24  SystemMessage systemMessage =
25      (SystemMessage)
26          new SystemPromptTemplate(systemString)
27              .createMessage(
28                  Map.of(
29                      "sourceLanguage",
30                      request.getSourceLanguage().name(),
31                      "targetLanguage",
32                      request.getTargetLanguage().name(),
33                      "sourceFramework",
34                      request.getSourceFramework().name(),
35                      "targetFramework",
36                      request.getTargetFramework().name(),
37                      "sourceDatabase",
38                      request.getSourceDatabase().name(),
39                      "targetDatabase",
40                      request.getTargetDatabase().name(),
41                      "outputFormat",
42                      request.getOutputFormat(),
43                      "format",
44                      getFormatConverter().getFormat(),
45                      "functionalRequirements",
46                      request.getFunctionalRequirements()));
47
48  // Mensaje del usuario: Proporciona el código concreto a migrar.
49  String userString =
50      """
51      Migra el siguiente código de {sourceFramework} a {targetFramework}.
52
53      **Código del Modelo de Dominio Origen (en {sourceLanguage}):**
```

```
        ```
        {domainModelCode}
        ```

        **Código del Repositorio Origen (en {sourceLanguage}):**
        ```
        {repositoryCode}
        ```

        Genera el código equivalente para `{targetFramework}` en el lenguaje {targetLanguage}, siguiendo todas las instrucciones y el formato de salida especificado. En la explicación, detalla los cambios clave en las anotaciones, la estructura del repositorio y cualquier consideración importante.
        """;

    UserMessage userMessage =
        (UserMessage)
            new PromptTemplate(userString)
                .createMessage(
                    Map.of(
                        "sourceLanguage", request.getSourceLanguage().name(),
                        "targetLanguage", request.getTargetLanguage().name(),
                        "sourceFramework", request.getSourceFramework().name(),
                        "targetFramework", request.getTargetFramework().name(),
                        "domainModelCode", request.getDomainModelCode(),
                        "repositoryCode", request.getRepositoryCode()));

    return new Prompt(List.of(systemMessage, userMessage));
}
```

C.2.2. Migración de la lógica de negocio

En este apartado se muestra el Código C.3 en el cual se puede observar el método Java que define la plantilla de prompt encargada de la construcción de los prompts cuya responsabilidad es adaptar el código de los servicios para que hagan uso de los nuevos repositorios asociados.

Código C.3: Plantilla para la migración de los servicios haciendo uso de los nuevos repositorios definidos

```
public Prompt buildPromptToMigrateServiceAndCustomQueries(
    OpenrouterMigrateCodeDatabaseRequestDto request, String scriptForDomainAndRepo) {

  // Mensaje del sistema: Define el rol para esta tarea avanzada.
  String systemString =
      """
      Actúa como un experto en migración de lógica de negocio entre diferentes lenguajes y frameworks.
```

```
8
9       Tu tarea es migrar código de perteneciente a la capa de servicio, ↩
            ↪ asumiendo que el Modelo de Dominio y los Repositorios ya han sido↩
            ↪  migrados a `{targetLanguage}` y `{targetFramework}`. Debes ↩
            ↪ adaptar el código que *utiliza* estos repositorios. El código que↩
            ↪  utiliza estos repositorios y el modelo de dominio ya migrado es ↩
            ↪ el siguiente:
10
11      {scriptForDomainAndRepo}
12
13      Instrucciones Clave:
14      1. **Adaptación de Servicios:** migra el código de los servicios para ↩
            ↪ que utilice las nuevas interfaces de repositorio migradas en {↩
            ↪ targetLanguage}. Presta especial atención a la gestión de ↩
            ↪ transacciones (Ej: `@Transactional` en Spring).
15      2. **Requisitos Funcionales:** Implementa los siguientes requisitos en ↩
            ↪ el código migrado: `{functionalRequirements}`. Esto puede incluir↩
            ↪  paginación, ordenación, proyecciones, etc.
16      3. **Ejemplo de Uso:** Utiliza el ejemplo de uso proporcionado para ↩
            ↪ guiar y verificar tu migración, mostrando cómo se llamaría al ↩
            ↪ nuevo código.
17
18      IMPORTANTE: La respuesta debe ser un JSON puro (RFC8259) que cumpla con↩
            ↪  el siguiente formato. No incluyas ` ```json ` ni ningún texto ↩
            ↪ fuera del JSON.
19      {format}
20      """;
21
22  SystemMessage systemMessage =
23      (SystemMessage)
24          new SystemPromptTemplate(systemString)
25              .createMessage(
26                  Map.of(
27                      "targetLanguage",
28                      request.getTargetLanguage().name(),
29                      "targetFramework",
30                      request.getTargetFramework().name(),
31                      "functionalRequirements",
32                      request.getFunctionalRequirements(),
33                      "format",
34                      getFormatConverter().getFormat(),
35                      "scriptForDomainAndRepo",
36                      scriptForDomainAndRepo));
37
38  // Mensaje del usuario: Proporciona el código de alto nivel y las consultas
39  // específicas.
40  String userString =
41      """
42      Continuando con la migración a {targetFramework}, ahora migra el código↩
            ↪  referente a la lógica de negocio del lenguaje {sourceLanguage} ↩
            ↪ al lenguaje {targetLanguage}.
```

```
43
44      **Código de la capa de servicio en {sourceLanguage}/{sourceFramework}**
45      ```
46      {businessLogicCode}
47      ```
48
49      **Ejemplo de Uso en la Capa de Servicio Origen (opcional, pero muy útil
            ↪ ):**
50      ```
51      {usageExampleCode}
52      ```
53
54      Genera el código equivalente para `{targetFramework}` en {
            ↪ targetLanguage}. En la explicación, justifica tus decisiones para
            ↪  la traducción de cada consulta, cómo se integra con los
            ↪ repositorios y cómo se ha implementado la gestión de
            ↪ transacciones o los requisitos funcionales.
55      """;
56
57    UserMessage userMessage =
58        (UserMessage)
59           new PromptTemplate(userString)
60               .createMessage(
61                   Map.of(
62                       "sourceLanguage",
63                       request.getSourceLanguage().name(),
64                       "targetLanguage",
65                       request.getTargetLanguage().name(),
66                       "sourceFramework",
67                       request.getSourceFramework().name(),
68                       "targetFramework",
69                       request.getTargetFramework().name(),
70                       "businessLogicCode",
71                       request.getBusinessLogicCode(),
72                       "usageExampleCode",
73                       Objects.toString(
74                           request.getUsageExampleCode(),
75                           "No se proporcionó un ejemplo de uso.")));
76
77    return new Prompt(List.of(systemMessage, userMessage));
78  }
```