UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Programa de Apoyo al Egreso para Alumnos con Práctica Profesional Supervisada

TÍTULO: Migración de base de datos modelo relacional a NoSQL

AUTOR/A: Bacci Facundo

DIRECTOR/A ACADÉMICO: Grigera Julián

DIRECTOR/A PROFESIONAL: Scigliotto Guillermo

CODIRECTOR/A ACADÉMICO:

CARRERA: Licenciatura en Sistemas

RESUMEN

PBAC es un sistema de software que opera como plataforma integral para la divulgación de las contrataciones estatales en la Provincia de Buenos Aires, sirviendo de punto de encuentro entre los usuarios compradores, mayormente vinculados a ministerios y organismos públicos, y los proveedores que participan en licitaciones para comercializar sus productos. El sistema posee dos bases de datos relacionales, la de COMPRAS que incluye todo lo relacionado al negocio de la aplicación, y la de NOTIFICACIONES. Esta ultima con el paso del tiempo adquirió un gran volumen de datos, y no estaba siendo aprovechada correctamente, lo cual generaba problemas de performance. Es por esto que se llevó a cabo un proyecto de migración para llevar las notificaciones a un motor de búsqueda mas aprovechable para el dominio del problema.

Palabras Claves

Aplicaciones web, ingeniería de software, base de datos, integración de proyectos, desarrollo Back-End, migración de bases de datos, base de datos NOSQL,

Conclusiones

La migración de la base de datos a un paradigma no relacional representó un esfuerzo técnico significativo que ha planteado diversos desafíos a lo largo del proyecto. La transición desde una arquitectura basada en SQL Server hacia una estructura orientada a documentos ha requerido no solo una revisión profunda de los modelos de datos, sino también la implementación de nuevas estrategias de gestión y almacenamiento de información. El proyecto cumplirá con las expectativas iniciales, mejorando notablemente la capacidad de manejo de grandes volúmenes de datos. Este cambio estructural permite una mayor flexibilidad en el desarrollo de futuras funcionalidades, sentando así bases sólidas para la evolución de la plataforma.

Trabajos Realizados

Análisis e investigación de herramientas a utilizar para llevar una base de datos SQL con problemas de escalado y eficiencia, a una arquitectura de documentos NoSQL más adecuada. Se dividió el proyecto en etapas, comenzando por el análisis y utilización de la herramienta de migración, para luego desarrollar la capa de la aplicación encargada de interactuar con la nueva base de datos.

Trabajos Futuros

Se propone a futuro una serie de lineamientos a desarrollar dentro de la aplicación teniendo en cuenta su funcionamiento actual. Entre ellas, implementar mejoras en los servicios que comunican los proyectos, migrar .net Framework a netcore, pensar una arquitectura de dockerizacion

Fecha de la presentación: NOVIEMBRE 2024

Índice General

Índice de Figuras

ANEXO 4 – Formato tesina PAE-PPS

1. Exposición de lo realizado en la PPS

Durante las prácticas se trabajó en una serie de tareas fundamentales orientadas a la mejora y modernización de un proyecto para la gestión de notificaciones y tareas en el contexto del sistema de compras públicas de la provincia de Buenos Aires. Estas actividades implicaron tanto un análisis de los procesos actuales como la identificación de áreas clave para su optimización, con especial enfoque en la migración de la base de datos subyacente, la cual estaba orientada hacia un esquema relacional. El principal objetivo de esta migración fue la implementación de una solución NoSQL, que ofreciera una mayor flexibilidad y eficiencia en el manejo de volúmenes masivos de datos, facilitando así la capacidad de escalar el sistema de manera horizontal para soportar su continuo crecimiento y evolución.

Este esfuerzo de modernización no sólo apunta a mejorar el rendimiento general del sistema, sino también a garantizar la continuidad operativa y su adecuación a las demandas futuras, tanto en términos de gestión de información como de velocidad de procesamiento. La transición hacia una base de datos NoSQL representa un desafío técnico importante, pero resulta fundamental para superar las limitaciones del modelo relacional anterior, especialmente en contextos donde la cantidad de datos es considerable.

El Sistema PBAC[1] (Provincia de Buenos Aires Compras) constituye una herramienta estratégica para el gobierno provincial, diseñada específicamente para gestionar y optimizar los procesos de compras y contrataciones públicas. Esta plataforma digital centraliza de manera eficiente toda la información relacionada con los procesos licitatorios, lo que facilita tanto la transparencia como la eficiencia en la adquisición de bienes y servicios. Su implementación ha permitido estandarizar procedimientos tales como la licitación, la contratación directa y el concurso de precios, asegurando que las distintas entidades provinciales cuenten con un sistema robusto que promueve la rendición de cuentas y el control exhaustivo de los recursos públicos.

En este contexto, se realizó un trabajo significativo en el desarrollo y ajuste de la API de notificaciones, un servicio clave del software mencionado, que permite la automatización de los procesos de comunicación dentro del ecosistema de PBAC. Esta API tiene la función de integrar y gestionar las notificaciones generadas a lo

[1] [1] PBAC: Normativa del sistema PBAC en la provincia de Buenos Aires. https://normas.gba.gob.ar/documentos/xDyQjOHy.pdf

largo de los procesos licitatorios y de contratación, asegurando que tanto las entidades públicas como los proveedores estén informados en tiempo y forma sobre el estado de las distintas etapas del proceso. La optimización de este sistema de notificaciones no solo agiliza la interacción entre los distintos actores involucrados, sino que también mejora la transparencia del sistema al garantizar una mayor trazabilidad de las comunicaciones. El proyecto de migración contribuye a fortalecer el impacto del Sistema PBAC en términos de modernización administrativa y mejora continua de la eficiencia operativa en la gestión pública.

2. Informe técnico

2.1 Resumen

Durante la Práctica Profesional Supervisada, se trabajó en el análisis y realización de un proyecto de migración de una base de datos relacional orientada a la gestión de notificaciones y tareas dentro de un sistema. Inicialmente, se llevó a cabo un análisis exhaustivo de las tablas involucradas y sus características, evaluando el volumen de datos y su almacenamiento.

Uno de los objetivos principales del proyecto fue realizar un proyecto de migración de las estructuras de datos desde una base de datos relacional (SQL Server)[2] a una base de datos NoSQL , en este caso MongoDB[3] . La elección de MongoDB fue clave debido a sus ventajas significativas en cuanto a la gestión de grandes volúmenes de datos, su capacidad de escalabilidad horizontal, y su flexibilidad al trabajar con datos no estructurados o semiestructurados.

Además, el sistema relacional, que supera los 100 GB de datos almacenados en SQL Server , representa un desafío en términos de rendimiento y escalabilidad. Con la migración a MongoDB, no solo se mejora el rendimiento, sino que también se logra una importante reducción en el espacio de almacenamiento utilizado. La estructura de documentos en MongoDB , junto con su capacidad para indexar y gestionar grandes cantidades de datos de manera eficiente, permite optimizar los

---

[2] SQL Server

https://learn.microsoft.com/es-es/sql/sql-server/?view=sql-server-ver16

[3] MongoDB

https://www.mongodb.com/es
https://www.mongodb.com/docs/drivers/csharp/current/

recursos y garantizar que el sistema pudiera seguir creciendo sin comprometer la performance.

Para facilitar la transición, se estudió la utilización de la herramienta *MongoDB Relational Migrator* [4], que automatiza parte del proceso y permite una migración exitosa desde SQL Server a MongoDB . También se implementaron modificaciones en la API de notificaciones, adaptando las capas de acceso a datos (DAO) para que funcionaran con MongoDB .

2.2 Palabras claves

Aplicaciones web, ingeniería de software, base de datos, integración de proyectos, desarrollo back-end, migración de base de datos, base de datos NOSQL

2.3 Marco conceptual

Si bien la migración de bases de datos es una tarea que se realiza frecuentemente en el ámbito de la industria, existen también múltiples reportes en la academia que puntualizan sobre desafíos surgidos del cambio de paradigma entre RDBMS y NoSQL . En un estudio sobre migración hacia NoSQL , Aggoune et al. (2020) proponen un método para transformar bases de datos objeto-relacionales en bases de datos orientadas a documentos, utilizando MongoDB como base. Un aspecto clave del proceso es la desnormalización, necesaria para adaptar los datos a un modelo flexible y semi-estructurado, característico de MongoDB . El método incluye la extracción del esquema relacional, el mapeo a un esquema orientado a documentos, la generación de documentos JSON y su posterior carga en la base de datos NoSQL , con mejoras en flexibilidad, escalabilidad y rendimiento, especialmente en aplicaciones de big data.

[4] MongoDB Relational Migrator

https://www.mongodb.com/docs/relational-migrator/

Por otro lado, El Alami et al. (2024) abordan la migración desde una perspectiva que integra conceptos de objetos, enriquecimiento semántico y metadatos. A través de un proceso automatizado, se transforman los esquemas relacionales en un formato enriquecido y aplanado, adecuado para la flexibilidad de NoSQL . Este enfoque se destaca por su capacidad para mantener la integridad de los datos y mejorar su calidad mediante el enriquecimiento semántico, permitiendo una transición más ágil y efectiva a modelos documentales. Este proceso incluye varias etapas, como el análisis del esquema, transformación de datos, validación y su posterior carga, resaltando la importancia del enriquecimiento semántico para la integridad y utilidad de los datos.

En un enfoque integral, Erraji et al. (2023) proponen un modelo de migración completo hacia MongoDB llamado "TMSDRDND", que divide el proceso en capas de transformación de estructura y semántica, junto con la migración de datos mediante un sistema ETL (Extract, Transform, Load). Este enfoque incluye el uso de herramientas como Mongoose para validar esquemas y asegurar la consistencia e integridad de los datos migrados. El uso de validadores durante la migración refleja el valor de MongoDB en la modernización de sistemas que requieren alta escalabilidad, flexibilidad y eficiencia para manejar grandes volúmenes de datos.

Finalmente, Shaikh et al. (2017) exploran la necesidad creciente de migrar desde SQL a MongoDB debido a las limitaciones de las bases de datos relacionales al manejar grandes volúmenes de datos no estructurados. Su enfoque se basa en una interfaz gráfica que facilita tanto la migración de datos como la conversión automática de consultas SQL a MongoDB , lo que reduce las barreras técnicas para los usuarios menos familiarizados con la sintaxis de MongoDB . Este tipo de herramientas automáticas es clave para minimizar errores y mejorar el rendimiento en la migración, lo que facilita una transición más eficiente hacia sistemas NoSQL modernos.

Cada uno de estos enfoques académicos nos brinda diferentes perspectivas sobre cómo superar las dificultades asociadas con la migración de bases de datos relacionales a NoSQL , destacando aspectos como la desnormalización, el enriquecimiento semántico, la automatización y la validación, elementos cruciales para una transición exitosa en proyectos que requieren flexibilidad y escalabilidad.

2.4 Trabajo Realizado

En esta sección se detallan las actividades llevadas a cabo durante el desarrollo del proyecto, abordando tanto el análisis del sistema preexistente como la selección de tecnologías y la implementación técnica. Se exploran distintos aspectos clave que permitieron definir y ejecutar las modificaciones necesarias para mejorar el

funcionamiento del sistema con la propuesta. En primer lugar, se realiza un estudio del sistema actual, evaluando su infraestructura y funcionamiento. Posteriormente, se presenta un análisis de las tecnologías más adecuadas para los cambios propuestos. Finalmente, se describe el desarrollo técnico, abarcando las implementaciones y ajustes realizados para alcanzar los objetivos del proyecto.

2.4.1 Estudio del sistema actual

El Sistema PBAC es una plataforma digital desarrollada por el Gobierno de la Provincia de Buenos Aires para gestionar y optimizar el proceso de compras y contrataciones públicas. Esta herramienta centraliza toda la información relacionada con los procesos licitatorios, lo que facilita la transparencia, la eficiencia y el control en la adquisición de bienes, servicios y obras por parte de las distintas entidades públicas provinciales.

El sistema permite a estas entidades realizar sus compras de manera más eficiente, aplicando procedimientos estandarizados como la licitación, la contratación directa o el concurso de precios. Una de sus principales características es la gestión de licitaciones electrónicas, lo que permite que los procesos se realicen de manera digital, fomentando una mayor participación de proveedores, mejorando la transparencia y acelerando los tiempos de ejecución. Además, el sistema garantiza un acceso público a la información sobre las compras y contrataciones, lo cual contribuye a mejorar la rendición de cuentas y aumentar la confianza en la administración pública. La centralización de los procedimientos de compra y contratación permite también una mejor optimización de los recursos públicos, reduciendo costos y mejorando la eficiencia operativa.

Para los proveedores, PBAC ofrece la posibilidad de registrarse y participar en los procesos licitatorios, lo que amplía las oportunidades de negocio tanto para proveedores locales como nacionales. De esta manera, PBAC se consolida como una herramienta clave en la estrategia del gobierno provincial para modernizar la gestión pública, no solo promoviendo la transparencia y la eficiencia, sino también impulsando la participación de más actores en los procesos licitatorios, al tiempo que facilita un control exhaustivo y accesible de los recursos públicos.

Infraestructura
El sistema actual se ejecuta sobre una arquitectura de base de datos relacional, específicamente en SQL Server , que centraliza la información de notificaciones y tareas de los usuarios. Mediante la siguiente query SQL se analizó el espacio en MB que representa cada tabla actualmente:

```
SELECT
```

```
t.NAME AS Tabla,
s.Name AS Esquema,
p.rows AS NumeroDeFilas,
CAST(ROUND(((SUM(a.total_pages) * 8) / 1024.00), 2) AS NUMERIC(36, 2)) AS TotalEspacio_MB,
CAST(ROUND(((SUM(a.used_pages) * 8) / 1024.00), 2) AS NUMERIC(36, 2)) AS
EspacioUtilizado_MB,
CAST(ROUND(((SUM(a.total_pages) - SUM(a.used_pages)) * 8) / 1024.00, 2) AS NUMERIC(36, 2))
AS EspacioNoUtilizado_MB
FROM
sys.tables t
INNER JOIN sys.indexes i ON t.OBJECT_ID = i.object_id
INNER JOIN sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
INNER JOIN sys.allocation_units a ON p.partition_id = a.container_id
LEFT OUTER JOIN sys.schemas s ON t.schema_id = s.schema_id
GROUP BY t.Name, s.Name, p.Rows
ORDER BY TotalEspacio_MB desc
```

La consulta SQL obtuvo como resultado un listado de cada una de las tablas involucradas con detalles generales de su contenido. El resultado completo con los detalles se pueden observar en la Figura 1.

	Tabla	Esquema	NumeroDeFilas	TotalEspacio_MB	EspacioUtilizado_MB	EspacioNoUtilizado_I
1	NotificacionEscritorio	dbo	131229089	74227.55	74217.77	9.77
2	NotificacionEscritorioHistorica	dbo	37275011	20360.15	20356.82	3.33
3	NotificacionMensajeria	dbo	6486540	20110.82	20108.31	2.51
4	TareaEscritorio	dbo	5389709	3738.61	3737.96	0.65
5	TareaEscritorioComprador	dbo	533403	326.85	326.70	0.15
6	AsociacionGestionEscritorioComprador	dbo	43741	3.06	2.43	0.63
7	temp	dbo	1758	0.14	0.09	0.05
8	Pliego	dbo	220	0.03	0.03	0.00
9	AvisoEmail	dbo	1	0.02	0.02	0.00
10	NotificacionMobile	dbo	0	0.00	0.00	0.00

Figura 1. Tamaño de la base de datos SQL.

Antes de analizar el modelo de la base de datos en detalle, es importante describir la estructura general del sistema actual. El modelo de la base de datos se basa en una serie de entidades que almacenan información relacionada con las notificaciones y tareas de los usuarios. Cada una de estas entidades representa un aspecto específico del flujo de trabajo dentro del sistema, lo que permite a los usuarios acceder a sus notificaciones, tareas asignadas y desde allí avanzar con sus procesos de compra en el sistema:

- **NotificacionEscritorio**
  Representa las notificaciones para los usuarios en su escritorio.

- **NotificacionMensajeria**
  Contiene las notificaciones relacionadas con mensajería que se envía vía

email.

- **TareaEscritorio**
  Representa las tareas asignadas a los usuarios proveedores en su escritorio.

- **TareaEscritorioComprador**
  Similar a TareaEscritorio, pero específicamente para las tareas relacionadas con los compradores.

- **AsociacionGestionEscritorioComprador**
  Esta entidad asocia las gestiones del escritorio de los compradores. Esto genera una relación entre los documentos que realiza el usuario durante el circuito de compra en la aplicación.

  Adicionalmente, existen otras cinco entidades que en la actualidad no están en uso, por lo cual no forman parte del proyecto de migración:

  - **NotificacionMobile**
  - **Pliego**
  - **AvisoEmail**
  - **temp**
  - **NotificacionEscritorioHistorica**

Se realizó un diagrama de la base de datos creado con la herramienta Database Diagram de **SQL SERVER**, detallando los atributos de cada tabla y la ausencia de relaciones explícitas entre ellas. Dicho diagrama se puede observar en la Figura 2.

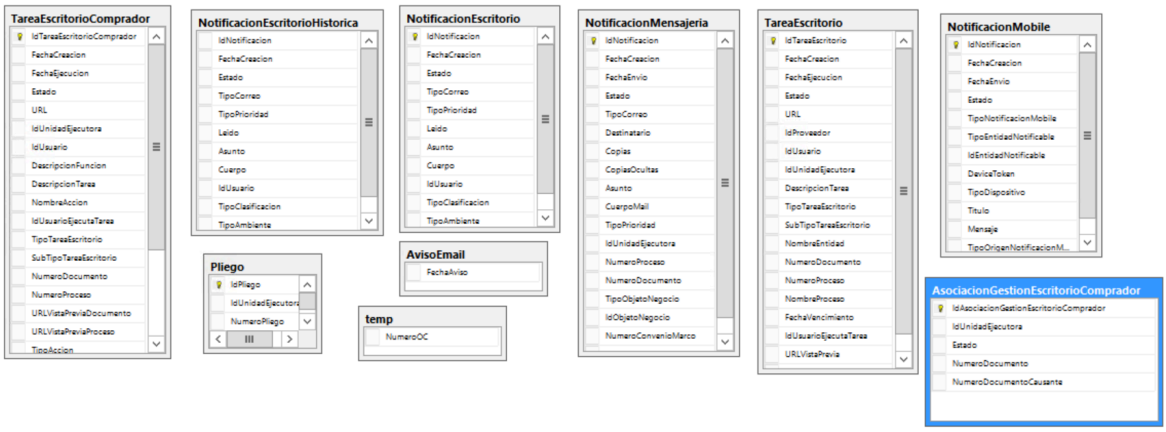

Figura 2. Diagrama de base de datos SQL.

Relaciones Implícitas en el modelo actual

En el modelo de base de datos actual, aunque existen interdependencias entre algunas entidades, éstas no están formalmente definidas a través de relaciones explícitas en el esquema. A continuación, se detallan las principales formas en que las entidades se interrelacionan de manera implícita: algunas de las entidades contienen un campo **idUsuario**, sin embargo, esta información no está directamente relacionada con un modelo de usuario dentro del mismo sistema. El **idUsuario** está desacoplado y pertenece a una base de datos separada, específicamente la base de datos de compras de la aplicación. Asimismo, algunas de las entidades contienen un campo **idProveedor**, sin embargo, esta información no está directamente relacionada con un modelo de usuario dentro del mismo sistema. El **idProveedor** está desacoplado y pertenece a una base de datos separada, la mencionada anteriormente. Por último, varias entidades están vinculadas a una Unidad Ejecutora, identificada por un campo específico (como **IdUnidadEjecutora**). Sin embargo, las unidades ejecutoras no están modeladas dentro de este sistema, ya que su gestión pertenece también a la base de datos de compras.

Adicionalmente, se evaluó el sistema de envío de correos electrónicos transaccionales ligado al subsistema de notificaciones. En el sistema actual, el envío de correos electrónicos se gestiona a través de un proceso automático que extrae los correos pendientes de la tabla **NotificacionMensajeria** en la base de datos SQL Server . Este proceso verifica periódicamente la tabla en busca de notificaciones pendientes de envío. Cuando se detectan correos por enviar, el sistema invoca un **Stored Procedure** que utiliza el agente **Database Mail** de SQL Server para realizar el envío. El Stored Procedure se encarga de procesar la información de cada correo electrónico (destinatarios, asunto, contenido, etc.) y, mediante la configuración de Database Mail, envía los correos a los destinatarios.

2.4.2 Estudio de tecnologías

En el contexto que nos encontramos, es crucial la elección de una tecnología que no solo satisfaga las necesidades actuales del sistema, sino que también ofrezca una base sólida para el futuro. El volumen de datos, la performance, la capacidad de escalar y la simplicidad en la transición son factores determinantes en esta decisión. Este análisis se centra en identificar los requisitos fundamentales que la nueva tecnología de base de datos debe cumplir, así como también las herramientas necesarias para que se pueda llevar a cabo la migración.

1. **Gestión del volumen de datos**:

   Dada la magnitud de los datos almacenados, la nueva tecnología debe estar equipada para manejar grandes volúmenes sin comprometer la eficiencia ni el rendimiento del sistema.

2. **Performance**:

   La presentación de la bandeja de notificaciones debe ser rápida y eficiente. Además, se requiere una baja latencia para la recuperación de datos.

3. **Escalabilidad**:

   Debe ser capaz de manejar el crecimiento continuo de los datos. Además, la base de datos debe escalar horizontalmente de manera efectiva.

4. **Simplicidad**:

   La tecnología elegida debe facilitar la migración desde una base de datos relacional.

Dado que el mundo de las BBDD NoSQL es muy heterogéneo, se estudió primero a nivel de familias, que es un nivel de organización relativamente conocido y utilizado para agrupar los diferentes productos. La selección de la familia NoSQL adecuada para la migración del sistema de notificaciones requiere una evaluación cuidadosa de tres tipos principales de bases de datos: **Clave/valor**, **documentos**, y **Familia de columnas**. A continuación, se presentan las características, ventajas y desventajas de cada tipo en función de los requisitos específicos del sistema actual (ver Tabla 1).

Tabla 1. Resumen de prestaciones de las diferentes familias de bases de datos NoSQL.

Familia NoSQL	Ventajas	Desventajas
Clave/Valor	- Bajo tiempo de latencia: Ideal para operaciones que requieren tiempos de respuesta extremadamente rápidos. - Simplicidad: Estructura simple que permite acceso rápido y eficiente a los datos.	- Flexibilidad limitada: No maneja bien datos estructurados o consultas complejas. - Desafíos en la persistencia: Aunque sistemas como Redis ofrecen persistencia, no es su punto fuerte.
Documentos	- Flexibilidad en el esquema: Almacena datos de forma natural y flexible, ideal para datos semi-estructurados. - Consultas poderosas: Soporta una amplia gama de operaciones de consulta. - Fácil migración: Herramientas y una comunidad robusta para migrar desde bases relacionales.	- Rendimiento: Puede no ser tan rápido como una base clave/valor para operaciones simples. - Sobrecarga de almacenamiento: Mayor uso de almacenamiento debido a la repetición de claves.
Familia de columnas	- Rendimiento en lecturas y escrituras en masa: Eficiente para grandes volúmenes de datos. - Optimización de almacenamiento: Almacenamiento en columnas que reduce espacio y mejora eficiencia en consultas.	- Complejidad en el modelo de datos: Más difícil de gestionar y consultar. - Curva de aprendizaje: Requiere mayor conocimiento para optimizar su uso y rendimiento.

Se realizó un análisis de las posibles tecnologías de bases de datos NOSQL **orientadas a documentos**, dado que es la opción que más fácilmente puede compatibilizarse con nuestro modelo del paradigma relacional. Entre las opciones de esta familia de BBDD NoSQL se optó por **MongoDB** por los siguientes motivos:

- **Gestión del volumen de datos**: MongoDB maneja grandes volúmenes de datos eficientemente a través de su modelo de documentos, permitiendo consultas rápidas y flexibles sin la necesidad de unir tablas.

- **Escalabilidad**: La capacidad de sharding automático de MongoDB permite escalar horizontalmente de manera eficiente, manejando el crecimiento

continuo de los datos sin comprometer el rendimiento.

- **Performance**: Aunque otras opciones ofrecen una latencia más baja, MongoDB ofrece un balance adecuado entre velocidad y durabilidad. Las operaciones de lectura y escritura en MongoDB son suficientemente rápidas para la mayoría de las aplicaciones de notificaciones.

- **Facilidad de migración**: MongoDB proporciona herramientas robustas para la migración desde bases de datos relacionales y una comunidad de apoyo que puede ayudar a resolver cualquier desafío durante el proceso de migración.

- **Durabilidad y persistencia**: MongoDB asegura que los datos se mantengan seguros y duraderos, lo que es crucial para aplicaciones que requieren un alto grado de confiabilidad y persistencia de datos.

Existen diversas tecnologías NoSQL de la familia de bases de datos orientadas a documentos, y cada una tiene características específicas que las hacen aptas para diferentes casos de uso. A continuacion, 3 tecnologías distintas a la elegida y sus diferencias con **MongoDB** :

- **CouchDB[5]:** Ofrece un modelo de documentos similar a MongoDB , pero su fuerte está en la replicación a nivel global y en su compatibilidad con dispositivos móviles a través de la sincronización offline/online. Sin embargo, MongoDB supera a CouchDB en términos de complejidad de consultas y en la flexibilidad de su lenguaje de consulta.

- **Cassandra[6]:** Aunque su modelo de datos puede manejar grandes volúmenes y alta disponibilidad a través de su arquitectura distribuida, se destaca más en el manejo de datos distribuidos en varios centros de datos. Además, Cassandra no se adapta tan bien como MongoDB , cuyo esquema flexible y potente API de consultas son mejores para manejar las entidades de API_NOTIFICACIONES.

---

[5] CouchDB

https://docs.couchdb.org/en/stable/

[6] Cassandra

https://cassandra.apache.org/doc/latest/

- **Amazon DynamoDB**[7]: es una opción de base de datos altamente escalable y manejada por AWS, que ofrece durabilidad y velocidad. A pesar de sus ventajas en sistemas donde el tiempo de respuesta es crítico, MongoDB proporciona una mayor flexibilidad en cuanto a la modelización de datos, lo que es un requisito clave en el diseño de la base de datos de API_NOTIFICACIONES para PBAC. DynamoDB también puede volverse costoso a medida que crecen los volúmenes de datos y la carga de consultas.

Estudio de tecnología de migración

Antes de embarcarse en el proceso de migración, es necesario evaluar las herramientas disponibles que facilitan este paso de un modelo relacional a uno basado en documentos. El éxito de la migración depende de cómo estas herramientas abordan aspectos clave como la preservación de la integridad de los datos, la eficiencia en la transformación de esquemas, su facilidad de uso y la compatibilidad.

Este estudio incluye la evaluación de tecnologías como **MongoDB Relational Migrator**, **Studio 3T**[8] y **Talend**[9] , siendo estas soluciones destacadas en el campo de la migración de datos. Cada una ofrece diferentes enfoques y funcionalidades, lo que requiere un análisis comparativo para seleccionar la herramienta más adecuada.

1. MongoDB Relational Migrator

Enfoque: Herramienta oficial de MongoDB diseñada específicamente para facilitar la migración de bases de datos relacionales (SQL) a MongoDB .

---

[7] Amazon DynamoDB

https://docs.aws.amazon.com/dynamodb/?icmpid=docs_homepage_featuredsvcs

[8] Studio 3T

https://studio3t.com/free/

[9] Talend

https://www.talend.com/resources/get-started-talend-open-studio-data-integration/

- **Automatización**: Ofrece una migración automatizada desde bases de datos relacionales a MongoDB con mínima intervención manual.
- **Mapeo inteligente**: Realiza un análisis automático de la estructura de la base de datos relacional y propone un esquema de documentos en MongoDB .
- **Integración nativa**: Como herramienta oficial de MongoDB , está diseñada para integrarse de forma óptima con la base de datos.
- **Simplicidad**: Está enfocada en simplificar la migración sin requerir grandes configuraciones o transformaciones complejas.

Ideal para: Usuarios que buscan una herramienta específica para MongoDB y requieren una migración relativamente sencilla y directa.

2. Studio 3T

Enfoque: Herramienta integral para trabajar con MongoDB , que incluye funcionalidades avanzadas de migración y desarrollo de consultas.

- **Soporte para múltiples bases de datos**: Permite conectar no solo a MongoDB , sino también a bases de datos SQL como MySQL, PostgreSQL, SQL Server , entre otras.
- **Mapeo flexible**: Ofrece opciones de mapeo personalizadas para adaptar mejor las estructuras de las tablas relacionales a colecciones de MongoDB .
- **Visualización**: Tiene una interfaz gráfica amigable y visual que facilita el diseño y la manipulación de datos, con herramientas para ver la estructura de documentos, mapear datos, y realizar transformaciones.
- **Funciones avanzadas**: No solo se utiliza para migración, sino también para desarrollo y administración diaria de bases de datos MongoDB .

Ideal para: Usuarios técnicos que desean control sobre la migración y necesitan una herramienta avanzada para trabajar tanto con datos migrados como con consultas diarias.

3. Talend

Enfoque: Plataforma de integración de datos que permite ETL (Extract, Transform, Load) para múltiples fuentes de datos, incluidas bases de datos SQL y MongoDB .

- **Multiplataforma**: Soporta una amplia variedad de fuentes de datos, desde SQL hasta NoSQL y sistemas en la nube.

- **Transformaciones complejas**: Talend es potente cuando se necesita realizar transformaciones avanzadas durante la migración, lo que permite aplicar lógica empresarial y limpieza de datos en el proceso.
- **ETL completo**: Talend es una herramienta de integración completa que se puede utilizar no solo para migración, sino también para integraciones continuas entre diferentes sistemas y procesos de carga de datos.
- **Interfaz de diseño visual**: Proporciona una interfaz gráfica donde se pueden diseñar flujos de trabajo de migración y transformación de datos.

Ideal para: Proyectos de migración más complejos o cuando se necesita una herramienta que combine la migración con otras tareas de integración de datos en un ecosistema más amplio.

La elección final recayó en **MongoDB Relational Migrator**. Si bien las otras opciones ofrecen un amplio rango de funcionalidades, MongoDB Relational Migrator se destacó por ser la solución oficial de MongoDB, lo que asegura una integración óptima.

Su facilidad de uso también fue un factor decisivo. Su enfoque automatizado y su capacidad para mapear esquemas relacionales a documentos de MongoDB con mínima intervención manual simplifican el proceso de migración, reduciendo el riesgo de errores y acelerando la adopción de la nueva base de datos. Esto lo convierte en una herramienta ideal para proyectos donde la prioridad es realizar la migración de forma eficiente y con un alto grado de compatibilidad con MongoDB.

Gracias a estas características, MongoDB Relational Migrator ofrece una solución confiable y eficiente para la transición hacia un sistema NoSQL, garantizando un rendimiento adecuado y una escalabilidad futura sin complicaciones técnicas adicionales.

2.4.3 Desarrollo técnico

En esta sección se abordarán los aspectos técnicos involucrados con respecto a la migración del sistema de notificaciones. Se detallarán las tecnologías empleadas y las modificaciones realizadas en el código fuente para garantizar una correcta integración con la infraestructura actual.

Uso de MongoDB Relational Migrator

El uso de esta herramienta fue fundamental para facilitar y agilizar el proceso de migración, permitiendo una transición más fluida desde el entorno relacional a un modelo NoSQL basado en documentos. La herramienta permitió establecer una conexión directa con la base de datos existente en SQL Server, y a partir de esta conexión, analizó la estructura relacional, generando automáticamente el modelo correspondiente para MongoDB. Este modelo representaba la estructura de documentos en MongoDB, lo cual simplificó considerablemente el proceso de transformación de los datos, garantizando que la migración preservara la integridad y organización de la información. Además de la generación del modelo, MongoDB Relational Migrator ofreció un servicio de migración de datos, que permitió transferir de manera eficiente los registros desde SQL Server a MongoDB . El proceso tomó aproximadamente 10 horas, y durante este tiempo, los datos se transformaron y almacenaron en documentos en MongoDB , respetando la estructura definida previamente en la herramienta.

Una vez finalizada la migración, se observaron mejoras significativas, como la reducción drástica en el volumen de almacenamiento requerido. Mientras que la base de datos en SQL Server supera los 100 GB, presentando desafíos en términos de rendimiento y escalabilidad, la base de datos migrada a MongoDB solo pesa 20 GB, lo que representó un cambio positivo en términos de eficiencia y gestión de recursos.

Comparación de performance entre querys: SQL Server y MongoDB

Se llevó a cabo una comparación del rendimiento de consultas en la tabla **TareaEscritorio** utilizando las herramientas proporcionadas por cada sistema de gestión de bases de datos para medir el tiempo de ejecución en milisegundos. Las consultas fueron ejecutadas en dos servidores distintos, pero con recursos idénticos, garantizando así una capacidad de procesamiento comparable

En primer lugar, se realizó una consulta SELECT en SQL Server para recuperar todas las tareas asociadas a un proveedor particular. La ejecución de esta consulta produjo un tiempo promedio de **180 a 200 ms** para un total de **2795 registros**.

```
SET STATISTICS TIME ON;

Select * from dbo.TareaEscritorio where IdProveedor  =13921

SET STATISTICS  TIME OFF;
```

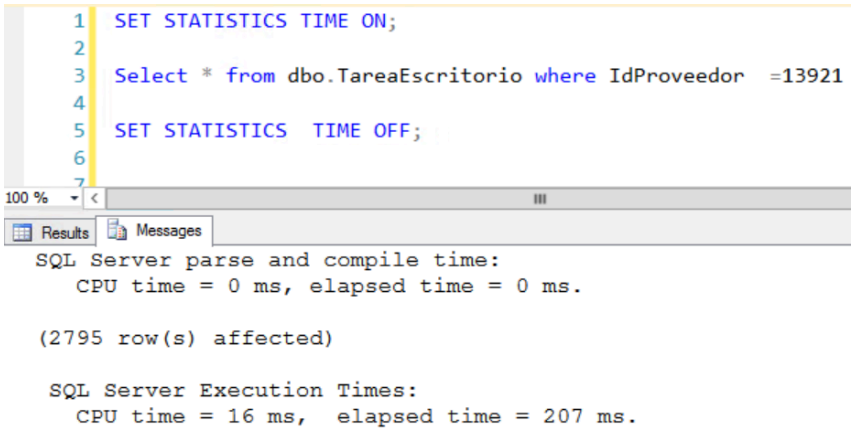

Figura 3. Query simple para comparar performance en SQL Server.

Posteriormente, se adaptó la misma consulta a MongoDB, ejecutándola en un fragmento de código en **Mongosh**, el shell de MongoDB. Para esta consulta, se obtuvo un tiempo promedio de **7 a 10 ms** para la misma cantidad de registros y el mismo ID de proveedor.

```
const start = Date.now();

db.TareaEscritorio.find({idProveedor:13921});

const end = Date.now();

const timeElapsed = end - start;

const cant = db.TareaEscritorio.countDocuments({idProveedor:13921});

console.log("Cantidad de registros", cant);

console.log("Tiempo de ejecución (ms):", timeElapsed);
```

```
> const start = Date.now();
  db.tareaEscritorio.find({idProveedor:13921});
  const end = Date.now();
  const timeElapsed = end - start;
  const cant = db.tareaEscritorio.countDocuments({idProveedor:13921});
  console.log("Cantidad de registros", cant);
  console.log("Tiempo de ejecución (ms):", timeElapsed);
< Cantidad de registros
< 2795
< Tiempo de ejecución (ms):
< 7
```

Figura 4. Query simple para comparar performance en MongoDB**.**

Los resultados obtenidos evidencian una mejora significativa en el rendimiento de las consultas realizadas en MongoDB en comparación con SQL Server. Esta mejora en el rendimiento tiene implicaciones directas y muy positivas para el sistema de notificaciones y tareas de PBAC. Al reducir drásticamente el tiempo de respuesta para las consultas, se puede lograr una mayor agilidad en la gestión de tareas y en el envío de notificaciones. Esto mejoraría así la experiencia general del usuario y la eficacia del flujo de trabajo.

En un entorno donde la rapidez y la eficiencia son cruciales, la implementación de MongoDB como gestor de base de datos representa un avance significativo hacia la modernización y optimización del sistema de PBAC.

Modificaciones para la integración de MongoDB en API_NOTIFICACIONES

En el proyecto de **API_NOTIFICACIONES** se realizaron varias modificaciones clave para permitir la transición de una base de datos SQL Server a MongoDB . Estos cambios incluyeron ajustes en los modelos de datos, en la capa de acceso a datos (**DAO**) y un cambio estructural significativo en el manejo de identificadores. En primer lugar, se añadieron anotaciones **[BsonElement()]** en los modelos de datos para mapear los campos de las clases con los correspondientes en la base de datos MongoDB . Estas anotaciones permiten alinear correctamente los documentos en MongoDB con las propiedades de las clases en el proyecto, facilitando la

persistencia y recuperación de datos. Incluso si los nombres de las propiedades en C# difieren de los nombres de los campos en MongoDB , el mapeo se realiza adecuadamente gracias a estas anotaciones.

```
[BsonElement("fechaCreacion")]
5 referencias
public DateTime FechaCreacion { get; set; }
```

Figura 5. Anotaciones en el modelo de datos.

Además, se desarrolló una nueva capa DAO (**Data Access Object**) para gestionar las conexiones y operaciones con MongoDB . Aunque la interfaz utilizada es la misma que la empleada anteriormente con SQL Server , los métodos se actualizaron para ser compatibles con MongoDB, incluyendo cambios en la manera de realizar consultas, actualizaciones y gestionar los datos. Para este propósito, se utilizó el Driver de MongoDB para C#, que facilita la manipulación de objetos de filtro y operaciones específicas con la nueva base de datos.

```
private readonly IMongoClient client;

private readonly IMongoDatabase mongoDatabase;

private readonly IMongoCollection<TareaEscritorio> collection;

1 referencia
public TareaEscritorioDAO(string connectionString, string databaseName)
{
    client = new MongoClient(connectionString);
    mongoDatabase = client.GetDatabase(databaseName);
    collection = mongoDatabase.GetCollection<TareaEscritorio>("tareaEscritorio");
}
```

Figura 6. Constructor utilizando el driver de MongoDB para C#.

```
public TareaEscritorio Save(TareaEscritorio tareaEscritorio)
{
    collection.InsertOne(tareaEscritorio);

    return tareaEscritorio;
}
```

Figura 7. Método para guardar una tarea en la base de datos nueva.

Un cambio estructural importante fue el manejo de identificadores. En lugar de los IDs autoincrementales de tipo int utilizados en SQL Server, se adoptó el **ObjectId** de MongoDB, lo que permite aprovechar la capacidad de MongoDB para generar identificadores únicos de forma automática y distribuida. Aunque el tipo de identificador cambió, se mantuvo la misma nomenclatura en la aplicación. Los identificadores se manejan como cadenas de texto (**string**) cuando se envían entidades a la aplicación y se recuperan de la misma forma. Al interactuar con la base de datos, estos identificadores se castean al tipo **ObjectId**.

```
[BsonId]
[BsonElement("_id")]
6 referencias
public new ObjectId Id { get; set; }
```

Figura 8. Identificador ObjectId de MongoDB.

Por otro lado, en el proyecto original **API_NOTIFICACIONES** con SQL Server , el envío de correos electrónicos se gestionaba mediante un proceso automático que obtenía información de la base de datos y ejecutaba un Stored Procedure (**sp_send_dbmail**) para enviar correos a los usuarios utilizando Database Mail. Con la migración a MongoDB , este enfoque ya no fue viable, por lo que se modificó el código del NotificacionMensajeriaDAO para enviar correos directamente desde el

código C# en lugar de depender de un Stored Procedure. la nueva implementación proporciona mayor flexibilidad, ya que permite gestionar y personalizar los correos directamente desde la lógica de la aplicación, eliminando la dependencia de procesos específicos de SQL.

Estas modificaciones han permitido que el proyecto API_NOTIFICACIONES se adapte a MongoDB , aprovechando sus capacidades de escalabilidad y flexibilidad, mientras se mantiene la misma estructura de datos y nomenclatura que antes.

2.5 Trabajo Futuro

Se propone una serie de lineamientos que permitirán mejorar y optimizar el funcionamiento del sistema en un futuro, garantizando que se mantenga actualizado con las mejores prácticas tecnológicas. Los principales puntos a desarrollar incluyen mejoras en los servicios de comunicación entre proyectos, la migración de .NET Framework[10] a .NET Core[11], y la implementación de una arquitectura de dockerización, pudiendo surgir otras propuestas o puntos de interés más adelante.

Plan de migración

El desarrollo del proyecto de migración, hasta el momento, ha sido probado únicamente en entornos de testeo y no ha podido implementarse en el entorno de producción debido a varias limitaciones organizacionales y técnicas. Estas limitaciones incluyen una falta general de recursos, lo cual limita la posibilidad de ampliar la infraestructura existente para integrar una base de datos NoSQL de forma productiva. Además, dado que hasta el momento el equipo solo ha gestionado bases de datos relacionales, la falta de capacitación en el mantenimiento y administración en NoSQL representa un desafío significativo.

No obstante, este proyecto de migración representa una deuda técnica considerable para el sistema PBAC. La necesidad de actualizar y optimizar el

---

[10] .NET Framework

https://learn.microsoft.com/es-es/dotnet/framework/

[11] .NET Core

https://learn.microsoft.com/en-us/aspnet/core/?view=aspnetcore-8.0&WT.mc_id=dotnet-35129-website

sistema había sido una preocupación mencionada con anterioridad, aunque no se había logrado concretar hasta ahora. Presentar este proyecto con su análisis exhaustivo marca un avance fundamental para la modernización del sistema, estableciendo los fundamentos necesarios para que, en el futuro cercano, se puedan resolver estas limitaciones y se logre implementar la migración con todas sus ventajas ya discutidas.

Mejoras en los servicios

Actualmente, los diferentes módulos y servicios de la aplicación PBAC se comunican mediante APIs y otros mecanismos de integración. No obstante, a medida que aumente la carga de datos y el tráfico, será crucial optimizar estas comunicaciones para mejorar la eficiencia y la escalabilidad del sistema. Se propone implementar las siguientes mejoras:

- **Uso de mensajería asíncrona**: Implementar servicios de mensajería asincrónica mediante tecnologías como **RabbitMQ**[12] o **Kafka**[13] permitirá manejar una mayor cantidad de solicitudes sin afectar el rendimiento del sistema, lo cual es especialmente útil en sistemas distribuidos.

- **Mejora de la resiliencia**: Incorporar patrones de diseño como **circuit breakers** para aumentar la tolerancia a fallos en la comunicación entre servicios.

- **Monitoreo y logging avanzado**: Implementar sistemas de monitoreo y trazabilidad para mejorar la capacidad de detectar y solucionar problemas en tiempo real.

[12] RabbitMQ

https://www.rabbitmq.com/docs

[13] Kafka

https://kafka.apache.org/documentation/

Migración de .NET Framework a .NET Core

Dada la creciente adopción de .NET Core (ahora conocido como **.NET**), se recomienda planificar la migración de los servicios desarrollados en **.NET Framework** a **.NET Core** . Esta migración proporcionará múltiples beneficios:

- **Multiplataforma**: .NET Core es compatible con varios sistemas operativos (Windows, Linux y macOS), lo que ofrece mayor flexibilidad en la infraestructura de servidores.

- **Mejor rendimiento**: .NET Core está optimizado para ofrecer un mejor rendimiento, lo que reducirá los tiempos de respuesta y mejorará el manejo de recursos.

- **Actualizaciones y soporte**: .NET Core recibe actualizaciones y mejoras constantes, mientras que .NET Framework ya no está recibiendo tantas mejoras, lo que a largo plazo podría afectar la seguridad y la compatibilidad.

- **Facilidad de integración con tecnologías modernas**: .NET Core facilita la integración con contenedores, arquitecturas de microservicios, y plataformas en la nube.

Arquitectura de *Dockerización*

La implementación de una arquitectura basada en contenedores mediante **Docker**[14] permitirá una mayor flexibilidad y portabilidad del sistema. Esta arquitectura tiene varios beneficios:

- **Aislamiento de servicios**: Cada microservicio puede ser empaquetado y desplegado de manera independiente, lo que facilita la escalabilidad y el mantenimiento. Si un servicio falla, no afectará a los demás.

- **Facilidad de despliegue**: Con contenedores, se puede automatizar el despliegue de la aplicación en diferentes entornos, asegurando que las dependencias y configuraciones sean consistentes.

---

[14] Docker

https://docs.docker.com/?_gl=1*qzrkmw*_gcl_au*MTY0NDQ3MDMzLjE3MjgwMDU5Mzk.*_ga*MTY4OTkzNjkyMS4xNzI4MDA1OTM5*_ga_XJWPQMJYHQ*MTcyODAwNTkzOS4xLjEuMTcyODAwNTkzOS42MC4wLjA.

- **Escalabilidad horizontal**: La infraestructura basada en Docker puede ser fácilmente escalada utilizando herramientas de orquestación como **Kubernetes**, lo que permitirá manejar picos de demanda de manera más eficiente.

- **Desarrollo y pruebas eficientes**: Los contenedores permiten a los desarrolladores y testers replicar los entornos de producción de forma local, facilitando el ciclo de vida del desarrollo y reduciendo los errores de configuración en diferentes entornos.

Referencias bibliográficas

(Sadalage 2013) Sadalage, Pramod J., and Martin Fowler. NoSQL distilled: a brief guide to the emerging world of polyglot persistence.Pearson Education, 2013.

(Shaikh 2017) Shaikh, N. F., Jadhav, A., Raina, C., Nagoshe, G., & Kale, S. (2017). Data migration from SQL To Mongodb. *Int J Eng Res Dev*, *13*(11), 32-36.

(Aggoune 2020) AGGOUNE, A., & NAMOUNE, M. S. (2020, February). A method for transforming object-relational to document-oriented databases. In *2020 2nd International Conference on Mathematics and Information Technology (ICMIT)* (pp. 154-158). IEEE.

(Erraji 2023) Erraji, A., Maizate, A., & Ouzzif, M. (2023). An integral approach for complete migration from a relational database to MongoDB. *Journal of the Nigerian Society of Physical Sciences*, 1089-1089.

(El Alami 2024) El Alami, A., Khourdifi, Y., El Mouden, Z. A., Lahmer, M., & Hasnaoui, M. L. (2024). Migrating Relational Databases to NoSQL-Oriented Documents Using Object-Oriented Concepts. *International Journal of Intelligent Engineering & Systems*, *17*(4).