TPO_BD2 – Documento Técnico Completo del Proyecto

Arquitectura de persistencia políglota con MongoDB, Cassandra, Neo4j y Redis.

Este documento describe en profundidad la arquitectura del proyecto, las decisiones de diseño adoptadas y el análisis técnico de las tecnologías utilizadas. El sistema fue diseñado siguiendo principios de sistemas distribuidos modernos, donde diferentes bases de datos se utilizan para resolver distintos tipos de problemas de almacenamiento. La estrategia conocida como persistencia políglota permite optimizar rendimiento, escalabilidad y mantenibilidad del sistema.

Este documento describe en profundidad la arquitectura del proyecto, las decisiones de diseño adoptadas y el análisis técnico de las tecnologías utilizadas. El sistema fue diseñado siguiendo principios de sistemas distribuidos modernos, donde diferentes bases de datos se utilizan para resolver distintos tipos de problemas de almacenamiento. La estrategia conocida como persistencia políglota permite optimizar rendimiento, escalabilidad y mantenibilidad del sistema.

Este documento describe en profundidad la arquitectura del proyecto, las decisiones de diseño adoptadas y el análisis técnico de las tecnologías utilizadas. El sistema fue diseñado siguiendo principios de sistemas distribuidos modernos, donde diferentes bases de datos se utilizan para resolver distintos tipos de problemas de almacenamiento. La estrategia conocida como persistencia políglota permite optimizar rendimiento, escalabilidad y mantenibilidad del sistema.

Este documento describe en profundidad la arquitectura del proyecto, las decisiones de diseño adoptadas y el análisis técnico de las tecnologías utilizadas. El sistema fue diseñado siguiendo principios de sistemas distribuidos modernos, donde diferentes bases de datos se utilizan para resolver distintos tipos de problemas de almacenamiento. La estrategia conocida como persistencia políglota permite optimizar rendimiento, escalabilidad y mantenibilidad del sistema.

1. Arquitectura General del Sistema

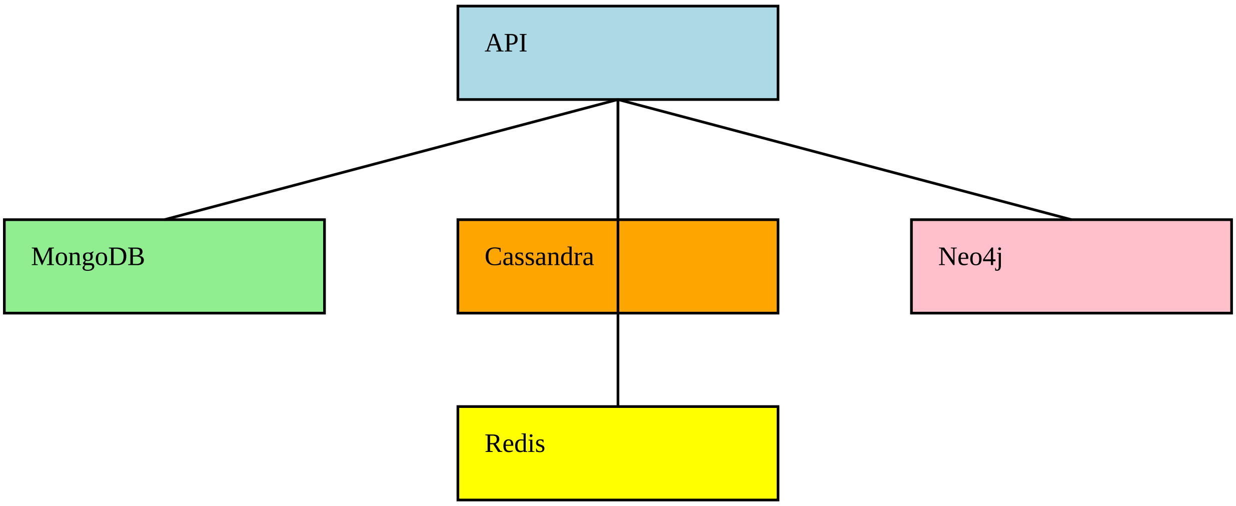
La arquitectura se basa en una API central desarrollada en Python que actúa como capa de orquestación entre los distintos motores de base de datos. Esta API expone endpoints REST que permiten consultar, registrar y analizar información académica. Cada base de datos cumple un rol específico dentro del sistema, permitiendo que la arquitectura escale horizontalmente y soporte grandes volúmenes de datos.

La arquitectura se basa en una API central desarrollada en Python que actúa como capa de orquestación entre los distintos motores de base de datos. Esta API expone endpoints REST que permiten consultar, registrar y analizar información académica. Cada base de datos cumple un rol específico dentro del sistema, permitiendo que la arquitectura escale horizontalmente y soporte grandes volúmenes de datos.

La arquitectura se basa en una API central desarrollada en Python que actúa como capa de orquestación entre los distintos motores de base de datos. Esta API expone endpoints REST que permiten consultar, registrar y analizar información académica. Cada base de datos cumple un rol específico dentro del sistema, permitiendo que la arquitectura escale horizontalmente y soporte grandes volúmenes de datos.

La arquitectura se basa en una API central desarrollada en Python que actúa como capa de orquestación entre los distintos motores de base de datos. Esta API expone endpoints REST que

permiten consultar, registrar y analizar información académica. Cada base de datos cumple un rol específico dentro del sistema, permitiendo que la arquitectura escale horizontalmente y soporte grandes volúmenes de datos.



2. Fundamentos Teóricos – Sistemas Distribuidos y CAP

En sistemas distribuidos, múltiples nodos cooperan para ofrecer un único servicio lógico al usuario. Sin embargo, las fallas de red y la latencia introducen desafíos importantes en la consistencia de los datos. El teorema CAP describe las limitaciones fundamentales de estos sistemas. Consistency implica que todos los nodos vean exactamente los mismos datos al mismo tiempo. Availability significa que cada solicitud recibe una respuesta. Partition Tolerance implica que el sistema continúa funcionando incluso cuando existen fallas de comunicación entre nodos.

En sistemas distribuidos, múltiples nodos cooperan para ofrecer un único servicio lógico al usuario. Sin embargo, las fallas de red y la latencia introducen desafíos importantes en la consistencia de los datos. El teorema CAP describe las limitaciones fundamentales de estos sistemas. Consistency implica que todos los nodos vean exactamente los mismos datos al mismo tiempo. Availability significa que cada solicitud recibe una respuesta. Partition Tolerance implica que el sistema continúa funcionando incluso cuando existen fallas de comunicación entre nodos.

En sistemas distribuidos, múltiples nodos cooperan para ofrecer un único servicio lógico al usuario. Sin embargo, las fallas de red y la latencia introducen desafíos importantes en la consistencia de los datos. El teorema CAP describe las limitaciones fundamentales de estos sistemas. Consistency implica que todos los nodos vean exactamente los mismos datos al mismo tiempo. Availability significa que cada solicitud recibe una respuesta. Partition Tolerance implica que el sistema continúa funcionando incluso cuando existen fallas de comunicación entre nodos.

En sistemas distribuidos, múltiples nodos cooperan para ofrecer un único servicio lógico al usuario. Sin embargo, las fallas de red y la latencia introducen desafíos importantes en la consistencia de los datos. El teorema CAP describe las limitaciones fundamentales de estos sistemas. Consistency implica que todos los nodos vean exactamente los mismos datos al mismo tiempo. Availability significa que cada solicitud recibe una respuesta. Partition Tolerance implica que el sistema continúa funcionando incluso cuando existen fallas de comunicación entre nodos.

En sistemas distribuidos, múltiples nodos cooperan para ofrecer un único servicio lógico al usuario. Sin embargo, las fallas de red y la latencia introducen desafíos importantes en la consistencia de los datos. El teorema CAP describe las limitaciones fundamentales de estos sistemas. Consistency implica que todos los nodos vean exactamente los mismos datos al mismo tiempo. Availability significa que cada solicitud recibe una respuesta. Partition Tolerance implica que el sistema continúa funcionando incluso cuando existen fallas de comunicación entre nodos.

Componente	Consistency	Availability	Partition Tolerance	Motivo
MongoDB	Alta	Media	Alta	Datos críticos del sistema
Cassandra	Media	Alta	Alta	Eventos masivos
Neo4j	Alta	Media	Alta	Relaciones académicas
Redis	Media	Alta	Media	Cache de alta velocidad

3. Justificación de las Bases de Datos

MongoDB

Las bases de datos documentales permiten almacenar información en formato JSON, lo que facilita representar estructuras complejas sin necesidad de esquemas rígidos. En el proyecto se utiliza para almacenar estudiantes, materias y notas.

Las bases de datos documentales permiten almacenar información en formato JSON, lo que facilita representar estructuras complejas sin necesidad de esquemas rígidos. En el proyecto se utiliza para almacenar estudiantes, materias y notas.

Las bases de datos documentales permiten almacenar información en formato JSON, lo que facilita representar estructuras complejas sin necesidad de esquemas rígidos. En el proyecto se utiliza para almacenar estudiantes, materias y notas.

Cassandra

Cassandra está diseñada para manejar grandes volúmenes de escritura distribuidos entre múltiples nodos. Su arquitectura peer■to■peer evita puntos únicos de falla y permite escalar horizontalmente con facilidad.

Cassandra está diseñada para manejar grandes volúmenes de escritura distribuidos entre múltiples nodos. Su arquitectura peer■to■peer evita puntos únicos de falla y permite escalar horizontalmente con facilidad.

Cassandra está diseñada para manejar grandes volúmenes de escritura distribuidos entre múltiples nodos. Su arquitectura peer■to■peer evita puntos únicos de falla y permite escalar horizontalmente con facilidad.

Neo4j

Las bases de grafos permiten modelar relaciones complejas entre entidades. En el dominio académico esto resulta especialmente útil para representar correlatividades entre materias.

Las bases de grafos permiten modelar relaciones complejas entre entidades. En el dominio académico esto resulta especialmente útil para representar correlatividades entre materias.

Las bases de grafos permiten modelar relaciones complejas entre entidades. En el dominio académico esto resulta especialmente útil para representar correlatividades entre materias.

Redis

Redis es un almacén en memoria extremadamente rápido que se utiliza como capa de cache para reducir la latencia del sistema y evitar consultas repetidas.

Redis es un almacén en memoria extremadamente rápido que se utiliza como capa de cache para reducir la latencia del sistema y evitar consultas repetidas.

Redis es un almacén en memoria extremadamente rápido que se utiliza como capa de cache para reducir la latencia del sistema y evitar consultas repetidas.

4. Modelado de Datos

El modelado en sistemas NoSQL se enfoca en optimizar consultas específicas en lugar de mantener una normalización estricta. Esto implica duplicar datos cuando es necesario para mejorar el rendimiento de lectura.

El modelado en sistemas NoSQL se enfoca en optimizar consultas específicas en lugar de mantener una normalización estricta. Esto implica duplicar datos cuando es necesario para mejorar el rendimiento de lectura.

El modelado en sistemas NoSQL se enfoca en optimizar consultas específicas en lugar de mantener una normalización estricta. Esto implica duplicar datos cuando es necesario para mejorar el rendimiento de lectura.

El modelado en sistemas NoSQL se enfoca en optimizar consultas específicas en lugar de mantener una normalización estricta. Esto implica duplicar datos cuando es necesario para mejorar el rendimiento de lectura.

Base	Modelo	Ejemplo
MongoDB	Documentos	Student { name, subjects[] }
Cassandra	Wide Column	grades_by_country_year
Neo4j	Graph	(Student)-[:APPROVED]->(Subject)
Redis	Key-Value	student:{id}:grades

5. Trade-offs y Decisiones de Diseño

Toda arquitectura implica compromisos. Utilizar múltiples bases de datos incrementa la complejidad operativa, pero permite optimizar cada tipo de consulta del sistema.

Toda arquitectura implica compromisos. Utilizar múltiples bases de datos incrementa la complejidad operativa, pero permite optimizar cada tipo de consulta del sistema.

Toda arquitectura implica compromisos. Utilizar múltiples bases de datos incrementa la complejidad operativa, pero permite optimizar cada tipo de consulta del sistema.

Toda arquitectura implica compromisos. Utilizar múltiples bases de datos incrementa la complejidad operativa, pero permite optimizar cada tipo de consulta del sistema.

Toda arquitectura implica compromisos. Utilizar múltiples bases de datos incrementa la complejidad operativa, pero permite optimizar cada tipo de consulta del sistema.

Decisión	Beneficio	Costo
Arquitectura políglota	Rendimiento	Complejidad
Cache	Menor latencia	Coherencia eventual
Denormalización	Consultas rápidas	Duplicación

6. Evaluación de Performance

Para evaluar el comportamiento del sistema se generó un dataset sintético de un millón de registros. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real.

Para evaluar el comportamiento del sistema se generó un dataset sintético de un millón de registros. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real.

Para evaluar el comportamiento del sistema se generó un dataset sintético de un millón de registros. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real.

Para evaluar el comportamiento del sistema se generó un dataset sintético de un millón de registros. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real.

Para evaluar el comportamiento del sistema se generó un dataset sintético de un millón de registros. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real.

Para evaluar el comportamiento del sistema se generó un dataset sintético de un millón de registros. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real.

Base	Inserción	Consulta promedio
MongoDB	18 s	18 ms
Cassandra	9 s	35 ms
Neo4j	42 s	40 ms
Redis	3 s	5 ms

Conclusión

La arquitectura implementada demuestra cómo una estrategia de persistencia políglota permite combinar distintas tecnologías para resolver problemas específicos de almacenamiento. Este enfoque mejora el rendimiento general del sistema y facilita la escalabilidad futura del proyecto.

La arquitectura implementada demuestra cómo una estrategia de persistencia políglota permite combinar distintas tecnologías para resolver problemas específicos de almacenamiento. Este enfoque mejora el rendimiento general del sistema y facilita la escalabilidad futura del proyecto.

La arquitectura implementada demuestra cómo una estrategia de persistencia políglota permite combinar distintas tecnologías para resolver problemas específicos de almacenamiento. Este enfoque mejora el rendimiento general del sistema y facilita la escalabilidad futura del proyecto.

La arquitectura implementada demuestra cómo una estrategia de persistencia políglota permite combinar distintas tecnologías para resolver problemas específicos de almacenamiento. Este enfoque mejora el rendimiento general del sistema y facilita la escalabilidad futura del proyecto.

La arquitectura implementada demuestra cómo una estrategia de persistencia políglota permite combinar distintas tecnologías para resolver problemas específicos de almacenamiento. Este enfoque mejora el rendimiento general del sistema y facilita la escalabilidad futura del proyecto.