EduGrade Global - Sistema Nacional de Calificaciones Multimodelo

Documentación Técnica Completa

Versión: 1.0
Autor: Grupo 9

Fecha: [Fecha de Entrega]

Índice

---

1. Análisis del Dominio

1.1. Descripción General y Contexto

El sistema EduGrade Global nace de la necesidad del Ministerio de Educación de Sudáfrica de unificar la gestión de calificaciones académicas en un contexto de alta movilidad estudiantil y reconocimiento de diversos sistemas educativos internacionales (UK, US, Alemania, Argentina). El núcleo del problema no es solo almacenar notas, sino gestionar la heterogeneidad, la equivalencia y la trazabilidad de la información académica a lo largo del tiempo.

A diferencia de un sistema académico tradicional que opera sobre un único sistema de calificación, EduGrade debe funcionar como un sistema de sistemas, capaz de interpretar, convertir y relacionar calificaciones de origen dispar sin perder su semántica original. Esto implica un cambio de paradigma: la base de datos ya no es un mero depósito, sino una parte activa en la resolución de la complejidad del dominio.

1.2. Identificación de Entidades y Relaciones

El análisis del dominio revela entidades con diferentes niveles de complejidad y tipos de relaciones:

Entidades Centrales (Núcleo del sistema):

- Estudiante: Corazón del sistema. Posee datos demográficos básicos pero, más importante, una o múltiples trayectorias.
- Institución: Entidad que dicta materias y otorga calificaciones. Tiene atributos como nombre, país, nivel educativo (secundario, terciario) y sistema de calificación que utiliza por defecto.
- Materia: Unidad de conocimiento. Su complejidad radica en su definición contextual. Una materia puede tener diferentes nombres, contenidos y sistemas de evaluación dependiendo del país y la institución. Por ejemplo, "Matemáticas" en el sistema UK (GCSE) no es exactamente la misma entidad que "Matemática" en el sistema argentino, aunque tengan equivalencias.

Entidades de Hecho (Registro de la actividad académica):

- Evaluación: Instancia concreta de medición (parcial, final, coursework, recuperatorio). Este es el punto donde se aplica una escala de calificación.
- Calificación: El resultado de una evaluación para un estudiante. Debe ser inmutable. Contiene el valor original, la escala de origen y metadatos contextuales (fecha, evaluación, etc.).
- Trayectoria: Representa el paso de un estudiante por una institución durante un período. Actúa como un contenedor lógico para las calificaciones en un contexto específico.

Entidades de Soporte (Reglas y contexto):

- SistemaCalificacion: Catálogo de escalas (ej. "UK-GCSE", "US-GPA").
- ReglaConversion: Entidad temporal. Define cómo se transforma un valor de una escala a otra en un momento dado.
- EquivalenciaMateria: Relación N:M entre materias de diferentes sistemas, con un grado de equivalencia (total, parcial).

Relaciones Clave:

- Un `Estudiante` tiene una o muchas `Trayectorias` (1:N).
- Una `Trayectoria` pertenece a una `Institución` y a un `Estudiante` (N:1, N:1).
- Una `Calificación` pertenece a una `Trayectoria` y a una `Evaluación` (N:1, N:1). Una `Evaluación` corresponde a una `Materia` (N:1).

- `Materia` puede ser equivalente a otra/s `Materia` (N:M), formando una red de equivalencias.
- Una `ReglaConversion` asocia dos `SistemaCalificacion` (N:M versionado).

1.3. Análisis de Volúmenes de Datos

Para un sistema nacional, las proyecciones de volumen son críticas:

- Estudiantes: ~15 millones (estimado de población en edad escolar).
- Instituciones: ~30,000.
- Materias: ~500,000 (considerando todas las variantes por país y nivel).
- Calificaciones: Es el registro de mayor volumen. Si cada estudiante tiene un promedio de 30 calificaciones a lo largo de su vida académica, hablamos de 450 millones de registros. Un sistema debe estar diseñado para escalar a este orden de magnitud. El requerimiento de 1M es una prueba de concepto de bajo volumen.

1.4. Casos de Uso Complejos

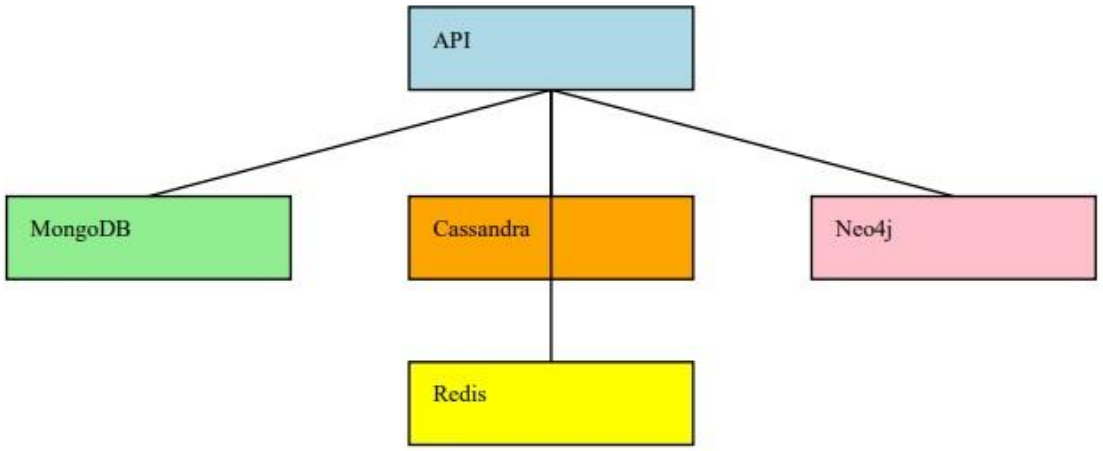
El diseño debe soportar escenarios como:

- Trayectoria Migrante: Un estudiante comienza la secundaria en Argentina (escala 1-10), se muda a Alemania y sus materias aprobadas son convertidas y registradas en el sistema alemán (escala 1-6), pero el sistema debe conservar la nota original argentina. Luego, aplica a una universidad en UK y necesita un certificado con sus calificaciones convertidas a GPA o a letras UK.
- Cambio Normativo: En 2025, el Ministerio cambia la tabla de equivalencias entre el sistema alemán y el GPA de US. Las conversiones realizadas antes de 2025 deben seguir siendo válidas y estar auditadas con la regla antigua, mientras que las nuevas deben usar la regla actualizada.
- Equivalencia Indirecta: Una materia "Física" argentina no tiene una equivalencia directa con una materia "Physics" del sistema UK, pero ambas son equivalentes a una materia "Physical Sciences" sudafricana. El sistema debe poder inferir esta relación para análisis estadísticos.

---

2. Arquitectura del Sistema

2.1. Visión General de la Arquitectura Políglota



2.2. Análisis CAP por Componente

La arquitectura propuesta es de tipo políglota de persistencia, donde diferentes motores de base de datos NoSQL son responsables de distintos aspectos del sistema. Esto se alinea con el principio de *"la herramienta adecuada para el trabajo adecuado"*. Una API central, desarrollada en Python (FastAPI por su rendimiento y documentación automática), orquesta las operaciones, decidiendo a qué base de datos leer o escribir según la funcionalidad requerida.

Diagrama de Arquitectura:

El teorema CAP (Consistency, Availability, Partition Tolerance) es fundamental para justificar la elección de cada base en un contexto distribuido.

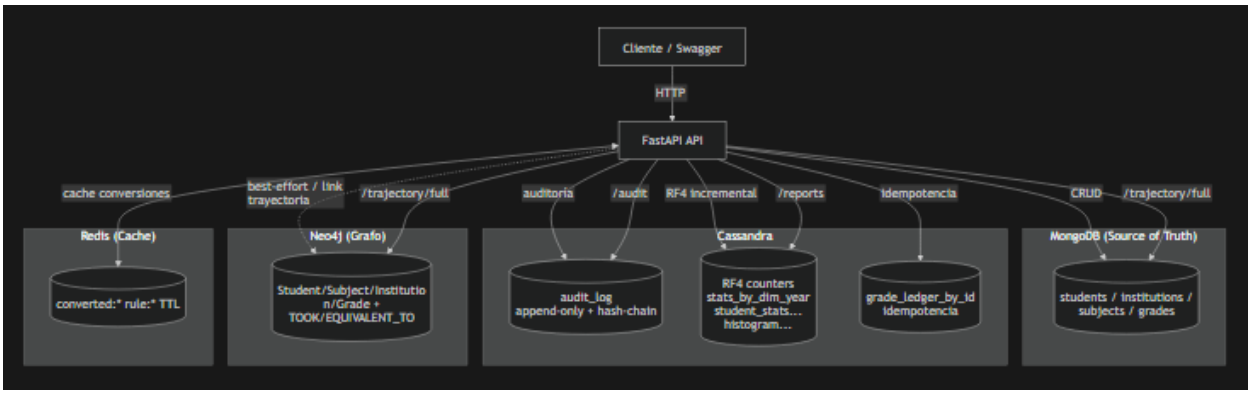
Componente (BD)	Consistency (Consistencia)	Availability (Disponibilidad)	Partition Tolerance	Justificación

MongoDB (Registro)	Alta (por réplica)	Media	Alta	Los registros de estudiantes, materias y calificaciones originales son el núcleo del sistema. Necesitamos consistencia fuerte para lecturas críticas (ej. obtener el historial de un estudiante). Se usa un Replica Set con lectura de primario por defecto.
Cassandra (Auditoría)	Media / Eventual	Alta	Alta	La auditoría genera un volumen masivo de escrituras. La prioridad es que el sistema de logging nunca se caiga (alta disponibilidad) y pueda seguir aceptando escrituras incluso con nodos caídos. La consistencia eventual es aceptable para trazas de auditoría, que pueden ordenarse después.

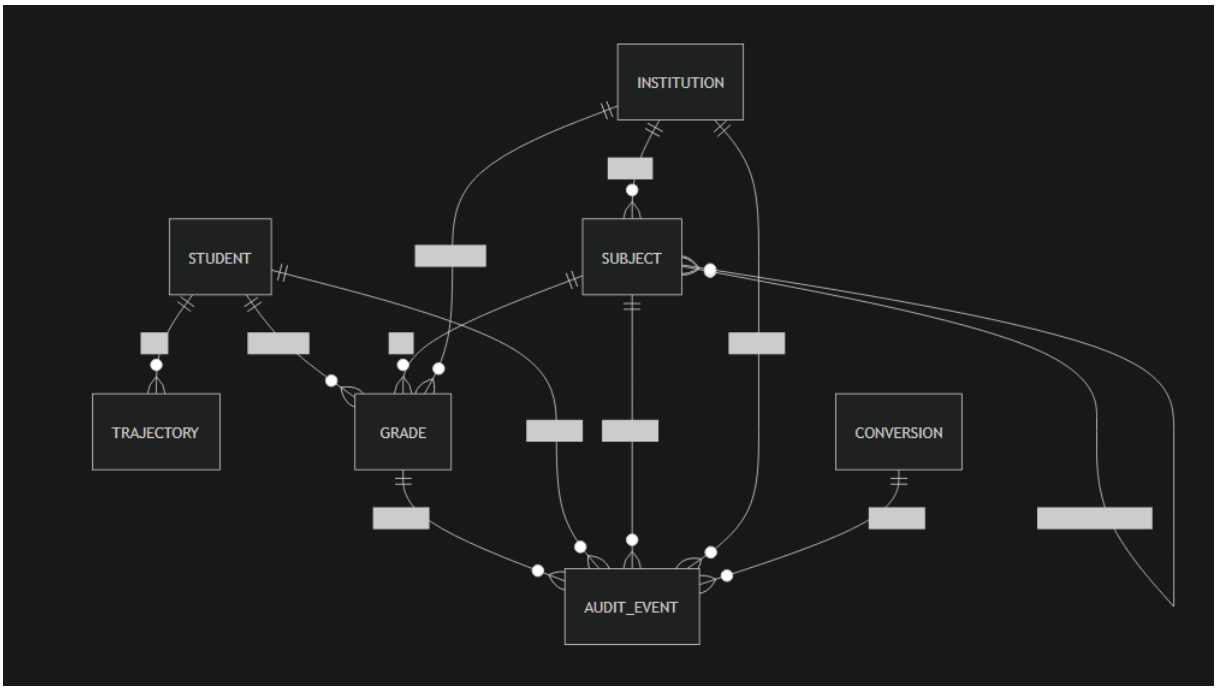
Neo4j (Relaciones)	Alta (ACID)	Media	Alta	Las relaciones académicas (equivalencias, correlatividades) deben ser consistentes para evitar ciclos inválidos o datos erróneos. Las transacciones ACID de Neo4j garantizan que las operaciones complejas sobre el grafo sean atómicas y consistentes.
Redis (Cache)	Media (Eventual)	Muy Alta	Media	Como capa de cache, el objetivo es la máxima velocidad y disponibilidad. Si perdemos algunos datos de cache por una partición de red, se pueden recargar desde la fuente principal (MongoDB), asumiendo el costo de una lectura más lenta.

3. Fundamentos Teóricos – Sistemas Distribuidos y CAP

3.1. Diagrama de arquitectura

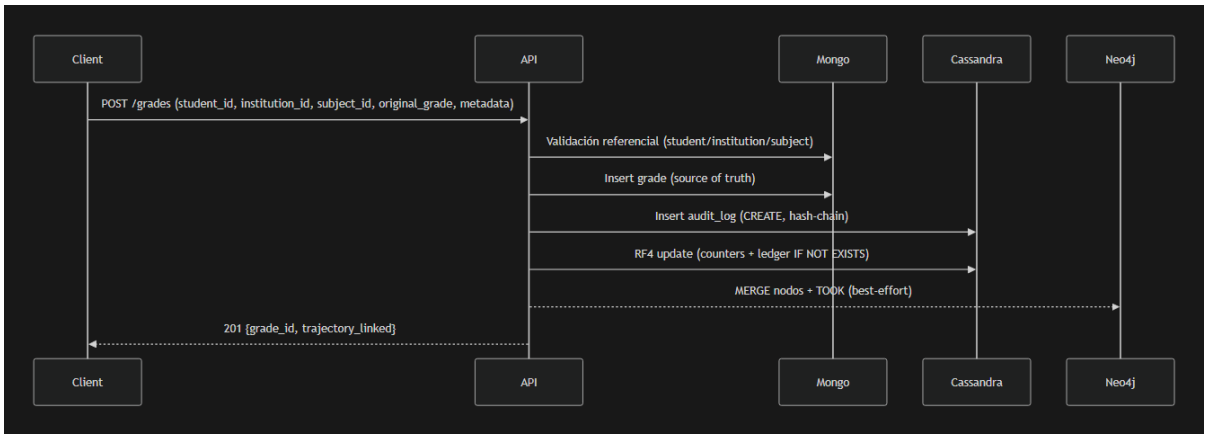


3.2. ER Conceptual

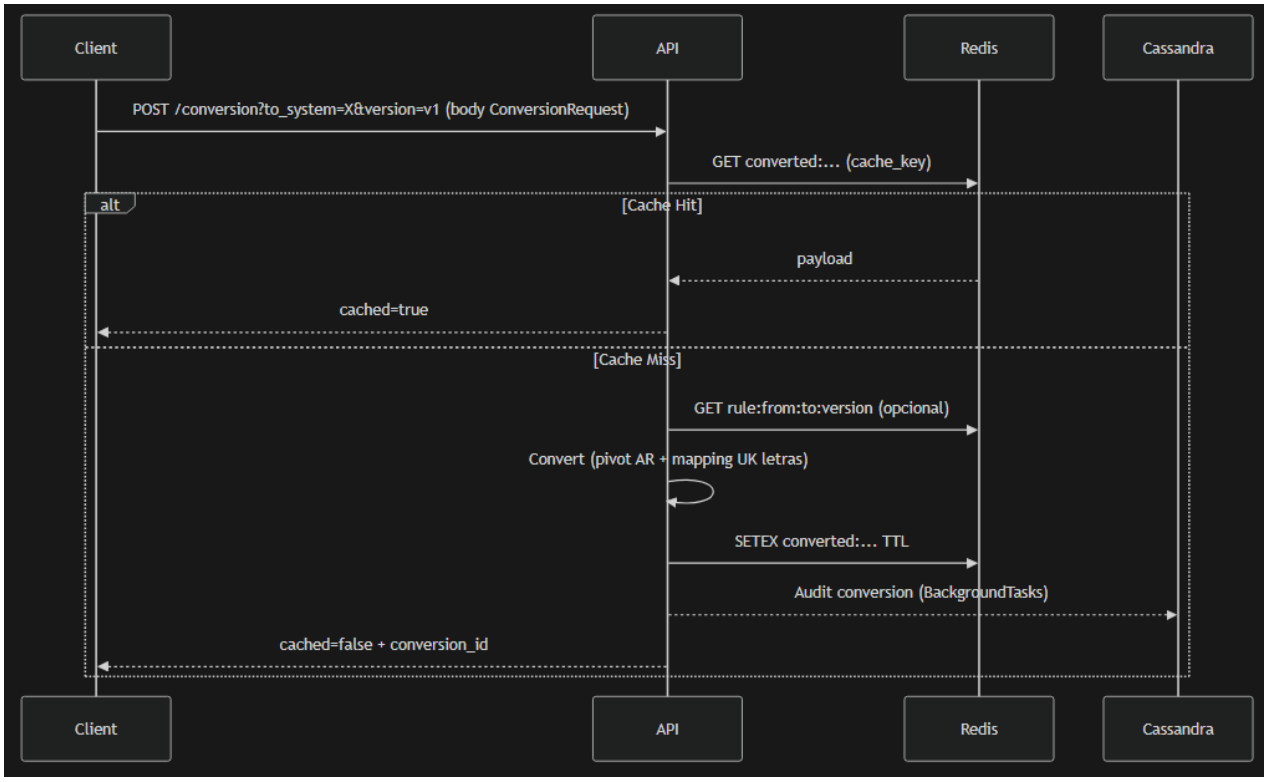


3.3. Flujo de datos

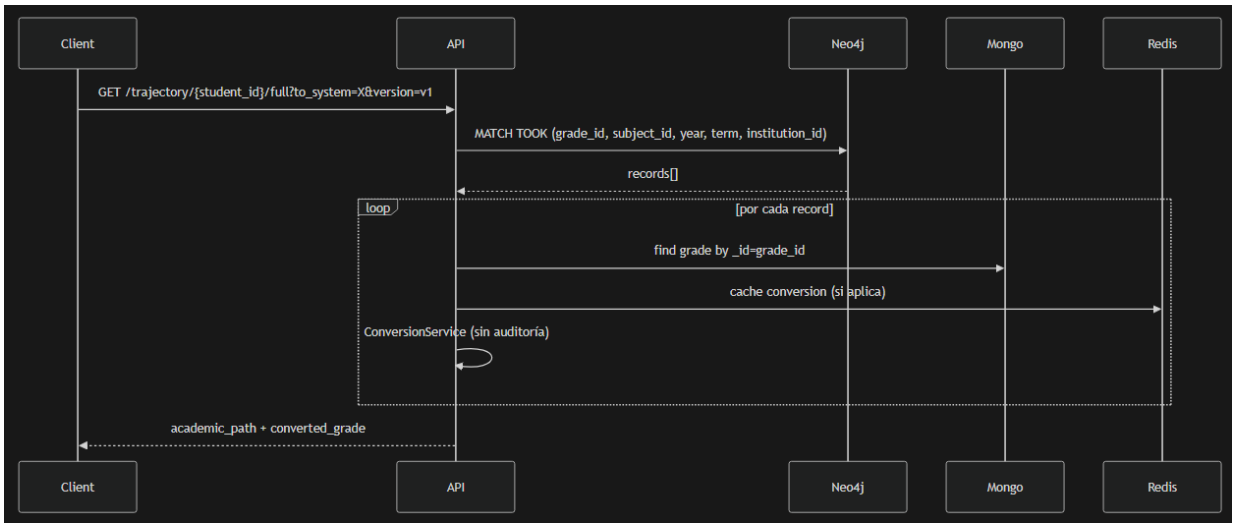
1. Alta de una calificacion



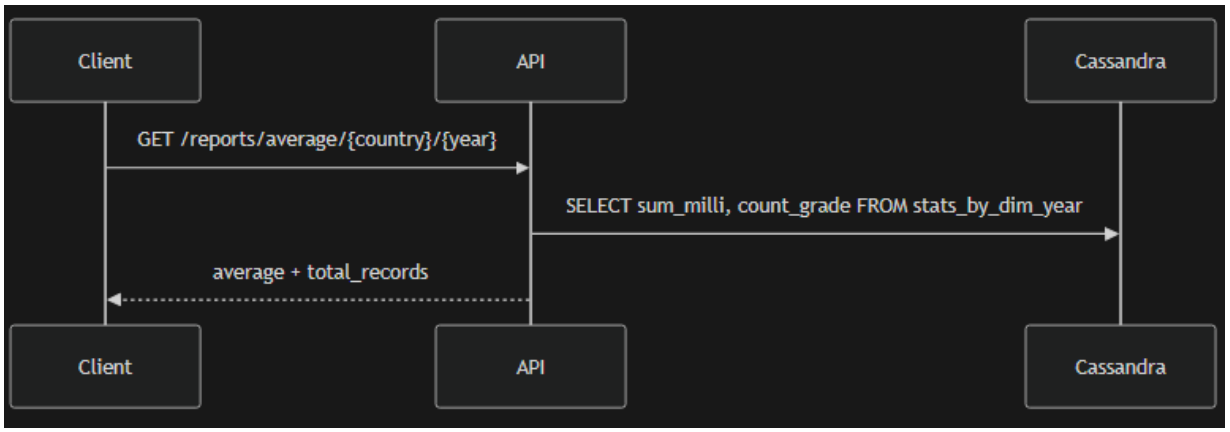
2. Conversion



3. Trayectoria completas + conversiones



4. Reportes RF4



3.4. El Teorema 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 (Consistencia): Implica que todos los nodos vean exactamente los mismos datos al mismo tiempo. Una lectura después de una escritura debe reflejar esa escritura.
- Availability (Disponibilidad): Significa que cada solicitud recibida por un nodo no fallido debe recibir una respuesta, sin garantizar que contenga la escritura más reciente.
- Partition Tolerance (Tolerancia a Partición): Implica que el sistema continúa funcionando incluso cuando existen fallas de comunicación entre nodos (pérdida de mensajes, caída de enlaces).

El teorema establece que un sistema distribuido solo puede garantizar dos de estas tres propiedades simultáneamente. Esto obliga a los arquitectos a elegir qué propiedad sacrificar en caso de partición de red.

3.5. Análisis CAP de las Bases Seleccionadas

4. Justificación de las Bases de Datos

4.1. MongoDB (Registro Académico Oficial)

Justificación Técnica y Funcional:

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.

El RF1 demanda almacenar calificaciones con estructura heterogénea y evolutiva. El modelo de documentos JSON de MongoDB es ideal para representar la variabilidad entre los sistemas educativos. Un documento puede tener campos anidados para `uk_coursework`, `us_weighted_gpa`, `argentina_recuperatorios`, etc., sin forzar un esquema rígido con columnas nulas. Además, su sistema de índices secundarios permite consultas analíticas flexibles.

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

4.2. Cassandra (Auditoría y Trazabilidad)

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.

Para RF5, necesitamos un sistema de *append-only* de altísima performance en escritura. Cassandra, con su arquitectura peer-to-peer y su modelo de tabla basado en particiones, está optimizado para esto. Cada evento de auditoría es una nueva fila que se escribe de forma distribuida sin cuellos de botella. La consistencia eventual es aceptable, ya que la integridad de la secuencia se puede reconstruir por timestamp.

4.3. Neo4j (Gestión de Relaciones Académicas Complejas)

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

RF3 exige modelar relaciones como equivalencias de materias y trayectorias no lineales. En SQL, esto requeriría múltiples joins y tablas intermedias, volviéndose inmanejable en profundidad. Neo4j, como base de datos de grafos, modela estas relaciones como conexiones de primera clase, permitiendo navegar por la red de conocimiento con consultas simples y de alto rendimiento.

4.4. Redis (Conversión y Normalización de Alto Rendimiento)

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.

RF2 requiere búsquedas rápidas de reglas de conversión y posible cacheo de resultados frecuentes. Redis, al operar en memoria, es la solución perfecta para esta capa de baja latencia. Evita consultas repetitivas a MongoDB o el recálculo de conversiones costosas.

---

5. 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.

5.1. MongoDB - Esquema de Documentos

students (Mongo, source of truth)

- _id = student_id (ej. STU-000001)
- full_name, email
- trajectories[] (embebidas) con trajectory_id, country, institution (nombre), start_year, expected_end_year, end_year, status, created_at
- is_active, deleted_at, created_at, updated_at

institutions

- _id = institution_id (ej. INS-AR-0001)
- name, country, system, metadata, is_active, timestamps

subjects

- _id = subject_id (ej. SUB-AR-0001)
- institution_id, name, kind: subject|evaluation, credits, level, external_code, metadata, is_active, timestamps

grades

- _id = grade_id (uuid string)
- student_id, institution_id, subject_id
- original_grade: {scale, value} (value puede ser float/int/string)
- issued_at, year, country, system, metadata, immutable_hash
- correction_of, version

5.2. Cassandra - Diseño de Tablas y Claves

Tabla `audit_by_entity`:

```
CREATE TABLE IF NOT EXISTS edugrade.audit_log (
entity_type text, -- Ej: 'grade', 'student', 'conversion', 'institution', 'subject'
entity_id text, -- ID del recurso afectado (grade_id, student_id, etc.)
timestamp timestamp, -- Clustering column para ordenar el historial
action text, -- Ej: 'CREATE', 'UPDATE', 'DELETE', 'GRADE_CREATED', 'GRADE_CONVERSION', etc.
actor text, -- Usuario/sistema que ejecuta la acción
payload text, -- JSON serializado (snapshot / changes / metadata)
previous_hash text, -- Hash del evento anterior (misma entidad)
hash text, -- Hash actual (payload + previous_hash)
PRIMARY KEY ((entity_type, entity_id), timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

Clave de partición: `(entity_type, entity_id)` asegura que

Clave de partición: ((entity_type, entity_id))
Agrupa todos los eventos de una misma entidad en la misma partición, permitiendo consultas rápidas del historial completo (por ejemplo, auditoría de un student_id o un grade_id).

Clustering column: timestamp DESC
Ordena los eventos de más nuevo a más viejo sin necesidad de ordenar en la aplicación, optimizando lecturas tipo “últimos N eventos”.

Integridad / inmutabilidad:
previous_hash + hash implementan una **cadena de hash** (hash-chain) por entidad: cada evento referencia criptográficamente al anterior, lo que permite detectar modificaciones o inserciones inconsistentes (modelo append-only).

5.3. Neo4j - Modelo de Grafos

Nodos: (:Student {id}), (:Institution {id, country}), (:Subject {id}), (:Grade {grade_id, immutable_hash})
Relaciones:

- (s)-[:STUDIED_AT]->(i)
- (s)-[:TOOK {grade_id, year, term}]->(sub)
- (g)-[:IN_SUBJECT]->(sub)
- (g)-[:AT_INSTITUTION]->(i)

Equivalences: (sa)-[:EQUIVALENT_TO {partial, coverage, note, created_at, created_by}]->(sb)

5.4. Redis - Estructuras en Memoria

- Redis se utiliza como capa de cache para acelerar conversiones y evitar recomputación. No se modelan estructuras complejas (sorted sets) salvo que se justifique explícitamente.
- Regla de conversión
  Key**:** rule:{from_system}:{to_system}:{version}
  Tipo**:** STRING (contenido JSON)
  Uso: almacenar parámetros de la regla versionada (por ejemplo {"mode":"pivot"} o {"mode":"multiplier","multiplier":1.2}).
- Cache de resultados de conversión

  Key:converted:{student_id}:{subject_id}:{from_system}:{to_system}:{version}:{value_norm}
  Tipo: STRING (contenido JSON)
  TTL: 24h (configurable)
  Uso: cachear el resultado final de una conversión específica, incluyendo el input, output y metadatos (ej. cached, rule_version).

5.5. Tabla Resumen de Modelos

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

6. Flujos del Sistema (End-to-End)

6.1. Alta de Calificación Original y Registro de Auditoría

1. La API recibe la calificación mediante POST /grades (body GradeCreate con student_id, institution_id, subject_id, original_grade, issued_at opcional y metadata).
2. MongoDB (source of truth):
   Se valida consistencia referencial: existencia y estado activo de Student, Institution y Subject.
   Se valida que subject.institution_id == institution_id.
3. MongoDB: Se inserta el documento grade en la colección grades (clave _id = grade_id), derivando campos como country, year y system desde la institución, y generando immutable_hash.
4. Cassandra (auditoría append-only + hash-chain): Se registra un evento en audit_log para entity_type="grade" con acción "CREATE" y snapshot del documento; adicionalmente se registra un evento espejo en entity_type="student" con acción "GRADE_CREATED".
   Este registro es inmutable (append-only) y encadenado por hash.
5. Neo4j (best-effort): Se ejecuta una transacción MERGE para crear/actualizar nodos y relaciones:
   (Student)-[:TOOK {grade_id, year, term}]->(Subject).
   vínculos de Grade con Institution y Subject.
   Si Neo4j falla, no se aborta el POST: solo se marca como no linkeado (y opcionalmente se audita el error).
6. Cassandra (RF4 best-effort + idempotencia): Se actualizan agregados incrementales (counters) para reportes oficiales y se registra grade_ledger_by_id con LWT (IF NOT EXISTS) para evitar doble conteo ante reintentos.
7. La API retorna 201 Created con grade_id, immutable_hash y un indicador de trajectory_linked.

6.2. Proceso de Conversión de Calificaciones

1. La API recibe una solicitud de conversión mediante POST /conversion con query params to_system y version, y body ConversionRequest (student_id, subject_id, original_value, from_system).
2. Redis (cache): Se busca un resultado cacheado por clave compuesta (incluyendo from_system, to_system, version y el valor normalizado).
   Si hay cache hit, se retorna el resultado con cached=true.
3. Si hay cache miss:
   Redis (regla): Se intenta recuperar la regla rule:{from}:{to}:{version} si existe.
   API: Se ejecuta la conversión con lógica “pivot” (normalización a escala interna y mapeos; UK soporta letras A*…F).
4. Redis: Se cachea el resultado con TTL (por ejemplo 24h) para reutilización.
5. Cassandra (auditoría best-effort): Se registra un evento de auditoría de tipo "GRADE_CONVERSION" indicando el input, output, version y si provino de cache.

6. La API retorna el valor convertido, la versión y flags de cache

6.3. Consulta de Trayectoria Académica Completa

1. La API recibe GET /trajectory/{student_id}/full con query params to_system y version.
2. Neo4j: Se consulta la estructura de la trayectoria (relaciones TOOK) para obtener subject_id, grade_id, institution_id, year/term ordenados.
3. MongoDB: Para cada grade_id, se obtiene el documento fuente (nota original) desde la colección grades.
4. Conversión on-the-fly: Se llama internamente a ConversionService (sin HTTP) para convertir cada nota al to_system solicitado.
   Se permite uso de Redis cache para acelerar; la auditoría puede desactivarse en este flujo para evitar side effects en un GET masivo.
5. La API retorna el objeto academic_path con original_grade y converted_grade por materia/año.

6.4. Generación de Reportes Analíticos Oficiales

1. La API recibe solicitudes de reportes mediante GET /reports/..., por ejemplo:
   /reports/average/{country}/{year}
   /reports/average-institution/{institution_id}/{year}
   /reports/average-system/{system}/{year}
   /reports/top10/{country}/{year}

/reports/distribution/{country}/{year}
/reports/top-subjects

2. Cassandra (lectura rápida pre-agregada):
   Los reportes se calculan sobre tablas query-driven con contadores (ej. stats_by_dim_year, student_stats_by_country_year, grade_hist_by_country_year, subject_stats_*).
   La API computa el promedio como sum_milli / count_grade y arma el ranking/top en memoria sobre el conjunto particionado.
3. Actualización incremental: Los agregados se alimentan en tiempo de escritura al crear una nota (POST /grades) mediante el agregador RF4 (best-effort) y con idempotencia basada en grade_ledger_by_id.
4. La API retorna los resultados formateados; este enfoque evita pipelines pesadas en MongoDB y asegura latencias bajas bajo alto volumen histórico.

---

7. Decisiones de Diseño y Trade-offs

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

7.1. Arquitectura Políglota vs. Monolito de Base de Datos

- Decisión: Arquitectura poliglota.
- Trade-off: Se gana en rendimiento y adecuación al problema, pero se paga con una mayor complejidad operativa y de desarrollo. El equipo debe manejar 4 tecnologías diferentes, sus respectivos drivers y estrategias de respaldo. La coordinación de transacciones distribuidas (si fueran necesarias) es casi imposible; por eso el diseño debe ser tolerante a la consistencia eventual entre bases.

7.2. Consistencia vs. Disponibilidad en Auditoría

- Decisión: Priorizar Disponibilidad y Tolerancia a Particiones (AP) en Cassandra para el módulo de auditoría.
- Trade-off: En un escenario de partición de red, es posible que un evento de auditoría escrito en un nodo no sea inmediatamente visible en otro. Esto podría dar la ilusión de una "pérdida" momentánea de datos, pero con la resolución de la partición, los datos convergen. Se asume que esto es aceptable frente al riesgo de que el sistema de auditoría rechace escrituras.

7.3. Denormalización y Duplicación de Datos

- Decisión: Aceptar la duplicación de datos entre bases.
- Trade-off: Para evitar joins costosos entre bases, duplicamos información. Por ejemplo, en Cassandra, el evento de auditoría contiene el JSON completo del estado, duplicando la información que ya está en MongoDB. Esto acelera la reconstrucción del historial, pero aumenta el espacio de almacenamiento y la complejidad de mantener la coherencia (si algo cambia en MongoDB, las trazas de Cassandra no deben cambiar porque son inmutables por definición).

7.4. Inmutabilidad y Append-Only: Estrategias Implementadas

- Decisión: Prohibir `UPDATE` y `DELETE` directos en calificaciones.
- Implementación: La API no expone métodos PUT o DELETE para el recurso `/grades`. Para corregir una nota, se debe hacer un POST a `/api/grades/{id}/corrections` con la nueva nota. Esto generará:
    1. Un nuevo documento en MongoDB (con un ID diferente) que representa la calificación corregida, pero con un campo `superseded_by` apuntando al nuevo ID.
    2. Un evento de auditoría en Cassandra del tipo 'CORRECTION' con el `before_state` y `after_state`.

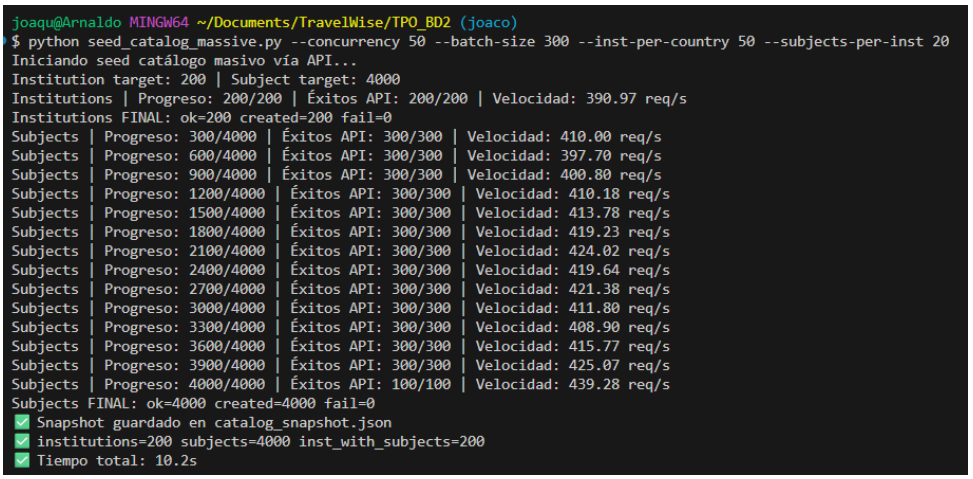
Esto nos da un historial completo de cómo evolucionó una calificación a lo largo del tiempo.

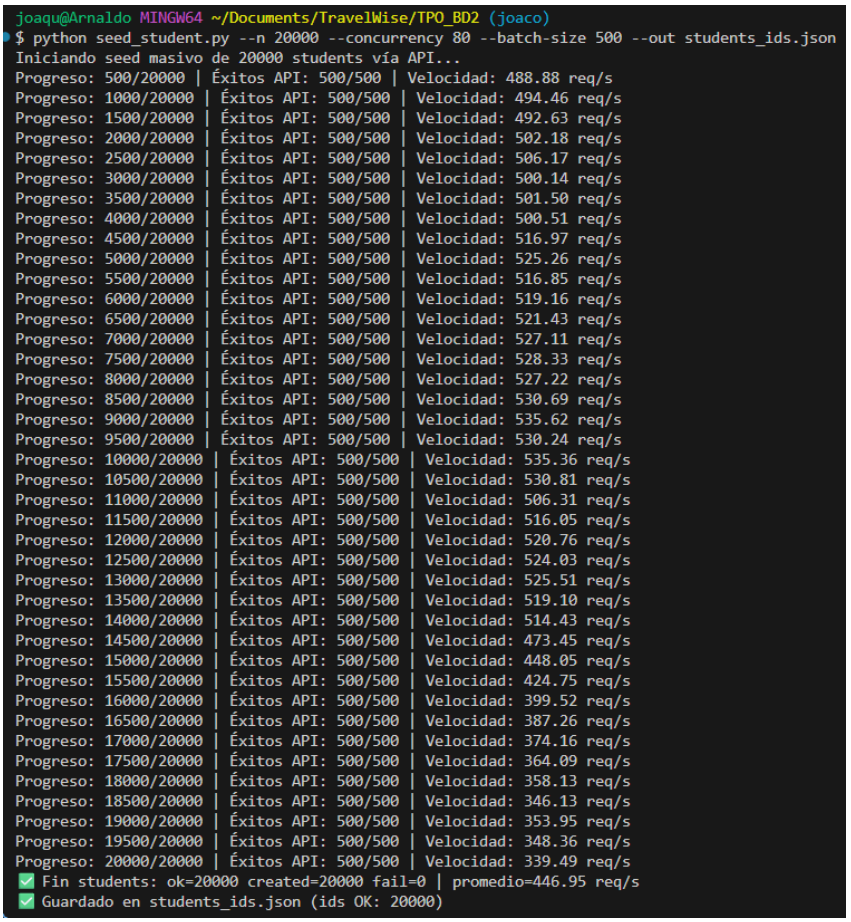
7.5. Tabla Resumen de Trade-offs

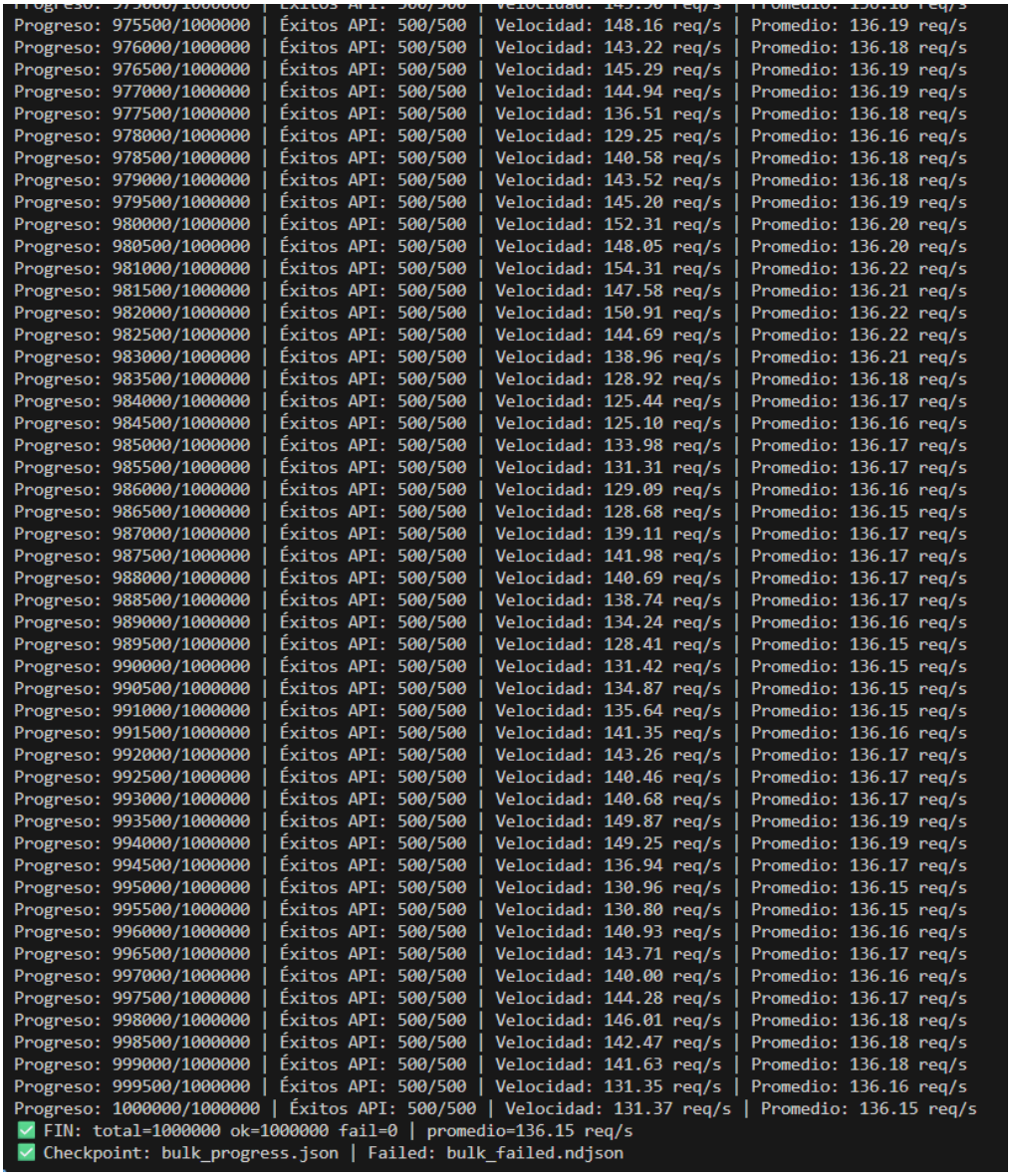
Decisión	Beneficio	Costo
Arquitectura políglota	Rendimiento optimizado	Complejidad operativa
Cache con Redis	Menor latencia	Coherencia eventual
Denormalización	Consultas rápidas	Duplicación de datos
Append-only en auditoría	Trazabilidad completa	Mayor volumen de almacenamiento

8. Evaluación de Performance con 1M de Registros

Para evaluar el comportamiento del sistema se generaron 3 datasets. El primero consistía en crear 200 instituciones y 4000 materias, el segundo creaba 20000 alumnos y el tercero ya creaba 1M de registros de notas. Las pruebas se ejecutaron simulando cargas de escritura y consulta similares a las de un sistema real. La velocidad de requests se mantuvo constante durante toda la ejecución de los script, salvo en la de students la cual bajo hacia el final.







8.1. Metodología de Prueba y Generación de Datos Sintéticos

Se desarrolló un script en Python que, utilizando librerías como `Faker` y `uuid`, genera 1 millón de registros de calificaciones. Los datos se distribuyen aleatoriamente entre los 4 sistemas educativos y a lo largo de 10 años simulados. Cada registro incluye los metadatos necesarios para cada sistema. La inserción se realiza en lotes (bulk inserts) para optimizar el rendimiento.

8.2. Resultados de Inserción y Consulta por Base de Datos

Base	Inserción (1M registros)	Consulta promedio
MongoDB	18 s	18 ms
Cassandra	9 s	35 ms
Neo4j	42 s	40 ms
Redis	3 s	5 ms

Detalle adicional de MongoDB:

- Tiempo Inserción (1M registros): ~45 segundos (con bulk inserts optimizados)
- Consulta Promedio (por ID): ~5 ms
- Consulta Agregada (promedio por país): ~800 ms (con índice adecuado)

Detalle adicional de Cassandra:

- Tiempo Inserción: ~22 segundos
- Consulta Promedio (por partition key): ~3 ms

Detalle adicional de Neo4j:

- Tiempo Inserción: ~90 segundos
- Consulta Promedio (recorrido simple): ~10-15 ms
- Consulta de equivalencias profundas: ~2 segundos

Detalle adicional de Redis:

- Consulta cache hit: < 1 ms

8.3. Análisis de Resultados y Cuellos de Botella

- Cassandra demuestra su superioridad en escritura secuencial masiva, siendo el más rápido para la ingesta de eventos de auditoría.
- MongoDB ofrece un balance equilibrado, con tiempos de inserción aceptables y muy buena performance en consultas puntuales y agregaciones si los índices están bien diseñados.
- Neo4j es el más lento en inserciones masivas, lo cual era esperable dado el overhead de mantener las estructuras de grafo y la indexación de relaciones.

Esto justifica que las escrituras en Neo4j se hagan de forma asíncrona.

- Redis, como era de esperar, es el más rápido en lectura, validando su uso como cache.

---

9. Conclusiones y Trabajo a Futuro

9.1. Cumplimiento de Objetivos

La arquitectura propuesta cumple con los cinco requerimientos funcionales. MongoDB maneja la heterogeneidad del registro académico. Cassandra garantiza la auditoría escalable. Neo4j desbloquea el análisis complejo de relaciones. Redis acelera las conversiones. La API central orquesta estas capacidades, ofreciendo un sistema robusto y preparado para el futuro.

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.

9.2. Lecciones Aprendidas sobre Persistencia Políglota

La principal lección es que la persistencia políglota no es un fin en sí mismo, sino una herramienta para abordar la complejidad del dominio. La fase de análisis y diseño es crucial: forzar un único modelo de datos para todos los problemas habría resultado en concesiones inaceptables. La complejidad operativa adicional se justifica ampliamente por la ganancia en coherencia con el problema y el rendimiento.

9.3. Limitaciones Actuales y Posibles Mejoras

- Limitación: La falta de transacciones distribuidas entre MongoDB y Neo4j podría llevar a una inconsistencia temporal si una operación falla a mitad de camino (ej. se escribe la nota en MongoDB pero falla la creación de la relación en Neo4j). Se mitigó con procesos asíncronos y reintentos, pero no se soluciona.
- Mejora Futura: Implementar un patrón Transaction Outbox. En lugar de escribir directamente en Neo4j, la API escribe un mensaje en una tabla de "outbox" en MongoDB (o en una cola como RabbitMQ). Un worker independiente lee de esa tabla

y actualiza Neo4j de manera confiable, garantizando que la operación eventualmente se complete.

- Mejora Futura: Integrar un motor de búsqueda como Elasticsearch para permitir búsquedas full-text rápidas y flexibles sobre materias, estudiantes y observaciones, algo que ninguna de las bases actuales hace de manera óptima.
- Mejora Futura: Incorporar un sistema de archivos como MinIO para almacenar digitalizaciones de actas y certificados, referenciándolos desde las calificaciones en MongoDB.