
# Ingeniería de datos II

# Tpo-Talentum+

Integrantes:

- Barreto Molas, Reinaldo
- Fuensalida, Mathias Lionel
- Garagaza, Agustina
- Lago De Azevedo Moura, Gustavo

# Modelo de Candidatos y Empleados

**Base de datos utilizada: Documental (MongoDB)**

Para modelar los perfiles de candidatos y empleados de la plataforma Talentum+, se eligió implementar una base de datos documental, concretamente MongoDB, debido a las siguientes razones:

## Justificación del modelo documental:

Los perfiles de talento en el sector IT presentan alta heterogeneidad y estructura flexible. No existe un esquema único:

- Algunos perfiles incluyen muchas experiencias laborales, otros ninguna.
- Algunos candidatos registran cursos, certificaciones, idiomas o proyectos personales, mientras que otros no.
- El historial de participación en procesos de selección y el feedback también es variable en cantidad y contenido.

Este tipo de datos no tabulares y altamente anidados no se adapta bien a un esquema rígido como el de una base relacional.

## Ventajas de usar MongoDB para este modelo:

1. **Esquema flexible**: No es necesario definir de antemano todas las columnas, lo que permite almacenar fácilmente estructuras variables entre perfiles.

2. **Documentos anidados**: Se pueden representar internamente arrays y subdocumentos, como listas de experiencias laborales, evaluaciones, entrevistas, certificaciones, etc.

3. **Alta velocidad de escritura y lectura**: MongoDB es ideal para sistemas donde los documentos se actualizan constantemente y se consultan con frecuencia, como perfiles en evolución.

4. **Facilidad de escalar horizontalmente**: Dado que Talentum+ busca escalabilidad y alta disponibilidad, MongoDB permite distribución de datos en múltiples nodos (sharding).

5. **Ideal para almacenar historial de cambios**: La plataforma requiere registrar evolución y actualizaciones en los perfiles. MongoDB facilita este seguimiento embebiendo arrays de historial con timestamps y eventos

## Estrategias Operacionales para el Modelo de Candidatos

Para garantizar la robustez, escalabilidad y disponibilidad de los datos de candidatos y empleados, que son el núcleo de la plataforma, se implementarán las siguientes estrategias:

- **Replicación y Alta Disponibilidad:** Utilizaremos MongoDB Replica Sets con una configuración mínima de un nodo primario y dos secundarios. Esto garantiza la alta disponibilidad mediante un proceso de failover automático: si el nodo primario falla, uno de los secundarios es promovido a nuevo primario en segundos. Además, permite distribuir la carga de lectura (Read Preference) hacia los nodos secundarios, optimizando el rendimiento para consultas de perfiles.

- **Escalabilidad Horizontal (Sharding):** A medida que la plataforma crezca a miles o millones de perfiles, la colección candidatos será particionada horizontalmente mediante Sharding. Como Shard Key, se utilizará un índice hasheado sobre el campo _id. Esta estrategia asegura una distribución perfectamente uniforme de los documentos a través de los shards, evitando la creación de "hotspots" (shards sobrecargados) y garantizando un escalado de escritura y lectura lineal.

- Backup y Recuperación: Se implementará una estrategia dual:

    1. Snapshots a nivel de sistema de archivos: Backups diarios automatizados (ej. snapshots de volúmenes EBS en AWS) para una recuperación rápida ante desastres.
    2. Backups lógicos con mongodump: Se ejecutarán cada 4 horas para permitir una recuperación a un punto en el tiempo (Point-in-Time Recovery). Estos backups se comprimirán y almacenarán de forma segura en un servicio de almacenamiento de objetos como Google Cloud Storage o AWS S3.

**Implementación:**

```
{
  "_id": "cand_001",
  "nombre": "Ana Torres",
  "email": "ana.torres@example.com",
  "telefono": "+54 911 2222-3333",
  "linkedin": "https://linkedin.com/in/anatorres",
  "experiencia": [
    {
      "empresa": "Oracle",
      "rol": "Data Analyst",
      "desde": "2019",
      "hasta": "2022",
       "proyectos_destacados": ["Migración de dashboards", "Optimización
de queries"]
    }
  ],
  "skills_tecnicos": ["SQL", "Python", "Power BI"],
     "skills_blandos":  ["Comunicación",  "Resolución  de  problemas",
"Trabajo en equipo"],
  "certificaciones": [
      { "nombre": "Data Science Professional Certificate", "entidad":
"IBM", "año": 2023 }
  ],
  "idiomas": [
    { "idioma": "Inglés", "nivel": "Avanzado" }
  ],
  "historial_procesos": [
    {
      "proceso_id": "proc_456",
      "empresa": "Globant",
```

```
      "estado": "entrevista técnica",
      "evaluaciones": [
        {
          "fecha": "2025-05-10",
          "tipo": "técnica",
          "puntuacion": 8.5,
          "comentarios": "Buen dominio de SQL y Python"
        }
      ]
    }
  ],
  "historial": [
    { "fecha": "2024-06-01", "cambio": "Agregó skill 'Power BI'" },
      { "fecha": "2025-05-12", "cambio": "Actualizó experiencia en
Oracle" }
  ]
}
```

**Consultas en MongoDB**

- Buscar todos los candidatos que tienen experiencia en "Oracle":

```
db.candidatos.find({ "experiencia.empresa": "Oracle" });
```

- Listar candidatos que hablen inglés en nivel avanzado:

```
db.candidatos.find({
  idiomas: { $elemMatch: { idioma: "Inglés", nivel: "Avanzado" } },
});
```

- Buscar candidatos con la certificación 'Data Science Professional Certificate':

```
db.candidatos.find({
  "certificaciones.nombre": "Data Science Professional Certificate",
});
```

- Filtrar candidatos que saben 'Python' y tienen más de 1 evaluación en su historial de procesos:

```
db.candidatos.find({
skills_tecnicos: "Python",
  "historial_procesos.evaluaciones.1": { $exists: true },
});
```

# Publicación de Búsquedas y Matching Inteligente

**Base de datos utilizada: Grafos Neo4j**

En Talentum+, la publicación de ofertas laborales se puede modelar naturalmente como un grafo, donde los nodos representan entidades como Empresas, Ofertas, Tecnologías, Candidatos, Habilidades, y las relaciones capturan conexiones entre ellos (por ejemplo, "OFRECE", "REQUIERE", "TIENE_SKILL").

## Justificación del uso de Neo4j

Las ofertas laborales y el matching inteligente son casos ideales para bases de datos de grafos porque:

- **Modelo relacional flexible:** Las relaciones entre candidatos, tecnologías, requisitos y ofertas se expresan directamente, sin necesidad de esquemas rígidos.
- **Consultas de matching avanzadas:** Los algoritmos de grafos y el lenguaje Cypher permiten consultas complejas para matching semántico, recomendación y análisis de caminos entre candidatos y ofertas.
- **Evolución del esquema:** Se pueden agregar nuevas entidades y relaciones fácilmente sin afectar la estructura existente.
- **Rendimiento en consultas conectadas:** Las consultas basadas en conexiones y patrones (ej. encontrar candidatos con skills coincidentes en la oferta) son más eficientes que en bases documentales o relacionales.

## Modelo de datos ejemplo en Neo4j

- Nodo **Empresa** con propiedades: nombre
- Nodo **Oferta** con propiedades: puesto, ubicación, descripción, fecha_publicación
- Nodo **Tecnologia** con propiedad: nombre
- Nodo **Beneficio** con propiedad: nombre
- Nodo **Idioma** con propiedad: nombre
- Nodo **Candidato** con propiedades: nombre, experiencia (años), etc.

## Relaciones típicas:

- (Empresa)-[**:OFRECE**]->(Oferta)
- (Oferta)-[**:REQUIERE_TECNOLOGIA**]->(Tecnologia)
- (Oferta)-[**:INCLUYE_BENEFICIO**]->(Beneficio)
- (Oferta)-[**:REQUIERE_IDIOMA**]->(Idioma)
- (Candidato)-[**:TIENE_SKILL**]->(Tecnologia)
- (Candidato)-[**:HABLA**]->(Idioma)

## Consultas en Neo4j

- Creación de una oferta de trabajo

```
CREATE (e:Empresa {nombre: "Globant"})
CREATE (o:Oferta {
  puesto: "Backend Developer",
  ubicacion: "Remoto",
   descripcion: "Buscamos un desarrollador backend para integrarse a un
equipo ágil con foco en microservicios.",
  fecha_publicacion: date("2025-06-01")
})
CREATE (t1:Tecnologia {nombre: "Node.js"})
CREATE (t2:Tecnologia {nombre: "MongoDB"})
CREATE (t3:Tecnologia {nombre: "Docker"})
CREATE (b1:Beneficio {nombre: "Home office"})
CREATE (b2:Beneficio {nombre: "Capacitaciones pagas"})
CREATE (b3:Beneficio {nombre: "Bono anual"})
CREATE (i1:Idioma {nombre: "Inglés intermedio"})

CREATE (e)-[:OFRECE]->(o)
CREATE (o)-[:REQUIERE_TECNOLOGIA]->(t1)
CREATE (o)-[:REQUIERE_TECNOLOGIA]->(t2)
CREATE (o)-[:REQUIERE_TECNOLOGIA]->(t3)
CREATE (o)-[:INCLUYE_BENEFICIO]->(b1)
CREATE (o)-[:INCLUYE_BENEFICIO]->(b2)
CREATE (o)-[:INCLUYE_BENEFICIO]->(b3)
CREATE (o)-[:REQUIERE_IDIOMA]->(i1)
```

- Encontrar todas las ofertas que requieren "MongoDB":

```
MATCH (o:Oferta)-[:REQUIERE_TECNOLOGIA]->(t:Tecnologia {nombre:
"MongoDB"})
RETURN o
```

- Ver ofertas con beneficios como "Home office":

```
MATCH (o:Oferta)-[:INCLUYE_BENEFICIO]->(b:Beneficio {nombre: "Home
office"})
RETURN o
```

- Buscar candidatos que hablan inglés y tienen al menos un skill requerido por una oferta:

```
MATCH (c:Candidato)-[:HABLA]->(:Idioma {nombre: "Inglés intermedio"})
MATCH
(c)-[:TIENE_SKILL]->(t:Tecnologia)<-[:REQUIERE_TECNOLOGIA]-(o:Oferta)
RETURN DISTINCT c, o
```

### Matching con candidatos en Neo4j

Supongamos que un candidato tiene nodos candidatos relacionados con tecnologías que domina y años de experiencia. Para hacer un matching por tecnologías y experiencia mínima:

```
MATCH (c:Candidato)-[:TIENE_SKILL]->(t:Tecnologia)
WHERE t.nombre IN ["Node.js", "Docker"] AND c.experiencia >= 2
RETURN c
```

### Ventajas del matching en Neo4j

- Las consultas pueden expresarse en términos de patrones y relaciones, no solo filtros sobre documentos.
- Fácil integración de nuevas dimensiones (ej. proyectos, certificaciones, recomendaciones).
- Posibilidad de usar algoritmos para matching semántico y ranking de candidatos.
- Mejor rendimiento en consultas con múltiples niveles de relaciones y datos conectados.

### Estrategias Operacionales para el Grafo de Matching

El grafo es fundamental para el matching y las recomendaciones. Su rendimiento y disponibilidad son críticos.

- **Replicación y Alta Disponibilidad (Causal Clustering):** Se recomienda un implementación de un Neo4j Causal Cluster en una configuración de 3 nodos Core. Este modelo Líder-Seguidor garantiza que todas las escrituras sean consistentes y coordinadas por un único líder. Si el líder falla, los nodos restantes eligen un nuevo líder, asegurando la continuidad del servicio. La consistencia causal garantiza que un usuario nunca verá efectos (ej. una recomendación) antes que sus causas (ej. haber añadido un skill), lo cual es vital para la experiencia de usuario.
- **Escalabilidad:** El escalado de lectura, que será la operación más frecuente en este componente (matching de muchos candidatos contra ofertas), se logrará añadiendo Read Replicas al clúster. Estas réplicas reciben los datos de los nodos Core pero no participan en los quórums de escritura, permitiendo manejar un volumen masivo de consultas de lectura de manera distribuida.
- **Backup y Recuperación:** Se recomienda realizar backups online diarios utilizando la herramienta neo4j-admin backup. Este procedimiento no requiere detener la base de datos, garantizando una operación 24/7. Los archivos de backup serán transferidos a un almacenamiento externo y versionado.

# Seguimiento de Procesos de Selección

**Base de datos utilizada: documental MongoDB**

La gestión de procesos de selección en Talentum+ implica manejar estados secuenciales, múltiples entidades relacionadas (candidatos,ofertas,entrevistas), y registrar.

## Justificación del modelo documental con MongoDB

- **Flexibilidad de esquema:** Los procesos de selección pueden variar en número de entrevistas, evaluaciones y notas, y MongoDB permite que cada documento almacene esa variabilidad sin alterar esquemas.
- **Historial embebido o separado:** Podemos guardar estados y cambios en el mismo documento o en colecciones aparte para auditar.
- **Operaciones atómicas a nivel documento:** MongoDB garantiza atomicidad dentro de un documento, útil para guardar procesos y sus detalles en un solo registro.
- **Consultas ágiles:** Se pueden indexar campos clave para acelerar búsquedas y reportes.

## Ejemplo de diseño de documentos en MongoDB

```
{
 "_id": ObjectId("..."),
 "id_candidato": ObjectId("..."),
 "id_oferta": ObjectId("..."),
 "estado_actual": "Entrevista Técnica",
 "fecha_inicio": ISODate("2025-05-01T00:00:00Z"),
 "fecha_fin": null,
 "historial_estados": [
  {"estado": "Preselección", "fecha": ISODate("2025-05-01T00:00:00Z")},
  {"estado": "Entrevista Técnica", "fecha": ISODate("2025-05-10T00:00:00Z")}
 ],
 "entrevistas": [
  {
   "fecha": ISODate("2025-05-10T10:00:00Z"),
   "entrevistador": "Juan Pérez",
   "feedback": "Buen conocimiento técnico, requiere mejora en comunicación."
  },
  {
   "fecha": ISODate("2025-05-15T15:00:00Z"),
   "entrevistador": "María Gómez",
   "feedback": "Sólido en experiencia, recomendado."
  }
 ]
```

### Ejemplo de consulta

- Obtener procesos activos de un candidato

```
db.procesos.find({
  id_candidato: ObjectId("id_del_candidato"),
  fecha_fin: null
})
```

### Ventajas

- **Esquema flexible y dinámico**: Permite almacenar procesos con diferentes estructuras (varias entrevistas, evaluaciones, campos personalizados) sin necesidad de modificar esquemas rígidos o hacer migraciones.
- **Documentos auto contenibles**: Toda la información relevante de un proceso (entrevistas, historial de estados, feedback) puede almacenarse en un solo documento, facilitando consultas rápidas y completas sin necesidad de joins complejos.
- **Alta escalabilidad**: MongoDB escala horizontalmente, lo que es ideal si la plataforma crece y se necesitan manejar grandes volúmenes de procesos y candidatos simultáneamente.

### Estrategias Operacionales para Procesos de Selección

Dado que este componente también utiliza MongoDB, se aplican las mismas estrategias de Replica Sets para alta disponibilidad y backups duales (snapshots y lógicos). Sin embargo, la estrategia de sharding es diferente:

- **Escalabilidad (Sharding):** Para la colección procesos, se propone una **Shard Key compuesta:** *{ id_empresa: 1, fecha_inicio: 1 }.* Esta clave agrupará los procesos de una misma empresa en los mismos shards, optimizando consultas muy comunes como "ver todos los procesos activos de la empresa X".
- **Consistencia en Operaciones Críticas:** Al actualizar el estado de un proceso (ej. de "Entrevista" a "Propuesta"), la operación de escritura utilizará un **Write Concern** de *majority*. Esto asegura que el cambio se ha replicado en la mayoría de los nodos del replica set antes de confirmarse, previniendo la pérdida de datos críticos ante un fallo del nodo primario.

# Gestión de Capacitación y Certificaciones

**Base de datos utilizada: documental MongoDB**

Para modelar el sistema de cursos, certificaciones y trayectorias de aprendizaje en Talentum+, se puede utilizar MongoDB, aprovechando su flexibilidad para almacenar documentos con relaciones embebidas o referencias entre entidades como cursos, candidatos y prerrequisitos

## Justificación del modelo documental con MongoDB

- **Representación flexible de relaciones:** Los cursos pueden contener listas de prerrequisitos como arreglos de IDs o documentos embebidos, permitiendo modelar dependencias entre cursos.
- **Rutas formativas personalizadas:** Se pueden almacenar trayectorias de aprendizaje como documentos que referencien cursos completados y pendientes.
- **Historial y recomendaciones:** Se pueden guardar documentos de candidatos con cursos realizados, fechas y certificaciones obtenidas para generar recomendaciones basadas en agregaciones y búsquedas.

**Ejemplo de modelado en MongoDB**

- Documento Curso

```
{
 "_id": ObjectId("curso_A"),
 "nombre": "Introducción a Python",
 "descripcion": "Curso básico de Python",
 "prerequisitos": [],   // sin prerrequisitos
 "modulos": [
   {"titulo": "Sintaxis básica", "duracion_horas": 5},
   {"titulo": "Estructuras de datos", "duracion_horas": 4}
 ]
}
```

- Documento Curso con prerrequisitos

```
{
 "_id": ObjectId("curso_B"),
 "nombre": "Machine Learning",
 "descripcion": "Curso avanzado de ML",
 "prerequisitos": [ObjectId("curso_A")],  // referencia a curso_A como prerrequisito
 "modulos": [
   {"titulo": "Regresión", "duracion_horas": 6},
   {"titulo": "Redes Neuronales", "duracion_horas": 8}
 ]
}
```

- Documento Candidato con cursos realizados

```
{
  "_id": ObjectId("cand_001"),
  "nombre": "Ana Torres",
  "cursos_realizados": [
    {
      "curso_id": ObjectId("curso_A"),
      "fecha_finalizacion": ISODate("2025-05-10T00:00:00Z")
    }
```

### Ventajas

- **Flexibilidad en el esquema:** Permite almacenar cursos, certificaciones y trayectorias con estructuras variables sin necesidad de esquemas rígidos, adaptándose fácilmente a cambios o extensiones.
- **Almacenamiento de datos semiestructurados:**Puede guardar módulos, prerrequisitos y detalles de cursos en documentos anidados, facilitando la representación natural de la información.
- **Consultas rápidas con índices:** MongoDB soporta índices en campos clave (como IDs de cursos, nombres y prerrequisitos), lo que acelera búsquedas y filtros para recomendaciones.
- **Escalabilidad horizontal:** Puede manejar grandes volúmenes de datos y usuarios, escalando eficientemente según crece la plataforma de capacitación.

### Ejemplo de consulta

- Búsqueda de cursos disponibles para un candidato. Supongamos que queremos encontrar cursos cuyos prerrequisitos el candidato ya completó pero que aún no tomó.

```
const cursosRealizados = db.candidatos.findOne(
   { _id: ObjectId("cand_001") },
   { cursos_realizados: 1 }
 ).cursos_realizados.map(c => c.curso_id);

   // Buscar cursos cuyos prerrequisitos estén contenidos en
cursosRealizados y que el candidato no haya realizado aún
 db.cursos.find({
   prerequisitos: { $all: cursosRealizados },
   _id: { $nin: cursosRealizados }
 })
```

# Gestión de Sesiones, Caché y Datos Temporales

## Base de datos utilizada: Clave-Valor en memoria (Redis)

Para responder a los requerimientos de alta disponibilidad, rendimiento instantáneo y gestión de datos volátiles como sesiones de usuario y caché, Talentum+ necesita un motor que priorice la velocidad por encima de todo. Una base de datos relacional o documental tradicional introduce latencia inaceptable para estas tareas.

### Justificación del modelo Clave-Valor (Redis)

- **Cacheo de Datos:** Almacenar temporalmente los resultados de consultas pesadas o datos accedidos frecuentemente (como perfiles de candidatos populares o los detalles de una oferta muy vista) reduce drásticamente la carga sobre las bases de datos persistentes (MongoDB, Neo4j) y mejora la velocidad de respuesta de la aplicación.
- **Gestión de Sesiones:** Guardar los tokens de sesión de los usuarios en Redis permite una validación de sesión casi instantánea en cada petición, lo cual es fundamental para una experiencia de usuario fluida y segura.
- **Datos en Tiempo Real:** Su velocidad la hace perfecta para contadores, leaderboards o notificaciones en tiempo real, como "hay 5 reclutadores viendo esta misma oferta ahora".

### Modelo de datos y Ejemplos en Redis

- Almacenar una sesión de usuario:

La clave puede ser el ID de sesión y el valor, el ID del usuario. Se establece un TTL (Time-To-Live) para que la sesión expire automáticamente.

```
# Almacena el ID de usuario 'user:123' para la sesión 'sess:xyzabc' por 3600 segundos (1 hora)
SETEX sess:xyzabc 3600 "user:123"

# Recuperar la sesión
GET sess:xyzabc
```

- Cachear el perfil de un candidato:

La clave es el ID del candidato y el valor es el documento JSON del perfil, serializado.

```
# Almacena el perfil del candidato 'cand_001' por 10 minutos (600 segundos)
SETEX cache:candidate:cand_001 600 '{"id": "cand_001", "nombre": "Ana Torres", "skills": ["SQL", "Python"]...}'

# Cuando la aplicación necesita el perfil de Ana, primero busca en Redis
GET cache:candidate:cand_001
# Si existe (cache hit), lo usa. Si no (cache miss), lo busca en MongoDB y luego lo guarda en Redis.
```

- Almacenar relaciones consultadas frecuentemente:

Podemos cachear las relaciones de un usuario, como sus mentores, para acceso rápido.

```
# Usando una estructura de Set para almacenar los mentores del usuario 'user_456'
SADD mentors:user_456 "mentor_A" "mentor_B"

# Obtener todos los mentores rápidamente
SMEMBERS mentors:user_456
```

### Estrategias Operacionales para Redis

- **Alta Disponibilidad (Redis Sentinel):** Para evitar que una caída del caché afecte a toda la plataforma, se implementará Redis Sentinel. Este sistema monitorea un clúster Maestro-Esclavo. Si el nodo maestro de Redis falla, Sentinel gestiona automáticamente la promoción de un esclavo a nuevo maestro y reconfigura la aplicación para que apunte al nuevo líder, minimizando el tiempo de inactividad.
- **Escalabilidad Horizontal (Redis Cluster):** Cuando la cantidad de datos en caché o el número de operaciones por segundo excedan la capacidad de un solo nodo, se migrará a Redis Cluster. Este modo particiona automáticamente el espacio de claves (keyspace) a través de múltiples nodos (sharding), permitiendo que el clúster crezca horizontalmente en capacidad de memoria y rendimiento.
- **Persistencia y Recuperación:** Aunque su función principal es en memoria, se configurará la persistencia AOF (Append Only File) con la política appendfsync everysec. Esto escribe cada comando en un fichero de log, ofreciendo un excelente equilibrio entre rendimiento y durabilidad. En caso de un reinicio completo del servicio, Redis puede reconstruir el estado en memoria a partir de este fichero, recuperando la mayor parte de los datos del caché.

# Sistema de Recomendaciones

## Base utilizada: Base de grafos (Neo4j)

El sistema de recomendaciones de Talentum+ está diseñado para generar sugerencias personalizadas basadas en intereses, cursos tomados y conexiones sociales dentro de la plataforma. Este componente exige un enfoque que priorice la eficiencia en la exploración de relaciones complejas entre entidades. Por esto, se eligió Neo4j como motor de grafos.

- Rutas de aprendizaje compartidas,
- Contactos en común,
- Intereses similares.
- Representa naturalmente relaciones entre candidatos, intereses, cursos y contactos.
- Es eficiente para consultas de caminos, intereses compartidos y colaboraciones.
- Flexibilidad para expandir relaciones y tipos de recomendaciones.
- Algoritmos nativos para recomendación y ranking.

### Ventajas de este enfoque

| Tecnología | Uso | Ventajas |
|---|---|---|
| Neo4j | Recomendaciones por relaciones sociales y formativas | ● Modelo natural para intereses, skills y contactos<br>● Recorrido de grafos para encontrar cursos similares<br>● Alta eficiencia en relaciones complejas |

**Modelo propuesto:**

Nodos

- Candidato
- Curso
- Skill
- Interés

Relaciones:

- (:Candidato)-**[:REALIZO]**->(:Curso)
- (:Curso)-**[:RELACIONADO_CON**]->(:Skill)
- (:Candidato)-**[:INTERESADO_EN]**->(:Skill)
- (:Candidato)**-[:CONOCE]**->(:Candidato)

**Ejemplos de consulta:**

- Relación de conocimiento basado en una relación de skill

```
(:Candidato)-[:CONOCE]->(:Candidato)
(:Candidato)-[:INTERESADO_EN]->(:Skill)

(:Candidato)-[:REALIZO {fecha: "2025-05-10"}]->(:Curso)
(:Curso)-[:RELACIONADO_CON]->(:Skill)
```

- Consulta: cursos que hicieron sus contactos, pero él no

```
MATCH (yo:Candidato {id: "cand_001"})-[:CONOCE]->(otro:Candidato)-[:REALIZO]->(curso:Curso)
 WHERE NOT (yo)-[:REALIZO]->(curso)
 RETURN DISTINCT curso
```

- Consulta: intereses compartidos

```
MATCH (yo:Candidato {id: "cand_001"})-[:INTERESADO_EN]->(skill)<-[:RELACIONADO_CON]-(curso:Curso)
 WHERE NOT (yo)-[:REALIZO]->(curso)
 RETURN DISTINCT curso
```

# Infraestructura y Escalabilidad

Para responder a las demandas de crecimiento de Talentum+, proponemos una arquitectura distribuida, multimodelo y resiliente, basada en los siguientes principios y tecnologías.

## Arquitectura Multimodelo y Desacoplada

Nuestra propuesta se basa en el principio de "usar la herramienta correcta para el trabajo", combinando diferentes motores de bases de datos especializados y comunicándolos de manera desacoplada.

Talentum+ adopta una **arquitectura shared nothing** ya que esta implica que **cada nodo del sistema tiene su propio almacenamiento, memoria y procesador**, y no comparte recursos físicos con otros nodos.

Esto aplica claramente a Talentum+ porque:

- MongoDB utiliza **sharding** y **replica sets**, donde cada shard/replSet opera en nodos separados.
- Neo4j usa **causal clustering**, con réplicas de lectura y un líder que coordina escrituras, todo distribuido por nodos sin disco compartido.
- Redis en modo cluster mantiene **cachés independientes replicadas**, cada una con su propia memoria y sin compartir recursos.

**Justificación por tipo de base de datos**

| Componente funcional | Motor elegido | Motivo |
|---|---|---|
| Candidatos y empleados | MongoDB | Perfiles flexibles, heterogéneos y con historial anidado. |
| Búsquedas y Ofertas | MongoDB | Almacenamiento maestro de las descripciones de ofertas. |
| Matching y Recomendación | Neo4j | Análisis de relaciones complejas, caminos y afinidad. |
| Procesos de Selección | MongoDB | Flujos de estado variables, con registros de auditoría y feedback. |
| Cursos y Trayectorias | MongoDB | Catálogos y progreso con relaciones de prerrequisitos (referencias). |
| Sesiones y Caché | redis | Acceso de alta velocidad a datos temporales y consultas frecuentes. |

## Componentes y motores

- **MongoDB:** se encarga de almacenar tanto perfiles de candidatos como ofertas laborales. Su modelo documental permite representar estructuras flexibles y dinámicas, incluyendo skills, certificaciones, experiencia laboral y requisitos de oferta.
- **Redis:** utilizado para gestionar sesiones activas, cachear consultas frecuentes y almacenar datos temporales con alta eficiencia.
- **Relaciones entre usuarios, mentores y equipos:** Estas conexiones se modelan en **MongoDB** mediante referencias cruzadas o subdocumentos anidados. **Redis** puede utilizarse para cachear relaciones consultadas con alta frecuencia, como vínculos de mentoring o membresía en equipos.
- **MongoDB** también permite almacenar evaluaciones, entrevistas, feedbacks y el historial completo del proceso de selección, utilizando subdocumentos anidados. Esto permite representar perfiles complejos y persistentes sin necesidad de joins, optimizando la consulta de trayectorias y desempeño histórico.

## Escalabilidad horizontal

- **MongoDB** soporta sharding automático y replica sets, permitiendo distribuir los datos entre múltiples nodos y balancear carga.
- **Redis** se implementa en modo clúster, permitiendo distribuir llaves entre nodos para escalar en lecturas y escrituras. Redis Sentinel proporciona alta disponibilidad mediante detección de fallos y failover automático.

**Alta disponibilidad y rendimiento**

- Réplicas en MongoDB garantizan disponibilidad y respaldo ante fallos de nodo.
- Redis TTL permite controlar el tiempo de vida de sesiones activas, liberando memoria automáticamente.
- MongoDB permite definir índices estratégicos para acelerar consultas sobre campos clave.
- Redis puede usarse con Pub/Sub para notificaciones en tiempo real o sistemas reactivos.

### Escalamiento Vertical vs. Horizontal

Talentum+ prioriza el **escalamiento horizontal**, utilizando:

- **Sharding en MongoDB** para particionar grandes volúmenes de perfiles, procesos y cursos entre nodos
- **Clustering en Neo4j** para balancear consultas complejas de grafos.
- **Redis en modo cluster** para cachear sesiones y recomendaciones con alta disponibilidad.

Esto permite crecer según demanda sin necesidad de detener el sistema.

## Flujo de Datos y Sincronización: Arquitectura Orientada a Eventos

Para mantener los datos consistentes entre los diferentes sistemas (ej. actualizar el grafo de Neo4j cuando un candidato añade un skill en MongoDB), se utilizará un Bus de Eventos como RabbitMQ o Apache Kafka.

**Flujo de Ejemplo:**

1. El servicio de perfiles recibe una petición para añadir "Python" al perfil de Ana Torres.
2. Actualiza el documento de Ana en MongoDB.
3. Tras confirmar la escritura, publica un evento candidate_skill_updated con { "candidateId": "cand_001", "skill": "Python" } en el bus de eventos.
4. El servicio de matching, que está suscrito a ese tema, consume el evento.
5. Ejecuta una consulta en Neo4j para crear la relación (:Candidato {id: 'cand_001'})-[:TIENE_SKILL]->(:Skill {nombre: 'Python'}).

Esta arquitectura garantiza el desacoplamiento y la resiliencia: si el servicio de matching está caído, los eventos se acumulan en la cola y se procesarán cuando vuelva a estar en línea.

## Estrategias de Disponibilidad, Escalado y Backup

| Tecnología | Estrategia de Alta Disponibilidad | Estrategia de Escalado | Estrategia de Backup |
|---|---|---|---|
| MongoDB | Replica Sets | Sharding (Clave Hashed/Compuesta) | Snapshots + mongodump para Point-in-Time Recovery |
| Neo4j | Causal Clustering (Nodos Core) | Read Réplicas | Backups online con neo4j-admin |
| Redis | Redis Sentinel | Redis Cluster | Persistencia AOF (Append-Only File) |

### Estrategia de Caching con Redis

Para optimizar la latencia y reducir la carga en las bases de datos principales, Redis se utilizará para:

- Gestión de Sesiones de Usuario: Almacenar tokens de sesión para una autenticación rápida.
- Caché de Consultas Frecuentes: Guardar resultados de búsquedas populares o perfiles muy visitados. Se usará un TTL (Time-To-Live) de 5-10 minutos para mantener los datos frescos.
- Rate Limiting: Controlar el número de peticiones por usuario para proteger el sistema.

### Seguridad y Privacidad

La protección de datos confidenciales es una prioridad.

- **Cifrado en Tránsito y en Reposo:** Se usará TLS para toda la comunicación y cifrado a nivel de disco para los datos almacenados.
- **Manejo de Datos Sensibles:** El feedback confidencial en los procesos de selección (procesos.entrevistas.feedback) se protegerá con Client-Side Field Level Encryption (CSFLE) de MongoDB. Esto cifra el dato en la aplicación antes de enviarlo a la base de datos, haciendo que sea ilegible incluso para un administrador con acceso a la BD.
- **Control de Acceso Basado en Roles (RBAC):** Cada microservicio se conectará a las bases de datos con usuarios que tengan los permisos mínimos necesarios para su función.

## Teorema de CAP

Talentum+ implementa una arquitectura distribuida que sigue el Teorema de CAP, el cual establece que en un sistema distribuido solo se pueden garantizar dos de estas tres propiedades al mismo tiempo: **Consistencia (C), Disponibilidad (A) y Tolerancia a particiones (P)**.

Cada base de datos utilizada en Talentum+ prioriza distintas combinaciones según sus funciones:

- **Neo4j (CP):** Para el sistema de matching, priorizamos **consistencia** y **tolerancia a particiones**, ya que es esencial que las relaciones entre candidatos, skills y ofertas estén siempre correctas antes de hacer recomendaciones. Se sacrifica disponibilidad en caso de fallos para garantizar precisión.

- **MongoDB (CP):** Para el almacenamiento de datos críticos como perfiles, procesos y certificaciones, priorizamos **consistencia** y **tolerancia a particiones**. La información debe estar correctamente sincronizada antes de ser confirmada, incluso si eso genera demoras ante fallos.

- **Redis (AP):** Para sesiones y caché priorizamos **alta disponibilidad** y **tolerancia a particiones**, asegurando respuestas rápidas, incluso si puede haber pequeños retrasos en la actualización. La consistencia no es crítica porque los datos se regeneran desde las bases persistentes.

Esta estrategia permite a Talentum+ ofrecer **precisión en lo importante** y **velocidad donde más se necesita**, combinando lo mejor de cada enfoque.

# Una Arquitectura Robusta y Evolutiva para el Futuro de Talentum+

La propuesta presentada aborda de manera integral el desafío de construir una plataforma de gestión de talento IT de nueva generación para Talentum+. Reconociendo la naturaleza heterogénea y relacional de los datos, así como los exigentes requisitos de escalabilidad, disponibilidad y rendimiento, hemos diseñado una **arquitectura distribuida** y **multimodelo** que se aleja de las soluciones monolíticas tradicionales para adoptar un enfoque especializado y de alto rendimiento.

La elección de una pila tecnológica compuesta por **MongoDB**, **Neo4j** y **Redis**, cada uno operando dentro de un ecosistema contenedorizado, no es arbitraria. Responde directamente a la necesidad de "utilizar la herramienta adecuada para cada trabajo":

**MongoDB** proporciona la **flexibilidad de esquema** necesaria para modelar los perfiles dinámicos de candidatos, los procesos de selección variables y los catálogos de cursos, permitiendo que la plataforma evolucione sin costosas migraciones.

**Neo4j** se erige como el motor de **matching y recomendaciones**, explotando su capacidad nativa para analizar relaciones complejas entre candidatos, skills, ofertas y contactos sociales, ofreciendo resultados que van más allá de simples filtros de palabras clave.

**Redis** actúa como el sistema nervioso de la plataforma, garantizando una e**xperiencia de usuario fluida e instantánea** a través de la gestión de sesiones y un cacheo inteligente que reduce la latencia y la carga sobre los sistemas persistentes.

Esta arquitectura, cohesionada mediante un **bus de eventos** para una comunicación asíncrona y desacoplada, está intrínsecamente diseñada para crecer. Las estrategias de **replicación (Replica Sets, Causal Clustering, Sentinel)** garantizan la alta disponibilidad, mientras que las de **escalado horizontal (Sharding, Read Replicas, Redis Cluster)** aseguran que Talentum+ pueda manejar un crecimiento exponencial de usuarios y datos sin sacrificar rendimiento. Además, las políticas de **backup, recuperación y seguridad** integradas en cada componente proporcionan la robustez y confiabilidad que una plataforma de misión crítica demanda.

En definitiva, esta propuesta no solo resuelve los desafíos técnicos actuales, sino que sienta las bases para un futuro innovador, permitiendo a Talentum+ anticiparse a las necesidades del negocio con agilidad, desde la implementación de dashboards analíticos y recomendaciones automáticas hasta la visualización de trayectorias profesionales complejas. Con esta arquitectura, Talentum+ estará equipado para convertirse en el líder indiscutible en la gestión del talento IT en Latinoamérica.