Universidad Argentina de la Empresa

Ingeniería de datos 2

TPO integrador - EDUSCALE - Sistema de admisión

Profesora: Joaquin Salas

Integrantes:

Deya, Sebastián - LU 1167157
Garcia Bosio, Alejo - LU 1167615
Sierra, Ignacio - LU 1152047

2025

REQUERIMIENTOS

REQUERIMIENTOS DEL CASO EDUSCALE

CONTEXTO Y VOLUMEN

- Soportar 1 millón de familias intentando inscribirse en una ventana de 2 horas
- Escritura masiva en pico

MODELO SAAS WHITE-LABEL

- Cada institución educativa (colegio, red, fundación) puede personalizar colores, logo, dominio y mensajes sin cambiar el core del sistema
- El frontend debe ser genérico pero tematizable
- Personalización mediante variables CSS y configuración por tenant

PERSISTENCIA POLÍGLOTA OBLIGATORIA

- Usar al menos 2 modelos NoSQL distintos
- Los datos deben modelarse explícitamente por tenant (institution_id como clave de particionamiento)

FRONTEND

- Debe ser SPA (React o Vue)
- Soporte multi-tenant
- Tematizable: temas, logos, textos, flujos ligeros por institución

BACKEND

- Arquitectura de microservicios o serverless
- Desacoplamiento entre frontend y validación

API PÚBLICA

- Arquitectura RESTful
- Autenticada mediante API Key
- Endpoints para consultar estado
- Al menos 2 endpoints obligatorios: GET /postulante/{id} y GET /inscripción/{id}
- Disponible para integración con sistemas internos (SiMS, ERP, Google Workspace)

INTEGRACIÓN Y CONSUMO

- Los datos deben estar disponibles mediante API RESTful segura
- Consumo por portales de gestión o futuros módulos del Data Lake

ANÁLISIS DEL DOMINIO: ENTIDADES, FLUJOS, VOLÚMENES Y PICOS (Basado en diagramas del CVA Fases A-C)

ENTIDADES PRINCIPALES IDENTIFICADAS

Entidades de Usuarios:

- Estudiante: aspirante a una beca en una institución educativa
- Administrador: personal del colegio que revisa expedientes
- Comité de Admisiones: grupo evaluador de candidatos

Entidades de Negocio - FASE A (Prospección):

- Registro Inicial: datos básicos del postulante (email, telefono, institución elegida)
- Session de Usuario: seguimiento de intentos de inscripción y estado
- Validación de Duplicados: control para evitar múltiples registros del mismo usuario

Entidades de Negocio - FASE B (Admisiones):

- Expediente de Admisión: documento central que agrega toda la información del candidato
- Documento Requerido: archivos que debe cargar el postulante (acta de nacimiento, comprobante académico, etc)
- Entrevista: entrevista personal con comité, puede ser presencial o virtual
- Evaluación: puntuación o resultado de cada entrevista
- Decisión del Comité: veredicto sobre aprobación, rechazo o lista de espera
- Resolución: resultado final de la admisión

Entidades de Negocio - FASE C (Inscripción y Alta):

- Matrícula: registro de inscripción del alumno aceptado

Entidades Transversales:

- Institución/Tenant: colegio o red educativa
- Configuración White-Label: temas, colores, logos, dominios por institución

FLUJOS DEL SISTEMA

Flujo FASE A - Prospección e Interés (0 a 2 horas en pico):

1. Usuario accede a portal white-label de institución
2. Ingresa datos básicos (nombre, email, telefono, apoderado)
3. Sistema valida no duplicado en Redis (rate limiting, estado en tiempo real)
4. Registro inicial se persiste en Redis con key institution_id:postulante_id
5. VOLUMEN EXTREMO: 1 millón de solicitudes en 2 horas = ~140,000 inscritos por minuto en pico

Flujo FASE B - Postulación y Admisiones (días/semanas posteriores):

1. Estudiante completa y envía formulario de interés
2. Documentos se almacenan en MongoDB (estructura jerárquica flexible)
3. Administrador revisa expediente completo (documentos + datos básicos)
4. Se agenda entrevista (si aplica)
5. Postulante asiste a entrevista o responde cuestionario
6. Comité evalúa: documentos + entrevista + antecedentes académicos
7. Se registra decisión (aprobado, rechazado, lista de espera)
8. Resultado se notifica a postulante
9. Expediente cambia de estado
10. VOLUMEN: ~200,000 expedientes en revisión simultáneamente (asumiendo conversión de FASE A)

Flujo FASE C - Inscripción y Alta (cuando se acepta):

1. Contrato se almacena en Cassandra con registro inmutable
2. Sistema inicia alta académica
3. Alumno queda visible en reportes y dashboards
4. VOLUMEN: ~50,000 inscripciones confirmadas (asumiendo ~5% de aceptación de 1M postulantes)

ENTIDADES DE CONTROL Y CONFIGURACIÓN

Tenant/Institución:

- institution_id (clave de particionamiento)
- nombre, dominio, email contacto
- configuración white-label (colores, logos, textos)

Configuración por Tenant (MongoDB):

- configuracion_tema (variables de colores, tipografía)
- logos (URL a almacenamiento externo)
- textos personalizados (bienvenida, instrucciones, mensajes)
- comité

VOLÚMENES Y PICOS

PICO MÁXIMO - FASE A (primer día hábil, 2 horas):

- Entrada esperada: 1,000,000 de inscripciones
- Tiempo: 2 horas = 120 minutos
- Promedio: 8,333 registros por minuto
- Pico máximo estimado (asimetría): 140,000 registros por minuto en peak de 10 minutos
- Operación dominante: ESCRITURA en Redis (validación, rate limiting)
- Lectura dominante: duplicados check en Redis
- Reserva de la transaccion en redis con TTL de 15 minutos

FASE B - Operación estable (dias a semanas):

- Expedientes en revisión simultánea: 200,000 (asumiendo ~20% de no-duplicados de FASE A)
- Lecturas diarias: documentos, evaluaciones (miles de accesos concurrentes)
- Escrituras diarias: comentarios, evaluaciones, decisiones (bajo volumen)
- Operación dominante: LECTURA en MongoDB (estructura flexible de expedientes)

FASE C - Baja concurrencia:

- Inscripciones confirmadas: 50,000 (asumiendo 25% tasa de aceptación)
- Escrituras inmutables en Cassandra: 50,000 registros
- Operación dominante: ESCRITURA inmutable en Cassandra

CARACTERÍSTICAS DE CARGA POR MODELO NOSQL

Redis (Clave-Valor):

- FASE A: escrituras masivas y hot (validación de duplicados, rate limiting)
- Patrón: institution_id:user_email → estado_inscripcion
- TTL: 24 horas para datos de sesión
- Baja latencia crítica para UX en pico extremo

MongoDB (Documental):

- FASE B: documentos con estructura flexible (PDF, entrevistas, comentarios jerárquicos)
- Patrón: expediente_id con arrays de documentos adjuntos y decisiones
- Lecturas frecuentes, escrituras moderadas
- Buscabilidad compleja (filtrar por estado, fecha entrevista, etc)

Cassandra (Tabular/Columnar):

- FASE C: escritura masiva concurrente con inmutabilidad
- Patrón: institution_id, alumno_id, fecha_inscripcion (partition key + clustering key)
- Time-series like: auditoría completa de cada inscripción

Redis:

- Desacoplamiento de FASE A frontal (respuesta rápida) de FASE B backend (validación lenta)
- Eventos: RegistroInicial, DocumentoCargado, DecisionComite, InscrpcionConfirmada
- Retry + Dead Letter Queue para fallos en integraciones con SiMS/ERP

WORKFLOW DE ADMISIONES (FASE B en detalle)

Estado 1 - Documentación:

- Postulante carga documentos requeridos
- Sistema valida tipos y tamaños

Estado 2 - Asignación a Entrevistador:

- Se agenda fecha

Estado 3 - Entrevista:

- Postulante asiste a entrevista (presencial o virtual link)

- Entrevistador registra puntuación/notas
- Estado: "ENTREVISTA" → "RECHAZADO/ACEPTADO"

Estado 4 - Revisión de Comité:

- Comité accede a expediente completo (documentos + entrevista)
- Realiza evaluación integral
- Vota: Aprobado / Rechazado
- Se registra decisión con fecha

Estado 5 - Resolución:

- Resultado se formaliza
- Expediente pasa a FASE C

JUSTIFICACIÓN DE VOLÚMENES

1 millón de postulantes en 2 horas = 500,000 inscriptores por hora

PERSISTENCIA POLÍGLOTA

REDIS - FASE A

Caso de uso: Registro inicial, validación duplicados, rate limiting (FASE A)

Por qué Redis:

- GET/SET en <5ms

- TTL automático (sesiones se expiran solas)

- INCR atómico para rate limiting

- Datos en RAM = latencia garantizada

MONGODB - FASE B

Caso de uso: Expediente de admisión, documentos, entrevistas, decisiones (FASE B)

Por qué MongoDB:

- Estructura flexible (documentos varían por institución)

- Búsquedas complejas: filtrar por estado + fecha

- Actualización parcial (agregar comentario sin reescribir todo)

- Arrays anidados sin 5+ JOINs

NEO4J - ANALYTICS

Caso de uso: Visualización de grafos, análisis de relaciones, patrones (ANALYTICS)

Por qué Neo4j:

- Modelo de grafos nativo (relaciones = ciudadanos de primera clase)

- Queries complejas de relaciones sin 5+ JOINs (MATCH en Cypher)

- Visualización interactiva: estudiantes-carreras-instituciones

- Análisis de patrones: "estudiantes con múltiples aplicaciones" en 1 query

CASSANDRA - FASE C

Caso de uso: Inscripción inmutable, auditoría

Por qué Cassandra:

- Write-optimized: 50k inscripciones concurrentes

- Append-only (sin UPDATE/DELETE) = auditoría garantizada

- Partición por institution_id = datos físicamente separados

- Time-series nativo: timestamp es clustering key

POR QUÉ NO UN SOLO MODELO

Solo Redis:

- ✗ 1M expedientes en RAM = costo infinito
- ✗ No soporta búsquedas complejas
- ✗ No persistencia a largo plazo

Solo MongoDB:

- ✗ FASE A degrada a 10-50ms (necesita <5ms)
- ✗ Permite UPDATE = riesgo legal en auditoría
- ✗ Memory overhead en configuración

Solo Cassandra:

- ✗ No soporta queries complejas (WHERE puntaje>7 AND estado='RESUELTO')
- ✗ Requiere schema fijo (inflexible para white-label)
- ✗ No optimizado para lectura de expedientes

Solo Neo4j:

- ✗ No optimizado para alta concurrencia de escritura (50k inscripciones)
- ✗ No tiene TTL automático para sesiones/cache
- ✗ Costo alto para almacenar documentos completos
- ✗ No time-series nativo como Cassandra

POLÍGLOTA: Cada modelo hace lo que mejor sabe

Redis = Rápido y simple (FASE A)

MongoDB = Flexible y buscable (FASE B)

Cassandra = Inmutable y auditables (FASE C)

FLUJO DEL DATO

Fase 1: Interés y Validación (El Escudo de Redis)

Todo comienza en el front-end, donde el postulante ingresa al portal de una universidad específica. Como muestra el primer diagrama de flujo, eligen un departamento y una carrera.

Esa información inicial (DNI, departamento_interes, carrera_interes) viaja primero a Redis, nuestro modelo en rojo. Redis actúa como un escudo de validación de altísima velocidad. Su única tarea en esta etapa es responder la pregunta "Ya existe postulación?" en microsegundos. Esto es crucial para manejar el pico masivo de 1 millón de usuarios y evitar que el sistema colapse con consultas duplicadas.

Fase 2: El Expediente Vivo (Mongo y Neo4j)

Si el DNI es nuevo, el usuario pasa al formulario web y completa sus datos personales (nombre, apellido, mail). En este punto, creamos el "expediente" principal del alumno en MongoDB, (el modelo verde MONGO-ESTUDIANTE).

Este documento en Mongo es el "expediente vivo" durante toda la Fase 2. Como muestra el diagrama de flujo, cada vez que el alumno avanza —'Análisis Datos', 'Asigna Comité', 'Entrevista' y 'Revisión'— su documento en Mongo se enriquece con nuevos datos: se le asigna un comité, se añade una fecha_entrevista, y finalmente una decisión y comentarios.

Paralelamente, cada cambio de estado se refleja en Neo4j (el modelo amarillo). Neo4j no guarda todo el expediente, sólo la relación. Así, podemos consultar en tiempo real el grafo de: (Estudiante 'Juan') -[APLICA_A {estado: 'en revisión'}]-> (Universidad 'X'). Esto nos da una visibilidad de las relaciones que Mongo o Cassandra no pueden.

Fase 3: El Registro Inmutable (Cassandra)

Una vez que la 'Revisión' termina, el flujo se bifurca. Si la decisión es "No", el proceso termina. Si es "Sí" (Aceptado), el alumno entra en la Fase 3: envía sus documentos (que actualizan el array documentos en Mongo) y completa la 'Inscripción'.

En ambos casos, ya sea un rechazo o una inscripción exitosa, el viaje del dato finaliza. En este punto, tomamos todos los datos relevantes del documento "vivo" de Mongo y los "aplanamos" para escribirlos una única vez en Cassandra (el modelo de tabla azul).

Cassandra se convierte en nuestro registro histórico, inmutable y de alta disponibilidad. Como se ve en el modelo, guarda la 'foto' final y permanente de la postulación (universidad_id, fecha_inscripcion, estudiante_id, dni, etc.), optimizado para auditoría y escritura masiva, cumpliendo así con el último requerimiento.

Diagrama

TEOREMA CAP

Fase A – Interés/Prospección (pico brutal de lecturas/escrituras cortas)

Redis (clave–valor) → AP: priorizamos disponibilidad y tolerancia a particiones para rate-limiting, detección de duplicados y locks efímeros. Se acepta consistencia eventual (los locks tienen TTL y luego se valida en fuente). Ventaja: latencia sub-ms aun con 1M de intents.

Trade-off: si hay partición, dos nodos podrían “ver” estados distintos por un rato; es aceptable porque la confirmación real ocurre luego en la base principal.

Fase B – Postulación/Admisiones (expediente, documentos, comité)

MongoDB (documental) → CP: priorizamos consistencia (esquemas validados, transacciones) y tolerancia a particiones; durante failover el cluster puede pausar escrituras (sacrifica algo de disponibilidad) para no corromper el expediente.

Neo4j (grafos): para relaciones y visualización del proceso, se usa de forma complementaria; si se cae, no bloquea la operación principal (sincronización asíncrona e idempotente).

Fase C – Inscripción/Alta y registro inmutable

Cassandra (wide-column) → AP con consistencia configurable: alta disponibilidad y throughput de escritura para el log inmutable de inscripciones/decisiones. Recomendación práctica: writes a QUORUM (equilibrio C/A) y reads a LOCAL_QUORUM/ONE según caso de uso de reporting. El diseño de PK particiona por institución y año para consultas previsibles y rápidas.

Trade-off: toleramos pequeñas ventanas de inconsistencia entre réplicas porque son datos históricos append-only y no impactan el flujo transaccional.

En síntesis: A=AP (velocidad y disponibilidad con validación posterior), B=CP (expediente consistente ante todo), C=AP (registro inmutable masivo con QUORUM para balance). Esto calza con los requerimientos de pico, multi-tenant y persistencia políglota del caso.