

Spring Data:

Persistencia en Sistemas Políglotas y Reactivos

Daniel García Costa

2026

¿Qué es una arquitectura políglota?

- Uso de **múltiples tecnologías de almacenamiento** en un mismo sistema
- Cada tecnología se utiliza según **sus fortalezas**
- No existe una única base de datos que resuelva todos los problemas

Hay que usar la tecnología adecuada para cada situación

Relacionales (SQL)

Tipo	Características principales	Casos de uso	Ejemplos
Relacional clásico	ACID, modelo tabular, integridad referencial	Aplicaciones empresariales, CRUD	PostgreSQL, MySQL
OLTP	Optimizado para muchas transacciones pequeñas	Sistemas operacionales (apps, APIs)	PostgreSQL, Oracle
OLAP / Data Warehouse	Optimizado para consultas analíticas complejas	BI, reporting, análisis de datos	Snowflake, Redshift
HTAP (híbrido)	Mezcla OLTP + OLAP en tiempo real	Sistemas con analítica en vivo	TiDB, Azure SQL HTAP
Distribuido	Escalado horizontal, replicación, particionado	Sistemas globales y escalables	CockroachDB, YugabyteDB

No Relacionales (NoSQL)

Tipo	Características principales	Casos de uso	Ejemplos
Documentales	Datos JSON/BSON, esquema flexible	APIs, datos semiestructurados	MongoDB, CouchDB
Clave-Valor	Acceso ultrarrápido por clave	Cache, sesiones, estado temporal	Redis, DynamoDB
Grafos	Relaciones complejas entre nodos	Redes sociales, recomendaciones	Neo4j
Columnar (Wide)	Columnas distribuidas, gran escalabilidad	Big Data, logs, eventos masivos	Cassandra, HBase
Búsqueda	Indexación y búsqueda full-text	Buscadores, analítica textual	Elasticsearch
Series temporales	Optimizadas para datos con timestamp	IoT, métricas, monitoring	InfluxDB, Timescale
Multi-modelo	Soportan varios modelos en un mismo motor	Sistemas híbridos complejos	ArangoDB, Cosmos DB

No hay una única solución óptima

Cada tipo resuelve un problema distinto

Por eso las arquitecturas modernas son **políglotas******

Complejidad

- Uso de múltiples tecnologías == mayor complejidad global
- Diferentes modelos de datos y APIs
- Necesidad de orquestar varios sistemas

Puntos de fallo

- Cada tecnología añade:
 - configuración propia
 - características específicas de despliegue
 - monitorización

Problemas más comunes

Consistencia de datos

- No hay transacciones distribuidas sencillas
- Posibles inconsistencias:
 - estado actualizado en un sistema pero no persistido en otro
 - fallos en mitad del proceso

Necesario diseñar tolerancia a fallos

Sincronización entre sistemas

- ¿Cuándo mover los datos?
 - en tiempo real
 - mediante eventos
 - mediante schedulers

El diseño del flujo de datos es crítico

Transformación de datos

- Cada sistema tiene su propio modelo:
 - entidades (Java)
 - DTOs
 - documentos (Mongo)
 - Key/Value (Redis)
 - ...

Dificultad de debugging

- Errores distribuidos entre servicios
- Fallos difíciles de rastrear:
 - llamadas remotas
 - errores de serialización
 - inconsistencias de datos

Conclusión

Mayor flexibilidad implica mayor responsabilidad

Caso práctico: Plataforma de Subastas

Problema

Diseñar un sistema capaz de:

- Gestionar subastas en tiempo real
- Registrar pujas concurrentes
- Mantener un histórico completo
- Escalar correctamente ante múltiples usuarios

Enfoque arquitectónico

Uso de una arquitectura **políglota**

- Diferentes tecnologías para distintos problemas
- Separación entre:
 - estado en vivo
 - persistencia histórica

Objetivo

Construir un sistema:

- escalable
- modular
- alineado con prácticas reales de arquitectura moderna

Tecnologías

- BD relacional -> productos, categorías, usuarios... (**integridad**)
- Redis -> estado activo de las subastas (**velocidad**)
- MongoDB -> almacenamiento histórico (**flexibilidad**)
- APIs REST -> integración entre servicios (**desacoplamiento**)
- Spring Data REST -> exponer repositorios como API (**agilidad**)

MongoDB (Base de datos documental)

Características principales

- Modelo basado en documentos (JSON / BSON)
- Esquema flexible (no fijo)
- Fácil evolución del modelo de datos
- Soporte natural para estructuras anidadas

Casos de uso

- Datos semiestructurados
- Históricos y auditoría
- Sistemas donde el modelo evoluciona
- APIs modernas (JSON-centric)

Ventajas

- Flexibilidad de diseño
- Alta productividad en desarrollo
- Modelo cercano al dominio de negocio
- Escalabilidad horizontal

Desventajas

- Menor control de integridad frente a SQL
- Redundancia de datos
- No ideal para transacciones complejas

Redis (Clave-Valor en memoria)

Características principales

- Base de datos en memoria
- Acceso extremadamente rápido (ms)
- Modelo clave-valor
- Soporte de TTL (expiración automática)
- Estructuras avanzadas (listas, sets, etc.)

Casos de uso

- Cache, Estado temporal, Sesiones, Sistemas en tiempo real

Ventajas

- Altísimo rendimiento
- Simplicidad de uso
- Ideal para datos efímeros
- TTL nativo

Desventajas

- Persistencia limitada (según configuración)
- No pensado como almacenamiento principal
- Consumo de memoria

Spring Data REST

¿Qué es?

- Componente de Spring que expone automáticamente repositorios como API REST
- Genera endpoints CRUD sin necesidad de controladores
- Basado en HATEOAS (Hypermedia)

¿Qué hace?

- Expone entidades directamente como recursos REST
- Genera endpoints como:
 - GET /api/items
 - GET /api/items/{id}
- Incluye enlaces (_links) para navegación entre recursos

¿Cuándo usarlo?

- Prototipos rápidos
- APIs CRUD simples
- Sistemas con bajo control de lógica de negocio
- Cuando se quiere reducir código boilerplate

¿Cuándo NO usarlo?

- Lógica de negocio compleja
- APIs públicas que requieren mayor personalización
- Necesidad de control fino sobre endpoints
- Cuando el modelo interno no debe exponerse directamente

Resolviendo los desafíos

No intentar evitar la complejidad, hay que hacerla explícita y controlarla

- Separación clara de responsabilidades
- Flujo de datos bien definido
- Uso de cada tecnología para lo que mejor sabe hacer

Diseño orientado a evitar acoplamientos innecesarios

Separación de responsabilidades

- **Relacional** -> garantiza integridad
- **Redis** -> estado en tiempo real
- **MongoDB** -> persistencia histórica
- **API** -> lógica de negocio

Beneficios

- Cada componente hace una sola cosa
- Menor complejidad interna por módulo
- Sistema más mantenible

Evitamos mezclar estado temporal con persistencia

Control del ciclo de vida de las subastas

Problema

- Redis elimina datos al expirar
- No podemos recuperar el estado después

Solución

Eliminamos dependencia de eventos internos

- Definimos el tiempo de vida de nuestros eventos
- Introducimos un **scheduler** para gestionar la persistencia

Flujo

1. Subasta activa en Redis
2. Scheduler detecta expiración
3. Persistencia en Mongo
4. Eliminación controlada

Control explícito del sistema

Transformación y enriquecimiento

Problema

- Datos distribuidos en varios servicios
- Representaciones incompletas

Solución

- Feign obtiene entidades base
- Resolución de relaciones
- Construcción de DTO completo (enriquecidos)

Reactividad

NO es necesaria, pero si conveniente

- Sistema con múltiples llamadas remotas
- Necesidad de gestionar concurrencia de forma eficiente
- Evitar bloqueos y mejorar rendimiento

Ventajas

- Mejor uso de recursos
- Alta capacidad de concurrencia
- Código expresivo para flujos complejos

Inconvenientes

- Mayor complejidad conceptual
- Manejo cuidadoso de errores
- Debugging más difícil

¿En qué casos merece la pena?

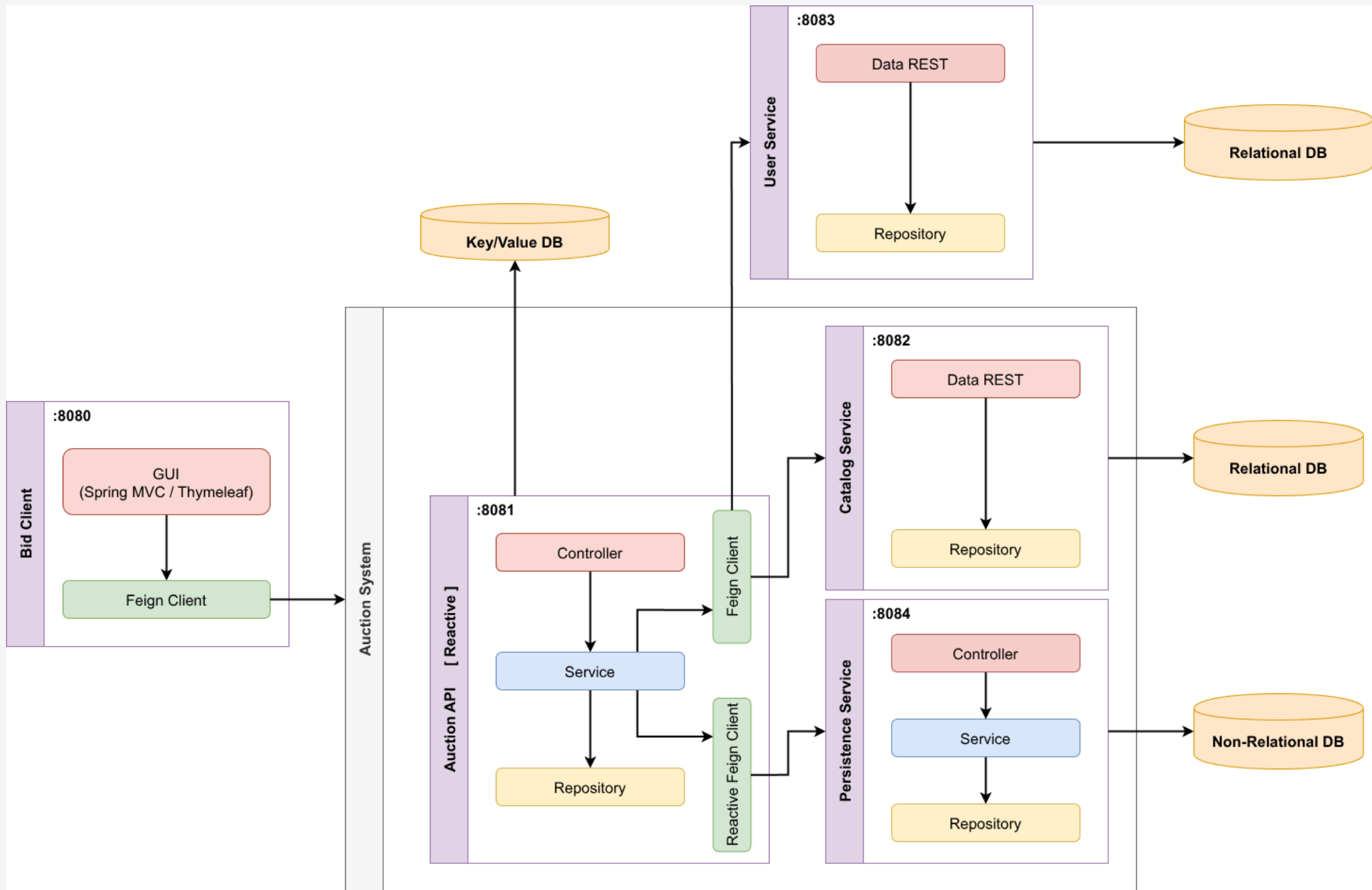
- Obtener datos
- Enriquecer información (usuario, categoría, etc.)
- Persistir resultado final

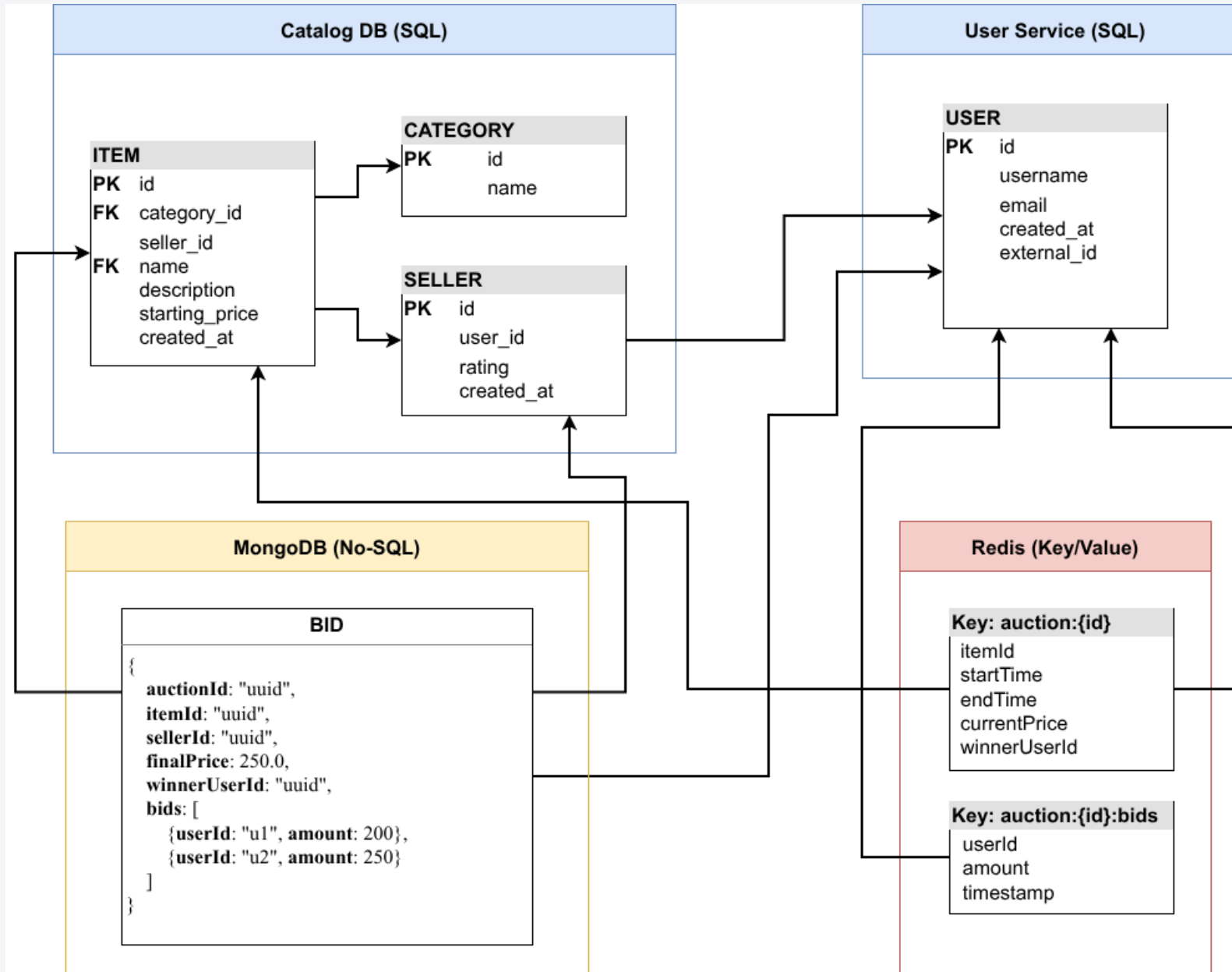
Todo en un pipeline no bloqueante

La reactividad es una forma distinta de modelar el flujo del sistema

Resultado

- Independencia de servicios externos
- Gestión de llamadas no bloqueante
- Datos listos para persistencia
 - Necesidad de integridad -> sistema relacional
 - Ciclo de vida -> Redis
 - Snapshot completo redundado -> Mongo





Todo esto está implementado en:

- **Auction API**
 - <https://inmaculados.uv.es/dagarcos/bidflow-auction-api>
- **Auction Catalog**
 - <https://inmaculados.uv.es/dagarcos/bidflow-auction-catalog>
- **Auction Persistence**
 - <https://inmaculados.uv.es/dagarcos/bidflow-auction-persistence>
- **Auction Users**
 - <https://inmaculados.uv.es/dagarcos/bidflow-users>
- **Bid Client**
 - <https://inmaculados.uv.es/dagarcos/bidflow-bid-client>

Vamos a ver algunas particularidades de la implementación