PROGETTO Distributed System

# AI RAG InPratica

Architettura Event-Driven per la trasformazione e la ricerca semantica su corpus documentali InPratica.

CANDIDATO
Lorenzo Leoni

Periodo
Dicembre 2025

# Il problema: risposte ai Quesiti

### Cosa sono i Quesiti e perchè nasce il progetto

I Quesiti sono un servizio offerto dall'azienda, molto apprezzato dai clienti, che consente di ottenere il parere di un esperto ad un costo contenuto su tematiche relative la nostra banca dati con risposte corredate da citazione e documentazione di riferimento.

### Problema

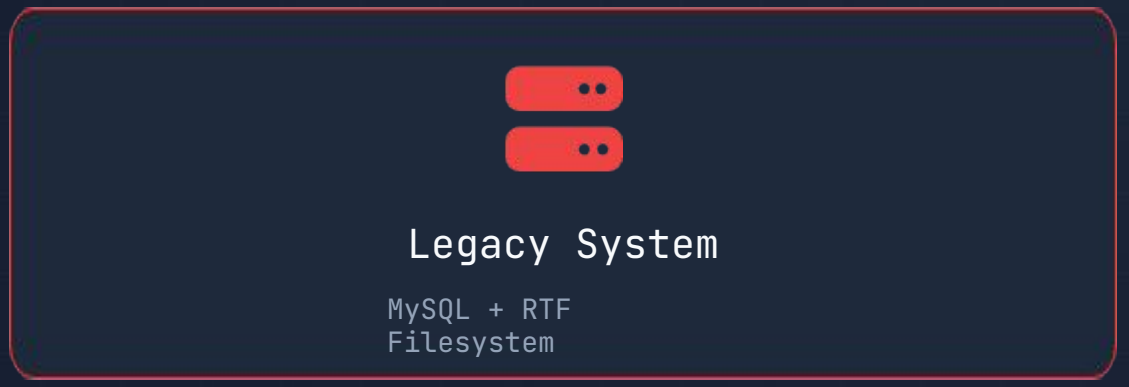
- Molto costosi per l'azienda
- Ricerca manuali dei documenti da parte degli autori e stesura della risposta
- Nessun controllo sull'utilizzo di AI in modo improprio

# Il Contesto: Da Legacy a Vettori

**Lo scenario di partenza**

Il sistema di ricerca esistente presenta sfide architetturali critiche per l'implementazione di funzionalità AI moderne:

- Nessun database di documenti formattati (composizione on-the-fly).
- Formato dati non strutturato (file RTF eterogenee e molto lunghi).
- Ricerca limitata al keyword matching esatto.



# Soluzione: Architettura Event-Driven

Il sistema è progettato come un layer di integrazione non invasivo, basato su 3 livelli distinti che comunicano asincronamente tramite **RabbitMQ**.

**Processing Layer (.NET)**

Responsabile dell'interfacciamento con il sistema Legacy. Esegue polling su MySQL e parsing intensivo dei file RTF.

**AI Layer (Python)**

Gestisce la logica RAG pura: generazione di embedding vettoriali e orchestrazione delle chiamate LLM (Claude).

**Infrastructure**

Persistenza: MongoDB (Documentale), Qdrant (Vettoriale) e RabbitMQ (Event Bus).

# Pipeline 1: sincronizzazione documenti

Questa pipeline gestisce la trasformazione del dato. È completamente disaccoppiata per non impattare le performance del sistema legacy.

## Step principali

- **Polling:** Rilevamento modifiche ogni 5 min.
- **Pubblicazione evento:** messaggio modifica su Rabbit
- **Transformation:** Conversione RTF -> Testo strutturato.
- **Pubblicazione evento:** messaggio conversione su Rabbit
- **Vectorization:** Generazione embedding 768-dim.

# Pipeline 1: Da RTF ai Vettori

### 1. Normalizzazione e Parsing(.NET)

Il DocumentProcessingService scarica i file RTF grezzi. Utilizzando LibreOffice headless, converte i file in testo, parsando la struttura gerarchica (titoli, paragrafi).

```
> Publish: TextProcessingEvent
```

### 2. Embedding (Python)

L'EmbeddingConsumer ascolta l'evento. Per ogni paragrafo, genera un vettore utilizzando un modello Transformer locale. I vettori vengono caricati su Qdrant associati ai metadati.

```
> Upsert: Qdrant Collection
```

# Pipeline 2: RAG

A differenza dell'ingestione, questo flusso è sincrono. Quando viene chiamato endpoint parte la pipeline di ricerca.

⚡Risposta in < 15 secondi.

**Step principali**

> **Classificazione query:** in albero di tassonomia
> **Retrieval:** ricerca semantica su Qdrant e filtri su metadati
> **Augmentation:** arricchimento contesto da Mongo con note citate per ogni paragrafo
> **Generazione bozza:** bozza di risposta con citazioni e possibilità di modifica
> **Generazione risposta:** risposta finale con stile Egaf.

# Interazione Utente: Due Workflow

Il sistema supporta due modalità di interazione distinte per adattarsi ai workflow esistenti senza forzare un cambio di abitudini.

### 1. Web App (Editoria)

Interfaccia dedicata per il team redazionale. Permette di inserire quesiti, revisionare le bozze generate dall'AI, correggere le citazioni e pubblicare la risposta ufficiale.

### 2. Ticketing (Autori)

Integrazione "invisibile" via Email. Quando un autore apre un ticket sul gestionale esistente, un servizio intercetta la richiesta e popola automaticamente la bozza di risposta.

# Web app - demo

Video demo

# Affidabilità in Produzione

**Idempotenza**

I consumer gestiscono messaggi duplicati senza corrompere i dati. Ogni operazione di scrittura (Upsert) è sicura anche se ripetuta.

**Dead Letter Queues**

I messaggi malformati (es. RTF corrotti) non bloccano la coda principale ma vengono spostati in DLQ per analisi manuale successiva.

**Replica Set**

MongoDB configurato in cluster (1 Primary, 2 Secondary). Garantisce continuità operativa anche in caso di crash di un nodo.

# Conclusioni

### ⚠ Sfide Affrontate

- **Parsing RTF Legacy:** Normalizzare documenti creati ”a mano” senza struttura fissa.
- **Sincronizzazione:** Mantenere allineati SQL, Mongo e Qdrant con latenza accettabile.
- **Orchestrazione servizi:** gestire ecosistema multi-tecnología

### Risultati Ottenuti

- Sistema in produzione con **1500+** documenti.
- Integrazione trasparente che non ha richiesto training per gli autori.
- Drastica riduzione del tempo di ricerca fonti.

# Fine

Grazie per l'attenzione

lorenzo.leoni5@studio.unibo.it