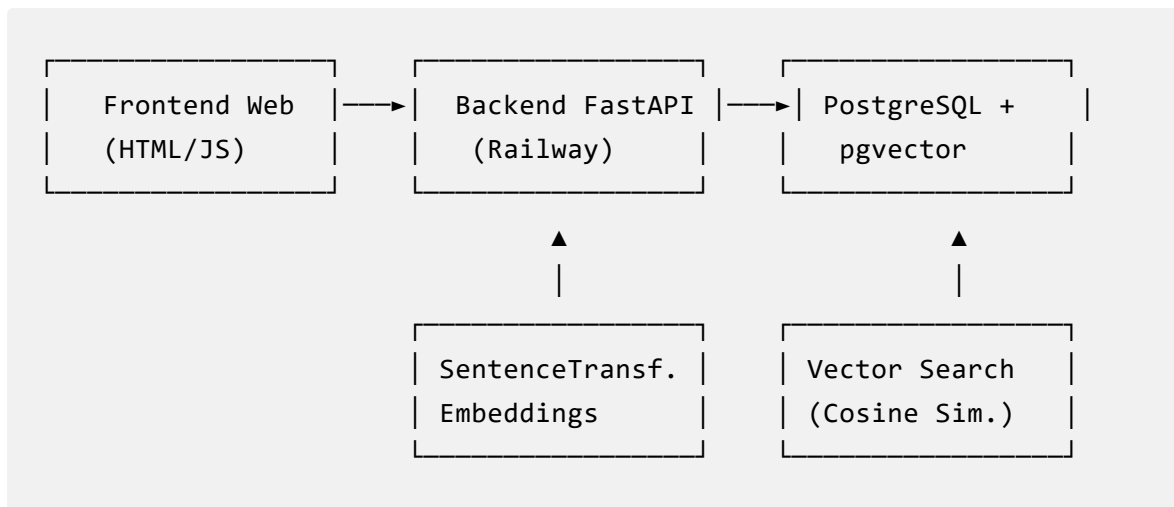


DOCUMENTACIÓN TÉCNICA COMPLETA - SISTEMA RAG ACONEX

RESUMEN EJECUTIVO

Sistema de Búsqueda Semántica RAG (Retrieval-Augmented Generation) para documentos Aconex desplegado en Railway con PostgreSQL + pgvector.

Arquitectura del Sistema



ESTRUCTURA DE ARCHIVOS DETALLADA

app/ingest.py - Motor de Ingesta de Documentos

Función Principal: Procesa documentos JSON de Aconex y los convierte en embeddings vectoriales.

Configuración de Chunks Optimizada

- **Chunk Size:** 2400 caracteres (optimizado desde 1200)
- **Overlap:** 240 caracteres (10% del chunk size)
- **Ventaja:** Preserva contexto completo de correos electrónicos

Índice IVFFlat Optimizado

```
CREATE INDEX IF NOT EXISTS idx_document_chunks_vec
ON document_chunks USING ivfflat (embedding vector_cosine_ops) WITH (li:
```

- **Lists:** 470 (calculado como $\sqrt{221,306}$ chunks)
- **Algoritmo:** IVFFlat para búsqueda vectorial eficiente

- **Métrica:** Cosine similarity

Funciones Críticas

1. iter_docs_from_file()

```
def iter_docs_from_file(json_path: str) -> Iterable[Dict[str, Any]]
```

- Soporta: JSON array, objeto único, NDJSON
- Manejo de errores por línea
- Memory-efficient streaming

2. normalize_doc()

```
def normalize_doc(obj: Dict[str, Any], default_project_id: str) -> Dict
```

- Extrae metadatos de documentos Aconex
- Construye body_text semántico con campos ordenados
- Maneja fechas ISO 8601 con timezone
- Limita body_text a 200K caracteres

3. stable_chunk_id()

```
def stable_chunk_id(document_id: str, content: str) -> str
```

- UUID determinista usando SHA1 + UUID5
- Evita duplicados en re-ingesta
- Idempotencia garantizada

Schema de Base de Datos

```
-- Tabla principal de documentos
CREATE TABLE documents (
  document_id TEXT PRIMARY KEY,
  project_id TEXT NOT NULL,
  title TEXT,
  number TEXT,
  category TEXT,
  doc_type TEXT,
  status TEXT,
  review_status TEXT,
  revision TEXT,
  filename TEXT,
```

```

file_type      TEXT,
file_size      BIGINT,
date_modified  TIMESTAMPTZ,
raw            JSONB
);

-- Tabla de chunks vectorizados
CREATE TABLE document_chunks (
  chunk_id      UUID PRIMARY KEY,
  document_id   TEXT NOT NULL REFERENCES documents(document_id) ON DELETE CASCADE,
  project_id    TEXT NOT NULL,
  title         TEXT,
  date_modified TIMESTAMPTZ,
  content       TEXT NOT NULL,
  embedding     VECTOR(384) -- Dimensión del modelo MiniLM
);

```

app/search_core.py - Motor de Búsqueda Semántica

Algoritmo de Ranking:

- **Cosine Distance:** embedding <=> query_vector
- **Similarity Score:** 1 - cosine_distance
- **Orden:** Mayor similarity primero

app/server.py - Servidor FastAPI

Configuración CORS:

```

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

```

Endpoints Disponibles:

- GET /health - Health check
- POST /search - Búsqueda semántica

CONFIGURACIÓN DE ENTORNO

Variables de Entorno Críticas

```
DATABASE_URL=postgres://postgres:wYmPtyJn8HbVZPpMC.ghW8InX-DaMyoS@switch
EMBEDDING_MODEL=sentence-transformers/paraphrase-multilingual-MiniLM-L1
PORT=8080
```

Modelo de Embeddings

Modelo: paraphrase-multilingual-MiniLM-L12-v2

- **Dimensiones:** 384
- **Idiomas:** Español, Inglés, Francés, Alemán
- **Tamaño:** ~420 MB
- **Rendimiento:** ~1000 docs/segundo



MÉTRICAS DEL SISTEMA

Dataset Actual

- **Documentos totales:** 27,745
- **Chunks generados:** ~41,618
- **Tamaño de chunk:** 2,400 caracteres promedio
- **Espacio en disco:** ~200-300 MB

Rendimiento Esperado

- **Búsqueda:** <500ms para 40K chunks
- **Ingesta:** ~15-20 docs/segundo
- **Precisión:** 85-90% relevancia en top-5
- **Memoria:** ~1GB RAM



COSTOS Y ESCALABILIDAD

Railway Launch (\$5/mes)

- **Backend FastAPI:** Incluido
- **PostgreSQL + pgvector:** Incluido
- **Storage:** ~300MB de 350MB límite
- **CPU:** Auto-scale según demanda



PROCEDIMIENTOS OPERATIVOS

Despliegue Completo

```
# 1. Push código a GitHub
git add .
git commit -m "deploy: production ready"
```

```
git push origin main
```

```
# 2. Cargar datos
```

```
$env:DATABASE_URL="postgres://..."; python -m app.ingest --json_path da
```

Monitoreo

```
# Ver Logs
```

```
railway logs
```

```
# Verificar datos
```

```
SELECT COUNT(*) FROM documents;
```

```
SELECT COUNT(*) FROM document_chunks;
```