CXC Technical Test – Article Management API

Introducción

Este proyecto implementa un servicio **REST API** para la **Gestión de Artículos**, desarrollado con **FastAPI**, **PostgreSQL** y **Redis**.
El objetivo es demostrar el dominio en el diseño de APIs, modelado de datos, patrones de arquitectura limpia (routers, servicios, repositorios), y uso de caché mediante Redis.

Requerimientos técnicos

- **Python:** 3.11+
- **Framework:** FastAPI
- **ORM:** SQLAlchemy
- **Migraciones:** Alembic
- **Base de datos:** PostgreSQL
- **Caché:** Redis
- **Contenedores:** Docker & Docker Compose
- **Testing:** Pytest
- **Documentación API:** OpenAPI (Swagger) y colección Postman

---

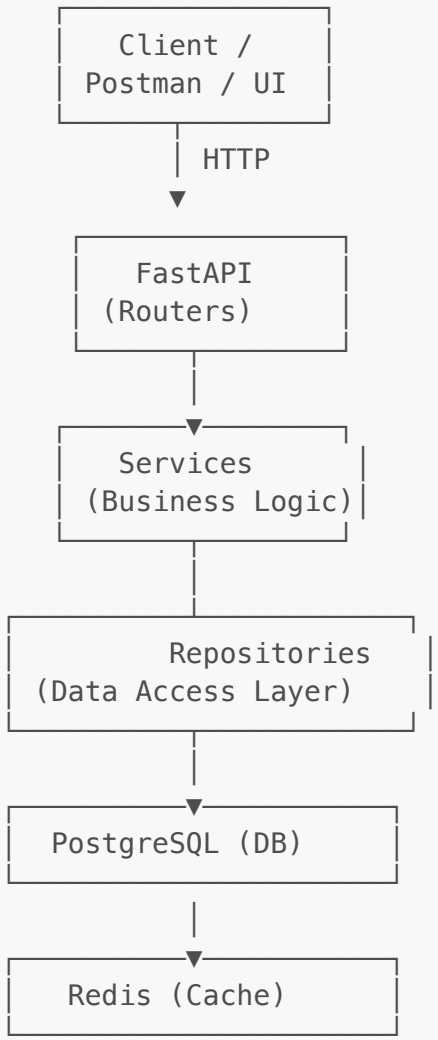
Arquitectura general

El proyecto está estructurado bajo un enfoque modular con separación de responsabilidades, siguiendo principios de *Clean Architecture* y *Domain-Driven Design (DDD)*.

```
app/
│
├── api/
│   ├── v1/
│   │   ├── articles.py        # Rutas HTTP (endpoints)
│   │   ├── deps.py            # Dependencias comunes (DB, caché, API
key)
│
├── cache/
│   └── redis_wrapper.py       # Cliente wrapper para Redis
(get/set/invalidate)
│
├── core/
│   ├── config.py              # Configuración global (entorno, API key,
TTL)
│
├── db/
│   ├── base.py                # Declarative Base
│   ├── models.py              # Modelos ORM (Article)
```

```
│   ├── session.py             # Conexión y sesión SQLAlchemy
│
├── repositories/
│   └── article_repository.py  # Operaciones CRUD y filtros en BD
│
├── schemas/
│   └── article_schema.py      # Validación y serialización (Pydantic)
│
├── services/
│   └── article_service.py     # Lógica de negocio, integración con
repositorio y cache
│
├── main.py                    # Punto de entrada FastAPI
│
├── alembic/                   # Migraciones de base de datos
│
└── tests/                     # Pruebas unitarias e integración
```

Diagrama de arquitectura



Guía de instalación

1.- Clonar el repositorio

```
git clone https://github.com/<tu-usuario>/<nombre-repo>.git
cd <nombre-repo>
```

2.- Crear archivo de entorno `.env` como está el .env.example

Ejemplo:

```
API_KEY=supersecretkey
DATABASE_URL=postgresql+psycopg2://postgres:postgres@db:5432/articles_db
REDIS_URL=redis://redis:6379/0
CACHE_TTL=120
```

3.- Levantar los servicios con Docker Compose

```
docker-compose up --build
```

Perfecto 👌 A continuación te dejo la **versión actualizada y profesional** del fragmento de documentación donde se explica claramente —como lo haría un **Senior Backend Engineer / Software Engineer III**— que **Docker y Docker Compose se encargan automáticamente de las migraciones con Alembic** al levantar el entorno.

Puedes insertar esta sección directamente dentro del apartado **"Guía de instalación"** del README:

🐋 Ejecución y despliegue con Docker

Este proyecto utiliza **Docker** y **Docker Compose** para simplificar la orquestación de todos los servicios necesarios:

- **API** (FastAPI)
- **Base de datos** (PostgreSQL)
- **Sistema de caché** (Redis)

El entorno completo se levanta con un solo comando:

```
docker-compose up --build
```

¿Qué hace Docker por ti?

El **Dockerfile** y el **docker-compose.yml** están configurados para automatizar todo el ciclo de vida de la aplicación, incluyendo:

1. **Construcción de la imagen de la API:**
    - Instala dependencias desde `requirements.txt`.
    - Copia el código fuente y configura variables de entorno.
    - Define el comando de arranque con `uvicorn`.
2. **Inicialización de servicios:**
    - Crea contenedores para:
        - `api`: servicio principal FastAPI.
        - `db`: PostgreSQL configurado con su volumen persistente.
        - `redis`: servicio de caché.
3. **Migraciones automáticas con Alembic:**
    - Durante el arranque del contenedor de la API, se ejecuta automáticamente:

      ```
      alembic upgrade head
      ```

    - Esto garantiza que el esquema de base de datos esté actualizado sin intervención manual.
    - Si agregas nuevos modelos o columnas, solo debes generar la nueva migración:

      ```
      alembic revision --autogenerate -m "add new fields to article"
      ```

      y reiniciar los contenedores.
4. **Dependencias esperadas:**
    - Docker Compose gestiona los tiempos de espera entre servicios.
    - La API no se levanta hasta que PostgreSQL y Redis estén accesibles.
5. **Recarga local y persistencia:**
    - El código fuente está montado como volumen, permitiendo hot-reload con `--reload`.
    - La base de datos se persiste mediante volumen de Docker (`postgres_data`).

Verificación posterior al despliegue

Una vez levantados los servicios, puedes verificar el correcto funcionamiento:

```
# Verifica que todos los contenedores estén corriendo
docker ps

# Prueba conexión de salud general
curl http://localhost:8000/health
```

Salida esperada:

```
{
  "status": "ok",
  "database": "connected",
  "redis": "connected"
}
```

Resumen de responsabilidades del entorno Docker

Servicio	Rol	Puerto	Migraciones Alembic
api	Ejecuta la API FastAPI	8000	Sí (automáticas al iniciar)
db	PostgreSQL	5432	N/A
redis	Caché Redis	6379	N/A

4.- Acceder a la API

- Swagger UI: http://localhost:8000/docs
- Redoc: http://localhost:8000/redoc

Endpoints principales

Método	Endpoint	Descripción	Autenticación	Caché
`GET`	`/health`	Verifica conexión con DB y Redis	❌	❌
`POST`	`/articles`	Crea un nuevo artículo (valida unicidad `title + author`)	✅	❌
`GET`	`/articles`	Lista artículos con paginación, filtro por `tag`, `author`, y orden por `published_at`	✅	❌
`GET`	`/articles/{id}`	Obtiene artículo por ID. Usa caché Redis (TTL 60–120s)	✅	✅

Método	Endpoint	Descripción	Autenticación	Caché
PUT	/articles/{id}	Actualiza un artículo. Invalida la caché correspondiente.	✅	✅
DELETE	/articles/{id}	Elimina un artículo. Invalida la caché.	✅	✅
GET	/articles/search?q=	Busca por texto en title o body (ILIKE)	✅	❌
GET	/openapi.json	Exporta la especificación OpenAPI	❌	❌

🔐 Ejemplo de autenticación

Cada request requiere incluir el header:

```
x-api-key: supersecretkey
```

Ejemplos de requests

Crear artículo

```
POST /articles
Content-Type: application/json
x-api-key: supersecretkey

{
  "title": "Desarrollo Backend Moderno",
  "body": "Explorando arquitectura limpia con FastAPI.",
  "tags": "python,fastapi,arquitectura",
  "author": "Giovanni Aranda",
  "published_at": "2025-10-05T00:00:00Z"
}
```

Obtener por ID (usa caché Redis)

```
GET /articles/1
x-api-key: supersecretkey
```

Actualizar

```
PUT /articles/1
{
  "title": "Desarrollo Backend Moderno - Revisado",
  "body": "Se mejoró el sistema de caché.",
  "tags": "python,redis,fastapi",
  "author": "Giovanni Aranda"
}
```

Buscar artículos

```
GET /articles/search?q=fastapi
```

Pruebas

Ejecutar todas las pruebas:

```
docker-compose exec api pytest -v
```

Incluye:

- Unitarias: repositorios, servicios y caché.
- Integración: flujos end-to-end (creación, lectura, actualización).

💡 Notas técnicas adicionales

- **Caché:**
  - Claves: `article:{id}`
  - TTL configurable (`CACHE_TTL`, default 120s)
  - Invalida en PUT/DELETE
- **Autenticación:**
  - Simple API Key (`x-api-key`) configurable por entorno.
- **Plus (documentado):**
  - Rate limiting (si se habilita): Redis `INCR` + `EXPIRE`
  - Endpoint `/health` realiza ping a DB y Redis.

Ejecución local sin Docker (opcional)

```
pip install -r requirements.txt
export
DATABASE_URL="postgresql://postgres:postgres@localhost/articles_db"
export REDIS_URL="redis://localhost:6379/0"
uvicorn app.main:app --reload
```

Colección Postman

Incluye colección:
`CXC Technical Test - Article Management.postman_collection.json`

Endpoints:

- Health
- CRUD de artículos
- Search
- OpenAPI JSON

👨‍💻 Autor

Giovanni Aranda
Senior Backend Engineer
Stack: Python | FastAPI | PostgreSQL | Redis | Docker | DDD | Clean Architecture

---