Postgres MCP + LangGraph Agent System (Guia en Espanol)

Esta guia complementa el README principal en ingles. Describe el sistema de asistencia para PostgreSQL construido con LangGraph, servidores MCP y una interfaz de chat en Streamlit.

Arquitectura rapida

- **Capa de Base de Datos**: `db.py` maneja conexiones async con `asyncpg`, pooling y operaciones de metadatos.
- **Servidor MCP**: `server4.py` expone herramientas sobre HTTP (incluye transporte SSE y streamable HTTP) y un health check.
- **Agentes**: `agent_langchain.py` (directo) y `agent_mcp_client.py` (cliente MCP) empaquetan herramientas en LangGraph.
- **UI**: `streamlit_chat.py` ofrece chat moderno con doble agente, temporizadores, exportacion y manejo de salidas de herramientas.

Flujo de datos

1. La UI recibe el mensaje del usuario. 2) Seleccion de agente (directo vs MCP). 3) Ejecucion de herramientas. 4) Operaciones en PostgreSQL. 5) Respuesta procesada y mostrada en la UI.

Archivos clave

- `db.py`: pool async, `list_tables`, `describe_table`, `get_table_sample`, `execute_sql`.
- `server4.py`: servidor FastMCP HTTP con herramientas y SSE; incluye `/health`.
- `agent_langchain.py`: agente LangGraph con herramientas directas (sin red).
- `agent_mcp_client.py`: agente LangGraph que descubre herramientas MCP via HTTP/SSE.
- `streamlit_chat.py`: UI Streamlit con controles avanzados y exportacion.
- `smoke_mcp.py`: pruebas de humo para el servidor MCP.

Configuracion

Variables de entorno (ver `.env.example`)

Base de datos:

```
PGHOST=localhost
PGPORT=5432
PGUSER=your_username
PGPASSWORD=your_password
PGDATABASE=your_database
PGSSL=false
PGPOOL_MIN_SIZE=1
PGPOOL_MAX_SIZE=10
PGPOOL_COMMAND_TIMEOUT=30
```

Servidor MCP:

```
POSTGRES_MCP_HOST=0.0.0.0
POSTGRES_MCP_PORT=8010
POSTGRES_MCP_PATH=/mcp
POSTGRES_MCP_URL=http://localhost:8010/mcp
```

AI/ML:

```
OPENAI_API_KEY=your_api_key
OPENAI_MODEL=gpt-4o-mini
```

Inicio rapido

1. Instalar deps: `pip install -r requirements.txt`.
2. Copiar `.env.example` a `.env` y ajustar credenciales/keys.
3. Levantar servidor MCP: `python -m postgres_gpt.server4` (health: `curl http://localhost:8010/health`).
4. Pruebas de humo (opcional): `python -m postgres_gpt.smoke_mcp`.
5. UI: `streamlit run postgres_gpt/streamlit_chat.py`.

Uso

- Explorar tablas: `list_tables`, `describe_table`, `get_table_sample` desde el chat.
- Consultas complejas: el agente arma SQL y ejecuta via `execute_sql`.
- Modos de agente: alternar entre directo (mas rapido) y cliente MCP (aislamiento/red).

Streamable HTTP vs SSE

- **Streamable HTTP**: mantiene request/response HTTP y envia chunks (chunked o HTTP/2) con tokens tempranos y un resumen final. Compatible con CDNs/proxies y permite payloads mixtos/binarios sin framing especial.
- **SSE**: `text/event-stream` simple, con `EventSource` y reintentos (`Last-Event-ID`). Ideal para feeds pequenos y solo texto.
- **Cuando usar cada uno**: elige streamable HTTP para respuestas multi-parte (deltas + resumen JSON) y compatibilidad de infraestructura; usa SSE para updates ligeros y cuando el cliente es solo navegador con `EventSource`.
- Referencia: Medium: Streamable HTTP vs SSE.

Monitoreo y debugging

- Logs en `agent_activity.log` con tiempos y errores.
- Health: `GET /health` en el servidor MCP.
- Metricas basicas: tiempos de consulta, uso de pool y pasos de razonamiento del agente.

Seguridad

- Usar SSL en produccion (`PGSSL=require`).
- Proteger `OPENAI_API_KEY` en entorno.
- Validar entradas SQL y limitar `execute_sql` en entornos no confiables.
- HTTPS y CORS apropiados segun despliegue.

Despliegue

- Local: todo en una maquina con PostgreSQL local.
- Contenedor: ver Dockerfile base en el README principal.
- Cloud: separar servidor MCP y UI; balanceador; PostgreSQL gestionado.

Contribucion

- Usar ramas de feature y PRs.
- Ejecutar `python -m postgres_gpt.smoke_mcp` antes de subir cambios.
- Mantener type hints, docstrings y logging estructurado.

Esta version en espanol es complementaria; el README original en ingles sigue siendo la fuente primaria y esta mas detallado.