# Documentación: Agente de IA para Base de Datos (Droguería)

## 1\. Resumen del Proyecto

Este proyecto es un \*\*asistente de chat inteligente\*\* que permite a un usuario hacer preguntas en lenguaje natural (como "¿cuántos productos de la marca X hay?") y obtener respuestas directamente desde la base de datos `drogueria4` de MySQL.

Está compuesto por tres partes principales:

1. Un \*\*Frontend\*\* (`index.html`): Una interfaz de chat bonita donde el usuario escribe.

2. Un \*\*Backend\*\* (`api.py`): Una API hecha en FastAPI que recibe las preguntas y entrega las respuestas.

3. Un \*\*Agente de IA\*\* (`agent.py`): El "cerebro" que traduce la pregunta del usuario a una consulta SQL, la ejecuta en la base de datos y luego traduce los resultados de vuelta a una respuesta amigable.

## 2\. Arquitectura (¿Cómo funciona?)

Para entender cómo funciona, imagina un restaurante:

1. \*\*El Cliente (Tú)\*\*: Eres el usuario en el `index.html` (el chat).

2. \*\*El Mesero (API)\*\*: Es el `api.py` (FastAPI). Toma tu pedido (la pregunta) y te trae la comida (la respuesta). No cocina, solo comunica.

3. \*\*El Chef (Agente)\*\*: Es el `agent.py`. Recibe el pedido del mesero. Es el experto que diseña el plato.

4. \*\*La Cocina (Herramientas)\*\*: El Chef tiene dos ayudantes:

\* \*\*El Traductor (`gemini.py`)\*\*: Ayuda al Chef a entender idiomas extraños (Lenguaje Natural $\leftrightarrow$ SQL).

\* \*\*El Asistente de Almacén (`mysql\_tool.py`)\*\*: Va a la bodega (la BD MySQL) a buscar los ingredientes (los datos).

5. \*\*La Bodega (Base de Datos)\*\*: Tu base de datos `drogueria4` en MySQL.

### Diagrama de Flujo

`index.html` $\rightarrow$ `api.py` (FastAPI) $\rightarrow$ `agent.py` (Agente) $\rightarrow$ `models/gemini.py` (para SQL) $\rightarrow$ `tools/mysql\_tool.py` (ejecuta SQL) $\rightarrow$ `Base de Datos MySQL`

Y el camino de regreso:

`Base de Datos MySQL` $\rightarrow$ `tools/mysql\_tool.py` (obtiene datos) $\rightarrow$ `agent.py` (Agente) $\rightarrow$ `models/gemini.py` (para respuesta natural) $\rightarrow$ `api.py` (FastAPI) $\rightarrow$ `index.html` (muestra la respuesta)

-----

## 3\. Componentes Detallados (Los Archivos)

### `index.html` (El Frontend)

\* \*\*Qué hace:\*\* Es un archivo único que contiene HTML (la estructura), CSS (los estilos bonitos del chat) y JavaScript (la lógica).

\* \*\*Función clave:\*\* Cuando envías un mensaje, el JavaScript usa la función `fetch` para enviar la pregunta al \*endpoint\* `/ask` de nuestra API. Luego, recibe la respuesta JSON y la muestra en el chat.

\* \*\*Truco especial:\*\* Modificamos el JavaScript para que reemplace los saltos de línea (`\n`) que envía el agente por etiquetas HTML (`<br>`), logrando que las listas se vean ordenadas.

### `api.py` (El Backend)

\* \*\*Qué hace:\*\* Es el servidor web que usa \*\*FastAPI\*\*. Es la puerta de entrada a nuestro agente.

\* \*\*Funciones clave:\*\*

\* `@asynccontextmanager (lifespan)`: Es una función muy importante. En lugar de crear un nuevo agente con cada pregunta (lo cual sería lento y costoso), crea \*\*una sola instancia\*\* (`agente\_global`) cuando la API se inicia. Esta instancia se reutiliza para todas las preguntas.

\* `CORSMiddleware`: Lo añadimos para solucionar un problema de seguridad del navegador (CORS), permitiendo que nuestro `index.html` pueda hacerle peticiones a la API aunque estén en "orígenes" diferentes (como `file://` y `http://127.0.0.1`).

\* `@app.post("/ask")`: Es el \*endpoint\* principal. Recibe el JSON (`{"question": "..."}`), se lo pasa al `agente\_global.ask()`, y devuelve la respuesta (`{"answer": "..."}`).

### `agent.py` (El Cerebro)

\* \*\*Qué hace:\*\* Es el orquestador. La clase `MCPAgent` es el corazón del proyecto.

\* \*\*Funciones clave:\*\*

\* `\_\_init\_\_`: El constructor. Inicializa el modelo (`GeminiModel`) y la herramienta de base de datos (`MySQLTool`).

\* `\_generate\_sql`: Esta es la \*\*primera magia\*\*. Toma la pregunta del usuario y el \*\*esquema\*\* de la base de datos (obtenido de `mysql\_tool.get\_schema()`) y le pide a Gemini que genere \*solo\* la consulta SQL.

\* `\_generate\_response`: Esta es la \*\*segunda magia\*\*. Toma la pregunta original, el SQL que se generó y los \*\*resultados\*\* de la base de datos, y le pide a Gemini que genere una respuesta amigable en texto plano.

\* \*\*Prompts (Instrucciones):\*\* Perfeccionamos los prompts en `\_generate\_response` para decirle a la IA:

1. "No uses NUNCA formato markdown (como \\*\\* o \\*)".

2. "Cuando presentes listas (...) usa guiones (-) y saltos de línea".

### `tools/mysql\_tool.py` (El Conector a BD)

\* \*\*Qué hace:\*\* Una clase dedicada a interactuar con MySQL.

\* \*\*Funciones clave:\*\*

\* `\_\_init\_\_`: Se conecta a la base de datos `drogueria4` usando las credenciales de `config.py`.

\* `get\_schema`: Una de las funciones \*\*más importantes\*\*. Se conecta a la BD y extrae la estructura de todas las tablas (columnas, tipos de datos, etc.). Esta información es la que el Agente le da a Gemini para que sepa qué tablas y columnas puede consultar.

\* `execute`: Toma el SQL generado por el agente y lo ejecuta de forma segura (solo permite consultas `SELECT`).

### `models/gemini.py` (El Traductor IA)

\* \*\*Qué hace:\*\* Es un "envoltorio" (wrapper) simple para la API de Google Gemini.

\* \*\*Función clave:\*\* `ask`: Recibe un prompt (una instrucción) y devuelve la respuesta de texto del modelo.

### `config.py` (La Configuración)

\* \*\*Qué hace:\*\* Almacena toda la información sensible y de configuración.

\* \*\*Contiene:\*\* `GEMINI\_API\_KEY` (tu clave secreta) y `MYSQL\_CONFIG` (host, usuario, contraseña, puerto y nombre de la base de datos).

-----

## 4\. Paso a Paso de Creación (Cómo lo Hicimos)

Así fue como construimos el proyecto, paso a paso:

1. \*\*Inicio (El Agente de Consola):\*\* Empezamos con un conjunto de archivos (`agent.py`, `config.py`, `main.py`, etc.). El `main.py` original era un script que se ejecutaba en la terminal (la consola).

2. \*\*Depuración Inicial:\*\* Nos encontramos con varios errores:

\* `ImportError` por `gimini.py` (un error de tipeo que corregimos a `gemini.py`).

\* `ImportError` por `DEFAULT\_DATABASE`. Esto pasaba porque el `main.py` original estaba pensado para SQLite y MySQL.

3. \*\*Enfoque en MySQL:\*\* Decidimos que solo usaríamos MySQL (`drogueria4`). Eliminamos la lógica de "datos quemados" (la tabla `empleados` de ejemplo) y modificamos `main.py` para que se conectara directamente a tu base de datos real.

4. \*\*El Salto a la Web (Creación de la API):\*\* Reemplazamos `main.py` por completo y creamos `api.py` usando \*\*FastAPI\*\*. Esto convirtió nuestro script de consola en un servidor web real.

5. \*\*Optimización (El `lifespan`):\*\* Creamos la función `lifespan` en `api.py` para asegurarnos de que el agente se cargara \*\*una sola vez\*\* al inicio y no con cada petición, mejorando la velocidad enormemente.

6. \*\*Creación del Frontend:\*\* Desarrollamos `index.html`. Usamos CSS para darle un aspecto de chat moderno y JavaScript (`fetch`) para conectarlo con el backend.

7. \*\*Solución de CORS:\*\* Al intentar conectar el chat con la API, nos topamos con un error de seguridad `CORS`. Lo solucionamos en `api.py` añadiendo `CORSMiddleware` para "darle permiso" al chat de hacerle peticiones a la API.

8. \*\*Refinamiento de Respuestas (Formato):\*\*

\* \*\*Problema 1:\*\* Las respuestas venían con `\*\*negrita\*\*` y `\*viñetas` (formato Markdown).

\* \*\*Solución 1:\*\* Modificamos el \*prompt\* en `agent.py` para ordenarle a la IA: "No uses NUNCA formato markdown".

\* \*\*Problema 2:\*\* Las listas salían en una sola línea de texto.

\* \*\*Solución 2:\*\* Fue un trabajo en dos partes:

1. Le dijimos al \*prompt\* de `agent.py` que usara "saltos de línea" para las listas.

2. Modificamos el JavaScript de `index.html` para que convirtiera esos saltos de línea (`\n`) en saltos de línea visibles (`<br>`).

## 5\. Cómo Ejecutar el Proyecto

1. \*\*Requisitos:\*\*

\* Tener Python instalado.

\* Tener tu servidor MySQL (ej: XAMPP, WAMP) corriendo.

\* Asegurarte de que la base de datos `drogueria4` exista y tenga tus tablas.

2. \*\*Instalación:\*\*

\* Abre una terminal en la carpeta del proyecto.

\* (Recomendado) Crea un entorno virtual: `python -m venv venv` y actívalo: `.\venv\Scripts\activate`.

\* Instala las dependencias:

```bash

pip install fastapi uvicorn google-generativeai mysql-connector-python

```

3. \*\*Configuración:\*\*

\* Abre `config.py`.

\* Pega tu `GEMINI\_API\_KEY`.

\* Verifica que los datos de `MYSQL\_CONFIG` (host, usuario, contraseña, puerto) sean correctos.

4. \*\*Ejecución:\*\*

\* \*\*Paso 1 (Iniciar el Backend):\*\* En tu terminal, ejecuta:

```bash

uvicorn api:app --reload

```

Verás que el servidor se inicia en `http://127.0.0.1:8000`.

\* \*\*Paso 2 (Abrir el Frontend):\*\* Ve a la carpeta de tu proyecto y simplemente \*\*haz doble clic en el archivo `index.html`\*\*. Se abrirá en tu navegador web.