Ficha Técnica — Módulo 6: API del Asistente Inteligente

Propósito del componente

La API del asistente inteligente actúa como la capa intermedia que conecta todos los módulos del sistema: el modelo generativo (Ollama), la lógica de intención (LangChain), la base de grafos (Neo4j) y la interfaz de interacción.
Su propósito es ofrecer un conjunto de endpoints claros y estables que permitan:

- Recibir consultas del usuario (tanto libres como guiadas),
- Comunicarse con el módulo generativo para interpretar la intención,
- Ejecutar consultas estructuradas en Neo4j,
- Devolver una respuesta clara y coherente en formato JSON,
- Facilitar el uso del sistema desde un frontend, una app o un script externo.

En términos simples, la API vuelve accesible el sistema experto a través de un servicio web.

Entradas

La API recibe diferentes tipos de entradas según el endpoint utilizado:
Entradas para consultas libres:

- texto del usuario ("¿El esquema Pedido cumple 2FN?")
- datos detectados por el LLM: esquema, forma normal, tipo de consulta

Entradas para el flujo guiado:

- nombre del esquema
- lista de atributos proporcionados por el usuario
- claves primarias
- dependencias funcionales declaradas paso a paso
- confirmaciones del usuario en lenguaje natural ("sí", "no", "agregar otra DF", etc.)

Salidas

La API produce respuestas en formato JSON, listas para ser consumidas por el frontend o el usuario final.
Ejemplos de salidas:

Para consultas libres:

- esquema evaluado
- forma normal solicitada
- intención detectada
- resultado (CUMPLE / NO_CUMPLE)
- motivos de violación
- explicación generada por el LLM

Para consultas teóricas:

- nombre de la FN
- lista de criterios
- explicación generada

Para el flujo guiado:

- estructura final del esquema creado
- confirmación de creación de nodos en Neo4j
- resultado de la evaluación
- explicación detallada

En todos los casos, la API devuelve respuestas coherentes y estructuradas.

Herramientas utilizadas y entorno

- Framework HTTP (dependiendo de la implementación: Flask/FastAPI o equivalente).
- Conexión con Ollama utilizando langchain_ollama.
- Conexión con Neo4j Aura mediante el driver oficial.
- Archivo .env para almacenar URI, usuario, contraseña y configuración del modelo.
- Pydantic models o validadores equivalentes para asegurar coherencia en parámetros.
- Formato JSON como estándar de comunicación.
- Ejecución local o despliegue en contenedor según necesidad.

Arquitectura o funcionamiento interno

La API funciona como un puente estructurado que recibe mensajes, los procesa y los devuelve transformados. El funcionamiento interno consta de tres flujos:

A. Flujo de consulta libre

1. El usuario envía un texto al endpoint de consulta.
2. La API pasa el texto al módulo generativo para que detecte intención y parámetros.
3. Se genera una consulta Cypher según intención y FN.
4. Se ejecuta sobre Neo4j.
5. El resultado se pasa nuevamente al LLM para generar una explicación final.
6. La API devuelve un JSON con el resultado final.

B. Flujo teórico

1. La API detecta que la intención es requisitos_fn.
2. Consulta en Neo4j los criterios que definen la FN.
3. El LLM transforma la información técnica en explicación natural.
4. Se retorna la respuesta final.

C. Flujo guiado de creación y evaluación de esquemas

La API interviene en pasos clave:

- recibe las respuestas parciales del usuario (“agrego un atributo”, “la PK es esta”, “hay una DF parcial”),
- normaliza esos textos mediante el LLM,
- convierte las respuestas en parámetros estructurados,
- llama a Neo4j para crear los nodos Atributo, Esquema y Dependencia Funcional,
- ejecuta la evaluación automática,
- devuelve el resultado final en formato JSON.

La API orquesta toda la interacción manteniendo el flujo coherente entre pasos.

Ver anexo: Capturas de API

Código relevante

El módulo incluye código que contiene:

- definición de endpoints
- validación de parámetros
- comunicación con el módulo de intención (route_query)
- creación de consultas Cypher a partir de parámetros
- manejo de errores
- generación de respuestas JSON

El proyecto está disponible en:
https://github.com/PauRodriguezz/EduDB-ProyectoIA

Ejemplo de instancia dentro del modelo conceptual

Ejemplo 1 — Consulta libre
Usuario: "¿Por qué el esquema Pedido no cumple la 2FN?"

- La API detecta intención estado_fn.
- Neo4j confirma dependencias parciales.
- LLM devuelve explicación detallada: "No cumple porque IDProducto y IDPedido determinan atributos que dependen solo parcialmente...".

Ejemplo 2 — Consulta teórica
Usuario: "¿Qué pide exactamente la 3FN?"

- Se recuperan criterios desde Neo4j.
- La API devuelve: requisitos, explicación del LLM y estructura clara.

Ejemplo 3 — Flujo guiado
Usuario carga: Atributos, PK, DF, confirmación de transitivas.

- La API arma nodos en Neo4j.
- Ejecuta el análisis final.
- Devuelve:
  "El esquema Factura NO cumple 3FN. Razón: presenta DF transitivas entre IDFactura → IDCliente → NombreCliente."

Ver anexo: Capturas de funcionamiento de API

Resultados obtenidos

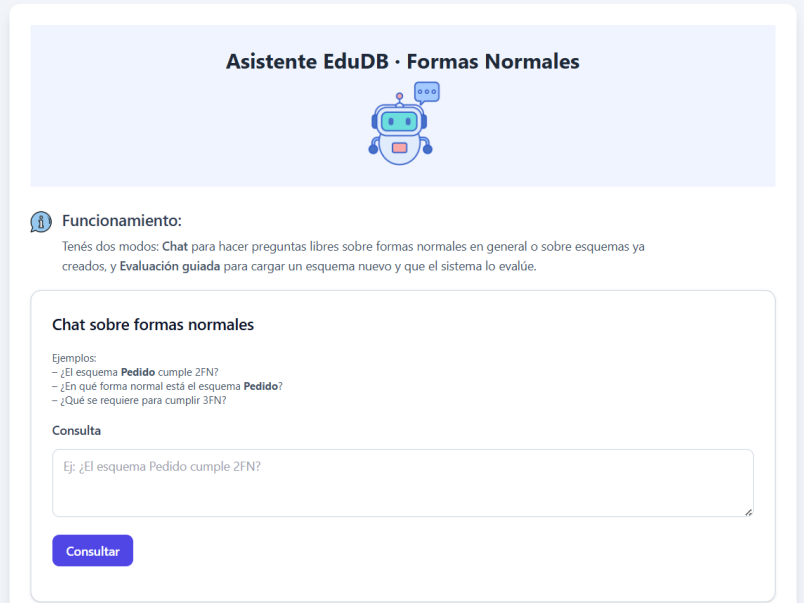
- Los endpoints respondieron correctamente con todos los escenarios probados.
- Las consultas libres generaron siempre parámetros coherentes y válidos.
- No se encontraron inconsistencias entre LLM, API y Neo4j.
- El flujo guiado funcionó sin pérdidas de información.
- La API permitió que la interacción sea fluida y natural.
- Los mensajes JSON fueron consistentes con la estructura predefinida.
- La evaluación final coincidió con sistemas tradicionales de normalización.
- Las explicaciones fueron claras para usuarios sin conocimiento técnico.

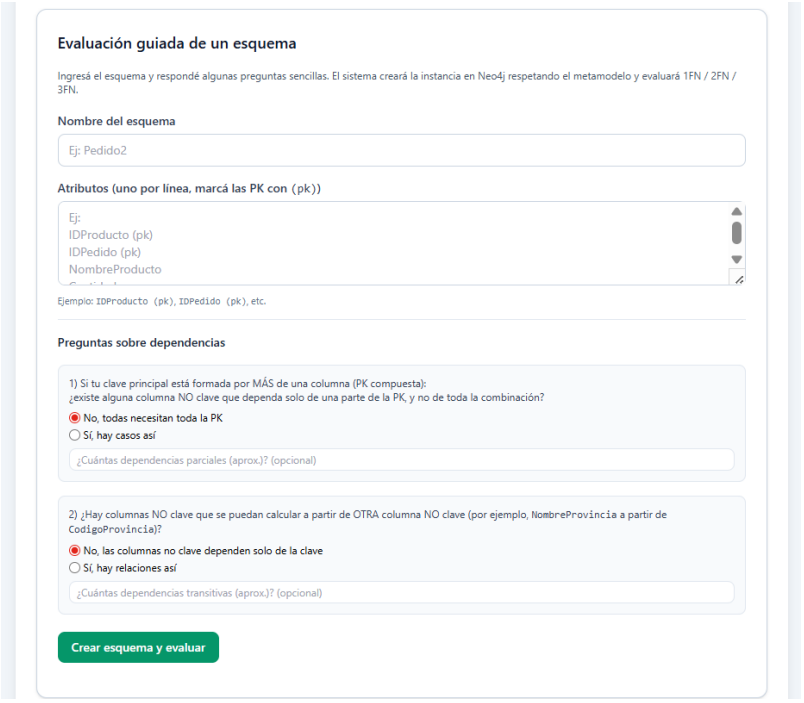
Observaciones y sugerencias

- Podría incluirse un endpoint para obtener la lista de esquemas disponibles.
- Sería útil implementar paginación y logs internos de solicitudes.
- Puede agregarse un modo "debug" para mostrar las consultas Cypher ejecutadas.
- A futuro, se podría permitir subir un archivo CSV para instanciar esquemas automáticamente.

Anexo

📌 Capturas de API





📌 Capturas de funcionamiento de API

Funcionamiento:

Tenés dos modos: **Chat** para hacer preguntas libres sobre formas normales en general o sobre esquemas ya creados, y **Evaluación guiada** para cargar un esquema nuevo y que el sistema lo evalúe.

Chat sobre formas normales

Ejemplos:
– ¿El esquema **Pedido** cumple 2FN?
– ¿En qué forma normal está el esquema **Pedido**?
– ¿Qué se requiere para cumplir 3FN?

Consulta

¿El esquema Pedido cumple 2FN?

Consultar

Esquema: Pedido
Forma normal: 2FN
Estado: NO_CUMPLE
Detalles NO_CUMPLE: `{"motivos":["Tiene dependencias parciales"],"parciales":2}`

Funcionamiento:

Tenés dos modos: **Chat** para hacer preguntas libres sobre formas normales en general o sobre esquemas ya creados, y **Evaluación guiada** para cargar un esquema nuevo y que el sistema lo evalúe.

Chat sobre formas normales

Ejemplos:
– ¿El esquema **Pedido** cumple 2FN?
– ¿En qué forma normal está el esquema **Pedido**?
– ¿Qué se requiere para cumplir 3FN?

Consulta

¿Qué se necesita para cumplir 3FN?

Consultar

Forma normal: 3FN

3FN exige que el esquema cumpla 2FN y que no existan dependencias transitivas: ningún atributo no clave puede depender de otro atributo no clave a través de una cadena de dependencias.

Crear esquema y evaluar

Esquema creado: Factura

1FN: CUMPLE
2FN: CUMPLE
3FN: NO CUMPLE

Detalle en Neo4j (CUMPLE / NO_CUMPLE):

Forma normal: 1FN
Estado: CUMPLE
`{"multival":0,"transitivas":2,"pk_compuesta":false,"parciales":0}`

Forma normal: 2FN
Estado: CUMPLE
`{"pk_compuesta":false,"parciales":0}`

Forma normal: 3FN
Estado: NO_CUMPLE
`{"transitivas":2,"motivo":"Tiene dependencias transitivas"}`