Memoria del Proyecto Grupo 2:

KAIRO - Asistente de Datos con GenAI

1. Introducción y Objetivo

KAIRO es un asistente inteligente de análisis de datos. En otras palabras, es un "Data Assistant" diseñado para democratizar el acceso a la información empresarial. Su objetivo es permitir que cualquier usuario consulte una base de datos PostgreSQL utilizando lenguaje natural, obteniendo no solo los datos crudos, sino también **visualizaciones automáticas** y **explicaciones narrativas**.

El proyecto surge ante una necesidad recurrente detectada en distintos clientes: la dificultad de acceder a la información almacenada en bases de datos corporativas sin depender constantemente de perfiles técnicos especializados. Esta situación genera cuellos de botella operativos en empresas, ralentiza la toma de decisiones y limita la autonomía de los equipos de negocio.

A diferencia de enfoques tradicionales, KAIRO utiliza **Llama 3 vía Groq** para razonar sobre la intención del usuario, lo que le permite entender preguntas complejas sin necesidad de mantener diccionarios de sinónimos manuales.

2. Arquitectura del Sistema

El sistema KAIRO se ha diseñado siguiendo el patrón de **microservicios**, donde cada componente tiene una responsabilidad única y se comunica con el resto mediante protocolos estándar (HTTP/JSON). Toda la solución está contenerizada con **Docker**, lo que garantiza que funcione igual en desarrollo y en producción.

- **Frontend React JavaScript:** Una *Single Page Application* (SPA) que actúa como interfaz de chat. Gestiona el estado de la conversación y renderiza dinámicamente tablas o gráficos según las instrucciones del backend, ofreciendo una experiencia fluida y reactiva.
- **Backend Flask API:** El núcleo orquestador. Funciona como un API Gateway seguro y *stateless* que valida las peticiones y coordina el flujo de datos entre el usuario, la inteligencia artificial y la base de datos.
- **Intelligence Engine Groq + Llama 3:** El "cerebro" del sistema. Aprovecha la velocidad de los chips LPU de Groq para realizar el razonamiento en tiempo real: detecta la intención del usuario, escribe el código SQL y narra los resultados en lenguaje natural.

- **Base de Datos (PostgreSQL):** El motor de almacenamiento relacional. Actúa como la "fuente de la verdad", garantizando la integridad de los datos y la precisión matemática en los cálculos.

3. El Corazón del Proyecto:

Prompt Engineering e IA (`app.py`)

En lugar de reglas gramaticales, KAIRO utiliza una **cadena de razonamiento (Chain of Thought)** dividida en tres fases manejadas por el backend:

A. Fase 1: Análisis de Intención (Router Semántico)

El sistema no ejecuta SQL a ciegas. Primero, utiliza el modo **JSON Mode** de Groq para clasificar qué quiere el usuario.

- **Input:** *"Grafícame las ventas por categoría"*
- **Prompt del Sistema:** Se le instruye para detectar si el usuario quiere datos (`"type": "data"`) o visualizaciones (`"type": "chart"`).
- **Output Estructurado:** El LLM devuelve un JSON limpio indicando incluso el tipo de gráfico recomendado (`bar`, `line`, `pie`).

B. Fase 2: Generación de SQL (Text-to-SQL)

Una vez entendida la intención, una segunda llamada al LLM genera el código SQL.

- **Contexto:** Se le inyecta el esquema de la base de datos (`DB_SCHEMA`) con las tablas `clientes` y `transacciones`.
- **Determinismo:** Se configura la temperature=0 para asegurar que el código generado sea preciso y reproducible, minimizando alucinaciones. Si le preguntas lo mismo 100 veces, te dará el mismo código SQL 100 veces. Esto es vital para evitar alucinaciones en consultas a bases de datos de empresas.

C. Fase 3: Respuesta Natural (Data-to-Text)

Finalmente, el sistema toma los datos crudos de la base de datos y utiliza el LLM para generar un resumen narrativo, explicando los hallazgos, basándose en los datos encontrados en la BBDD, al usuario de forma "amable y profesional".

---

4. Visualización frontend

El frontend está construido con **React** y React Router DOM, ofreciendo una interfaz dinámica y modular. Incluye sistemas de autenticación, rutas protegidas, dashboards personalizados según el rol del usuario y un módulo de chat interactivo. Para la visualización de información se utilizan **gráficos dinámicos con Chart.js**, y los estilos se gestionan mediante **Sass/SCSS,** facilitando un diseño consistente y reutilizable. La sesión del usuario se mantiene mediante **LocalStorage**, mejorando la experiencia de uso.

El backend se ha desarrollado con Node.js y Express, implementando una API REST encargada de la autenticación, la gestión de usuarios y la comunicación con la API externa de Data Science. La seguridad se refuerza mediante **JSON Web Tokens (JWT),** express-session y encriptación de contraseñas con bcrypt. El sistema de roles y permisos (Admin, Manager y Worker) controla el acceso a los distintos endpoints y funcionalidades

5. Implementación y Despliegue

La implementación de KAIRO se sustenta en tres pilares tecnológicos seleccionados estratégicamente para maximizar la velocidad, la precisión y la portabilidad del sistema.

A. Inferencia de Alto Rendimiento: Groq API & LPU

El núcleo de procesamiento en tiempo real se basa en **Groq**, una plataforma de inferencia que redefine la velocidad en la IA generativa.

- **Arquitectura LPU (Language Processing Unit):** A diferencia de las GPUs tradicionales diseñadas para el paralelismo de datos (gráficos), KAIRO utiliza el chip LPU de Groq. Este hardware está diseñado específicamente para la naturaleza secuencial del procesamiento de lenguaje natural (LLMs).
- **Ventaja Competitiva:** La LPU elimina los cuellos de botella de memoria habituales en las GPUs (HBM), ofreciendo una **inferencia de latencia determinista** y ultra-baja. Esto es crítico para la experiencia de usuario (UX) en un chatbot: la respuesta se genera casi instantáneamente ("token-per-second" masivo), haciendo que la conversación se sienta fluida y humana, en lugar de esperar segundos a que se "escriba" la respuesta.
- **Eficiencia:** Además de la velocidad, esta arquitectura ofrece una mayor eficiencia energética por token generado, alineando el proyecto con prácticas de *Green AI*.

B. Inteligencia Generativa: Llama-3.3-70b

Como cerebro cognitivo, se ha seleccionado el modelo **Llama 3** de Meta, servido a través de la API de Groq.

- **Razonamiento Complejo (SQL):** Con 70 mil millones de parámetros, este modelo posee una capacidad de razonamiento lógico superior a modelos más pequeños. Es

capaz de entender esquemas de bases de datos relacionales complejos y traducir preguntas ambiguas en lenguaje natural a consultas SQL sintácticamente perfectas (PostgreSQL).

- **Instruction Following (Formato JSON):** Una característica vital para este proyecto es su capacidad para seguir instrucciones estrictas de formato. El modelo garantiza que la salida sea siempre un objeto JSON válido (con campos `sql`, `type`, `chart_type`), lo cual es indispensable para que el backend (Flask) pueda procesar la respuesta programáticamente sin romperse.

C. Contenerización y Despliegue: Docker

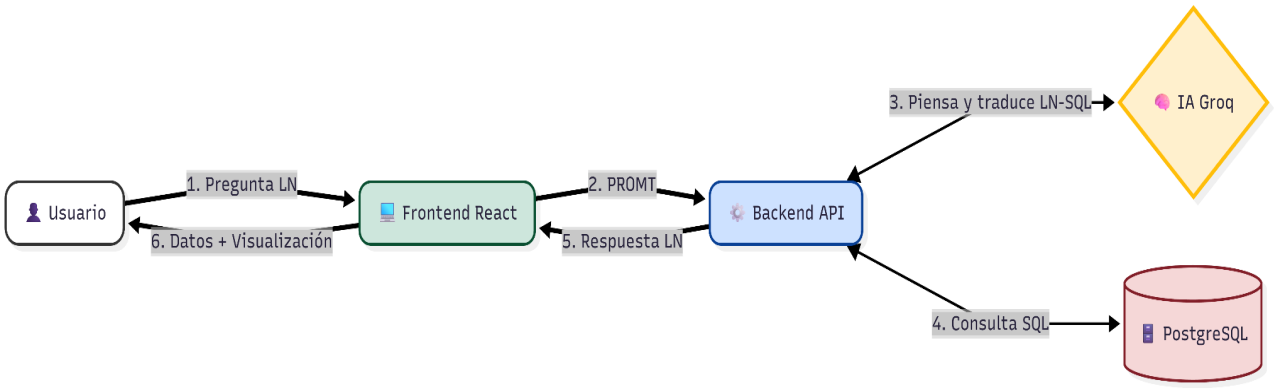
Para garantizar la reproducibilidad y escalabilidad del software, todo el backend ha sido contenerizado.

- **Entorno Ligero:** Se utiliza una imagen base `python:3.9-slim` en el `Dockerfile`. Esta versión minimalista de Linux/Python reduce drásticamente el tamaño de la imagen final y la superficie de ataque de seguridad, conteniendo solo las librerías necesarias (`flask`, `groq`, `psycopg2`).
- **Infraestructura como Código (IaC):** Gracias a Docker, el entorno de desarrollo local es idéntico al de producción. Esto elimina el clásico problema de *"en mi máquina funciona"*.
- **Despliegue en la Nube (Render):** El contenedor se despliega automáticamente en **Render**, una plataforma Cloud PaaS. Al estar dockerizado, el despliegue es agnóstico a la infraestructura subyacente, permitiendo migrar el servicio a AWS, Azure o Google Cloud sin cambiar ni una línea de código.

---

6. Diagrama de Flujo

Este diagrama muestra cómo KAIRO procesa una consulta desde que el usuario escribe hasta que ve el gráfico.



7. Ventajas Competitivas

1. **Flexibilidad Lingüística:** A diferencia de otros modelos basados en palabras clave o reglas, KAIRO entiende el contexto. Puede procesar "los productos que más lo petan" o "best sellers" sin necesidad de programar sinónimos explícitos.
2. **Capacidad Visual:** El sistema no solo extrae datos, sino que **decide** cuál es la mejor forma de visualizarlos (Barras, Líneas, Tarta) según la naturaleza de la pregunta.
3. **Velocidad Extrema:** Al usar Groq, la latencia de generación de SQL es mínima, ofreciendo una experiencia de usuario fluida.
4. **Resiliencia:** Si la consulta SQL falla o los datos son complejos, el asistente tiene la capacidad de explicar el error o resumir la información en lugar de simplemente fallar.

---

8. Posibles Mejoras y Roadmap Futuro

Aunque KAIRO es funcional y robusto, el desarrollo de un asistente de datos es un proceso iterativo. A continuación, se detallan las líneas de evolución propuestas para escalar el proyecto:

1. **Memoria Conversacional (Contexto):**
    - Implementar un historial de chat para que el usuario pueda hacer preguntas de seguimiento (ej: *"¿Y en 2024?"* o *"Fíltralo por ciudad"*) sin repetir toda la frase.
2. **Agente de Autocorrección (Self-Healing):**
    - Crear un bucle de mejora donde, si la consulta SQL falla, la IA analice el error de la base de datos y reescriba el código automáticamente hasta que funcione.

3. **Visualización Avanzada (Code Interpreter):**
    - Permitir que la IA genere y ejecute código Python real para crear gráficos complejos (mapas de calor, histogramas, scatter plots) en lugar de solo usar plantillas predefinidas.
4. **Escalabilidad con RAG:**
    - Utilizar *Retrieval-Augmented Generation* para soportar bases de datos con cientos de tablas, inyectando en el contexto solo las tablas relevantes para la pregunta actual.