Shift-Left AI: Inteligencia Artificial de Cero Latencia

Cuadernillo | Vol. 1

mercedev.es — 2026-05-02 | Fase 9 (Inteligencia y Autonomía)

El Desafío (Síntoma)

Queríamos que *Merci*, la asistente virtual de la web, ofreciera un saludo inteligente y contextualizado basado en el artículo exacto que el visitante está leyendo en la Biblioteca. La aproximación habitual en la industria es conectar el JavaScript del frontend (navegador) a la API de un LLM (Large Language Model - Modelo de Lenguaje Grande) como OpenAI o Google Gemini.

Sin embargo, esta arquitectura tradicional choca frontalmente con nuestras directrices fundacionales: 1. **Fuga de Secretos:** Exponer la API Key en el código cliente es un riesgo crítico de seguridad. 2. **Rendimiento (100/100):** La latencia de una petición de IA en tiempo real añade entre 2 y 5 segundos al *Time to Interactive* (Tiempo hasta ser Interactivo), destrozando la experiencia de usuario y las métricas *Core Web Vitals*.

La Maniobra (Lógica)

Aplicamos el principio de **Shift-Left AI** (mover la inteligencia hacia la izquierda del pipeline, a la fase de compilación).

1. **El Lóbulo Frontal en Python (`merci-brain.py`):** Desarrollamos un orquestador local que se ejecuta durante la fase de *Build*. Este script escanea los metadatos YAML de cada artículo Markdown de la Biblioteca y consulta a la API de Gemini (usando nuestra llave segura del entorno local) para generar un saludo a medida.
2. **Memoria Estática:** Las respuestas de la IA se compilan en un pequeño diccionario estático (`brain_data.json`) que se deposita en la carpeta pública.
3. **Consumo en Frontend:** El código Vanilla JS de *Merci* carga asíncronamente este JSON al iniciar la página. Si la ruta actual (`window.location.pathname`) coincide con una entrada del JSON, el asistente sobrescribe su saludo genérico con el pensamiento de la IA.

El Aprendizaje / Deuda Técnica

Autodescubrimiento de Modelos y Rate Limiting

Durante el desarrollo chocamos con dos grandes problemas de la API gratuita de Google (v1beta): los alias de los modelos cambian o desaparecen provocando errores 404, y los límites de velocidad (Rate Limits) son estrictos (ej. 20 peticiones/día o 5 peticiones/minuto).

Soluciones implementadas: - **Service Discovery (Autodescubrimiento):** En lugar de *hardcodear* el nombre del modelo (`gemini-1.5-flash`), el script interroga dinámicamente a Google sobre qué modelos están disponibles hoy y filtra mediante *subcadenas* (`1.5-flash`) para esquivar versiones experimentales inestables. - **Degradación Elegante (Graceful Degradation):** Si Google nos rechaza una petición por exceder la cuota (`HTTP 429`), el orquestador captura el error e inyecta una respuesta estática de contingencia (`[Fallback]`) en el JSON en lugar de colapsar. En la siguiente ejecución, el script intentará sobrescribir ese fallback con una respuesta real.

Conclusión

Al procesar la Inteligencia Artificial en el servidor seguro durante la compilación, dotamos a la interfaz web de capacidades generativas con un coste de red de **0 milisegundos**, 100% de seguridad y cero consumo de cuota de API en producción.