Taller práctico de IA para TI (Connect)
AI Incident Analyzer

Documento de guía — 28 January 2026

1. Descripción del taller

En este taller el equipo construye un mini-producto que demuestra cómo "producto-izar" un LLM: convertir texto (logs/incidentes) en un análisis estructurado y accionable, expuesto como un API, probado con Postman y consumido desde una web estática publicada en GitHub Pages.

Objetivo funcional

Dado un bloque de logs o un incidente (anonimizado), el sistema devuelve un JSON con:
• Resumen del evento
• Señales/patrones observados
• Hipótesis con confianza (0–100)
• Checks verificables para validar hipótesis
• Mitigaciones seguras (sin downtime)
• Criterios de escalamiento
• Preguntas faltantes para completar diagnóstico

Resultados esperados

- Un backend local (Node.js + Express) con endpoint POST /ai/analyze que llama a un LLM y devuelve JSON.
- Un cliente web estático (HTML/JS) para pegar logs y visualizar la respuesta.
- Pruebas funcionales del endpoint en Postman (incluye variación por temperatura y validación de errores).
- Comprensión práctica de: prompt (system vs user), temperatura, contrato de salida (JSON) y manejo de fallos.

Duración sugerida

90 minutos (puede adaptarse a 60 o 120).

Agenda sugerida (90 min)

0–10	Contexto y reglas de seguridad (no PII, no secretos).
10–35	Backend API: construir y levantar /health y /ai/analyze.
35–50	Postman: pruebas, errores típicos, temperatura 0.2 vs 0.8.
50–70	Frontend estático: UI simple que consume /ai/analyze.

70–85	GitHub Pages: publicar la carpeta web (vitrina).
85–90	Cierre: lecciones aprendidas y siguientes pasos (RAG, despliegue backend).

2. Requisitos y prechequeo

Herramientas

- Mac/Windows con Node.js 18+ (ideal 20+).
- VS Code instalado.
- Postman instalado.
- Cuenta de GitHub.
- Una API key válida de un proveedor LLM (OpenAI / Gemini / Claude). Esta guía usa OpenAI.

Reglas de seguridad (obligatorias)

- No pegar PII ni datos sensibles (nombres de clientes, teléfonos, placas, correos, identificaciones).
- No pegar tokens, passwords, service accounts ni secretos. Nunca.
- Usar logs ficticios o anonimizados. Si hay IDs, reemplazar por placeholders.
- El archivo .env NO se sube a GitHub.

Comprobación rápida

```
node -v
npm -v
```

3. Paso a paso técnico (desde cero)

3.1 Crear estructura del proyecto

En una terminal:

```
cd ~
rm -rf ai-incident-analyzer
mkdir ai-incident-analyzer
cd ai-incident-analyzer
mkdir server web
```

3.2 Backend (Node + Express) — CommonJS

Entrar a la carpeta server e inicializar el proyecto:

```
cd server
npm init -y
npm i express cors dotenv
```

Agregar script dev en package.json (sin editar manualmente):

```
npm pkg set scripts.dev="node index.js"
```

Crear el archivo de variables de entorno (.env):

```
touch .env
# editar .env con tu editor preferido
```

Contenido recomendado para .env (sin comillas, sin espacios alrededor del =):

```
PORT=3001
OPENAI_API_KEY=TU_KEY_REAL_AQUI
MODEL=gpt-4.1-mini
```

Crear index.js en server/ con el API:

```
const express = require("express");
const cors = require("cors");
const dotenv = require("dotenv");

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json({ limit: "1mb" }));

const PORT = process.env.PORT || 3001;
const MODEL = process.env.MODEL || "gpt-4.1-mini";

app.get("/health", (req, res) => res.json({ ok: true }));

app.post("/ai/analyze", async (req, res) => {
 try {
  const { text, temperature = 0.2 } = req.body || {};
  if (!text || typeof text !== "string") {
   return res.status(400).json({ error: "Missing 'text' (string)." });
  }
  if (!process.env.OPENAI_API_KEY) {
   return res.status(500).json({ error: "Missing OPENAI_API_KEY in .env" });
  }

  const system = [
   "Eres un Senior SRE/IT Ops en un entorno tipo Connect (operación 24/7).",
   "Reglas:",
   "- No inventes datos. Si falta evidencia, marca como 'hipótesis'.",
   "- Devuelve SOLO JSON válido (sin markdown, sin texto extra).",
   "- Prioriza acciones seguras (sin downtime) y checks verificables.",
```

```
    "- Incluye confidence 0-100 por hipótesis."
  ].join("\n");

  const schema = {
    summary: "string (máx 5 líneas)",
    signals: ["string"],
    hypotheses: [
      { title: "string", why: "string", checks: ["string"], confidence: 0 }
    ],
    safe_mitigations: ["string"],
    escalate_when: ["string"],
    questions_to_ask: ["string"]
  };

  const user = `LOGS / CONTEXTO:\n${text}\n\nDevuelve EXACTAMENTE este JSON
(misma estructura):\n${JSON.stringify(schema, null, 2)}`;

  const r = await fetch("https://api.openai.com/v1/responses", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "Authorization": `Bearer ${process.env.OPENAI_API_KEY}`
    },
    body: JSON.stringify({
      model: MODEL,
      temperature,
      input: [
        { role: "system", content: system },
        { role: "user", content: user }
      ]
    })
  });

  const rawText = await r.text();
  if (!r.ok) {
    return res.status(500).json({ error: "LLM call failed", details: rawText });
  }

  let data;
  try { data = JSON.parse(rawText); }
  catch { return res.status(500).json({ error: "Unexpected LLM response", details:
rawText }); }
```

```
  const outputText =
    data.output_text ||
    (Array.isArray(data.output)
      ? data.output.map(o => (o.content || []).map(c => c.text || "").join("")).join("")
      : "");

  let parsed;
  try { parsed = JSON.parse(outputText); }
  catch { return res.status(500).json({ error: "Model did not return valid JSON", raw:
outputText }); }

  return res.json(parsed);
 } catch (e) {
  return res.status(500).json({ error: "Server error", details: String(e) });
 }
});

app.listen(PORT, () => console.log(`Server running: http://localhost:${PORT}`));
```

Levantar el backend:

```
npm run dev
```

3.3 Pruebas con curl (sanity checks)

En otra terminal, validar salud:

```
curl http://localhost:3001/health
```

Probar análisis (ejemplo ficticio):

```
curl -X POST http://localhost:3001/ai/analyze   -H "Content-Type: application/json"   -d
'{"text":"2026-01-28 10:01:22 ERROR auth 401 token invalid\n2026-01-28 10:01:25 WARN
clock_skew=120s\nUsers report intermittent login failures.","temperature":0.2}'
```

3.4 Pruebas con Postman

Crear una colección con 2 requests:

- GET http://localhost:3001/health
- POST http://localhost:3001/ai/analyze (Body: raw JSON)

Body recomendado para /ai/analyze:

```
{
 "text": "Pega aquí logs anonimizados o un caso ficticio...",
 "temperature": 0.2
}
```

Ejercicio de comparación: repetir el mismo request con temperature=0.8 y observar diferencias (variabilidad y riesgo de suposiciones).

3.5 Frontend estático (web)

Crear el cliente web que consume el API:

```
cd ../web
# crear index.html (copiar/pegar el contenido)
# abrir en navegador o usar Live Server de VS Code
```

Contenido mínimo sugerido para web/index.html:

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <title>AI Incident Analyzer</title>
 <style>
  body { font-family: Arial, sans-serif; background:#f6f6f6; }
  .c { max-width: 900px; margin: 30px auto; background:#fff; padding: 18px; border-
radius: 12px; }
  textarea,input { width:100%; padding:10px; margin:6px 0 12px; }
  button { padding:10px 14px; cursor:pointer; }
  pre { background:#111; color:#0f0; padding:12px; border-radius:10px; overflow:auto; }
 </style>
</head>
<body>
 <div class="c">
  <h1>AI Incident Analyzer</h1>
  <label>API URL</label>
  <input id="apiUrl" value="http://localhost:3001" />
  <label>Temperatura</label>
  <input id="temp" type="number" min="0" max="1" step="0.1" value="0.2" />
  <label>Logs / Contexto (anonimizado)</label>
  <textarea id="text" rows="10"></textarea>
  <button id="run">Analizar</button>
  <pre id="out"></pre>
 </div>

 <script>
  const out = document.getElementById("out");
  document.getElementById("run").onclick = async () => {
   out.textContent = "Procesando...";
   const apiUrl = document.getElementById("apiUrl").value.trim();
   const temperature = Number(document.getElementById("temp").value);
```

```
    const text = document.getElementById("text").value;

    try {
      const r = await fetch(apiUrl + "/ai/analyze", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({ text, temperature })
      });
      const data = await r.json();
      out.textContent = JSON.stringify(data, null, 2);
    } catch (e) {
      out.textContent = "Error: " + e;
    }
  };
 </script>
</body>
</html>
```

3.6 Publicar la UI en GitHub Pages

Pages publica contenido estático. Publicaremos solo la carpeta web/. El backend queda local (en el taller) porque maneja secretos.

En la raíz del proyecto (ai-incident-analyzer/):

```
cd ..
cat > .gitignore << 'EOF'
server/.env
server/node_modules
.DS_Store
EOF

git init
git add .
git commit -m "AI Incident Analyzer demo"
# crear repo en GitHub y luego:
# git remote add origin <URL>
# git push -u origin main
```

En GitHub: Settings → Pages → Deploy from branch → main → /web. El sitio se publicará como vitrina del cliente web.

4. Guía de facilitación (qué observar)

Demostración 1: Control por prompt

Cambiar el texto del system prompt para exigir “checks verificables” o “citar evidencia” y comparar la salida.

Demostración 2: Temperatura

Ejecutar el mismo input con temperature=0.2 vs 0.8 en Postman y discutir consistencia vs creatividad.

Demostración 3: Anti-alucinación

Probar inputs incompletos y validar que el modelo produzca preguntas y marque hipótesis (no afirmaciones).

5. Troubleshooting (fallos típicos y solución)

Error 401 / invalid_api_key	La OPENAI_API_KEY es inválida o no está cargando desde .env. Revisar .env (sin comillas ni espacios).
fetch is not defined	Node < 18. Actualizar Node a 18+.
Model did not return valid JSON	Bajar temperature a 0.1–0.2 y/o endurecer instrucciones (solo JSON).
CORS error en navegador	Backend debe tener cors habilitado (ya incluido). Asegurar que el API URL sea correcto.

6. Extensiones (post-taller)

- Desplegar el backend en Cloud Run/Render/Fly.io y apuntar el frontend a esa URL (manteniendo secretos seguros).
- Agregar validación de esquema (Zod/JSON Schema) y reintentos cuando el modelo no devuelva JSON.
- Agregar RAG: una mini-KB de políticas/procedimientos internos y exigir citas a secciones de la KB.
- Registrar métricas: tiempo de respuesta, tasa de JSON inválido, top errores, etc.