# Bitácora de Implementación y Ejecución

## Sistema de Gestión de Riesgos para Activos Digitales

*(Proyecto de Auditoría – UPT | Abril 2025)*

---

# 1. Entorno de trabajo

| Componente | Versión | Observaciones |
|---|---|---|
| **SO** | Windows 10 Pro 64 bit | Local del auditor |
| **PowerShell** | 7.4 (pwsh) | Terminal integrada en VS Code |
| **VS Code** | 1.89 | Carpeta abierta: `D:\UPT\AUDITORÍA DE SISTEMAS\Examen U1\A analizar\AuditoriaRiesgos` |
| **Git** | 2.44 | Clonación del repositorio base |
| **Node.js** | 18.x | Instalado a nivel sistema |
| **Python** | 3.12 (venv `.venv`) | Activado con `\.venv\Scripts\activate` |
| **Ollama** | 0.6.6 | Instalador oficial – agrega ruta al PATH |

**Ruta raíz del proyecto** utilizada en todos los pasos:
`D:\UPT\AUDITORÍA DE SISTEMAS\Examen U1\A analizar\AuditoriaRiesgos`

---

# 2. Clonación del repositorio base

```
git clone https://github.com/OscarJimenezFlores/CursoAuditoria.git
cd CursoAuditoria\AuditoriaRiesgos   # se renombró lógicamente a
AuditoriaRiesgos
```

---

# 3. Instalaciones y configuraciones realizadas

## 3.1 Front-end

```
# Dentro de la raíz del proyecto
npm install           # descarga dependencias
npm run build         # genera carpeta dist/ (Vite + React 18 +
Ant-Design)
```

## 3.2 Backend (Flask)

```
python -m venv .venv
```

```
\.venv\Scripts\activate
pip install flask openai requests
```

**Ajustes de código**

- `app.py` ➜ Se añadió autenticación ficticia y rutas `/analizar-riesgos` y `/sugerir-tratamiento`.
- Se cambió la línea **model** → `model="llama3"` en funciones `obtener_riesgos()` y `obtener_tratamiento()`.

### 3.3 Ollama & modelo LLaMA 3

```
# Descarga del modelo
ollama pull llama3

# Ejecución permanente (puerto 11434 por defecto)
ollama run llama3     # terminal 1
```

Comprobación:

```
curl http://localhost:11434/v1/models  # -> 200 OK
```

---

# 4. Ejecución paso a paso

| Terminal | Comando | Ubicación |
|---|---|---|
| **T-1** | `ollama run llama3` | Raíz del proyecto (modelo) |
| **T-2** | `\.venv\Scripts\activate``python app.py` | Raíz (backend escuchando en **:5500**) |
| **T-3** | `python auto_eval.py` | Raíz (script automático) |

---

# 5. Script `auto_eval.py`

- Crea la lista de 5 activos (servidor BD, aplicación banca, firewall, VPN, BDD clientes).
- Envía POST → `/analizar-riesgos` y posteriormente a `/sugerir-tratamiento`.
- Guarda log en `resultados_riesgos.txt`.

Ubicación del archivo:
`D:\...\AuditoriaRiesgos\resultados_riesgos.txt`

---

# 6. Problemas y soluciones encontradas

| Incidencia | Síntoma | Solución |
|---|---|---|
| `ollama` no reconocido | *CommandNotFound* | Añadir `C:\Users\<user>\AppData\Local\Programs\Ollama` al PATH |
| `ModuleNotFoundError: requests / flask / openai` | Backend no inicia | `pip install <paquete>` dentro del venv |
| `openai.NotFoundError model "ramiro:instruct"` | 404 en endpoints | Cambiar a `model="llama3"` o `ollama pull modelo_personal` |
| Conexión rechazada a `localhost:5500` | `WinError 10061` en script | Iniciar **app.py** antes de `auto_eval.py` |

---

# 7. Resultado final

- **Backend** accesible en `http://localhost:5500/login` → autenticación `admin / 1234`.
- **Frontend** servido desde `dist/index.html` después del login.
- **Script automático** genera reporte con riesgos, impactos y tratamientos, almacenado en `resultados_riesgos.txt`.

---

# 8. Archivos clave del proyecto

```
AuditoriaRiesgos/
├─ app.py                  # servidor Flask + endpoints
├─ auto_eval.py            # script de evaluación automática
├─ package.json            # dependencias frontend
├─ vite.config.js          # configuración Vite
├─ src/                    # código React
├─ dist/                   # build generado por Vite
└─ resultados_riesgos.txt  # salida del análisis
```

---

# 9. Próximos pasos sugeridos

1. Diseñar plantilla PDF para exportar `resultados_riesgos.txt`.
2. Dockerizar backend + Ollama para despliegue sencillo.
3. Implementar control de sesiones y *logout* seguro.

# Anexos de Código

## Anexo A — `auto_eval.py`

Script en Python que automatiza el ciclo completo "analizar ➜ sugerir tratamiento" para los cinco activos solicitados y genera `resultados_riesgos.txt`.

```
"""auto_eval.py - Ejecución automática de análisis y tratamientos
ISO 27001
Autor: <tu nombre>
Fecha: abril 2025

Requisitos previos:
• Ollama ejecutando el modelo llama3 en http://localhost:11434
   $ ollama run llama3
• Backend Flask ejecutándose en http://localhost:5500
   $ python app.py
• Paquete Python requests instalado
   $ pip install requests
"""
import requests

URL_BACKEND = "http://localhost:5500"
ARCHIVO_SALIDA = "resultados_riesgos.txt"

activos = [
    {"activo": "Servidor de Base de Datos", "tipo": "Base de Datos"},
    {"activo": "Aplicación Web de Banca", "tipo": "Aplicación"},
    {"activo": "Firewall Perimetral", "tipo": "Seguridad"},
    {"activo": "VPN Corporativa", "tipo": "Infraestructura"},
    {"activo": "Base de Datos Clientes", "tipo": "Información"}
]

def analizar_y_tratar(activo: dict, fp):
    """Lanza los dos endpoints del backend y escribe resultados en
disco."""
    fp.write(f"\n🔍 Activo: {activo['activo']} ({activo['tipo']})\n")
    print(f"\n🔍 Analizando: {activo['activo']}")

    # 1) Analizar riesgos
    r = requests.post(f"{URL_BACKEND}/analizar-riesgos",
json={"activo": activo["activo"]})
    if r.status_code != 200:
        msg = f"❌ Error análisis: {r.text}"
        fp.write(msg + "\n")
        print(msg)
        return

    riesgos = r.json()["riesgos"]
    impactos = r.json()["impactos"]

    # 2) Para cada riesgo, pedir tratamiento
    for riesgo, impacto in zip(riesgos, impactos):
        fp.write(f"⚠️ Riesgo: {riesgo}\n➡️ Impacto: {impacto}\n")
        print(f"⚠️ Riesgo: {riesgo} — Impacto: {impacto}")
```

```
        t = requests.post(f"{URL_BACKEND}/sugerir-tratamiento", json={
            "activo": activo["activo"],
            "riesgo": riesgo,
            "impacto": impacto
        })
        if t.status_code == 200:
            tratamiento = t.json()["tratamiento"]
            fp.write(f"💊 Tratamiento: {tratamiento}\n\n")
            print(f"💊 Tratamiento: {tratamiento}\n")
        else:
            err = f"❌ Error tratamiento: {t.text}"
            fp.write(err + "\n\n")
            print(err)

if __name__ == "__main__":
    with open(ARCHIVO_SALIDA, "w", encoding="utf-8") as fp:
        fp.write("📋 RESULTADO DEL ANÁLISIS AUTOMATIZADO DE RIESGOS\n"
+ "="*50 + "\n")
        for act in activos:
            analizar_y_tratar(act, fp)
    print(f"\n☑ Archivo generado: {ARCHIVO_SALIDA}")
```

---

# Anexo B – Modificaciones en `app.py`

Solo se cambiaron las funciones que interactúan con Ollama para apuntar al modelo **`llama3`** (sin sufijo) que fue descargado con `ollama pull llama3`.

## 1. Cliente OpenAI (sin cambios, referencia a puerto 11434)

```
client = OpenAI(
    base_url="http://localhost:11434/v1",
    api_key="ollama"  # requerido pero ignorado por Ollama
)
```

## 2. Función `obtener_riesgos`

```
def obtener_riesgos(activo: str):
    response = client.chat.completions.create(
        model="llama3",   # <-- antes: "ramiro:instruct" /
"llama3:instruct"
        messages=[
            {"role": "system", "content": "Responde en español, eres
una herramienta para gestión de riesgos ISO 27001..."},
            {"role": "user", "content": "mi raspberry pi"},
            {"role": "assistant", "content": "• **Acceso no
autorizado**: ..."},
            {"role": "user", "content": activo}
        ]
    )
    texto = response.choices[0].message.content
    patron = r"\*\*\s*(.+?)\*\*:\s*(.+?)\.(?=\s*\n|\s*$)"
    pares = re.findall(patron, texto)
    riesgos  = [p[0] for p in pares]
    impactos = [p[1] for p in pares]
    return riesgos, impactos
```

## 3. Función `obtener_tratamiento`

```
def obtener_tratamiento(entrada: str):
    response = client.chat.completions.create(
        model="llama3",   # <-- antes: "ramiro:instruct" /
"llama3:instruct"
        messages=[
            {"role": "system", "content": "Responde en español, eres
una herramienta para gestión de riesgos ISO 27001..."},
            {"role": "user", "content": "mi telefono movil;Acceso no
autorizado;un atacante puede acceder..."},
            {"role": "assistant", "content": "Establecer un bloqueo de
la pantalla..."},
            {"role": "user", "content": entrada}
        ]
    )
    return response.choices[0].message.content
```

**Nota:** No hubo más cambios en rutas, lógica de sesión ni validaciones. El backend permanece escuchando en `http://0.0.0.0:5500`.

---

## Fin de Anexos