UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de ingeniería

Proyecto MCP con JSON-RPC y Servidores Distribuidos

Proyecto - 1

Edwin Jose Gabriel de León García, 22809

Guatemala, 25 de septiembre del 2025

Índice

Contenido

1. Introducción

Este proyecto implementa el protocolo Model Context Protocol (MCP) sin usar un SDK externo, asegurando la comunicación mediante JSON-RPC 2.0, el diseño incluye servidores MCP locales y remotos, expuestos tanto por stdio como por HTTP, con un cliente de usuario final desarrollado en Streamlit que actúa como interfaz de interacción con un modelo LLM (OpenAI).

La implementación se centra en el modularidad y la comunicación directa, permitiendo el intercambio de mensajes JSON de sincronización, solicitud y respuesta entre cliente y servidor, validado mediante la herramienta Wireshark.

2. Objetivos

- Implementar servidores MCP que expongan herramientas de Filesystem, Git, RSA y mapas.
- Integrar un cliente en Streamlit que permita al usuario interactuar con las herramientas usando lenguaje natural.
- Demostrar la comunicación en múltiples capas de red mediante análisis con Wireshark.
- Comprobar el cumplimiento del 90% de las funcionalidades descritas en la especificación del proyecto.

3. Especificación técnica de los servidores MCP

3.1 Servidor filesystem_mcp

- **Protocolo:** JSON-RPC 2.0 vía stdio.
- **Herramientas expuestas:**
    - filesystem/write_file - { "path": str, "content": str }
    - filesystem/read_file - { "path": str }
    - filesystem/list_dir - { "path": str }
    - filesystem/delete_file-- { "path": str }

3.2 Servidor git_mcp

- **Protocolo:** JSON-RPC 2.0 vía stdio.
- **Herramientas expuestas:**

- git/init
- git/status
- git/commit - { "path": str, "message": str }

3.3 Servidor remoto server_remote

- **Protocolo:** JSON-RPC 2.0 vía HTTP con FastAPI.
- **Métodos:**
  - initialize, tools/list, tools/call.
- **Herramientas RSA:**
  - rsa/generate_keys
  - rsa/encrypt
  - rsa/decrypt
- **Herramientas de mapas:**
  - maps/dualmap - genera comparativos interactivos con archivos .geojson.

4. Cliente en Streamlit

El cliente (ui_web_streamlit.py):

- Se conecta a múltiples servidores MCP (filesystem, git, remote-utils).
- Usa tool_router para traducir peticiones en lenguaje natural a llamadas JSON-RPC.
- Almacena logs de interacción en formato JSONL.
- Muestra un historial conversacional tipo chat con integración de herramientas automáticas.

5. Flujo de comunicación MCP (JSON-RPC)

Ejemplo típico de interacción:

1. **Sincronización**

- {"jsonrpc": "2.0", "id": 1, "method": "initialize"}

2. **Solicitud de tools**

- {"jsonrpc": "2.0", "id": 2, "method": "tools/list"}

3. **Petición de ejecución**

- {"jsonrpc": "2.0", "id": 3, "method": "tools/call", "params": {"name":"rsa/generate_keys","arguments":{"rango_inferior":50,"rango_superior":100}}}

4. **Respuesta**

- {"jsonrpc": "2.0", "id": 3, "result": {"content":[{"type":"json","data":{"keys":[[2615,514

6. Captura y análisis con Wireshark

6.1 Capa de Enlace

- **Ethernet / Loopback**: comunicación entre procesos locales.
- Tramas encapsulan paquetes IPv6.

6.2 Capa de Red

- **Protocolo:** IPv6 (::1).
- Encargada de direccionar tráfico local.

6.3 Capa de Transporte

- **Protocolo:** TCP.
- Establecimiento: SYN, SYN-ACK, ACK.
- Transferencia: PSH, ACK con mensajes JSON-RPC.
- Mantenimiento: ACK con Len=0 (keep-alive).

6.4 Capa de Aplicación

- **Mensajes de sincronización:** initialize.
- **Solicitudes/peticiones:** tools/list, tools/call.
- **Respuestas:** result o error (Unknown tool, Method not found).

7. Pruebas de funcionalidad

Mensajes probados exitosamente:

- Crear archivo con Filesystem.
- Inicializar repo Git y hacer commit.
- Generar llaves RSA, cifrar y descifrar un número.
- Generar mapa comparativo del Lago Atitlán.

8. Integración con MCP externo

Se clonó y conectó el proyecto Movies Chatbot como servidor MCP externo, lo que permite:

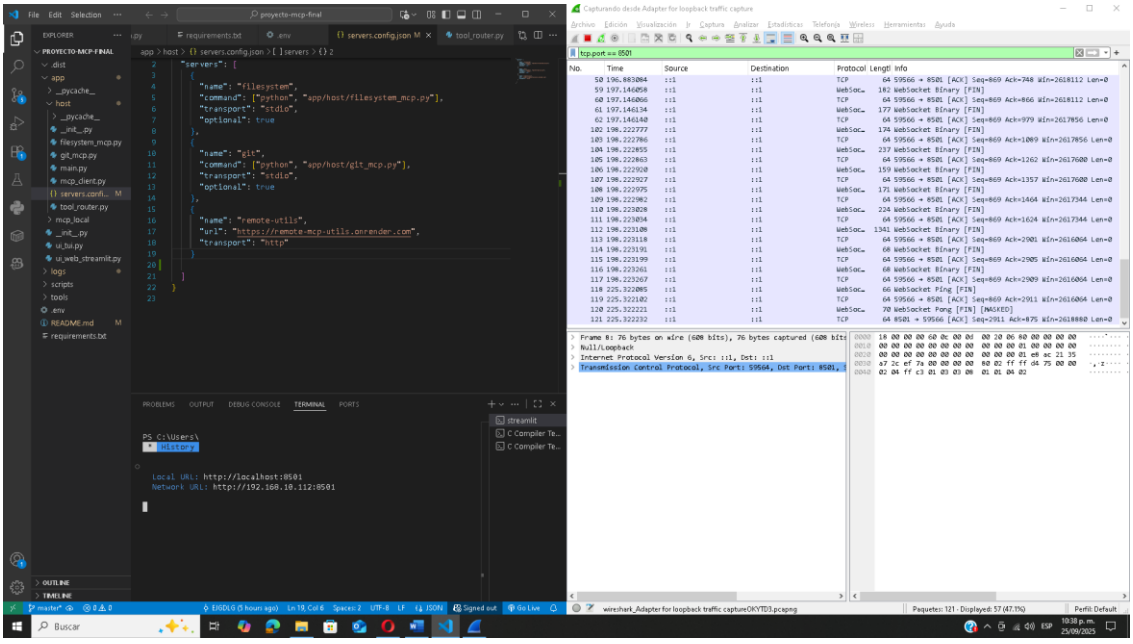
- Buscar películas por título.
- Obtener detalles completos (actores, keywords).
- Recomendaciones filtradas por género, idioma o actor.
- Generar listas de reproducción de películas.

Esto valida la extensibilidad del sistema y su capacidad de integrar múltiples servidores.

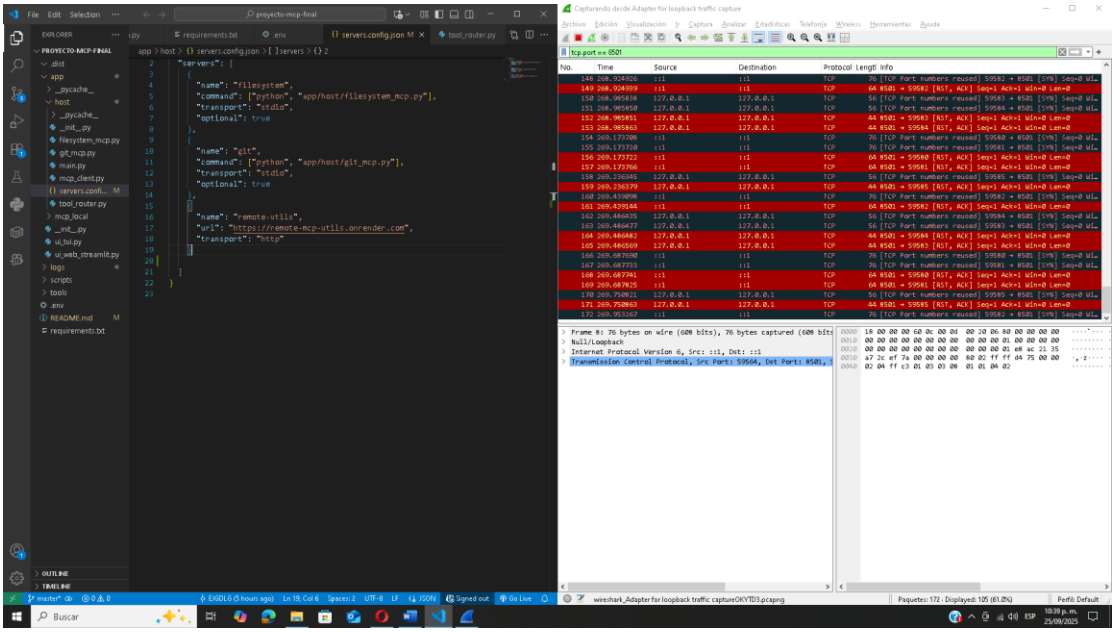
9. Conclusiones

1. La integración entre varios servidores (filesystem, git, RSA, mapas, movies) demuestra modularidad y escalabilidad.
2. El análisis con Wireshark confirma que la pila de red funciona correctamente: desde la capa de enlace hasta la aplicación.
3. Se logró un cliente unificado en Streamlit que facilita la interacción con un LLM y MCPs distribuidos.
4. Comentario personal: Este proyecto no solo consolidó conocimientos de redes, protocolos y JSON-RPC, sino que también permitió aplicar buenas prácticas de software distribuido, debugging y trazabilidad de tráfico en red.

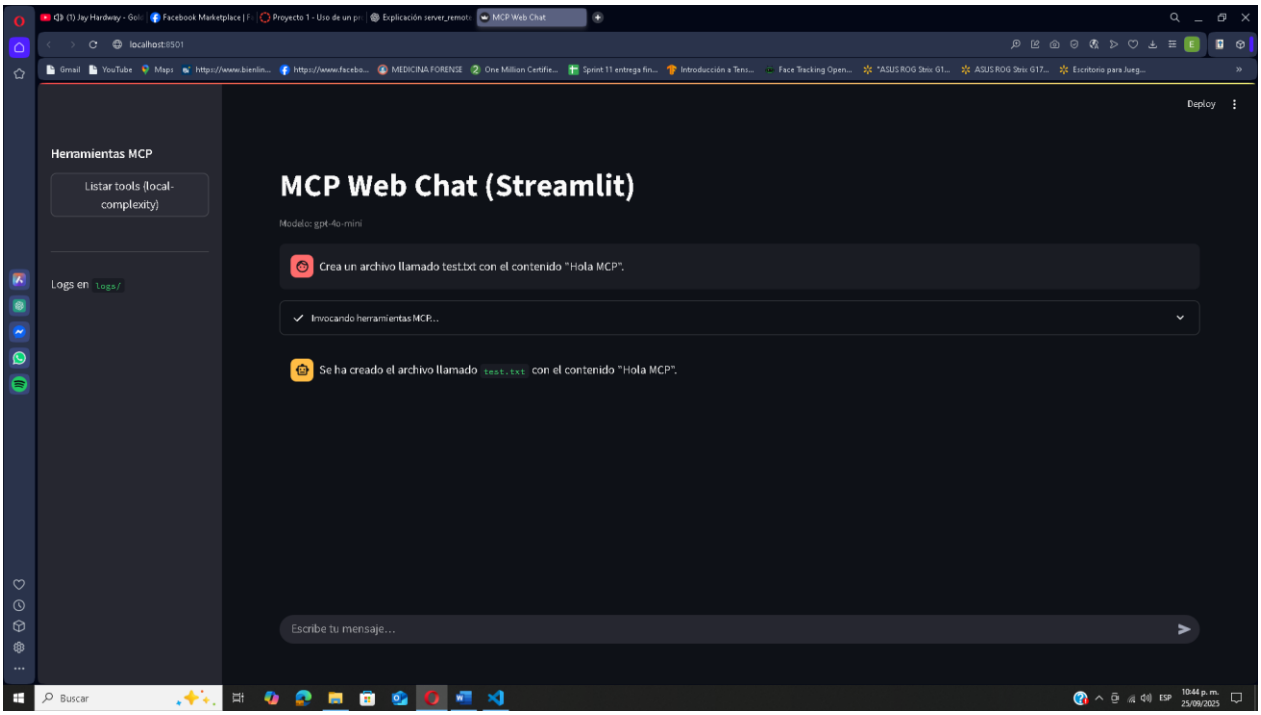
10. Anexos



Wireshark 1: captura con el chatbot encendido



Wireshark 2: captura con el chatbot apagado



UI: implementación de una User Interface (UI)- Streamlit