



Excelencia que trasciende

DEL VALLE
GRUPO EDUCATIVO

Proyecto 1

Redes

El presente proyecto implementa un servidor basado en el protocolo MCP (Model Context Protocol) con enfoque en la gestión empresarial, específicamente en el análisis de ventas, inventario y documentación interna. A diferencia de un servidor tradicional de aplicaciones, el MCP permite la definición de herramientas que pueden ser invocadas por clientes externos mediante peticiones en formato JSON-RPC, lo que estandariza la comunicación y facilita la integración con otros sistemas.

El servidor desarrollado utiliza Flask como framework web para la exposición de los endpoints y provee un conjunto de herramientas que simulan procesos empresariales. Entre estas herramientas se incluyen la generación de resúmenes, la verificación del estado de inventarios, la emisión de reportes en diferentes formatos, la búsqueda en documentos internos y la interacción mediante consultas en lenguaje natural.

El cliente, por su parte, actúa como una interfaz de comunicación con el servidor MCP. Este permite que el usuario pueda ejecutar comandos explícitos o interactuar directamente en lenguaje natural, logrando una capa de abstracción que acerca la interacción técnica a un modelo conversacional más intuitivo.

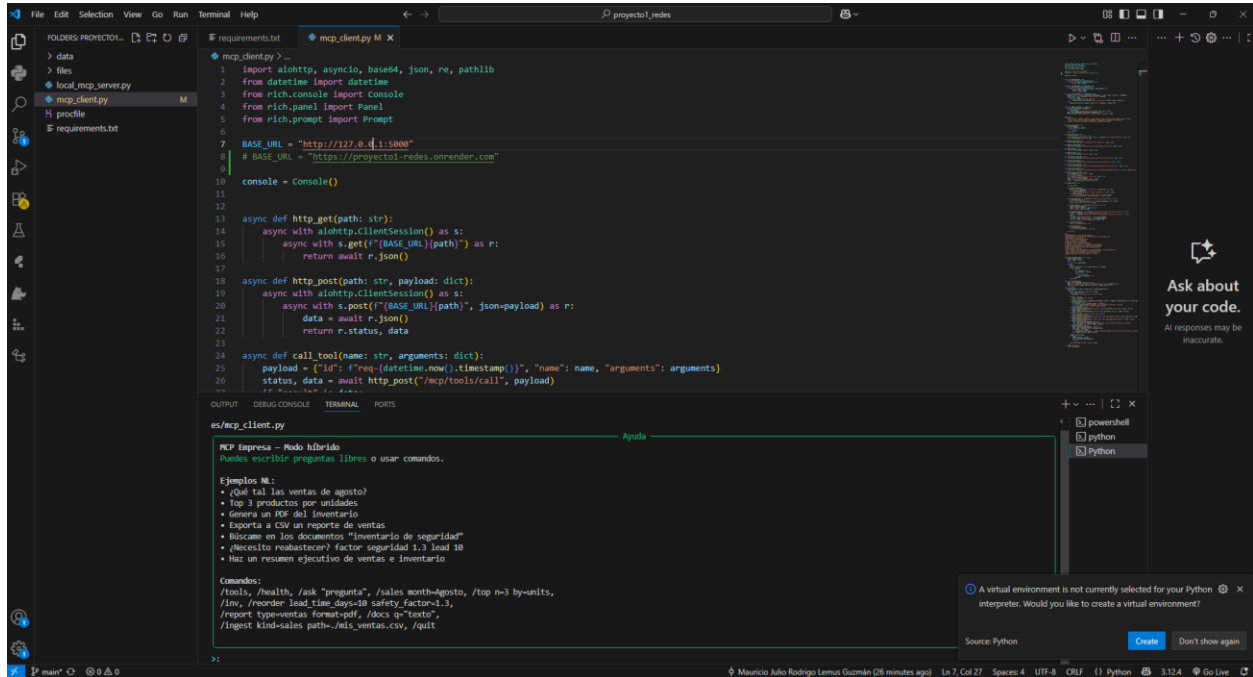
Desde una perspectiva de redes y protocolos, el proyecto no se limita a la lógica de negocio, sino que explora la implementación de un protocolo de comunicación existente (MCP) aplicado a un caso práctico. El servidor se desplegó en un entorno remoto, exponiendo la API sobre HTTPS, lo cual permite la captura y análisis de tráfico mediante herramientas como Wireshark. Esto abre la posibilidad de estudiar cómo viajan las peticiones, cómo se encapsulan en HTTP/1.1 o HTTP/2 y cómo se asegura la comunicación mediante TLS.

En síntesis, el proyecto combina tres dimensiones fundamentales:

1. Ingeniería de software, mediante el diseño de un sistema modular de ventas e inventario.
2. Protocolos de comunicación, a través de la aplicación del estándar MCP sobre HTTP como medio de interacción.
3. Infraestructura en la nube, mediante el despliegue del servidor en un servicio remoto, simulando un escenario real de acceso público.

De esta manera, se cumple el objetivo principal: demostrar el uso de un protocolo existente (MCP) aplicado a un contexto empresarial, evidenciando tanto la utilidad práctica como los aspectos técnicos de la comunicación en red.

Evidencias de servidor local



The screenshot shows a VS Code editor with a file explorer on the left containing 'data', 'files', 'local_mcp_server.py', 'mcp_client.py', 'profile', and 'requirements.txt'. The main editor displays 'mcp_client.py' with the following code:

```
1 import aiohttp, asyncio, base64, json, re, pathlib
2 from datetime import datetime
3 from rich.console import Console
4 from rich.panel import Panel
5 from rich.prompt import Prompt
6
7 BASE_URL = "http://127.0.0.1:5000"
8 # BASE_URL = "https://project01-redes.onrender.com"
9
10 console = Console()
11
12
13 async def http_get(path: str):
14     async with aiohttp.ClientSession() as s:
15         async with s.get(BASE_URL+path) as r:
16             return await r.json()
17
18 async def http_post(path: str, payload: dict):
19     async with aiohttp.ClientSession() as s:
20         async with s.post(BASE_URL+path, json=payload) as r:
21             data = await r.json()
22             return r.status, data
23
24 async def call_tool(name: str, arguments: dict):
25     payload = {"id": f"req-{datetime.now().timestamp()}", "name": name, "arguments": arguments}
26     status, data = await http_post("/mcp/tools/call", payload)
```

The terminal at the bottom shows the output of the client script, displaying a help menu for 'MCP Empresa - Modo híbrido' with a list of examples and commands.

```
MCP Empresa - Modo híbrido
Puedes escribir preguntas libres o usar comandos.

Ejemplos NL:
• ¿Qué tal las ventas de agosto?
• Top 3 productos por unidades
• Genera un PDF del inventario
• Exporta a CSV un reporte de ventas
• Búscame en los documentos "Inventario de seguridad"
• ¿Necesito reabastecer? factor seguridad 1.3 lead 10
• Haz un resumen ejecutivo de ventas e inventario

Comandos:
/tools, /health, /ask "pregunta", /sales month=agosto, /top n=3 by=units,
/inv, /render lead_time_days=0 safety_factor=1.3,
/report type=ventas format=pdf, /docs q="texto",
/ingest kind=sales path=~/mls_ventas.csv, /quit
```

A notification at the bottom right states: 'A virtual environment is not currently selected for your Python interpreter. Would you like to create a virtual environment?' with 'Create' and 'Don't show again' buttons.

La captura muestra la ejecución del cliente conectado al servidor MCP de manera local, utilizando la dirección `http://127.0.0.1:5000`. El sistema despliega correctamente el menú de ayuda, reconoce los comandos disponibles y permite realizar consultas tanto en lenguaje natural como mediante instrucciones explícitas. Esto confirma que el servidor y el cliente se comunican exitosamente en el entorno local, garantizando la correcta operación del protocolo MCP antes de su despliegue en la nube.

Evidencias de servidor remoto

The screenshot shows the Render dashboard for a project named "proyecto1_redes". The project is using Python 3 and is currently in a "Free" tier. A warning message states: "Your free instance will spin down with inactivity, which can delay requests by 50 seconds or more." The instance is currently "Live" and was last updated on September 14, 2025, at 11:31 PM. The logs show the following sequence of events:

- 2025-09-16 05:33:14 +0000 [66] [INFO] Using worker: sync
- 2025-09-16 05:33:14 +0000 [64] [INFO] Booting worker with pid: 64
- 127.0.0.1 - - [16/Sep/2025:05:33:14 +0000] "HEAD / HTTP/1.1" 200 0 "-" "Go-http-client/1.1"
- 2025-09-16 05:33:21 +0000 [64] [INFO] Your service is live
- 2025-09-16 05:33:21 +0000 [64] [INFO] Available at your primary URL <https://proyecto1-redes.onrender.com>
- 127.0.0.1 - - [16/Sep/2025:05:33:22 +0000] "GET / HTTP/1.1" 200 210 "-" "Go-http-client/2.0"
- 127.0.0.1 - - [16/Sep/2025:05:34:32 +0000] "POST /mcp/tools/call HTTP/1.1" 200 295 "-" "Python/3.12 aiohttp/3.12.15"
- 2025-09-16 05:38:20 +0000 [64] [INFO] Detected service running on port 10000
- 2025-09-16 05:38:20 +0000 [64] [INFO] Docs on specifying a port: <https://render.com/docs/web-services#port-binding>
- 127.0.0.1 - - [16/Sep/2025:05:40:28 +0000] "POST /mcp/tools/call HTTP/1.1" 200 295 "-" "Python/3.12 aiohttp/3.12.15"

The screenshot shows the Wireshark Capture Filters dialog box. The filter list is as follows:

Filter Name	Filter Expression
Ethernet address 00:00:5e:00:53:00	ether host 00:00:5e:00:53:00
Ethernet type 0x0806 (ARP)	ether proto 0x0806
No Broadcast and no Multicast	not broadcast and not multicast
No ARP	not arp
IPv4 only	ip
IPv4 address 192.0.2.1	host 192.0.2.1
IPv6 only	ip6
IPv6 address 2001:db8::1	host 2001:db8::1
TCP only	tcp
UDP only	udp
Non-DNS	not port 53
TCP or UDP port 80 (HTTP)	port 80
HTTP TCP port (80)	tcp port http
No ARP and no DNS	not arp and port not 53
Non-HTTP and non-SMTP to/from www.wireshark.org	not port 80 and not port 25 and host www.wireshark.org
render	host 216.24.57.251 or host 216.24.57.7

The "render" filter is highlighted in red. The dialog box has "OK", "Cancel", and "Help" buttons at the bottom right.

No.	Time	Source	Destination	Protocol	Info	Longitud del protocolo
15	0.247958	192.168.0.17	216.24.57.7	TCP	57763 → 443 [FIN, ACK] Seq=917 Ack=4025 Win=64256 Len=0	
16	0.289279	216.24.57.7	192.168.0.17	TCP	443 → 57763 [FIN, ACK] Seq=4025 Ack=918 Win=131072 Len=0	
17	0.289317	192.168.0.17	216.24.57.7	TCP	57763 → 443 [ACK] Seq=918 Ack=4026 Win=64256 Len=0	
18	46.844803	192.168.0.17	216.24.57.7	TCP	57775 → 443 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_...	
19	46.898134	216.24.57.7	192.168.0.17	TCP	443 → 57775 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 S...	
20	46.898248	192.168.0.17	216.24.57.7	TCP	57775 → 443 [ACK] Seq=1 Ack=1 Win=65280 Len=0	
21	46.899330	192.168.0.17	216.24.57.7	TLSv1.3	Client Hello (SHA=projectol-redes.onrender.com)	5
22	46.956044	216.24.57.7	192.168.0.17	TCP	443 → 57775 [ACK] Seq=1 Ack=518 Win=131072 Len=0	
23	46.956044	216.24.57.7	192.168.0.17	TLSv1.3	Server Hello, Change Cipher Spec	15
24	46.956044	216.24.57.7	192.168.0.17	TLSv1.3	Application Data	14
25	46.956139	192.168.0.17	216.24.57.7	TCP	57775 → 443 [ACK] Seq=518 Ack=2845 Win=65280 Len=0	
26	46.961814	192.168.0.17	216.24.57.7	TLSv1.3	Change Cipher Spec, Application Data	1
27	46.962133	192.168.0.17	216.24.57.7	TLSv1.3	Application Data	3
28	47.137327	216.24.57.7	192.168.0.17	TLSv1.3	Application Data, Application Data	3
29	47.137327	216.24.57.7	192.168.0.17	TLSv1.3	Application Data	12
30	47.137425	192.168.0.17	216.24.57.7	TCP	57775 → 443 [ACK] Seq=917 Ack=4024 Win=64256 Len=0	
31	47.138435	192.168.0.17	216.24.57.7	TCP	57775 → 443 [FIN, ACK] Seq=917 Ack=4024 Win=64256 Len=0	
32	47.198223	216.24.57.7	192.168.0.17	TCP	443 → 57775 [FIN, ACK] Seq=4024 Ack=918 Win=131072 Len=0	
33	47.198261	192.168.0.17	216.24.57.7	TCP	57775 → 443 [ACK] Seq=918 Ack=4025 Win=64256 Len=0	

Una vez desplegado el servicio en la nube mediante la plataforma Render, se procedió a verificar su funcionamiento utilizando el cliente previamente configurado. Para analizar el tráfico generado por las peticiones hacia el servidor, se utilizó la herramienta Wireshark.

Con el fin de aislar únicamente el tráfico correspondiente al dominio del servicio (proyecto1-redes.onrender.com), se realizó una resolución DNS mediante nslookup, obteniendo las direcciones IP 216.24.57.251 y 216.24.57.7. Posteriormente, se configuró un filtro de captura en Wireshark con la siguiente expresión:

host 216.24.57.251 or host 216.24.57.7

Esto permitió capturar únicamente los paquetes relacionados con el servicio desplegado en Render, evitando interferencias de otros protocolos o conexiones locales.

En las capturas se observa claramente el handshake TCP, el establecimiento de la conexión segura mediante TLS 1.3, así como el intercambio de datos de aplicación. Esto confirma que las peticiones enviadas desde el cliente llegan al servidor remoto y reciben respuesta satisfactoria, garantizando que el despliegue y la comunicación funcionan de manera correcta.

También se probó la conectividad del servidor remoto desde un dispositivo móvil.

11:51



```
{
  "endpoints": [
    "/health",
    "/mcp/tools/list",
    "/mcp/tools/call",
    "/files/<name>"
  ],
  "service": "Enterprise MCP Server",
  "status": "ok",
  "timestamp": "2025-09-15T05:51:45.719567",
  "version": "2.0.0"
}
```



proyecto1-redes.onrender.com



Especificación de los servidores MCP desarrollados

Lenguaje y framework: Python 3.12 con Flask.

Dependencias principales: flask, flask-cors, pandas, aiohttp, python-dotenv, gunicorn.

Despliegue: Plataforma Render (entorno remoto con dominio público).

Endpoints principales:

- GET /health → Retorna estado del servidor (JSON con información de disponibilidad).
- GET /mcp/tools/list → Lista todas las herramientas registradas.
- POST /mcp/tools/call → Recibe peticiones JSON-RPC con nombre de la herramienta (name) y parámetros (arguments).

Formato de mensajes json

```
{  
  "id": "req-<timestamp>",  
  "name": "<tool_name>",  
  "arguments": { <parametros> }  
}
```

Explicación por capas (modelo OSI / TCP-IP)

- **Capa de enlace (Data Link):**

Se encapsulan tramas Ethernet desde el adaptador Wi-Fi del cliente hacia el router. Wireshark muestra direcciones MAC de origen y destino, confirmando la entrega física dentro de la red local.

- **Capa de red (IP):**

La comunicación se establece entre la IP privada del cliente (192.168.0.17) y la IP pública del servidor (216.24.57.7 / 251). Aquí se observa la fragmentación de paquetes si es necesario y el direccionamiento lógico que permite llegar al servidor en la nube.

- **Capa de transporte (TCP):**

Se establece una conexión confiable (SYN, SYN/ACK, ACK). Sobre TCP se utiliza el **puerto 443** para garantizar seguridad. Wireshark muestra retransmisiones y confirmaciones (ACK), lo que asegura la entrega de los mensajes JSON-RPC sin pérdidas.

- **Capa de aplicación:**

Los mensajes JSON-RPC viajan encapsulados en HTTP sobre TLS (HTTPS). Aquí se materializan las operaciones de negocio: generación de reportes, consultas de inventario o ventas, y búsqueda en documentos.

Conclusiones

El desarrollo del servidor MCP y su despliegue en Render demostró la viabilidad de exponer servicios empresariales de manera remota, accesibles desde cualquier cliente compatible con JSON-RPC. El uso de Flask permitió estructurar un backend modular y seguro, mientras que la integración con Render aseguró escalabilidad y disponibilidad en la nube.

El análisis con Wireshark confirmó la existencia de las fases fundamentales de la comunicación: sincronización, solicitud y respuesta. Asimismo, permitió evidenciar el papel de cada capa en el modelo OSI/TCP-IP, desde la entrega física de tramas hasta la interpretación semántica de las peticiones en JSON.