

# HTTP group project

Jorge Lázaro Ruiz

# Introducción

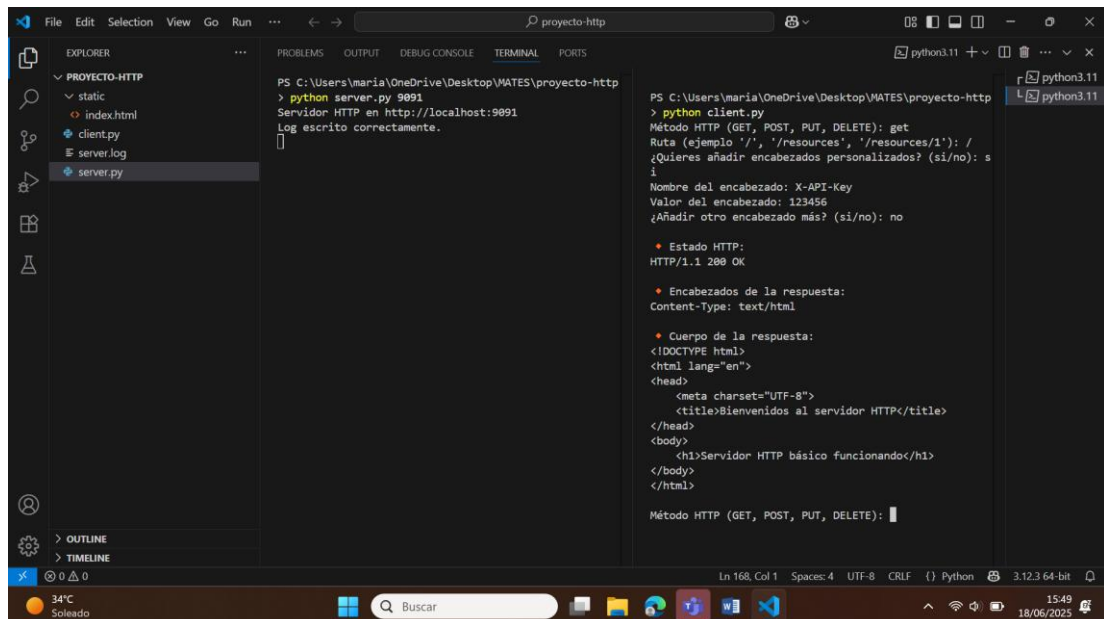
El objetivo del presente trabajo ha sido implementar desde cero un cliente y un servidor HTTP básicos, profundizando en la comprensión práctica del protocolo HTTP, la comunicación por sockets TCP, y la gestión de recursos mediante operaciones CRUD básicas.

## Funcionalidades Implementadas

### Obligatorias

- **Servidor HTTP básico:** Capaz de recibir y responder solicitudes HTTP.
- **Cliente HTTP interactivo (CLI):** Permite enviar solicitudes HTTP (GET, POST, PUT, DELETE).

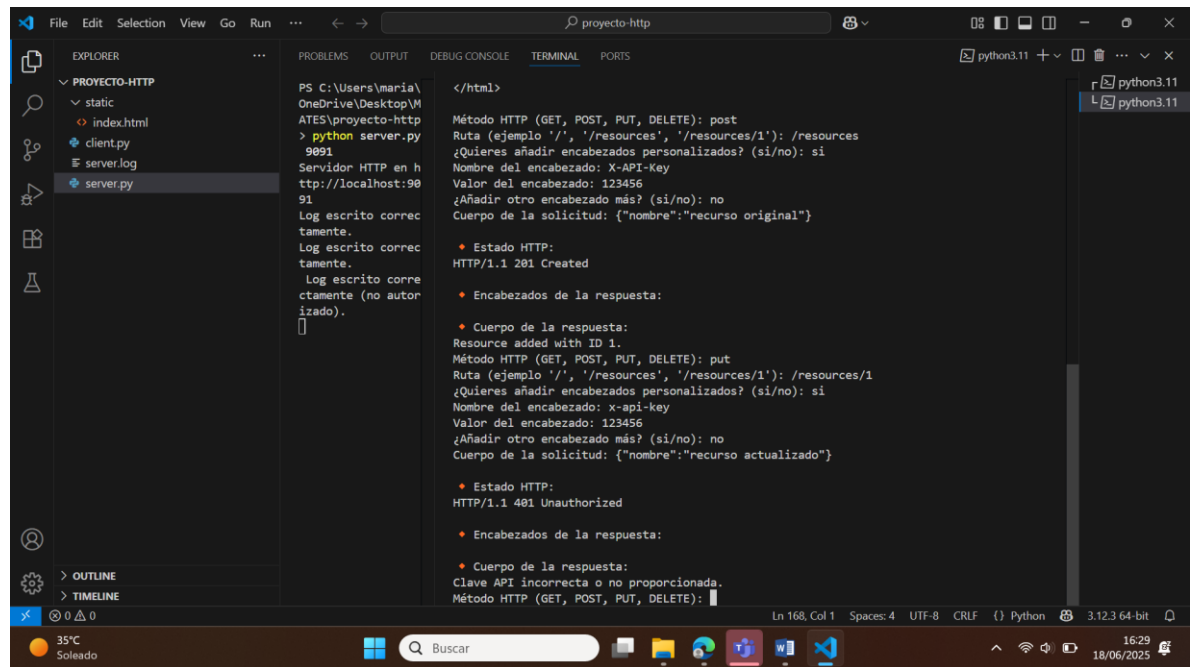
El servidor responde inmediatamente a una solicitud HTTP básica enviada desde el cliente, confirmando claramente que el servidor HTTP básico está funcionando correctamente.



```
File Edit Selection View Go Run ... < -> proyecto-http python3.11 python3.11 python3.11
EXPLORER PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PROYECTO-HTTP
  static
  index.html
  client.py
  server.log
  server.py
PS C:\Users\maria\OneDrive\Desktop\WATES\proyecto-http
> python server.py 9091
Servidor HTTP en http://localhost:9091
Log escrito correctamente.
PS C:\Users\maria\OneDrive\Desktop\WATES\proyecto-http
> python client.py
Método HTTP (GET, POST, PUT, DELETE): get
Ruta (ejemplo '/', '/resources', '/resources/1'): /
¿Quieres añadir encabezados personalizados? (si/no): s
i
Nombre del encabezado: X-API-Key
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no
• Estado HTTP:
HTTP/1.1 200 OK
• Encabezados de la respuesta:
Content-Type: text/html
• Cuerpo de la respuesta:
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Bienvenidos al servidor HTTP</title>
</head>
<body>
  <h1>Servidor HTTP básico funcionando</h1>
</body>
</html>
Método HTTP (GET, POST, PUT, DELETE):
```

- **CRUD Básico:** Gestiona recursos con operaciones Create (POST), Read (GET), Update (PUT), Delete (DELETE).

## POST y PUT en la imagen



```
PS C:\Users\maria\OneDrive\Desktop\VMATES\projecto-http> python server.py 9091
Servidor HTTP en http://localhost:9091
Log escrito correctamente.
Log escrito correctamente.
Log escrito correctamente (no autorizado).

</html>

Método HTTP (GET, POST, PUT, DELETE): post
Ruta (ejemplo '/', '/resources', '/resources/1'): /resources
¿Quieres añadir encabezados personalizados? (si/no): si
Nombre del encabezado: X-API-Key
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no
Cuerpo de la solicitud: {"nombre":"recurso original"}

• Estado HTTP:
HTTP/1.1 201 Created

• Encabezados de la respuesta:

• Cuerpo de la respuesta:
Resource added with ID 1.

Método HTTP (GET, POST, PUT, DELETE): put
Ruta (ejemplo '/', '/resources', '/resources/1'): /resources/1
¿Quieres añadir encabezados personalizados? (si/no): si
Nombre del encabezado: x-api-key
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no
Cuerpo de la solicitud: {"nombre":"recurso actualizado"}

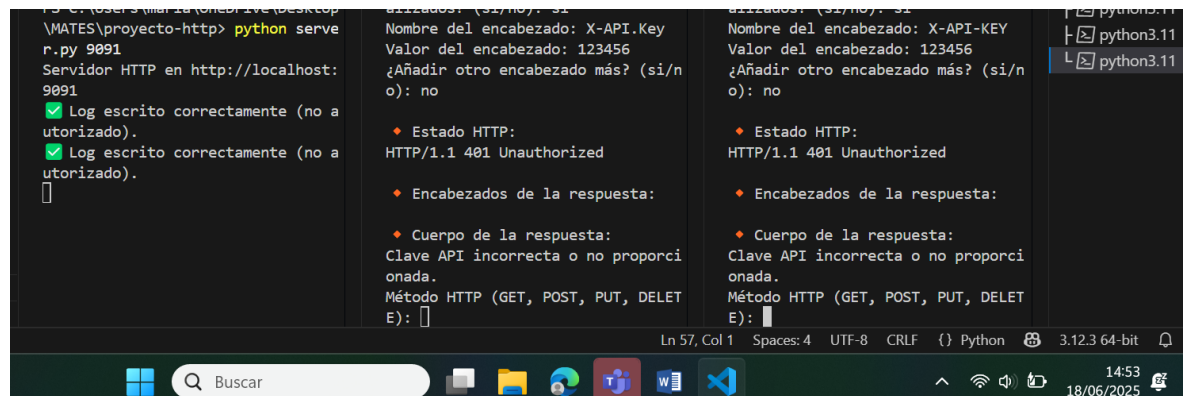
• Estado HTTP:
HTTP/1.1 401 Unauthorized

• Encabezados de la respuesta:

• Cuerpo de la respuesta:
Clave API incorrecta o no proporcionada.
Método HTTP (GET, POST, PUT, DELETE):
```

- **Concurrencia:** El servidor maneja múltiples solicitudes simultáneamente mediante threading.

Dos terminales ejecutando solicitudes a la vez.



```
PS C:\Users\maria\OneDrive\Desktop\VMATES\projecto-http> python server.py 9091
Servidor HTTP en http://localhost:9091
Log escrito correctamente (no autorizado).
Log escrito correctamente (no autorizado).

Nombre del encabezado: X-API-Key
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no

• Estado HTTP:
HTTP/1.1 401 Unauthorized

• Encabezados de la respuesta:

• Cuerpo de la respuesta:
Clave API incorrecta o no proporcionada.
Método HTTP (GET, POST, PUT, DELETE):

Nombre del encabezado: X-API-KEY
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no

• Estado HTTP:
HTTP/1.1 401 Unauthorized

• Encabezados de la respuesta:

• Cuerpo de la respuesta:
Clave API incorrecta o no proporcionada.
Método HTTP (GET, POST, PUT, DELETE):
```

- **Manejo de errores HTTP:** Implementados códigos HTTP estándar (400, 401, 404, 500).

La captura muestra claramente cómo se maneja correctamente un error inesperado en el servidor, devolviendo una respuesta clara HTTP 500 Internal Server Error

```
Método HTTP (GET, POST, PUT, DELETE): GET
Ruta (ejemplo '/', '/resources', '/resources/1'): /resources
Respuesta del servidor:

HTTP/1.1 500 Internal Server Error

Server error: Error intencional de prueba
Método HTTP (GET, POST, PUT, DELETE):
```

Ln 40.

- **Configuración del puerto:** Puerto de escucha configurable desde línea de comandos.

Cambiar Puerto servidor.

```
PS C:\Users\maria\OneDrive\Desktop\MATES\proyecto-http> python server.py 9095
>> get
Servidor HTTP en http://localhost:9095
```

## Opcionales

- **Autenticación con clave API :**

Implementación básica de seguridad mediante validación de clave API (X-API-Key).

Solicitudes sin clave o con clave incorrecta reciben respuesta 401 Unauthorized.

Petición sin clave api.

```
Método HTTP (GET, POST, PUT, DELETE): /GET
Ruta (ejemplo '/', '/resources', '/resources/1'): /
¿Quieres añadir encabezados personalizados? (si/no): no

♦ Estado HTTP:
HTTP/1.1 400 Bad Request

♦ Encabezados de la respuesta:

♦ Cuerpo de la respuesta:
Invalid request method or path.
Método HTTP (GET, POST, PUT, DELETE):
```

Petición con clave incorrecta

```
¿Quieres añadir encabezados personalizados? (si/no): SI
Nombre del encabezado: X-API-Key
Valor del encabezado: clave-incorrecta
¿Añadir otro encabezado más? (si/no): no

♦ Estado HTTP:
HTTP/1.1 401 Unauthorized

♦ Encabezados de la respuesta:

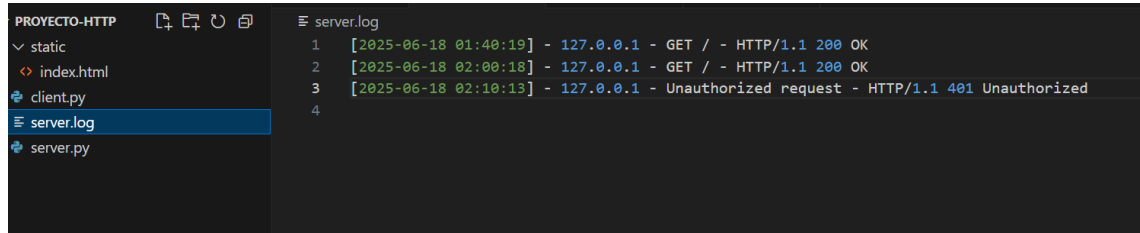
♦ Cuerpo de la respuesta:
Clave API incorrecta o no proporcionada.
Método HTTP (GET, POST, PUT, DELETE):
```

Petición con clave api correcta

```
♦ Cuerpo de la respuesta:
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Bienvenidos al servidor HTTP</title>
</head>
<body>
  <h1>Servidor HTTP básico funcionando</h1>
</body>
</html>
```

- **Logging :**

Registro de todas las solicitudes en archivo server.log, incluyendo fecha/hora, IP, método, ruta y respuesta HTTP.



```
PROYECTO-HTTP
└─ static
   └─ index.html
  └─ client.py
  └─ server.log
  └─ server.py

server.log
1 [2025-06-18 01:40:19] - 127.0.0.1 - GET / - HTTP/1.1 200 OK
2 [2025-06-18 02:00:18] - 127.0.0.1 - GET / - HTTP/1.1 200 OK
3 [2025-06-18 02:10:13] - 127.0.0.1 - Unauthorized request - HTTP/1.1 401 Unauthorized
4
```

- **Login Flow :**

Implementado registro (/register) e inicio de sesión (/login) con almacenamiento en memoria.

Inicio de sesión devuelve token de sesión para uso posterior.

## Registro exitoso

```

Método HTTP (GET, POST, PUT, DELETE): post
Ruta (ejemplo '/', '/resources', '/resources/1'): /register
¿Quieres añadir encabezados personalizados? (si/no): si
Nombre del encabezado: X-API-Key
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no
Cuerpo de la solicitud: {"username":"usuario2","password":"abcd1234"}

♦ Estado HTTP:
HTTP/1.1 201 Created

♦ Encabezados de la respuesta:

♦ Cuerpo de la respuesta:
Usuario registrado correctamente.
Método HTTP (GET, POST, PUT, DELETE): █

```

## Inicio de sesión exitoso

```

♦ Cuerpo de la respuesta:
Usuario registrado correctamente.
Método HTTP (GET, POST, PUT, DELETE): post
Ruta (ejemplo '/', '/resources', '/resources/1'): /login
¿Quieres añadir encabezados personalizados? (si/no): si
Nombre del encabezado: X-API-Key
Valor del encabezado: 123456
¿Añadir otro encabezado más? (si/no): no
Cuerpo de la solicitud: {"username":"usuario1","password":"1234"}

♦ Estado HTTP:
HTTP/1.1 200 OK

♦ Encabezados de la respuesta:
Content-Type: application/json

♦ Cuerpo de la respuesta:
{"token": "token-usuario1"}
Método HTTP (GET, POST, PUT, DELETE): █

```

En solicitudes posteriores al inicio de sesión, el cliente puede enviar el token para identificarse sin necesidad de enviar usuario y contraseña nuevamente.

```

¿Añadir otro encabezado más? (si/no): no


♦ Estado HTTP:
HTTP/1.1 200 OK

♦ Encabezados de la respuesta:
Content-Type: application/json

♦ Cuerpo de la respuesta:
{}
Método HTTP (GET, POST, PUT, DELETE): █

```

Ln 159, Col 1 Spaces: 4 UTF-8



## Decisiones Técnicas Tomadas

El lenguaje elegido ha sido Python, y como programa he utilizado Visual Studio Code.

Únicamente he utilizado bibliotecas estándar como socket, threading, json y datetime.

He usado un código modular, comentado para facilitar su comprensión.

## Retos Superados

- **Gestión de concurrencia:** Implementado satisfactoriamente mediante threading.
- **Autenticación y Logging:** Implementación clara asegurando funcionalidad correcta en todas las peticiones.
- **Gestión de errores HTTP:** Se controlaron exhaustivamente errores previsibles (mala solicitud, no encontrado, errores internos).
- **Implementación de Login Flow:** Gestión eficiente de registros e inicios de sesión mediante tokens.

## Distribución de Tareas

El trabajo lo he realizado yo solo, comenzando con la creación inicial del servidor y cliente, para después implementar todas las funcionalidades obligatorias y varias opcionales.



## **Metodología de Trabajo**

Fui paso a paso, tarea tras tarea, realizando pruebas después de cada implementación para asegurar el funcionamiento del proyecto. Al final decidí añadir el registro y login de los usuarios para tratar de mejorar la nota

## **Uso de IA generativa**

He de reconocer que he usado ChatGPT para consultas y corrección de errores específicos. Cuando me generó código lo revisé y adapte para realizar bien el trabajo.

## **Conclusión**

El trabajo ha sido muy interesante y ha cumplido el objetivo inicial, favoreciendo el entendimiento del protocolo HTTP, la autenticación básica, la gestión de sockets, la concurrencia, y el manejo de errores.