Realizado por:
Andrea Catalán

SERVICIOS API

HTTP

Realizado por:
Alberto Bernet

Índice de CONTENIDOS

Introducción

1. En esta práctica se implementará un Cliente HTTP que enviará solicitudes a un Servidor HTTP, el cual responderá con un mensaje.

2. Utilizaremos la API java.net.http de Java 11 para realizar estas operaciones.

3. El objetivo es comprender cómo se comunican las aplicaciones a través del protocolo HTTP.

¿Qué es... un Cliente HTTP?

- Un Cliente HTTP es un programa que envía solicitudes HTTP a un servidor y procesa sus respuestas.
- Se utiliza para comunicarse con APIs y servidores web.
- Java proporciona herramientas para realizar peticiones HTTP de manera sencilla.

Ejemplos de uso

Obtener información desde APIs, como por ejemplo sobre noticias o el clima.

Consultar datos de una base de datos remota.

Descargar páginas web o archivos.

03

API de Java para Cliente en HTTP

Desde Java 11, la API **java.net.http** proporciona clases clave para trabajar con HTTP:

➔ Clases principales:

HttpClient:
Maneja las solicitudes HTTP.

HttpRequest:
Representa una solicitud HTTP.

HttpResponse:
Contiene la respuesta del servidor.

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://ejemplo.com"))
        .GET()
        .build();
HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
System.out.println(response.body());
```

04

Otros métodos HTTP

Aunque en esta práctica solo usamos GET, existen otros métodos HTTP que permiten diferentes operaciones en una API:

- **POST → Envía datos al servidor para ser procesados.**

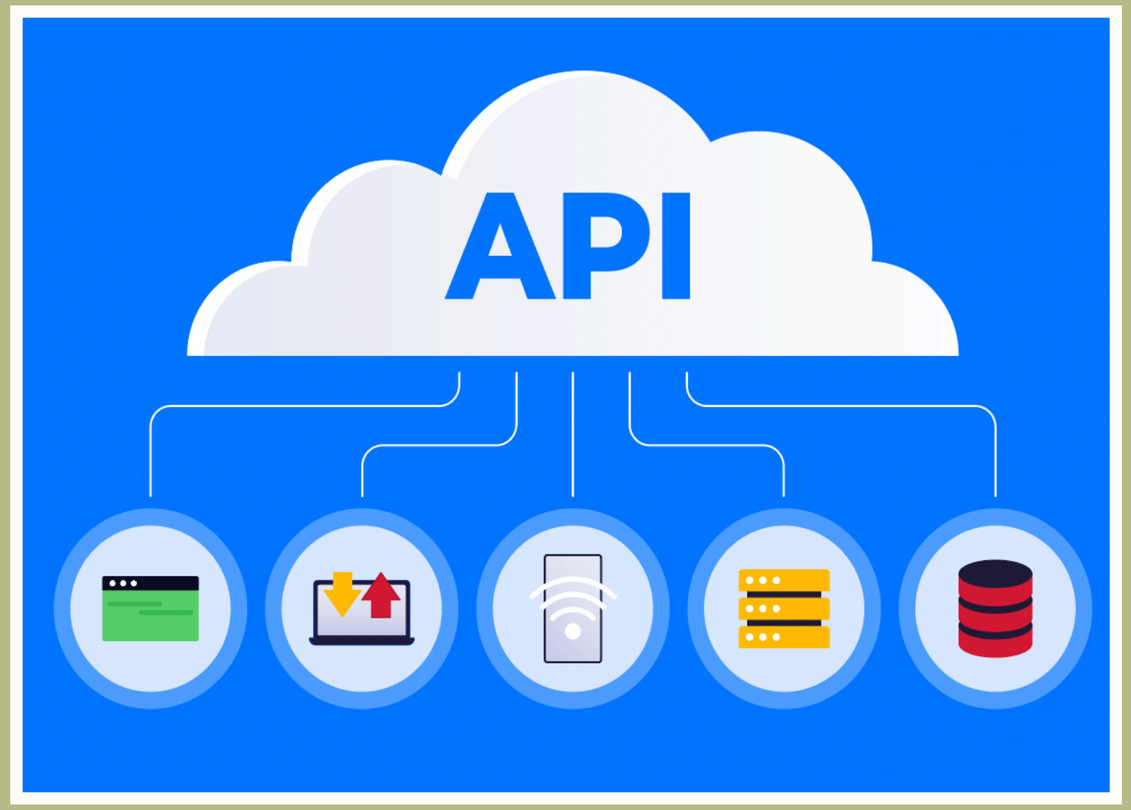
```
HttpRequest requestPost = HttpRequest.newBuilder()
        .uri(URI.create("http://localhost:8080/recurso"))
        .POST(HttpRequest.BodyPublishers.ofString( body: "nuevo contenido"))
        .build();
```

- **PUT → Actualiza datos en el servidor.**

```
HttpRequest requestPut = HttpRequest.newBuilder()
        .uri(URI.create("http://localhost:8080/recurso"))
        .PUT(HttpRequest.BodyPublishers.ofString( body: "actualización"))
        .build();
```

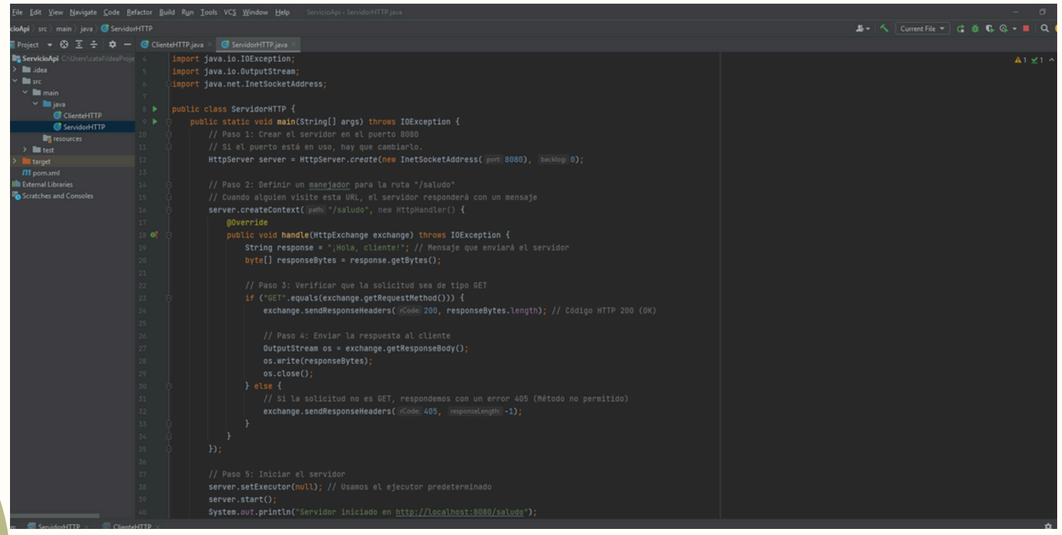
- **DELETE → Elimina un recurso del servidor.**

```
HttpRequest requestDelete = HttpRequest.newBuilder()
        .uri(URI.create("http://localhost:8080/recurso"))
        .DELETE()
        .build();
```

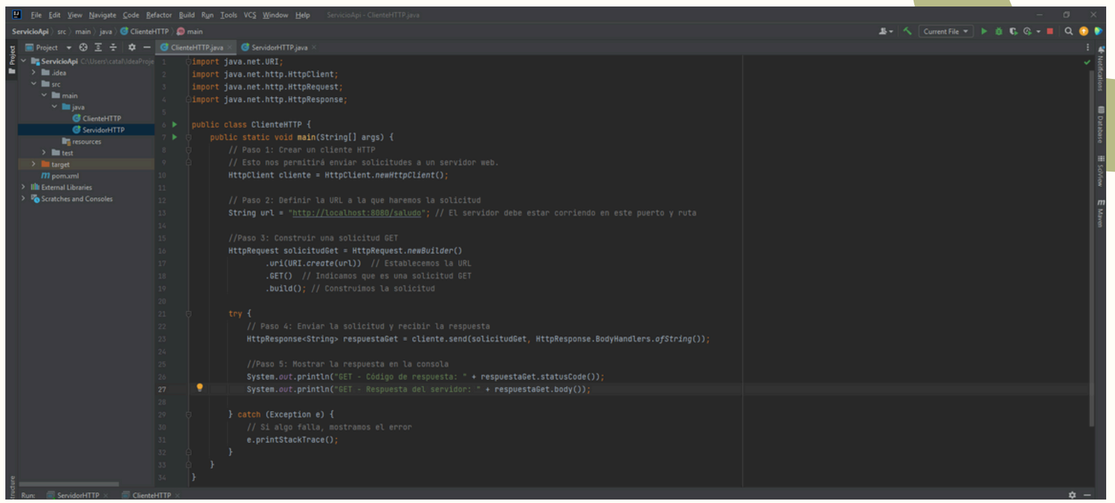


CÓDIGO DE IMPLEMENTACIÓN

Clase ServidorHTTP



```java
import java.io.IOException;
import java.io.OutputStream;
import java.net.InetSocketAddress;

public class ServidorHTTP {
    public static void main(String[] args) throws IOException {
        // Paso 1: Crear el servidor en el puerto 8080
        // Si el puerto está en uso, hay que cambiarlo.
        HttpServer server = HttpServer.create(new InetSocketAddress( port: 8080),  backlog: 0);

        // Paso 2: Definir un manejador para la ruta "/saludo"
        // Cuando alguien visite esta URL, el servidor responderá con un mensaje
        server.createContext( path: "/saludo", new HttpHandler() {
            @Override
            public void handle(HttpExchange exchange) throws IOException {
                String response = "¡Hola, cliente!"; // Mensaje que enviará el servidor
                byte[] responseBytes = response.getBytes();

                // Paso 3: Verificar que la solicitud sea de tipo GET
                if ("GET".equals(exchange.getRequestMethod())) {
                    exchange.sendResponseHeaders( rCode: 200, responseBytes.length); // Código HTTP 200 (OK)

                    // Paso 4: Enviar la respuesta al cliente
                    OutputStream os = exchange.getResponseBody();
                    os.write(responseBytes);
                    os.close();
                } else {
                    // Si la solicitud no es GET, respondemos con un error 405 (Método no permitido)
                    exchange.sendResponseHeaders( rCode: 405,  responseLength: -1);
                }
            }
        });

        // Paso 5: Iniciar el servidor
        server.setExecutor(null); // Usamos el ejecutor predeterminado
        server.start();
        System.out.println("Servidor iniciado en http://localhost:8080/saludo");
```



```java
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class ClienteHTTP {
    public static void main(String[] args) {
        // Paso 1: Crear un cliente HTTP
        // Esto nos permitirá enviar solicitudes a un servidor web.
        HttpClient cliente = HttpClient.newHttpClient();

        // Paso 2: Definir la URL a la que haremos la solicitud
        String url = "http://localhost:8080/saludo"; // El servidor debe estar corriendo en este puerto y ruta

        //Paso 3: Construir una solicitud GET
        HttpRequest solicitudGet = HttpRequest.newBuilder()
                .uri(URI.create(url))  // Establecemos la URL
                .GET()  // Indicamos que es una solicitud GET
                .build(); // Construimos la solicitud

        try {
            // Paso 4: Enviar la solicitud y recibir la respuesta
            HttpResponse<String> respuestaGet = cliente.send(solicitudGet, HttpResponse.BodyHandlers.ofString());

            //Paso 5: Mostrar la respuesta en la consola
            System.out.println("GET - Código de respuesta: " + respuestaGet.statusCode());
            System.out.println("GET - Respuesta del servidor: " + respuestaGet.body());

        } catch (Exception e) {
            // Si algo falla, mostramos el error
            e.printStackTrace();
        }
    }
}
```

Clase ClienteHTTP

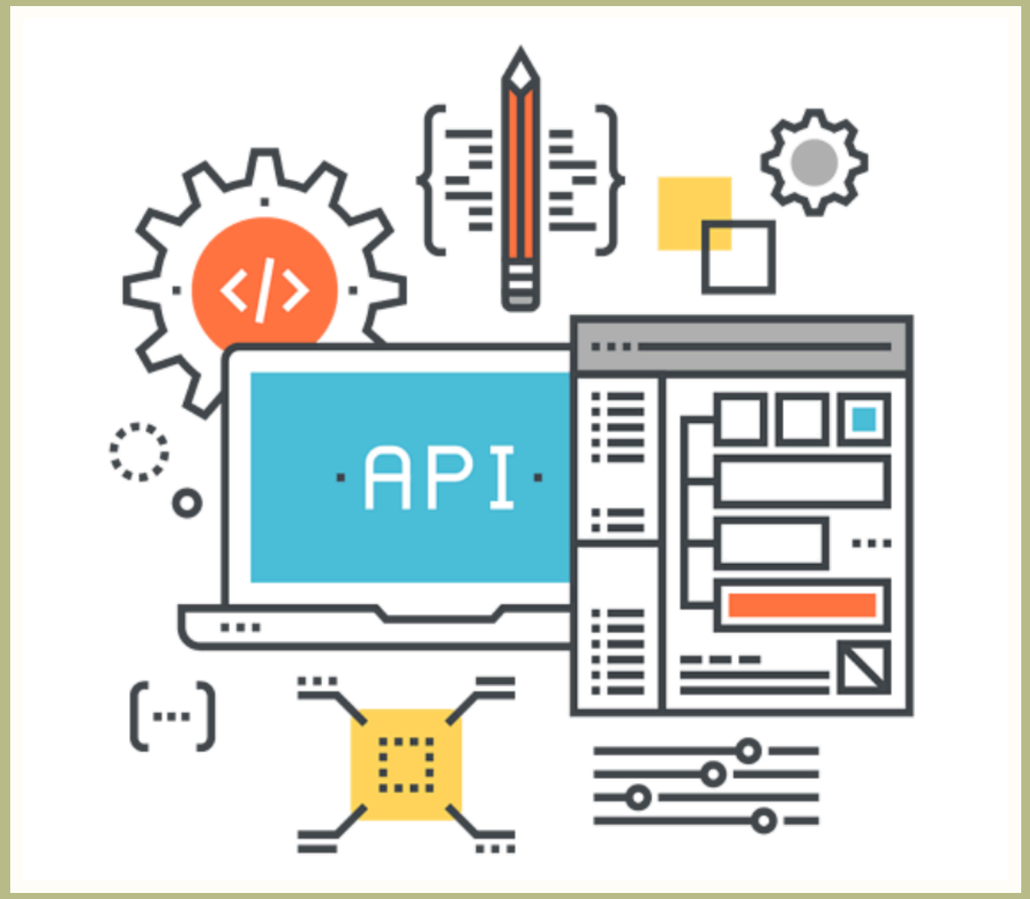
Explicación del código:

Servidor HTTP

- **Crea un servidor en el puerto 8080.**
- **Define un manejador (/saludo) que responde con "¡Hola, cliente!"**
- **Verifica que la solicitud sea de tipo GET.**
- **Envía la respuesta con un código 200 que significa "OK, todo bien".**

Cliente HTTP

- **Crea una instancia de HttpClient para manejar la conexión.**
- **Construye una solicitud HTTP GET con la URL del servidor.**
- **Envía la solicitud y obtiene la respuesta.**
- **Imprime el código de estado y el contenido de la respuesta.**



EJECUCIÓN Y SALIDA

01 Iniciar el servidor ejecutando ServidorHTTP.java

Mensaje en consola: Servidor iniciado en http://localhost:8080/saludo

02 Ejecutar el cliente con ClienteHTTP.java

Salida esperada en consola:

Dificultades que pueden

Encontrarse

Puerto ocupado:

Al intentar levantar el servidor en el puerto 8080, tuvimos un pequeño problema al no darnos cuenta que NetBeans lo estaba usando con TomCat.

Errores de conexión:

Obviamente debemos asegurarnos de que el servidor se está ejecutando antes de hacer la petición.

Problemas con permisos:

Algunos entornos requieren permisos de red para escuchar en un puerto. Para solucionarlo deberíamos ejecutar el programa con permisos de administrador.

Resumen Final

SE IMPLEMENTÓ UN CLIENTE Y UN SERVIDOR HTTP USANDO JAVA.

SE UTILIZÓ JAVA.NET.HTTP.HTTPCLIENT PARA ENVIAR SOLICITUDES HTTP.

SE CREÓ UN SERVIDOR CON COM.SUN.NET.HTTPSERVER.HTTPSERVER QUE RESPONDE A SOLICITUDES GET.

SE APRENDIÓ SOBRE LA ESTRUCTURA DE LAS PETICIONES Y RESPUESTAS HTTP.

10

Por último:

Conclusión

Conclusiones

- **Este proyecto nos ha ayudado a comprender fácilmente el funcionamiento de la comunicación cliente-servidor usando el API de Java. Tiene una implementación muy sencilla, y nos permite entender el funcionamiento de HTTP, aunque sea a bajo nivel, sin el uso de librerías externas.**

Muchas

GRACIAS