PROGRAMACIÓN DE SERVICIOS Y PROCESOS – TAREA 5

📦 Tarea: Servidor HTTP de Curiosidades del Mundo

📖 Descripción del Problema:

- Desarrolla un **servidor HTTP multicliente en Java** capaz de responder a peticiones GET desde el navegador.
- El servidor debe estar preparado para atender a **varios clientes simultáneamente** usando **hilos**.
- Debe responder con diferentes páginas HTML en función de la URL solicitada:
  - /: página de bienvenida
  - /curiosidad: muestra una curiosidad aleatoria
  - /contacto: información de contacto
  - Ruta desconocida: mostrar error 404

⚖️ Requisitos:

Servidor HTTP:

- Escuchar en el puerto 8080 o 8066.
- Leer la primera línea de la petición GET del cliente.
- Determinar la ruta solicitada (/, /curiosidad, etc.).
- Enviar una respuesta HTTP completa con cabecera + HTML.
- Cada conexión debe ser gestionada en un hilo independiente.

Respuestas esperadas:

- /: HTML de bienvenida.
- /curiosidad: muestra una de varias curiosidades aleatorias.
- /contacto: página con datos ficticios de contacto.
- *Cualquier otra ruta*: página 404 con mensaje de error.

💾 Archivos Utilizados:

- Paginas.java: contiene las respuestas HTML.
- Mensajes.java: contiene las líneas de cabecera HTTP.

🌐 Comunicación HTTP:

- El cliente (navegador) solicita una URL vía GET.
- El servidor detecta la ruta solicitada.
- El servidor responde con contenido HTML correspondiente.

💡 Ejemplo de Ejecución:

Navegador:

http://localhost:8086/
http://localhost:8086/curiosidad
http://localhost:8086/contacto
http://localhost:8086/noexiste

Servidor:

Servidor HTTP escuchando en el puerto 8086
Cliente conectado desde: /127.0.0.1
Petición: GET /curiosidad HTTP/1.1
Cliente conectado desde: /127.0.0.1
Petición: GET /contacto HTTP/1.1

🛠️ Indicaciones de entrega

- Entregar un fichero ZIP que incluya:
  - El proyecto Java completo.
  - Un documento explicativo con:
    - Descripción de la solución.
    - Capturas de pantalla de peticiones desde el navegador.
    - Explicación de cómo se gestionan los hilos.
- **Nomenclatura:** PSP05_Tarea04_apellido1_apellido2_nombre.zip

✨ Consejos y Recomendaciones

- Empieza adaptando el ejemplo de servidor HTTP de la unidad.
- Crea una clase manejadora que implemente Runnable.
- Evita errores de concurrencia usando una instancia de hilo por cliente.
- Asegúrate de cerrar todos los recursos abiertos.

📊 Criterios de Calificación: Total 10 puntos

- **(3 puntos):** Servidor acepta múltiples conexiones simultáneas.
- **(2 puntos):** Gestión correcta de las rutas solicitadas.
- **(2 puntos):** Uso adecuado de hilos y concurrencia.
- **(1 punto):** Código claro, estructurado y comentado.
- **(1 punto):** Documento explicativo con capturas.
- **(1 punto):** Funcionamiento comprobado desde el navegador.

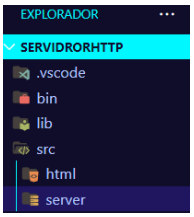
🔧 Resultados de aprendizaje y criterios de evaluación

Desarrolla aplicaciones que ofrecen servicios en red, utilizando librerías de clases y aplicando criterios de eficiencia y disponibilidad:

a) Se han analizado librerías que permitan implementar protocolos estándar de comunicación en red.
b) Se han programado clientes de protocolos estándar de comunicaciones y verificado su funcionamiento.
c) Se han desarrollado y probado servicios de comunicación en red.
d) Se han analizado los requerimientos necesarios para crear servicios capaces de gestionar varios clientes concurrentes.
e) Se han incorporado mecanismos para posibilitar la comunicación simultánea de varios clientes con el servicio.
f) Se ha verificado la disponibilidad del servicio.
g) Se han depurado y documentado las aplicaciones desarrolladas.

Resolución de la tarea

Para la resolución de esta tarea se ha desarrollado un proyecto Java, que contiene dos apartados uno almacenar el servidor Http, que gestionará la conexión para mostrar controlar el acceso a la paginas web. Y otro que contendrá el HTML y las imágenes que contienen las páginas.



Server

Para la configuración del servidor se han creado tres clases, una contendrá al main que ejecutará el servidor, otra que gestionará los clientes y otra que cargará el contenido de la web.

Veamos en profundidad el proceso que realiza cada una de ellas, vamos a comenzar viendo la clase denominada WebSites, será la encargada de devolver la web que ha de mostrar según haya seleccionado el usuario.

WebSites.java

La función principal de esta clase es la de leer los ficheros html y sus recursos y devolverlos a través de servidor.

Esta clase tendrá dos variables de tipo String una para almacenar la ruta donde se encuentran los ficheros HTML y otra para almacenar la dirección web de la API que se va a usar para mostrar unos chistes relacionados con informática.

Además, habrá tres métodos uno público y dos privados.

- **obtenerRespuestas:** este método devolverá una String con el HTML que se mostrará en la web.

  Este método tiene tres variables, una para almacenar el protocolo HTML y el tipo de conexión 200 ok, que indica que es una conexión exitosa. Otra para para almacenar el protocolo HTML y tipo de conexión 404 que indica que no se ha encontrado, por último, habrá una que contendrá el código HTLM, que se pintará en la web.

  Para que se devuelva la web exacta solicitada por el usuario, se utilizará un switch para mostrar una web u otra.

  El GET que se reciba se enviará al método cargarCuerpo, y en el caso que se reciba una curiosidad, también se obtendrá el chiste a través del método obtenerCuriosidadDesdeApi.

  Por último, en caso de que no se encuentre el GET solicitado, se mostrará la web error 404.

  En caso de que el cuerpo sea null, se mostrará un de forma manual que no se ha encontrado la web – error 404.

```
public String obtenerRespuesta(String pagina) {

    // Se establece el tipo de protocolo HTTP y el tipo 200 para establecer una
    // conexión exitosa
    String cabecera = "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n";
    // Se estable el tipo de protocolo HTTT y el tipo 404 para indicar que no se ha
    // encontrado la página.
    String cabecera404 = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n\r\n";
    // Variable para almacenar el cuerpo del HTML que se va a mostrar.
    String cuerpo;

    // Control de flujo switch para selecionar la página web a mostar.
    switch (pagina) {
        // Si se recibe "/" se mostrará el HTML index
        case "/":
            cuerpo = cargarCuerpo(html:"index.html");
            // Se finaliza el switch
            break;
        // Si se recibe "/curiosidad" se mostrará una curiosidad.
        case "/curiosidad":
            // Llamamos a la API para obtener un chiste
            String curiosidad = obtenerCuriosidadDesdeAPI();
            // Se obtiene la web curiosidades y se remplaza el contenido por la curiosidad
            // (chiste)
            cuerpo = cargarCuerpo(html:"curiosidades.html").replace(target:"{{curiosidad}}", curiosidad);
            // Se finaliza el switch
            break;
        // Si se recibe "/contacto" se mostrará el HTML del contancto
        case "/contacto":
            cuerpo = cargarCuerpo(html:"contacto.html");
            // Se finaliza el switch
            break;
        // En caso que se reciba un GET que no coincida con ningún HMTL de nuestra WEB
        // se mostrará el HTML de Error 404
        default:
            cuerpo = cargarCuerpo(html:"error404.html");
            // Se finaliza el switch y se muestra la web error 404.
            return cabecera404 + cuerpo;
    }

    // Si el cuerpo está vacío o null, se muestra un error
    if (cuerpo == null || cuerpo.isEmpty()) {
        return cabecera404 + "<html><body><h1>Error al cargar la página.</h1></body></html>";
    }

    // Se devuelve el HTML completo para mostrar la WEB.
    return cabecera + cuerpo;
}
```

- **obtenerCuriosidadDesdeApi:** este método obtendrá de una API externa un chiste y se devolverá el String para incorporarlo a la página web Curiosidades. Esta controlado la conexión con la API, para controlar que se devuelva algo en caso de no haber conexión con la API.

```
private String obtenerCuriosidadDesdeAPI() {
    try {
        // Creamos la URI utilizando el método URI.create() y luego la convertimos a URL
        URI uri = URI.create(API_URL);
        // Se convierte el URI en URL
        URL url = uri.toURL();
        // Se establece la conexión
        HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setRequestMethod(method:"GET");
        // Tiempo de espera para la conexión
        connection.setConnectTimeout(timeout:5000);
        // Tiempo de espera para leer el get.
        connection.setReadTimeout(timeout:5000); // Timeout para leer

        // Se lee la respuesta de la API
        BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
        // Varaibel para obtener la linea
        String inputLine;
        // StringBuilder para almacenar todo los datos recibidos
        StringBuilder response = new StringBuilder();

        // Se recorren todas las líneas de la lectura
        while ((inputLine = in.readLine()) != null) {
            // Se almacenan todas las líneas en el StringBuilder.
            response.append(inputLine);
        }

        // se finaliza el BuferedReader
        in.close();

        // Se contruje un Json para con los daos recibidos.
        JSONObject jsonResponse = new JSONObject(response.toString());

        // Se obtiene le tipo
        String tipo = jsonResponse.getString(key:"type");

        // Control para obtener un chiste compuesto o un chiste simple
        if (tipo.equals(anObject:"single")) {
            // Caso de chiste simple
            return "\"" + jsonResponse.getString(key:"joke") + "\"";
        } else if (tipo.equals(anObject:"twopart")) {
            // Caso de chiste en dos partes
            return "\"" + jsonResponse.getString(key:"setup") + "\"<br><em>" + jsonResponse.getString(key:"delivery")
                    + "</em>";
        }

        // En caso de que el tipo no sea conocido
        return "\"No hay chiste disponible.\"";

    } catch (IOException e) {
        // Si hay un error en la conexión o al leer la respuesta, mostramos un mensaje
        // error
        e.printStackTrace();
        return "Error al cargar el chiste desde la API.";
    }
}
```

- **cargarCuerpo:** este método devolverá el HTML completo para mostrarlo en la Web. Recibe un String con el nombre del fichero HTLM que se tiene que obtener. Este código estará dentro de un try-catch, para controlar los accesos a las rutas y lectura de los datos.

  En caso de que surja un error se devolverá un mensaje de error

  La construcción de la página web esta compuesta por dos ficheros, uno que es base para mostrar en todas las páginas y otro que contiene el contenido que se va a mostrar en el main de la web.

  Cuando se cargue la web se devuelve un String con todo el código HTLM para pintar la web.

```java
private String cargarCuerpo(String html) {
    try {
        // Leemos la base del HTML
        String baseHTML = Files.readString(Paths.get(HTML_DIR + "base.html"));
        // Leemos el contenido de la página HTML solicitada
        String contenido = Files.readString(Paths.get(HTML_DIR + html));
        // Insertamos el contenido de la página dentro de la base
        return baseHTML.replace(target:"{{content}}", contenido);
    } catch (IOException e) {
        // Si ocurre un error al leer los archivos, lo imprimimos en consola
        e.printStackTrace();
        // En caso de error, devolvemos una página de error
        return "<html><body><h1>Error al cargar el contenido de la página.</h1></body></html>";
    }
}
```

Ahora vamos a ver la clase ClientHandler, esta clase maneja las conexiones individuales de los clientes en el servidor HTTP.

ClientHandle.java

Esta clase como se ha comentado anteriormente implementa un Runnable que permitirá ejecutarse en hilos separados para gestionar las conexiones individuales.

Encontraremos dos atributos, un Socket y un Objeto WebSites, para mostrar las páginas seleccionadas.

Habrá un constructor, para inicializar el socket y el objeto WebSites que nos permitirá manejar las respuestas.

El método runnable de la clase, gestionará la lógica principal para las peticiones HTTP, además, leerá la petición GET que se recibe del cliente. También se controlará los recursos estáticos y dinámicos, de este modo se podrán cargar las imágenes que necesita la web para ser mostrada correctamente. Si finalizará la conexión con el cliente.

```java
@Override
public void run() {
    // Método para leer la peticiones del cliente - se controla a través de un try-with-out.
    try (BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            OutputStream out = socket.getOutputStream()) {

        // Se almacena la línea recibida.
        String requestLine = in.readLine();

        // Si la línea es distita a null y comienza con "GET"
        if (requestLine != null && requestLine.startsWith(prefix:"GET")) {
            // Se muestra la peticón recibida por el cliente
            System.out.println("Petición: " + requestLine.toString());
            // Se obtiene el paht para la página que se va a mostrar.
            String path = requestLine.split(regex:" ")[1];
            // Si se recibe un petición de resource
            if (path.startsWith(prefix:"/resource/")) {
                // Se llama al método para obtener las imágenes para la web.
                servirArchivoEstatico(path, out);
            // De lo contraio
            } else {
                // Se obtiene la pagina a mostrar
                String response = selPaSites.obtenerRespuesta(path);
                // Se envía la respuesta recibida.
                out.write(response.getBytes());
            }
        }
    // Se controal las excepciones
    } catch (IOException e) {
        e.printStackTrace();
    // Se finaliza si o si la conexión
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Método para devolver las imágenes (Recursos) que se van a mostrar en las paginas que se soliciten en por el método GET. Se recibirá la ruta que contiene los recursos, y un OutputStream para poder pintar en la web las imágenes

Este método buscará en el directorio html, el recurso solicitado, determinará el tipo del contenido y enviará la cabercerá correspondiente y se transmite la imagen solicitada. En caso de que no exista se enviará un error 404.

```
private void servirArchivoEstatico(String path, OutputStream out) {
    try {
        // Directorio raíz donde están las imágenes, ajusta si cambia tu estructura
        String filePath = "src/html" + path;
        Path archivo = Paths.get(filePath);

        // Se comprueba si existe el archivo solicitado.
        if (!Files.exists(archivo)) {
            // Se llama al método para devolver error 404
            enviar404(out);
            // Se finaliza el método
            return;
        }

        // Se obtiene archivo
        String contentType = Files.probeContentType(archivo);
        // Si el contenido es null se le asigan el tipo
        if (contentType == null) contentType = "application/octet-stream";

        // Se almacena los bytes del archivo.
        byte[] contenido = Files.readAllBytes(archivo);

        // Se construye la cabecera HTTP
        PrintWriter writer = new PrintWriter(out, autoFlush:false);
        // Protocolo de conexión y estado
        writer.print(s:"HTTP/1.1 200 OK\r\n");
        // tipo de contenido
        writer.print("Content-Type: " + contentType + "\r\n");
        // Longitud del contenido
        writer.print("Content-Length: " + contenido.length + "\r\n");
        // Líenas en blaco obligatoria
        writer.print(s:"\r\n");
        // Nos aseguramos que se envien los datos inmediatamente.
        writer.flush();

        //Se envía el contenido
        out.write(contenido);
        // Nos aseguramos que se envien los datos inmediatamente.
        out.flush();

    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Por último, se ha creado un método que construye la cabecera para devolver un error 404 cuando no se encuentre el recurso solicitado.

```
private void enviar404(OutputStream out) throws IOException {
    // Se contruye el HTML a mostrar
    String mensaje = "HTTP/1.1 404 Not Found\r\nContent-Type: text/html\r\n\r\n"
            + "<html><body><h1>404 - Archivo no encontrado</h1></body></html>";
    // Se envia el mensaje
    out.write(mensaje.getBytes());
}
```

Ahora que ya tenemos todo la lógica de nuestra aplicación, vamos a configurar el servidor HTTP.

ServerHTTP.java

En esta clase se va a implementar el servidor HTTP que permitirá varias conexiones simultáneamente. Lo primero se instanciará el puerto con el puerto 8086, ya que el 8080 esta cubierto por otra aplicación en mi dispositivo.

Seguidamente se establecerá un main, para ejecutar el proceso del servidor. En el main, se creará un servidor con el número de puerto indicado, además habrá un bucle infinito, que establecerá las conexiones con el cliente, e indicará que se a establecido la conexión con el cliente.

Se crea un gestor que controlará las peticiones del cliente, asi se mostrarán las solicitudes que haga el cliente. Este manejador se enviará a un hilo independiente para poder gestionar las peticiones de forma independiente.

```java
// Se inicializa el puerto, se utiliza el 8086
private static final int PUERTO = 8086;

/**
 * Método main para ejecutar la aplicación y establecer el servidor.
 */
Run | Debug
public static void main(String[] args) {
    // Se controla el serverSoket, con un try-whit-resource para finalziar el
    // servidor automaticamente.
    // Se crea el servidor
    try (ServerSocket serverSocket = new ServerSocket(PUERTO)) {
        // Se imprime que se ha establecido la conexión
        System.out.println("Servidor HTTP escuchando en el puerto " + PUERTO);
        // Bucle infinito que se mantiene a la escucha.
        while (true) {
            // Se crea un cliente
            Socket clienteSocket = serverSocket.accept();
            // Se informa que el cliente se ha conectado
            System.out.println("Cliente conectado desde: " + clienteSocket.getInetAddress());
            // Se crea el getor para el cliente
            ClientHandler manejador = new ClientHandler(clienteSocket);
            // Se ejecuta en un hilo independiente
            new Thread(manejador).start();
        }

        // En caso de error se muestra un mensaje de error.
    } catch (IOException e) {
        System.err.println("Error al iniciar el servidor: " + e.getMessage());
    }
}
```

Ejecución de la aplicación

A continuación, se va a ejecutar el servidor para comprobar su funcionamiento. Lo primero que vamos hace es comprobar si el servidor se ejecuta correctamente.

Como se puede observar en la siguiente imagen se puede ver como el servidor devuelve un mensaje informado de que está a la escucha.

```
C:\00-PROGRAMACION\PSP\TAREAS\Tarea_5\ServidrorHTTP> c: && cd c:\00-PROGRAMACION\PSP\TAREAS\Tarea_5\ServidrorHTTP
\dmiap\AppData\Local\Temp\cp_csxo9gmb0qqbtiqfttc02vvv2.argfile server.ServerHTTP "
Servidor HTTP escuchando en el puerto 8086
```

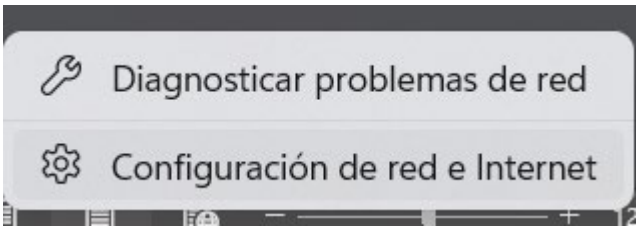
A continuación, se realizará una conexión a través del navegador

El servidor mostrará el siguiente mensaje de que se ha establecido una conexión con los datos del cliente que se ha conectado.

```
C:\00-PROGRAMACION\PSP\TAREAS\Tarea_5\ServidrorHTTP> c: && cd c:\00-PROGRAMACION\PSP\TAREAS\Tarea_5\S
\dmiap\AppData\Local\Temp\cp_csxo9gmb0qqbtiqfttc02vvv2.argfile server.ServerHTTP "
Servidor HTTP escuchando en el puerto 8086
Cliente conectado desde: /0:0:0:0:0:0:0:1
Cliente conectado desde: /0:0:0:0:0:0:0:1
Petición: GET / HTTP/1.1
Petición: GET /resource/personalImagen.jpg HTTP/1.1
```

Ahora vamos a conectarnos a través de una table, lo primero que haremos para poder conectarnos desde otro dispositivo es obtener la IP a la que esta conectado al Router.

Para eso pulsamos con el botón derecho sobre la conexión y seleccionamos la opción “Configuración de red e Internet”. Esto abrirá la configuración de la red y buscamos el IPv4.



Con este dato ya se podrá obtener la conexión desde otro dispositivo conectado a la red. Si se configurara el Router para habilitar el puerto, se podría crear acceder desde internet. Pero este paso se va a omitir porque no lo veo necesario para la tarea.

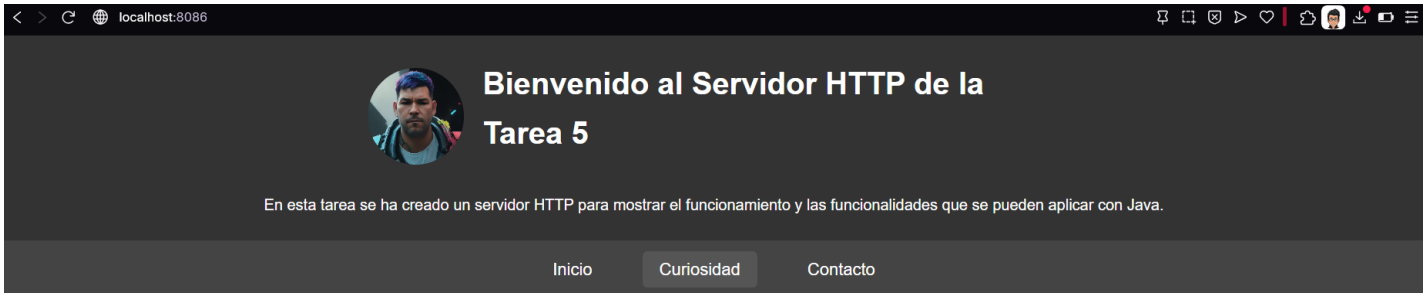
En el navegador de la Tablet escribiremos la dirección IPv4 del ordenador seguida de dos puntos y el número de puerto (192.168.1.132:8086).

En el servidor veremos que se ha conectado un nuevo cliente. Además, se puede observar que la IP es diferente a la consulta que ha se ha realizado desde el ordenador.

```
Petición: GET /resource/personalImagen.jpg HTTP/1.1
Cliente conectado desde: /192.168.1.129
Petición: GET / HTTP/1.1
Cliente conectado desde: /192.168.1.129
Petición: GET /resource/personalImagen.jpg HTTP/1.1
Cliente conectado desde: /192.168.1.129
Petición: GET /favicon.ico HTTP/1.1
```

Seguidamente vamos a llamar a las siguientes vistas y enviar los “GET” con las peticiones que se y ver cómo actúa el servidor.

Vamos a solicitar la página curiosidad.



Para esto se pulsará sobre la opción curiosidad del menú que se ha creado para mostrar las páginas y realizar las peticiones.

El servidor recibirá la petición y devolverá la vista con la solicitud.

```
Petición: GET /favicon.ico HTTP/1.1
Cliente conectado desde: /0:0:0:0:0:0:0:1
Cliente conectado desde: /0:0:0:0:0:0:0:1
Petición: GET /curiosidad HTTP/1.1
Petición: GET /resource/personalImagen.jpg HTTP/1.1
```

Se solicita la página contacto

Al igual que la pagina anterior se utilizará el menú para realizar la solicitud y devolver la página Contacto.

```
Cliente conectado desde: /0:0:0:0:0:0:0:1
Petición: GET /contacto HTTP/1.1
Petición: GET /resource/personalImagen.jpg HTTP/1.1
Cliente conectado desde: /0:0:0:0:0:0:0:1
Petición: GET /resource/contacto.jpg HTTP/1.1
```

Por último, se ha agregado un footer a la página para poder abrir la página web de no encontrado, o también se puede introducir cualquier parámetro tras la barra "/" ejemplo se insertará la palabra buscar tras la barra "/".

```
Cliente conectado desde: /0:0:0:0:0:0:0:1
Petición: GET /buscar HTTP/1.1
Petición: GET /resource/personalImagen.jpg HTTP/1.1
Cliente conectado desde: /0:0:0:0:0:0:0:1
Petición: GET /resource/error404.jpg HTTP/1.1
```

De este modo hemos conseguido hacer que nuestro servidor funcione correctamente.

HTMLs

Para las páginas web se creado una página base que se le cambiará el contenido, para ir mostrando los diferentes contenidos. Esta base no permite reutilizar el código y evitar tener que crear html muy extensos.