Servidor WEB

Elaborada por: M. en C. Ukranio Coronilla

En esta práctica vamos a implementar un servidor web al cual se pueda acceder mediante un navegador web de manera interactiva mediante la incorporación de HTML, JavaScript y CSS.

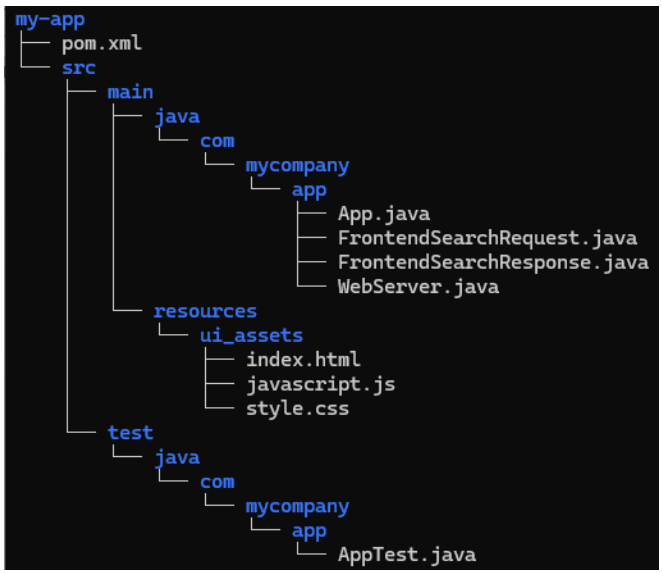
Descargue el archivo my-app.rar y para descomprimirlo instale primero la librería unrar:

```
sudo apt-get install unrar
```

Posteriormente descomprima con:

```
unrar x my-app.rar
```

El proyecto maven adjunto contiene un servidor web funcional el cual contiene el siguiente árbol de directorios:



Para crear el archivo jar sólo debe ejecutar dentro de la carpeta my-app :

```
mvn clean compile assembly:single
```

y como en ejemplos anteriores dentro de la carpeta `target` se encontrará el archivo JAR con todas las dependencias. Al ejecutar el archivo JAR se levantará un servidor web en el puerto 3000.

Acceda al servidor mediante un navegador web para verificar que funciona correctamente y después retomando sus conocimientos en aplicaciones web, revise con un editor de texto los assets que se encuentran en la carpeta `ui_assets` para verificar que se están incorporando en la aplicación web. Dele un vistazo también a los cuatro códigos Java disponibles en la carpeta `app` para hacerse una idea general del funcionamiento de esta aplicación web.

Posteriormente realice las siguientes actividades y suba al teams de su equipo las respuestas correspondientes (se recomienda también comentar su código con los conocimientos obtenidos al responderlas):

a) ¿Qué devuelve el servidor cuando intento acceder a los endpoints declarados en el archivo WebServer.java: `/status`, `/`, `/ui_assets/` y `/procesar_datos`?
b) Imprima un mensaje del tipo "`Se recibió el método HTTP x al acceder al endpoint y`" dentro del archivo *WebServer.java* para imprimir el método HTTP que se envía y el endpoint al que accede el cliente web en una consulta. Agregue la captura de pantalla como comprobante para visualizar el orden y tipo de estas solicitudes.
c) En el método `handleRequestForAsset` del archivo *WebServer.java* imprima el valor de la variable `String asset` para ver cuál es el asset que solicita el cliente web. Adicionalmente en el método `sendResponse` agregue la impresión del número de bytes enviados para verificar que coinciden en tamaño con los archivos enviados. Agregue la captura de pantalla como comprobante.
d) En el código *WebServer.java* ¿cuál es el propósito del método readUiAsset y en particular qué hace la instrucción `getClass().getResourceAsStream(asset)`?
e) ¿Cuál es el propósito del método `addContentType()` ?
f) Después de recibir los assets, al introducir la frase en el navegador y dar click en enviar, ¿qué método HTTP se ejecuta, a qué endpoint accede y qué método del código Java se ejecuta?
g) Para el inciso anterior ¿qué valor tiene el header “Content-type” cuando el método POST llega al servidor (imprimir su valor y enviar la captura de pantalla)? *Sugerencia: Revisar como se accede a los headers de una solicitud POST en el código explicado en el video “Servidor HTTP el código fuente”*.

h) En la práctica “Google Gson” utilizamos la librería de Gson para convertir objetos JSON en objetos Java. En este caso en su lugar se utiliza la librería **Jackson** debido a que es más rápida que Gson y más conveniente para las altas cargas de trabajo que puede tener un servidor. Investigue para el código dentro del método `handleTaskRequest` en la clase WebServer: ¿Para qué se utiliza el método de Jackson `readValue`? ¿qué se almacena en la variable `String frase` (imprímala)? ¿qué se almacena en el objeto `frontendSearchResponse`? ¿Para qué se utiliza el método de Jackson `writeValueAsBytes`?

i) Además de los assets HTML, CSS y JS, el navegador **Chrome** solicita un cuarto asset (algunos navegadores no lo solicitan). ¿Cuál es y que significa? Realice las modificaciones necesarias para que también se envíe el cuarto asset y se muestre en el navegador. Envíe la captura de pantalla correspondiente para demostrar que funciona.