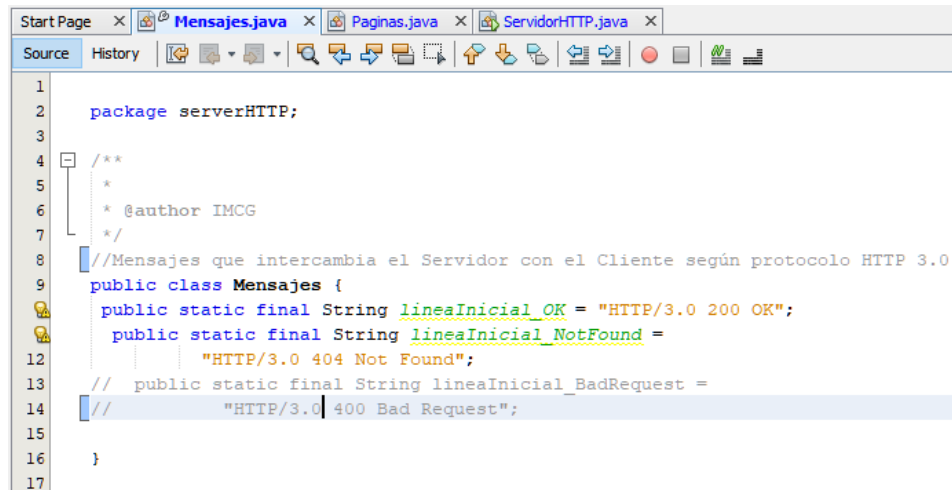


EJERCICIO 1

Se nos pide modificar el proyecto del apartado 5.1 del temario para añadir la fecha de hoy en la cabecera de respuesta del servidor.

Lo primero que voy a hacer es cambiar dentro de la clase mensajes, el mensaje inicial que emite el servidor de http 1.1 a http 3.0 ya que cuando he probado la aplicación el navegador considera que http 1.1 es poco fiable y no me deja ejecutarlo por motivos de seguridad. La clase quedaría así:

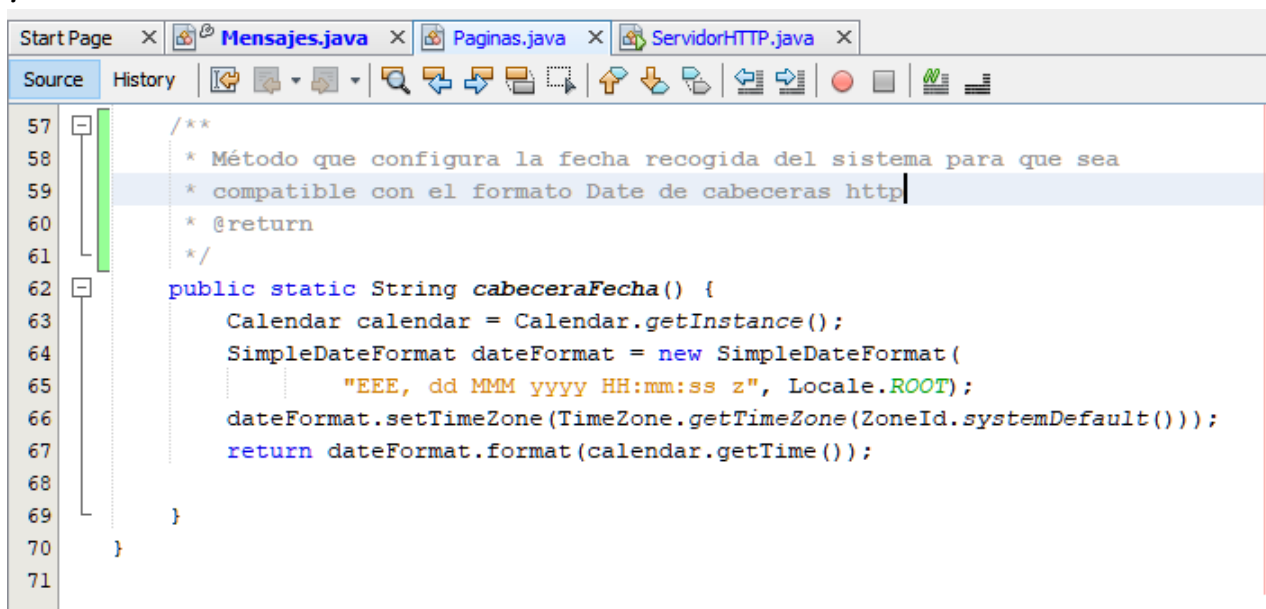


```

1
2  package serverHTTP;
3
4  /**
5   *
6   * @author IMCG
7   */
8  //Mensajes que intercambia el Servidor con el Cliente según protocolo HTTP 3.0
9  public class Mensajes {
10     public static final String lineaInicial_OK = "HTTP/3.0 200 OK";
11     public static final String lineaInicial_NotFound =
12         "HTTP/3.0 404 Not Found";
13     // public static final String lineaInicial_BadRequest =
14     // "HTTP/3.0 400 Bad Request";
15
16 }
17

```

Lo siguiente que voy a hacer será crear un método que nos devuelva la fecha de hoy, con el formato válido para una cabecera de http (*Date: Fri, 31 Dec 2003 23:59:59 GMT*):



```

57
58  /**
59   * Método que configura la fecha recogida del sistema para que sea
60   * compatible con el formato Date de cabeceras http
61   * @return
62   */
63  public static String cabeceraFecha() {
64      Calendar calendar = Calendar.getInstance();
65      SimpleDateFormat dateFormat = new SimpleDateFormat(
66          "EEE, dd MMM yyyy HH:mm:ss z", Locale.ROOT);
67      dateFormat.setTimeZone(TimeZone.getTimeZone(ZoneId.systemDefault()));
68      return dateFormat.format(calendar.getTime());
69  }
70
71

```

Devuelve un String con la cadena de fecha en el formato cabecera

Ya solo nos queda añadir la cabecera “Date” a la respuesta del servidor dentro de la clase servidorHTTP:

```

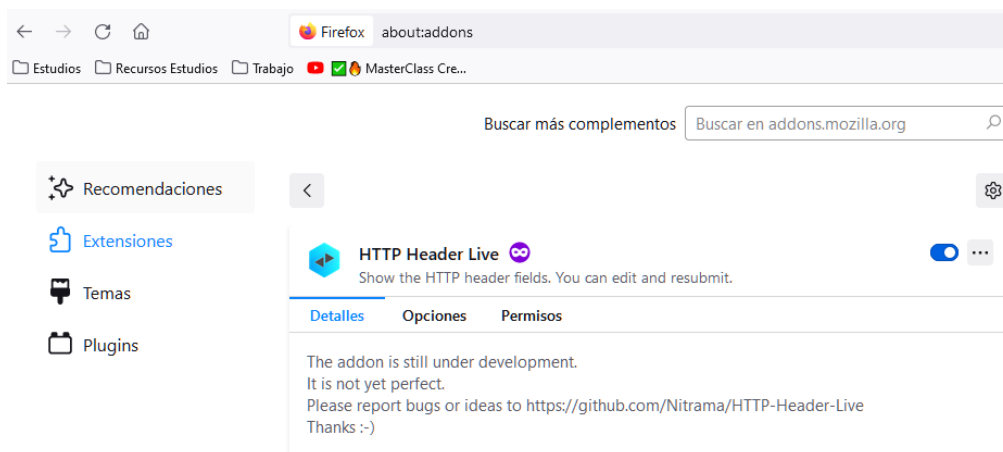
85
86 //si corresponde a la página de inicio
87 if (peticion.length() == 0 || peticion.equals("/")) {
88 //sirve la página
89 html = Paginas.html_index;
90 printWriter.println(Mensajes.lineaInicial_OK);
91 printWriter.println(Paginas.primerCabecera);
92 printWriter.println("Content-Length: " + html.length() + 1);
93 //Añadida cabecera Date al proyecto
94 printWriter.println("Date: "+Paginas.cabeceraFecha());
95 printWriter.println("\n");
96 printWriter.println(html);
97 } //si corresponde a la página del Quijote
98 else if (peticion.equals("/quijote")) {
99 //sirve la página
100 html = Paginas.html_quijote;
101 printWriter.println(Mensajes.lineaInicial_OK);
102 printWriter.println(Paginas.primerCabecera);
103 printWriter.println("Content-Length: " + html.length() + 1);
104 //Añadida cabecera Date al proyecto
105 printWriter.println("Date: "+Paginas.cabeceraFecha());
106 printWriter.println("\n");
107 printWriter.println(html);
108 } //en cualquier otro caso
109 else {
110 //sirve la página
111 html = Paginas.html_noEncontrado;
112 printWriter.println(Mensajes.lineaInicial_NotFound);
113 printWriter.println(Paginas.primerCabecera);
114 printWriter.println("Content-Length: " + html.length() + 1);
115 //Añadida cabecera Date al proyecto
116 printWriter.println("Date: "+Paginas.cabeceraFecha());
117 printWriter.println("\n");
118 printWriter.println(html);
119 }
120
121 }
122

```

Añadimos la cabecera a cada una de las posibles respuestas

Pruebas de la aplicación:

Para probar este ejercicio y poder ver las cabeceras he instalado como complemento dentro del navegador Firefox el llamado “HTTP live header” que muestra en una ventana las cabeceras tanto enviadas como recibidas por el servidor:



Ejecuto la aplicación en Netbeans, lo que me va a crear el servidor http y se va a quedar en espera de petición cliente:

```

Output - Ejercicio_1 (run) x
init:
Deleting: D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_1\build\build-jar.properties
deps-jar:
Updating property file: D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_1\build\build-jar.properties
Compiling 1 source file to D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_1\build\classes
compile:
run:
El Servidor WEB se está ejecutando y permanece a la escucha por el puerto 8066.
Escribe en la barra de direcciones de tu explorador preferido:

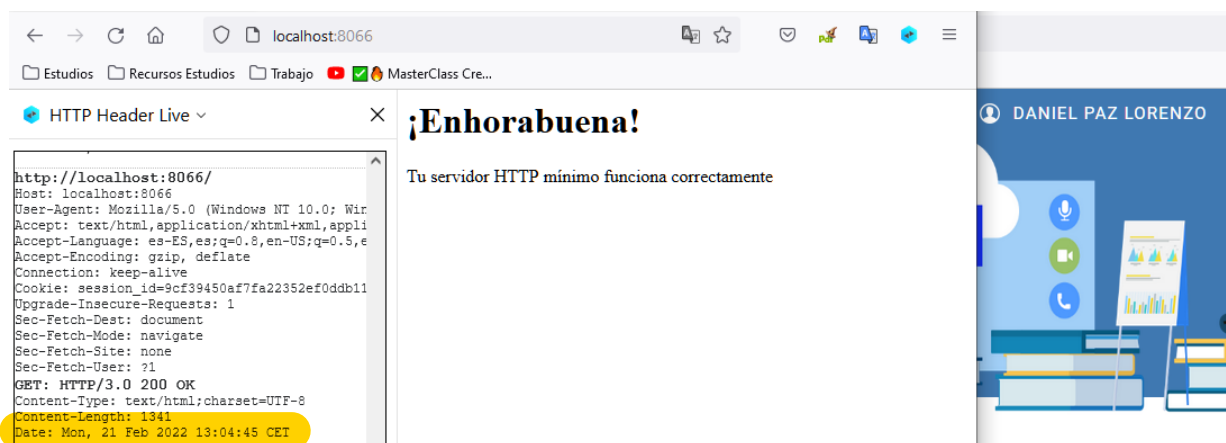
http://localhost:8066
para solicitar la página de bienvenida

http://localhost:8066/quijote
para solicitar una página del Quijote,

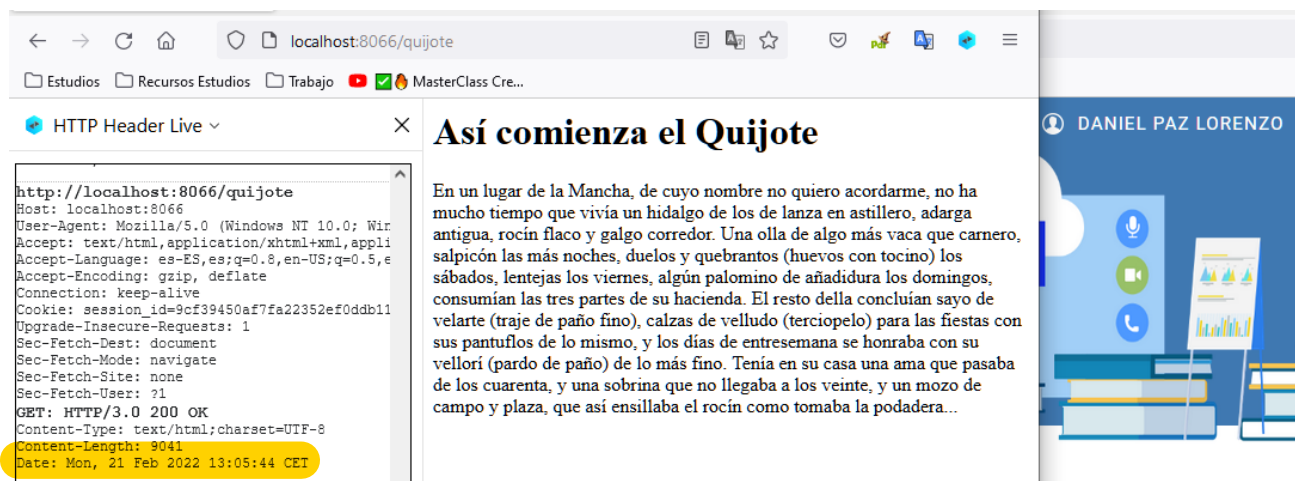
http://localhost:8066/q
para simular un error
Date: Mon, 21 Feb 2022 12:57:40 CET

```

Solicito las 3 páginas que contiene el servidor a través de mi navegador Firefox:



Vemos el resultado de la consulta <http://localhost:8066> con las cabecera Date a la izquierda



Vemos el resultado de la consulta <http://localhost:8066/quijote> con las cabecera Date a la izquierda

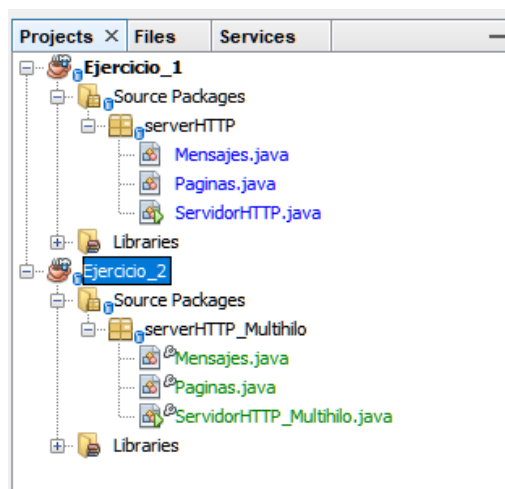


Vemos el resultado de la consulta `http://localhost:8066 /q` con las cabecera `Date` a la izquierda

EJERCICIO 2

En este apartado se nos pide modificar el ejercicio anterior para que el servidor acepte concurrencia de peticiones de cliente, es decir que sea multihilo.

Lo primero que haré será crear un nuevo proyecto donde voy a copiar las clases del apartado anterior:



Las clases “Mensajes” y “Páginas” las voy a reutilizar tal cual están sin realizar ningún cambio porque no es necesario. La que voy a modificar será la clase “servidorHTTP” renombrada como “servidorHTTP_Multihilo”. Voy a decirle que herede de la clase Thread para hacerla multihilo y añadiré un constructor que reciba como parámetro el socket de cliente:

```
class ServidorHTTP_Multihilo extends Thread {

    Socket socketCliente;

    public ServidorHTTP_Multihilo(Socket socketCliente) {
        this.socketCliente = socketCliente;
    }
}
```

Añado el método “Run” como sobrescrito de la clase Thread y dentro de el voy a colocar todo el código que anteriormente contenía el método de “procesarPetición()” porque será este método el que se encargue ahora de procesar las peticiones de cada hilo de cliente. Una vez hecho esto podemos ya borrar el método “procesarPetición()”:



```

60      @Override
61      public void run() {
62          InputStreamReader inSR = null;
63          try {
64              //variables locales
65              String peticion;
66              String html;
67              //Flujo de entrada
68              inSR = new InputStreamReader(
69                  socketCliente.getInputStream());
70              //espacio en memoria para la entrada de peticiones
71              BufferedReader bufLeer = new BufferedReader(inSR);
72              //objeto de java.io que entre otras características, permite escribir
73              //'línea a línea' en un flujo de salida
74              PrintWriter printWriter = new PrintWriter(
75                  socketCliente.getOutputStream(), true);
76              //mensaje petición cliente
77              peticion = bufLeer.readLine();
78              //para compactar la petición y facilitar así su análisis, suprimimos todos
79              //los espacios en blanco que contenga
80              peticion = peticion.replaceAll(" ", "");
81              //si realmente se trata de una petición 'GET' (que es la única que vamos a
82              //implementar en nuestro Servidor)
83              if (peticion.startsWith("GET")) {
84                  //extrae la subcadena entre 'GET' y 'HTTP/1.1'
85                  peticion = peticion.substring(3, peticion.lastIndexOf("HTTP"));
86
87                  //si corresponde a la página de inicio
88                  if (peticion.length() == 0 || peticion.equals("/")) {
89                      //sirve la página
90                      html = Paginas.html_index;
91                      printWriter.println(Mensajes.lineaInicial_OK);
92                      printWriter.println(Paginas.primerCabecera);
93                      printWriter.println("Content-Length: " + html.length() + 1);
94                      printWriter.println("Date: " + Paginas.cabeceraFecha());
95                      printWriter.println("\n");
96                      printWriter.println(html);
97                  } //si corresponde a la página del Quijote
98                  else if (peticion.equals("/quijote")) {
99                      //sirve la página
100                     html = Paginas.html_quijote;
101                     printWriter.println(Mensajes.lineaInicial_OK);
102                     printWriter.println(Paginas.primerCabecera);
103                     printWriter.println("Content-Length: " + html.length() + 1);
104                     printWriter.println("Date: " + Paginas.cabeceraFecha());
105                     printWriter.println("\n");

```

Método Run que va a procesar la petición del cliente

Ya solo queda modificar el método “main” para que procese cada solicitud de cliente como un hilo y así poder aceptar peticiones de manera concurrente:

```

32  /**
33  *
34  * procedimiento principal que crea un hilo a cada petición entrante y
35  * asigna un socket cliente, por donde se enviará la respuesta una
36  * vez procesada
37  *
38  * @param args the command line arguments
39  */
40  public static void main(String[] args) throws IOException, Exception {
41
42      //Asociamos al servidor el puerto 8066
43      ServerSocket socServidor = new ServerSocket(8066);
44      imprimeDisponible();
45      Socket socCliente;
46
47      //ante una petición entrante, procesa la petición por el socket cliente
48      //por donde la recibe
49      while (true) {
50          //a la espera de peticiones
51          socCliente = socServidor.accept();
52          //atiendo un cliente
53          System.out.println("Atendiendo al cliente ");
54          //crea un nuevo hilo para despacharla por el socketCliente que le asignó
55          new ServidorHTTP_Multihilo(socCliente).start();
56          //Muestra mensaje una vez terminada la petición
57          System.out.println("cliente atendido");
58      }
59  }

```

Pruebas de la aplicación:

Una vez tenemos configurada ya la nueva aplicación vamos a ejecutarla lo que nos va a generar el servidor:

```

serverHTTP_Multihilo.ServidorHTTP_Multihilo
Output - Ejercicio_2 (run)
Updating property file: D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_2\build\build-jar.properties
Created dir: D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_2\build\classes
Created dir: D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_2\build\empty
Created dir: D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_2\build\generated-sources\ap-source-output
Compiling 3 source files to D:\Curso DAM\Repositorios GitHub\Asignaturas-2-DAM\PSP\Tareas\Tarea 5\TAREA_5_PSP\Ejercicio_2\build\classes
compile:
run:
El Servidor WEB se está ejecutando y permanece a la escucha por el puerto 8066.
Escribe en la barra de direcciones de tu explorador preferido:

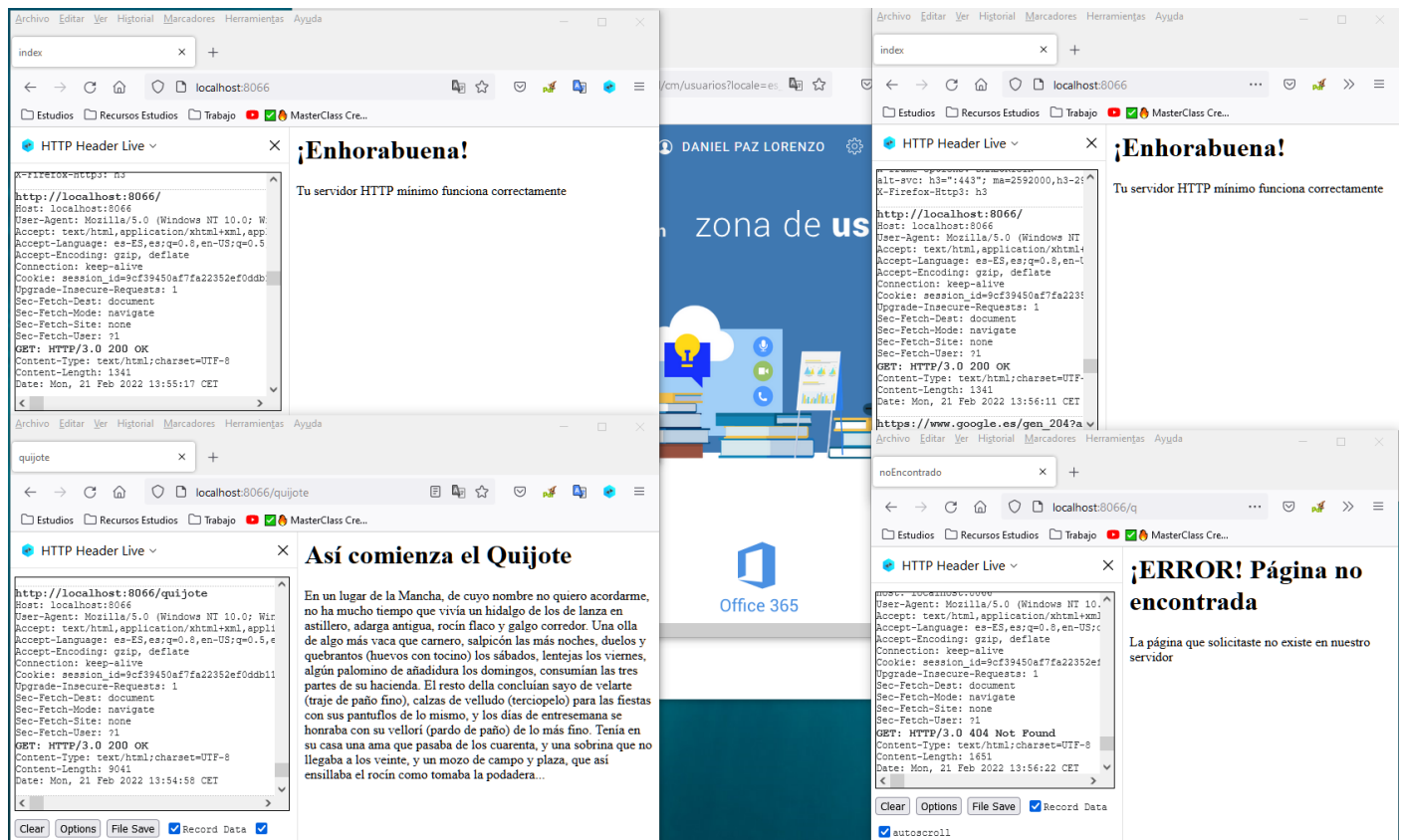
http://localhost:8066
para solicitar la página de bienvenida

http://localhost:8066/quijote
para solicitar una página del Quijote,

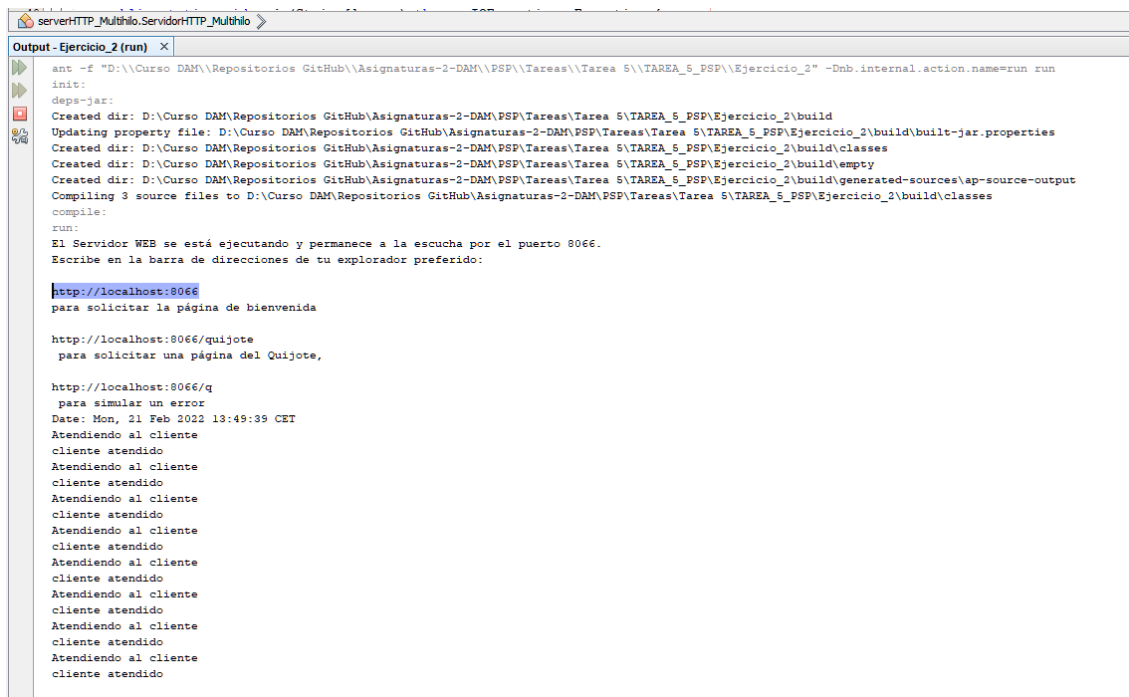
http://localhost:8066/q
para simular un error
Date: Mon, 21 Feb 2022 13:49:39 CET

```


Para probar la concurrencia abriremos 4 ventanas de nuestro navegador y en cada una solicitaremos una página al servidor creado por la aplicación:



Podemos ver en el navegador las respuestas del servidor a cada petición



A la vez en Netbeans comprobamos en la salida que se han aceptado y atendido las peticiones de cada cliente