

Laboratorio 5

Introducción a proyectos web

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Profesor:

Iván Darío Viasus Quintero

Curso:

CVDS - 2

Fecha De Entrega:

21-09-2022

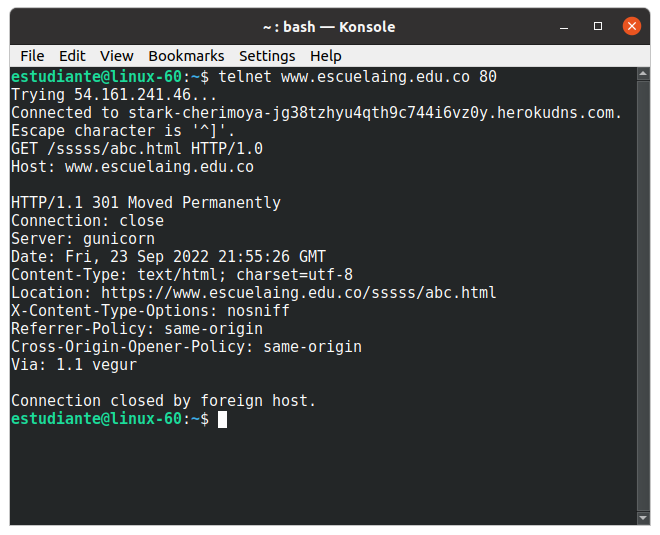
PARTE I. - JUGANDO A SER UN CLIENTE HTTP

1. Abra una terminal Linux o consola de comandos Windows.
2. Realice una conexión síncrona TCP/IP a través de Telnet al siguiente servidor:

* Host: www.escuelaing.edu.co
* Puerto: 80

Teniendo en cuenta los parámetros del comando telnet:

|  |
| --- |
| telnet HOST PORT |



1. Antes de que el servidor cierre la conexión por falta de comunicación:

* Revise la página 36 del RFC del protocolo HTTP, sobre cómo realizar una petición GET. Con esto, solicite al servidor el recurso ‘sssss/abc.html’, usando la versión 1.0 de HTTP.
* Asegúrese de presionar ENTER dos veces después de ingresar el comando.

Texto

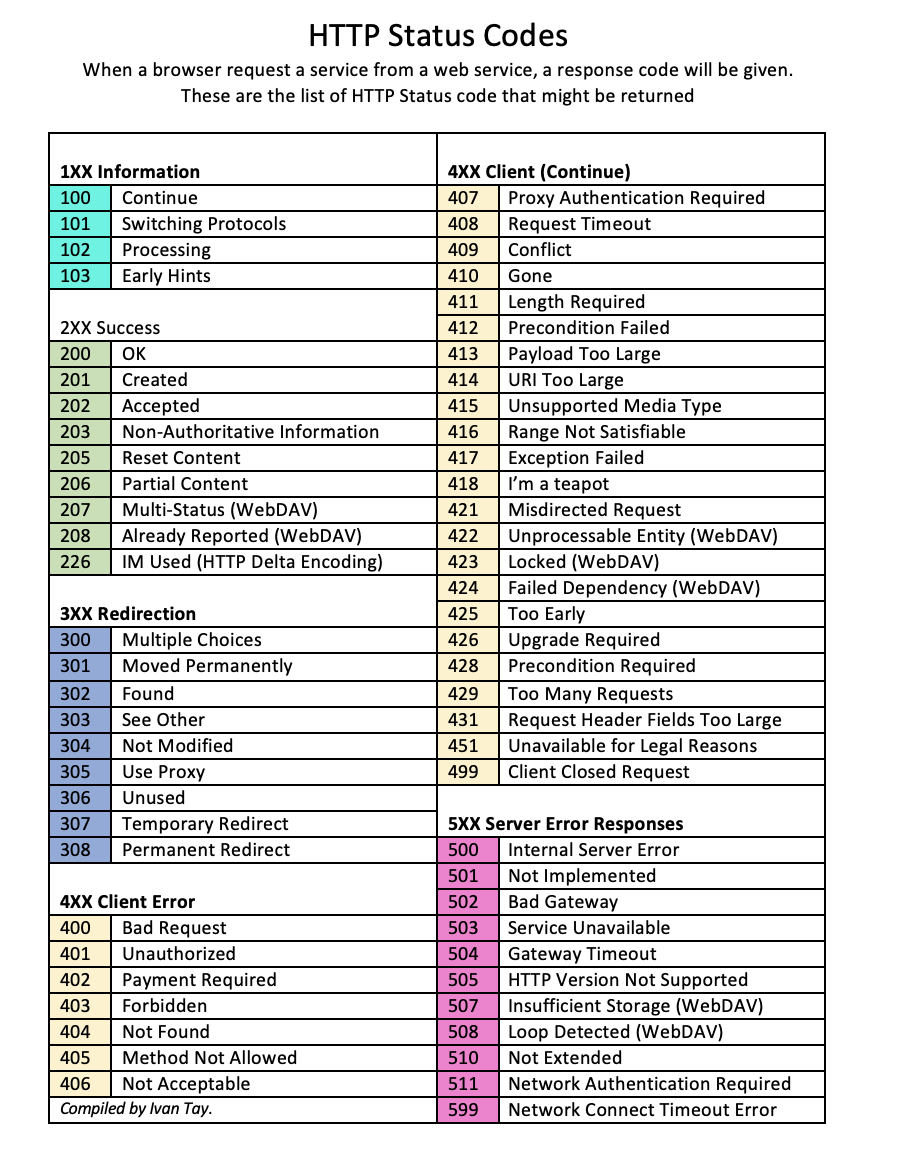
Descripción generada automáticamente

* Revise el resultado obtenido. ¿Qué Código de error sale?, revise el significado del mismo en la lista de códigos de estado HTTP.

**301 (MOVED PERMANENTLY)** los datos solicitados han sido transferidos a una nueva dirección.

* ¿Qué otros códigos de error existen?, ¿En qué caso se manejarán?
* 100s: Códigos informativos que indican que la solicitud iniciada por el navegador continúa.
* 200s: Los códigos con éxito regresaron cuando la solicitud del navegador fue recibida, entendida y procesada por el servidor.
* 300s: [Códigos de redireccionamiento](https://kinsta.com/es/blog/redirecciones-de-wordpress/) devueltos cuando un nuevo recurso ha sido sustituido por el recurso solicitado.
* 400s: Códigos de error del cliente que indican que hubo un [problema con la solicitud](https://kinsta.com/es/base-de-conocimiento/400-bad-request/).
* 500s: Códigos de error del servidor que indican que la solicitud fue aceptada, pero que [un error en el servidor impidió que se cumpliera](https://kinsta.com/es/blog/500-internal-server-error/).

Dentro de cada uno de estos tipos, existe una variedad de códigos de servidor y pueden ser devueltos por el servidor. Cada código individual tiene un significado específico y único



1. Realice una nueva conexión con telnet, esta vez a:

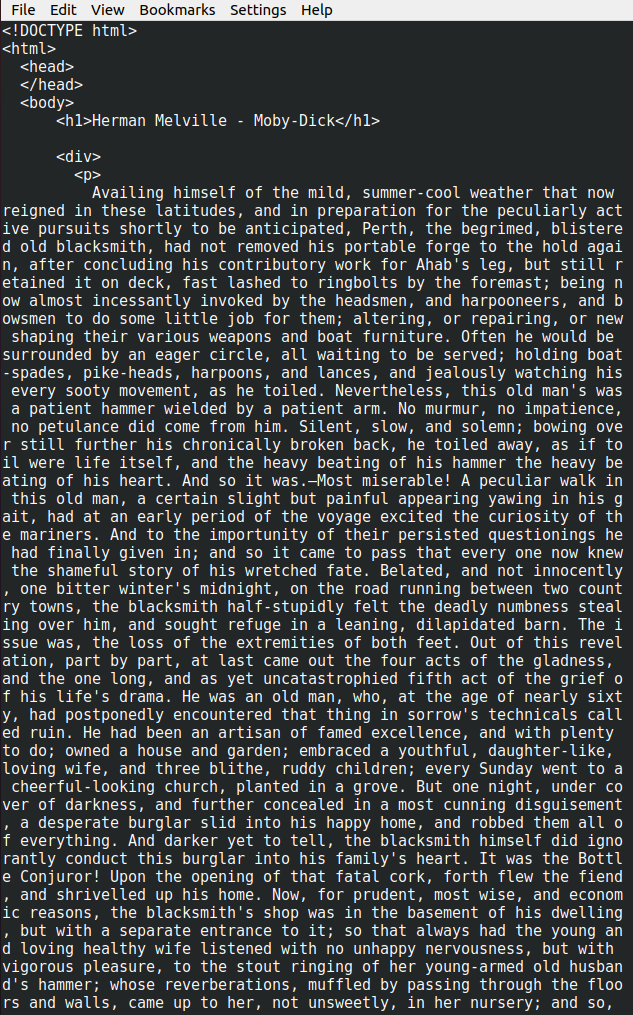
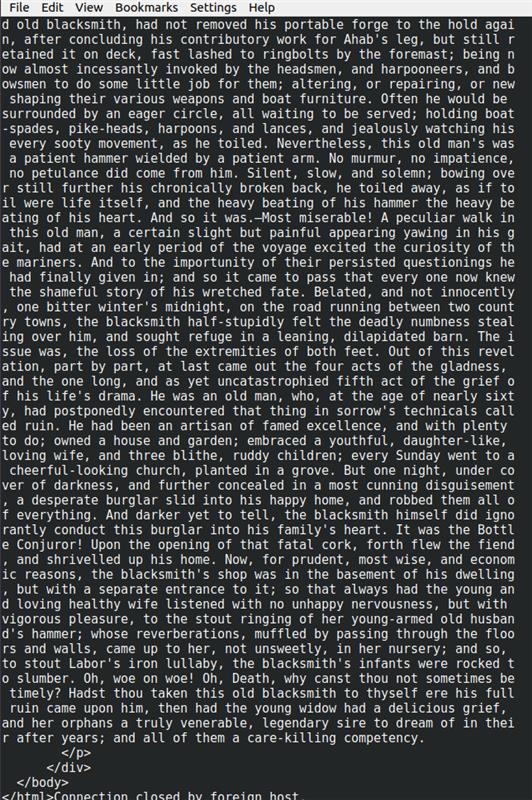
* Host: [www.httpbin.org](http://www.httpbin.org)
* Puerto: 80
* Versión HTTP: 1.1

Ahora, solicite (GET) el recurso /html. ¿Qué se obtiene como resultado?



Como podemos observar en esta imagen el código error que nos salió fue el 200 (OK) que significa que la solicitud se llevo a cabo de manera correcta.

Adicional a esto al solicitar el recurso /html, también nos mostró lo que podemos ver el las siguientes imágenes:

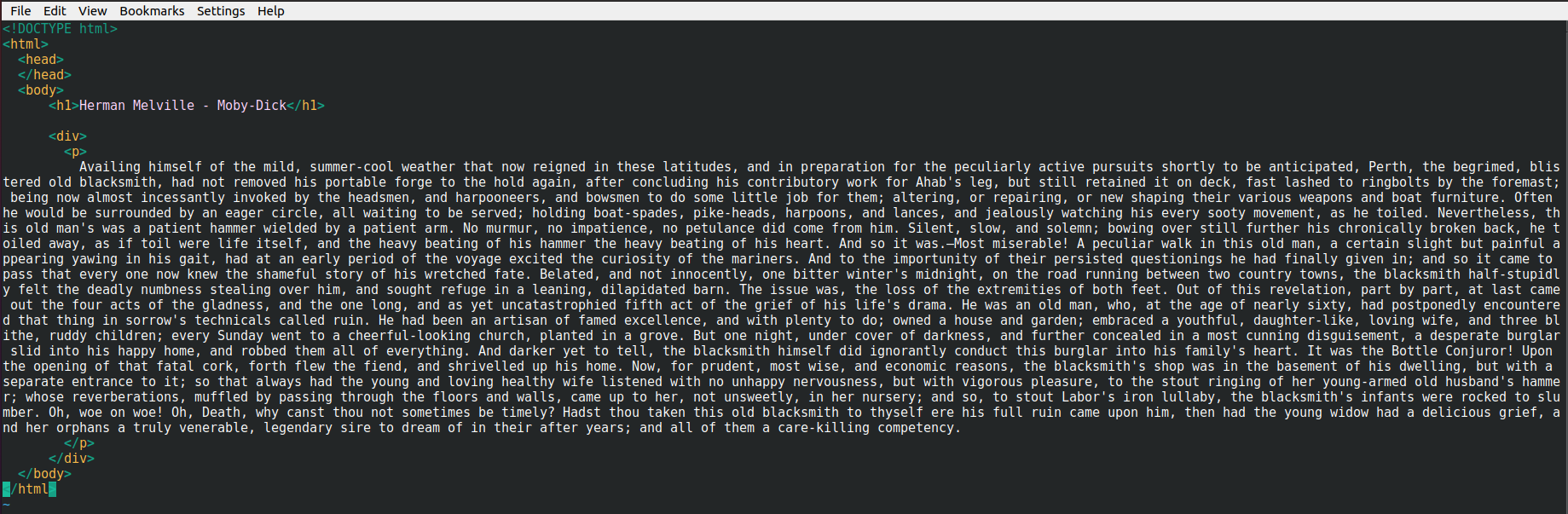
 

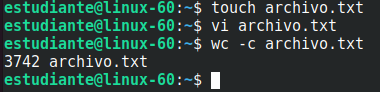
¡Muy bien!, ¡Acaba de usar del protocolo HTTP sin un navegador Web! Cada vez que se usa un navegador, éste se conecta a un servidor HTTP, envía peticiones (del protocolo HTTP), espera el resultado de las mismas, y -si se trata de contenido HTML- lo interpreta y dibuja.

1. Seleccione el contenido HTML de la respuesta y cópielo al cortapapeles CTRL-SHIFT-C. Ejecute el comando wc (word count) para contar palabras con la opción -c para contar el número de caracteres:

|  |
| --- |
| wc -c |

Pegue el contenido del portapapeles con CTRL-SHIFT-V y presione CTRL-D (fin de archivo de Linux). Si no termina el comando wc presione CTRL-D de nuevo. No presione más de dos veces CTRL-D indica que se terminó la entrada y puede cerrarle la terminal. Debe salir el resultado de la cantidad de caracteres que tiene el contenido HTML que respondió el servidor.





Como podemos observar en la imagen al ingresar el comando, nos dio como resultado que la cantidad de caracteres que tienen el contenido HTML es de 3742

Claro está, las peticiones GET son insuficientes en muchos casos. Investigue: ¿Cuál es la diferencia entre los verbos GET y POST? ¿Qué otros tipos de peticiones existen?

* **GET:** con este método, los datos que se envían al servidor**se escriben en la misma dirección URL**
* **POST:** este método introduce los parámetros **en la solicitud HTTP**para el servidor. Por ello, no quedan visibles para el usuario. Además, la capacidad del método POST es ilimitada.

**Diferencias Principales**

* El método GET coloca los parámetros dentro de la URI mientras que el método POST los agrega al cuerpo.
* GET se utiliza esencialmente para obtener la información. En cambio, el objetivo del método POST es actualizar los datos.
* Los resultados de la consulta POST no pueden marcarse, mientras que los resultados de la consulta GET pueden marcarse porque existen en forma de [URL](https://pc-solucion.es/2018/05/14/url/).
* En el método GET la información es visible en la URL lo que aumenta las vulnerabilidades y el riesgo de hacking. Por el contrario, el método POST no muestra variables en la URL y también se pueden utilizar múltiples técnicas de codificación, lo que lo hace resistente.
* Cuando se utiliza el método GET en el formulario, sólo se aceptan caracteres [ASCII](https://pc-solucion.es/2018/04/26/ascii/) en los tipos de datos. Por el contrario, el método POST no enlaza tipos de datos de formulario y permite caracteres binarios y ASCII.
* El tamaño variable en el método GET es de aproximadamente 2000 caracteres. A la inversa, el método POST permite hasta 8 Mb de tamaño variable.
* Los datos del método GET se pueden almacenar en [caché](https://pc-solucion.es/2018/05/08/cache/), mientras que los datos del método POST no.

**Otros tipos de peticiones**

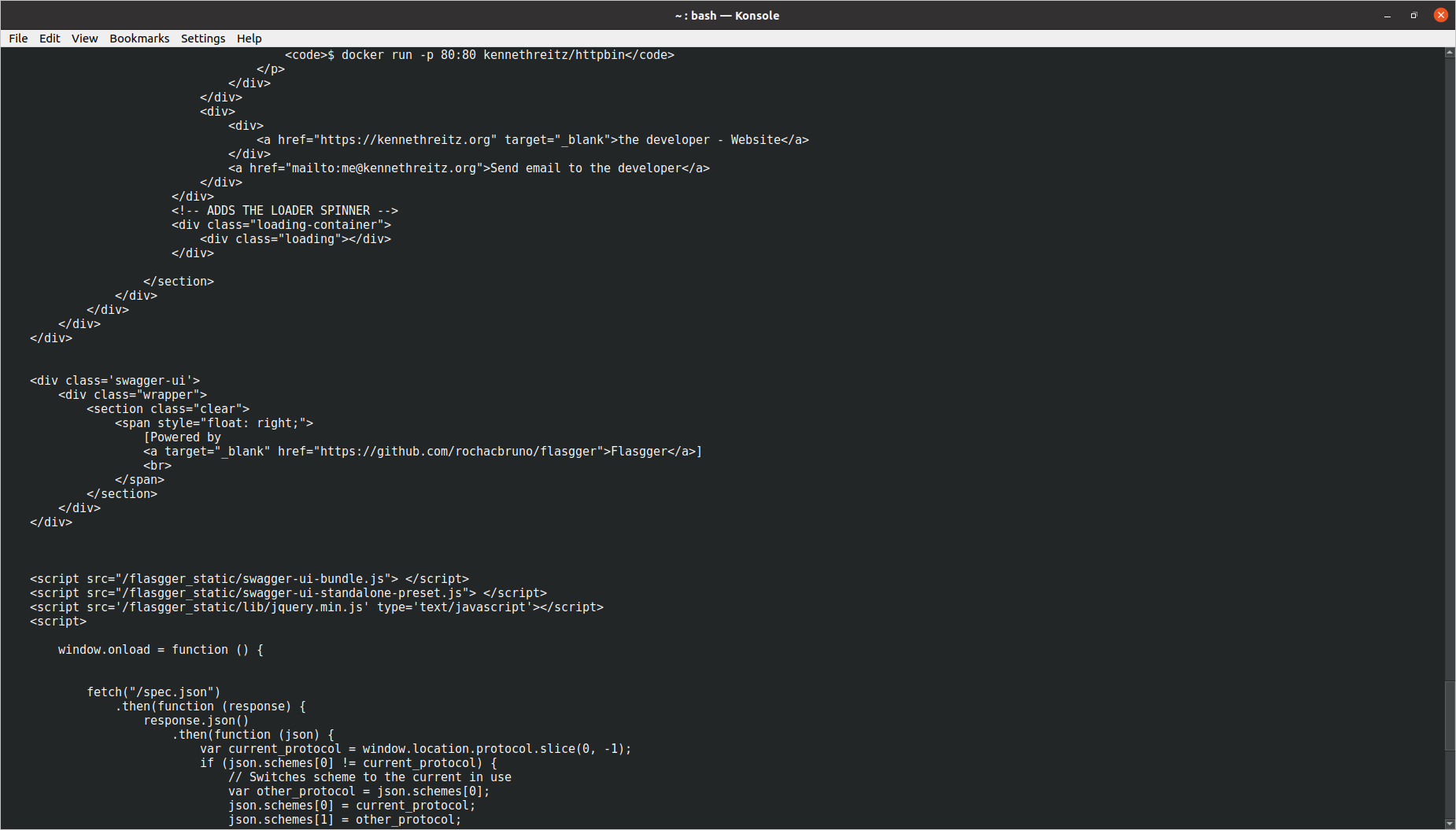
* [**HEAD**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/HEAD)**:** El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
* [**PUT**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT)**:** El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
* [**DELETE**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/DELETE)**:** El método DELETE borra un recurso en específico.
* [**CONNECT**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT)**:** El método CONNECT establece un túnel hacia el servidor identificado por el recurso.
* [**OPTIONS**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/OPTIONS)**:** El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.
* [**TRACE**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/TRACE)**:** El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
* [**PATCH**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH)**:** El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

1. En la práctica no se utiliza telnet para hacer peticiones a sitios web sino el comando curl con ayuda de la línea de comandos:

|  |
| --- |
| curl www.httpbin.org |







Texto

Descripción generada automáticamente

Utilice ahora el parámetro -v y con el parámetro -i:

|  |
| --- |
| curl -v [www.httpbin.org](http://www.httpbin.org)  curl -i www.httpbin.org |

* curl -v [www.httpbin.org](http://www.httpbin.org)

Captura de pantalla de computadora

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* curl -i [www.httpbin.org](http://www.httpbin.org)

Captura de pantalla de computadora

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

¿Cuáles son las diferencias con los diferentes parámetros?

* **curl**

Se usa en líneas de comando o scripts para transferir datos. También se utiliza en automóviles, televisores, enrutadores, impresoras, equipos de audio, teléfonos móviles, tabletas, decodificadores, reproductores multimedia y es la columna vertebral de transferencia de Internet para miles de aplicaciones de software que afectan a miles de millones de seres humanos a diario.

|  |
| --- |
| curl options URLs |

* **curl -v**

Muestra desde la conexión hasta los encabezados y los datos devueltos.

* **curl -i**

Devuelve los encabezados http

PARTE II. - HACIENDO UNA APLICACIÓN WEB DINÁMICA A BAJONIVEL.

En este ejercicio, va a implementar una aplicación Web muy básica, haciendo uso de los elementos de más bajo nivel de Java-EE (Enterprise Edition), con el fin de revisar los conceptos del protocolo HTTP. En este caso, se trata de un módulo de consulta de clientes Web que hace uso de una librería de acceso a datos disponible en un repositorio Maven local.

1. Para esto, cree un proyecto Maven nuevo usando el arquetipo de aplicación Web estándar maven-archetype-webapp y realice lo siguiente:

|  |
| --- |
| mvn archetype:generate -DgroupId=edu.eci.cvds -DartifactId=web -DarchetypeArtifactId=maven-archetype- webapp -Dpackage=edu.eci.cvds.tdd -DarchetypeVersion=1.0 |

1. Revise la clase SampleServlet incluida a continuación, e identifique qué hace:



Revise qué valor tiene el parámetro ‘urlPatterns’ de la anotación @WebServlet, pues este indica qué URLs atiende las peticiones el servlet.

Imagen que contiene Gráfico

Descripción generada automáticamente

1. En el pom.xml, modifique la propiedad "packaging" con el valor "war". Agregue la siguiente dependencia



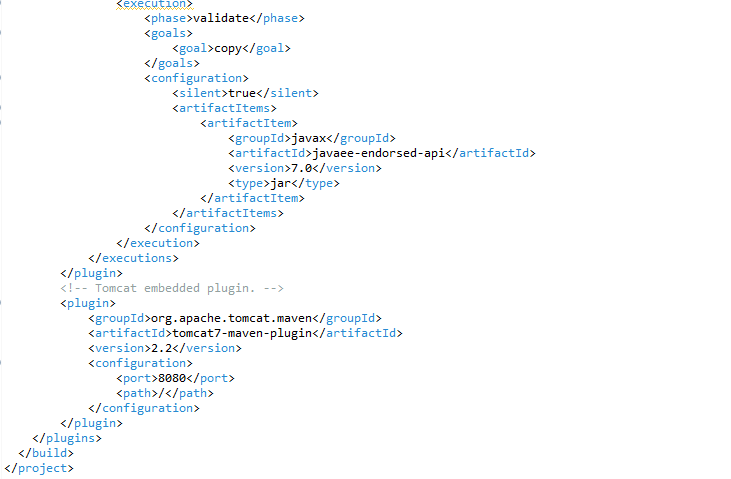
y agregue la seccion build al final del tag project en el archivo pom.xml:

Interfaz de usuario gráfica, Texto, Aplicación

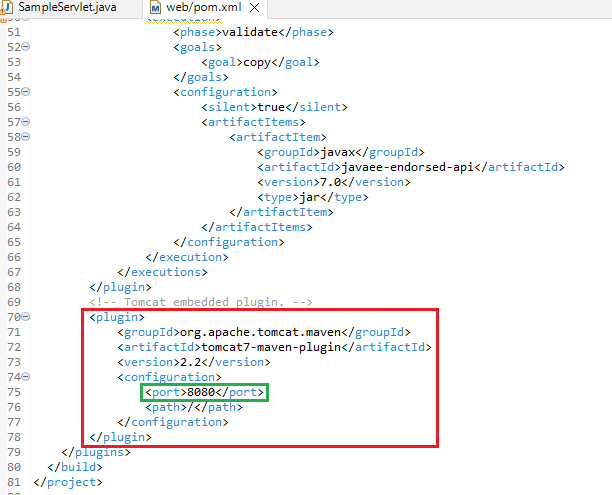
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

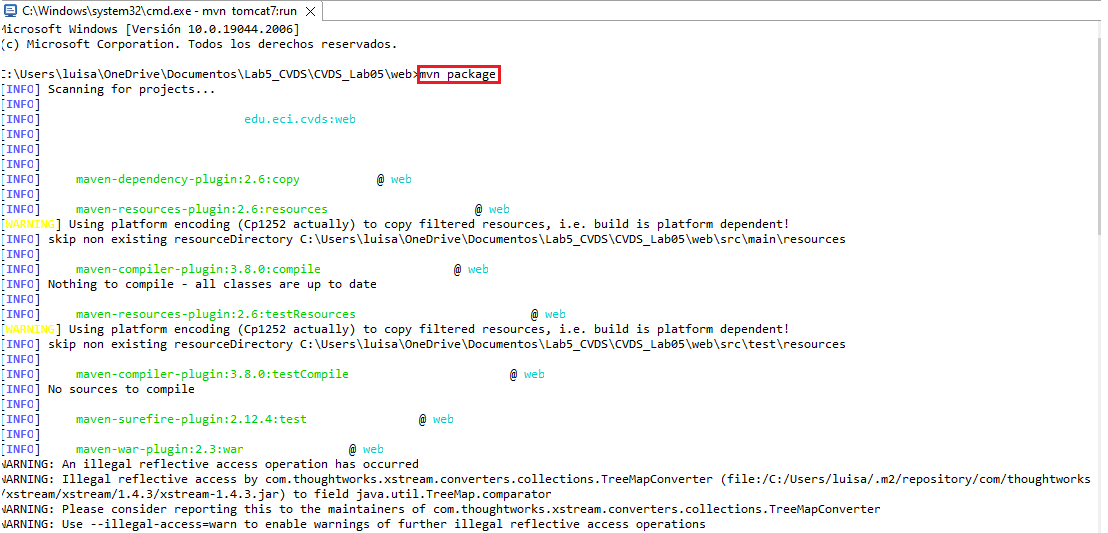


1. Revise en el pom.xml para qué puerto TCP/IP está configurado el servidor embebido de Tomcat (ver sección de plugins).



1. Compile y ejecute la aplicación en el servidor embebido Tomcat, a través de Maven con:

|  |
| --- |
| mvn package  mvn tomcat7:run |

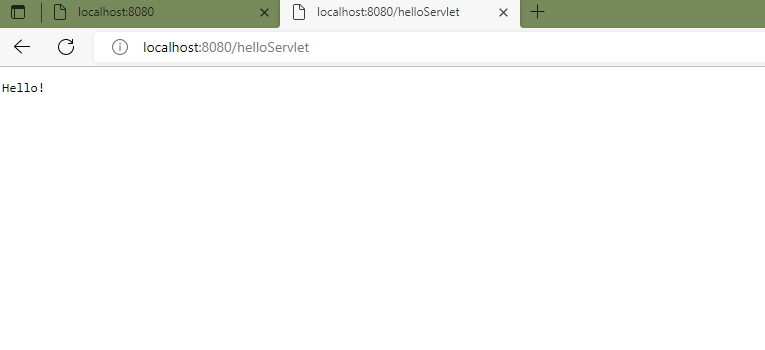


Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

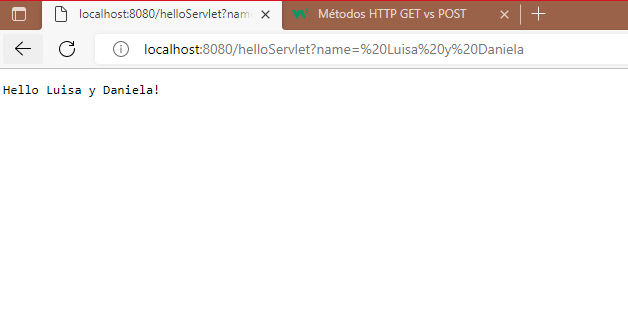
1. Abra un navegador, y en la barra de direcciones ponga la URL con la cual se le enviarán peticiones al ‘SampleServlet’. Tenga en cuenta que la URL tendrá como host ‘localhost’, como puerto, el configurado en el pom.xml y el path debe ser el del Servlet. Debería obtener un mensaje de saludo.

El resultado al escribir <http://localhost:8080/helloServlet>



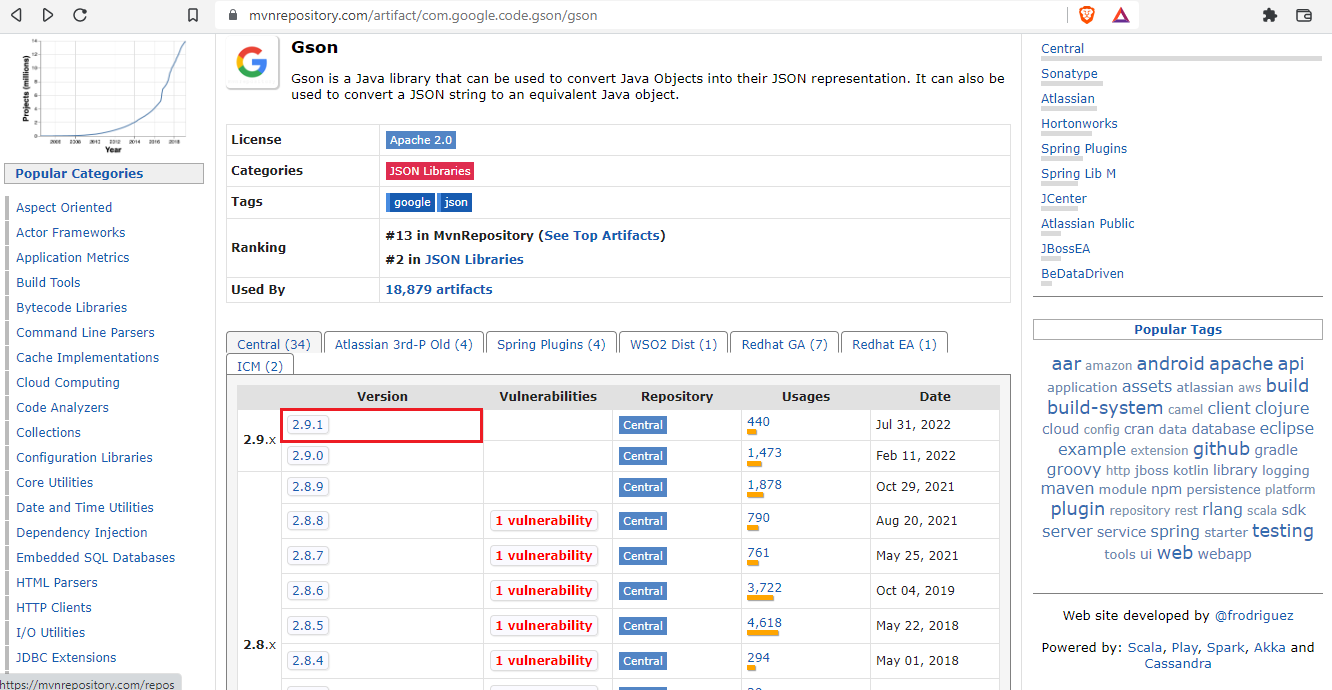
1. Observe que el Servlet ‘SampleServlet’ acepta peticiones GET, y opcionalmente, lee el parámetro ‘name’. Ingrese la misma URL, pero ahora agregando un parámetro GET (si no sabe como hacerlo, revise la documentación en <http://www.w3schools.com/tags/ref_httpmethods.asp>).

La solicitud se hace de la siguiente manera [localhost:8080/helloServlet?name= Luisa y Daniela](http://localhost:8080/helloServlet?name=%20Luisa%20y%20Daniela)



1. Busque el artefacto gson en el repositorio de maven y agregue la dependencia.

Buscamos la dependencia en el repositorio en maven y escogemos la última opción.

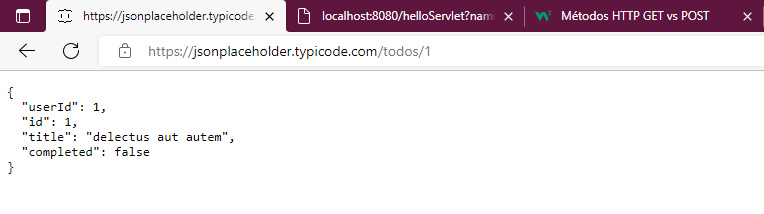




Agregamos la dependencia en pom.xml



1. En el navegador revise la dirección https://jsonplaceholder.typicode.com/todos/1. Intente cambiando diferentes números al final del path de la url.



Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

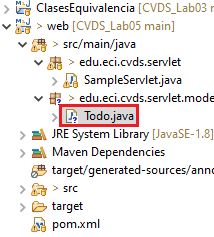
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Basado en la respuesta que le da el servicio del punto anterior, cree la clase edu.eci.cvds.servlet.model.Todo con un constructor vacío y los métodos getter y setter para las propiedades de los "To Dos" que se encuentran en la url indicada.

Creación de la clase



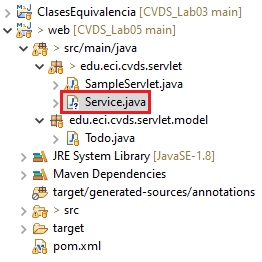
Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

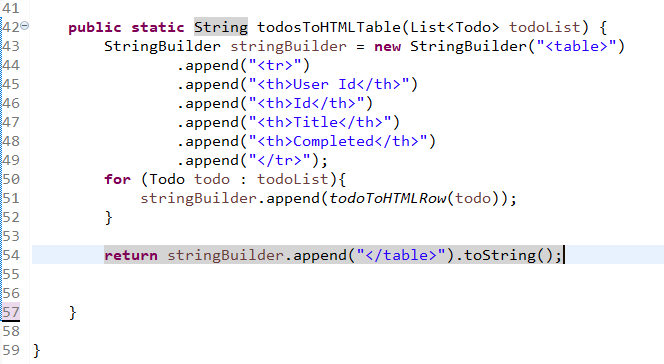
1. Utilice la siguiente clase para consumir el servicio que se encuentra en la dirección url del punto anterior:





Texto

Descripción generada automáticamente

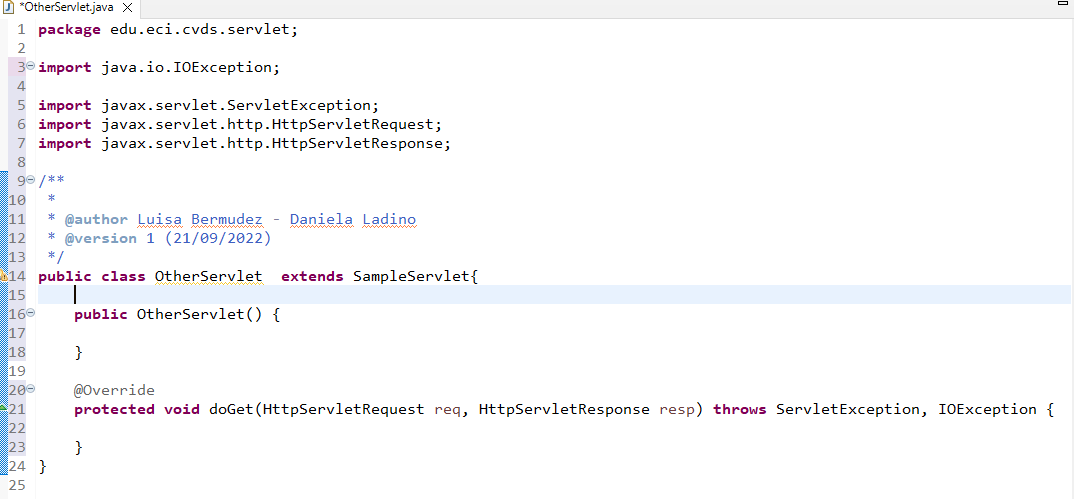


1. Cree una clase que herede de la clase HttpServlet (similar a SampleServlet), y para la misma sobrescriba el método heredado doGet. Incluya la anotación @Override para verificar –en tiempo de compilación- que efectivamente se esté sobrescribiendo un método de las superclases.

La clase que creamos es *OtherServlet*

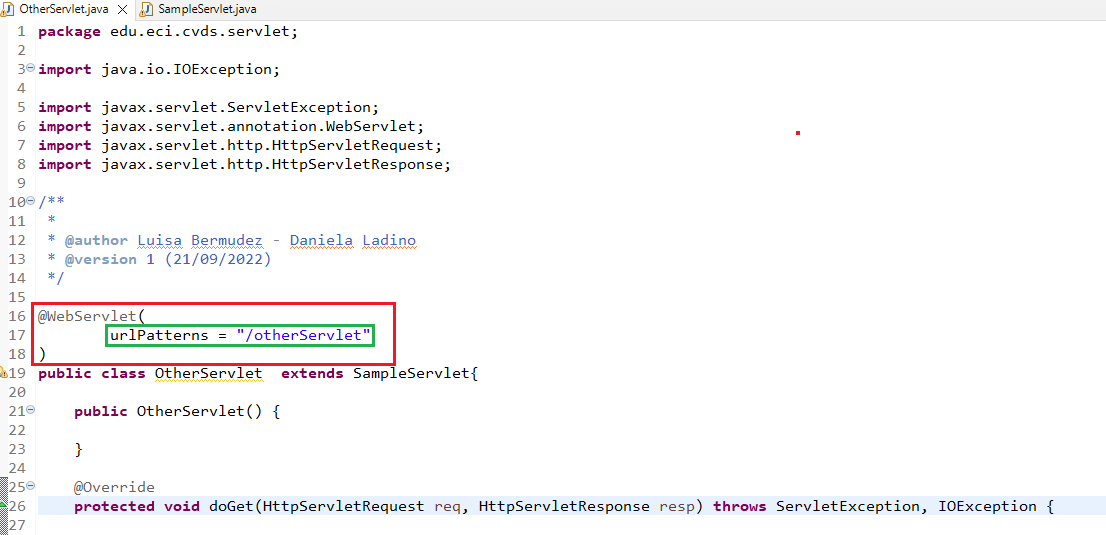
Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja



1. Para indicar en qué URL el servlet interceptará las peticiones GET, agregue al método la anotación @WebServlet, y en dicha anotación, defina la propiedad urlPatterns, indicando la URL (que usted defina) a la cual se asociará el servlet

*El urlPatterns que asignamos es urlPatterns = “/otherServlet”*

**

1. Teniendo en cuenta las siguientes métodos disponibles en los objetos ServletRequest y ServletResponse recibidos por el método doGet:

* response.setStatus(N); <- Indica con qué código de error N se generará la respuesta. Usar la clase HttpServletResponse para indicar el código de respuesta.
* request.getParameter(param); <- Consulta el parámetro recibido, asociado al nombre ‘param’
* response.getWriter() <- Retorna un objeto PrintWriter a través del cual se le puede enviar la respuesta a quien hizo la petición.
* response.setContentType(T) <- Asigna el tipo de contenido (MIME type) que se entregará en la respuesta. Implemente dicho método de manera que:
* Asuma que la petición HTTP recibe como parámetro el número de id de una lista de cosas por hacer (todo), y que dicha identificación es un número entero.
* Con el identificador recibido, consulte el ítem por hacer de la lista de cosas por hacer, usando la clase "Service" creada en el punto 10.
* Si el ítem existe:
* Responder con el código HTTP que equivale a ‘OK’ (ver referencia anterior), y como contenido de dicha respuesta, el código HTML correspondiente a una página con una tabla que tenga los detalles del ítem, usando la clase "Service" creada en el punto 10 para crear la tabla.
* Sel ítem no existe:
* Responder con el código correspondiente a ‘no encontrado’, y con el código de una página HTML que indique que no existe un ítem con el identificador dado.
* Si no se pasó parámetro opcional, o si el parámetro no contiene un número entero, devolver el código equivalente a requerimiento inválido.
* Si se genera la excepción MalformedURLException devolver el código de error interno en el servidor
* Para cualquier otra excepción, devolver el código equivalente a requerimiento inválido



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

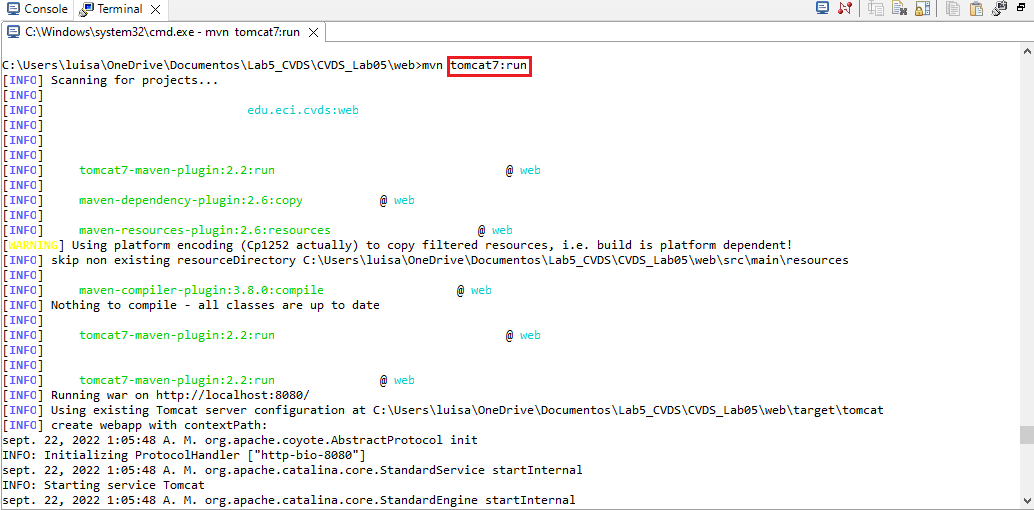
Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

1. Una vez hecho esto, verifique el funcionamiento de la aplicación, recompile y ejecute la aplicación.

Texto

Descripción generada automáticamente con confianza media



1. Intente hacer diferentes consultas desde un navegador Web para probar las diferentes funcionalidades.

* No se le manda ningún Id

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

* Cuando en vez de un numero se le manda letras

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

* Cuando el Id no es valido

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

* Cuando el valor es valido

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

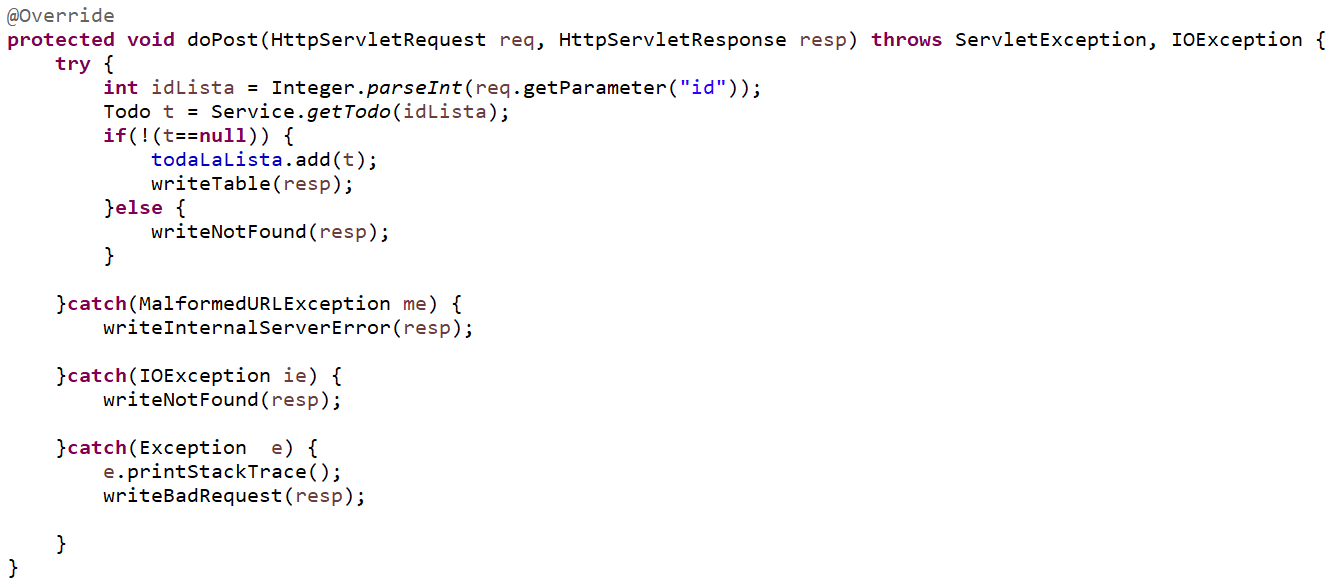
* Error interno en el servidor

Interfaz de usuario gráfica, Aplicación, Sitio web

Descripción generada automáticamente

PARTE III

1. En su servlet, sobreescriba el método doPost, y haga la misma implementación del doGet.



1. Cree el archivo index.html en el directorio src/main/webapp/index.html de la siguiente manera:

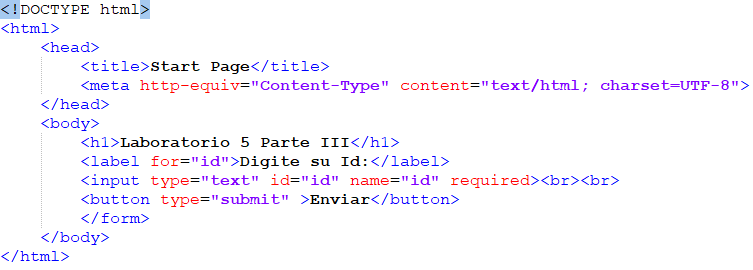
Interfaz de usuario gráfica, Aplicación

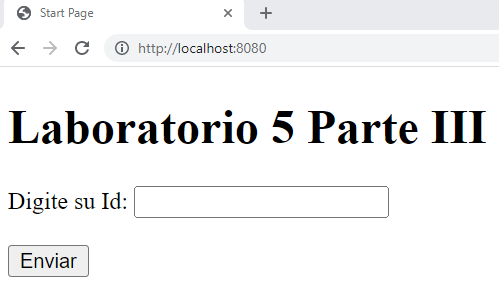
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

1. En la página anterior, cree un formulario que tenga un campo para ingresar un número (si no ha manejado html antes, revise http://www.w3schools.com/html/) y un botón. El formulario debe usar como método ‘POST’, y como acción, la ruta relativa del último servlet creado (es decir la URL, pero excluyendo ‘http://localhost:8080/’).





1. Revise este ejemplo de validación de formularios con JavaScript y agréguelo a su formulario, de manera que -al momento de hacer ‘submit’- desde el browser se valide que el valor ingresado es un valor numérico.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

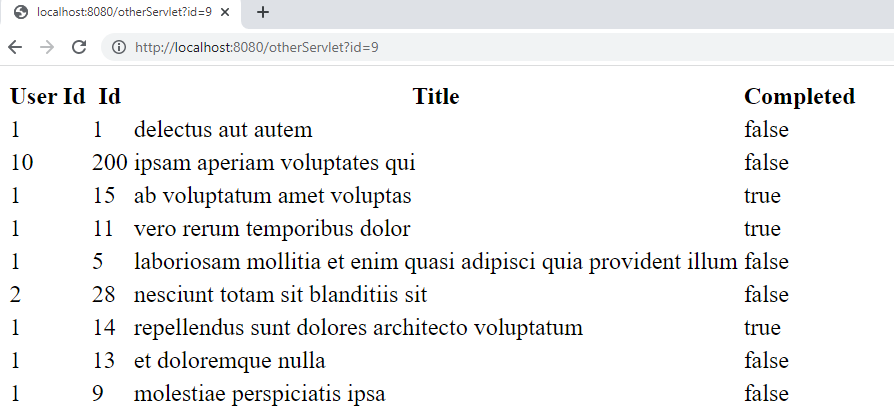
1. Recompile y ejecute la aplicación. Abra en su navegador en la página del formulario, y rectifique que la página hecha anteriormente sea mostrada. Ingrese los datos y verifique los resultados. Cambie el formulario para que ahora en lugar de POST, use el método GET. ¿Qué diferencia observa?

* **Método POST**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

* **Método GET**



Como podemos observar con el método GET, los datos que se envían al servidor**se escriben en la misma dirección URL**

1. ¿Qué se está viendo? Revise cómo están implementados los métodos de la clase Service.java para entender el funcionamiento interno.

Service implementa una tabla de datos en la que podemos visualizar el User Id, Id, Title y Completed por cada uno de los id que ingresa el usuario

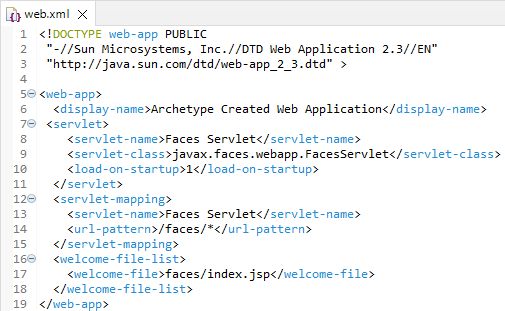
PARTE IV. - FRAMEWORKS WEB MVC – JAVA SERVER FACES /PRIME FACES

En este ejercicio, usted va a desarrollar una aplicación Web basada en el marco JSF, y en una de sus implementaciones más usadas: PrimeFaces. Se trata de un juego en línea para adivinar un número, en el que el ganador, si atina en la primera oportunidad, recibe $100.000. Luego, por cada intento fallido, el premio se reduce en $10.000.

1. Al proyecto Maven, debe agregarle las dependencias más recientes de javax.javaee-api, com.sun.faces.jsf-api, com.sun.faces.jsf-impl,javax.servlet.jstl y Primefaces (en el archivo pom.xml).



1. Para que configure automáticamente el descriptor de despliegue de la aplicación (archivo web.xml), de manera que el framework JSF se active al inicio de la aplicación, en el archivo web.xml agregue la siguiente configuración:



1. Revise cada una de las configuraciones agregadas anteriormente para saber qué hacen y por qué se necesitan. Elimine las que no se necesiten.
2. Ahora, va a crear un Backing-Bean de sesión, el cual, para cada usuario, mantendrá de lado del servidor las siguientes propiedades:
3. El número que actualmente debe adivinar (debe ser un número aleatorio).
4. El número de intentos realizados.
5. El premio acumulado hasta el momento.
6. El estado del juego, que sería una cadena de texto que indica si ya ganó o no, y si ganó de cuanto es el premio.

Para hacer esto, cree una clase que tenga:

* el constructor por defecto (sin parámetros)
* los métodos get/set necesarios dependiendo si las propiedades son de escritura o lectura
* coloque las anotaciones:
* @ManagedBean, incluyendo el nombre: @ManagedBean (name = "guessBean").
* @ApplicationScoped.

A la implementación de esta clase, agregue los siguientes métodos:

* guess: Debe recibir un intento de adivinanza y realizar la lógica para saber si se adivinó, de tal forma que se ajuste el valor del premio y/o actualice el estado del juego.
* restart: Debe volver a iniciar el juego (inicializar de nuevo el número a adivinar, y restaurar el premio a su valor original).

1. Cree una página XHTML, de nombre guess.xhtml (debe quedar en la ruta src/main/webapp). Revise en la página 13 del manual de PrimeFaces, qué espacios de nombres XML requiere una página de PrimeFaces y cuál es la estructura básica de la misma.
2. Con base en lo anterior, agregue un formulario con identificador guess\_form con el siguiente contenido básico
3. Al formulario, agregue:
4. Un elemento de tipo <p:outputLabel> para el número que se debe adivinar, sin embargo, este elemento se debe ocultar. Para ocultarlo, sepuede agregar el estilo display: none; al elemento. Una forma de hacerlo es por medio de la propiedad style.

* En una aplicacion real, no se debería tener este elemento, solo se crea con el fin de simplificar una prueba futura.

1. Un elemento <p:inputText> para que el usuario ingrese su número.
2. Un elemento de tipo <p:outputLabel> para mostrar el número de intentos realizados.
3. Un elemento de tipo <p:outputLabel> para mostrar el estado del juego.
4. Un elemento de tipo <p:outputLabel> para mostrar en cuanto va el premio.

Y asocie dichos elementos al BackingBean de sesión a través de su propiedad value, y usando como referencia el nombre asignado:

1. Al formulario, agregue dos botones de tipo <p:commandButton>, uno para enviar el número ingresado y ver si se atinó, y otro para reiniciar el juego.
2. El botón de envío de adivinanza debe tener asociado a su propiedad update el nombre del formulario en el que se agregaron los campos antesdescritos, de manera que al hacer clic, se ejecute un ciclo de JSF y se refresque la vista.
3. Debe tener también una propiedad actionListener con la cual se le indicará que, al hacer clic, se ejecutará el método guess, creado en elbacking-bean de sesión:
4. El botón de reiniciar juego tendrá las mismas propiedades de update y actionListener del otro con el valor correspondiente:
5. Para verificar el funcionamiento de la aplicación, agregue el plugin tomcat-runner dentro de los plugins de la fase de construcción (build). Tenga encuenta que en la configuración del plugin se indica bajo que ruta quedará la aplicación:
6. mvn package
7. mvn tomcat7:run

Si no hay errores, la aplicación debería quedar accesible en la URL: <http://localhost:8080/faces/guess.xhtml>

1. Si todo funcionó correctamente, realice las siguientes pruebas:
2. Abra la aplicación en un explorador. Realice algunas pruebas con el juego e intente adivinar el número.
3. Abra la aplicación en dos computadores diferentes. Si no dispone de uno, hágalo en dos navegadores diferentes (por ejemplo Chrome y Firefox;incluso se puede en un único navegador usando una ventana normal y una ventana de incógnito / privada). Haga cinco intentos en uno, y luegoun intento en el otro. ¿Qué valor tiene cada uno?
4. Aborte el proceso de Tomcat-runner haciendo Ctrl+C en la consola, y modifique el código del backing-bean de manera que use la anotación@SessionScoped en lugar de @ApplicationScoped. Reinicie la aplicación y repita el ejercicio anterior.

* ¿Coinciden los valores del premio?
* Dado la anterior, ¿Cuál es la diferencia entre los backing-beans de sesión y los de aplicación?

1. Por medio de las herramientas de desarrollador del explorador (Usando la tecla "F12" en la mayoría de los exploradores):

* Ubique el código HTML generado por el servidor.
* Busque el elemento oculto, que contiene el número generado aleatoriamente.
* En la sección de estilos, deshabilite el estilo que oculta el elemento para que sea visible.
* Observe el cambio en la página, cada vez que se realiza un cambio en el estilo.
* Revise qué otros estilos se pueden agregar a los diferentes elementos y qué efecto tienen en la visualización de la página.
* Actualice la página. Los cambios de estilos realizados desaparecen, pues se realizaron únicamente en la visualización, la respuesta delservidor sigue siendo la misma, ya que el contenido de los archivos allí almacenados no se ha modificado.
* Revise qué otros cambios se pueden realizar y qué otra información se puede obtener de las herramientas de desarrollador.

1. Para facilitar los intentos del usuario, se agregará una lista de los últimos intentos fallidos realizados: