

Laboratorio 5

Introducción a proyectos web

Integrantes:

Luisa Fernanda Bermúdez Girón

Karol Daniela Ladino Ladino

Profesor:

Ivan Dario Viasus Quintero

Curso:

CVDS - 2

Fecha De Entrega:

08-09-2022

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 |

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

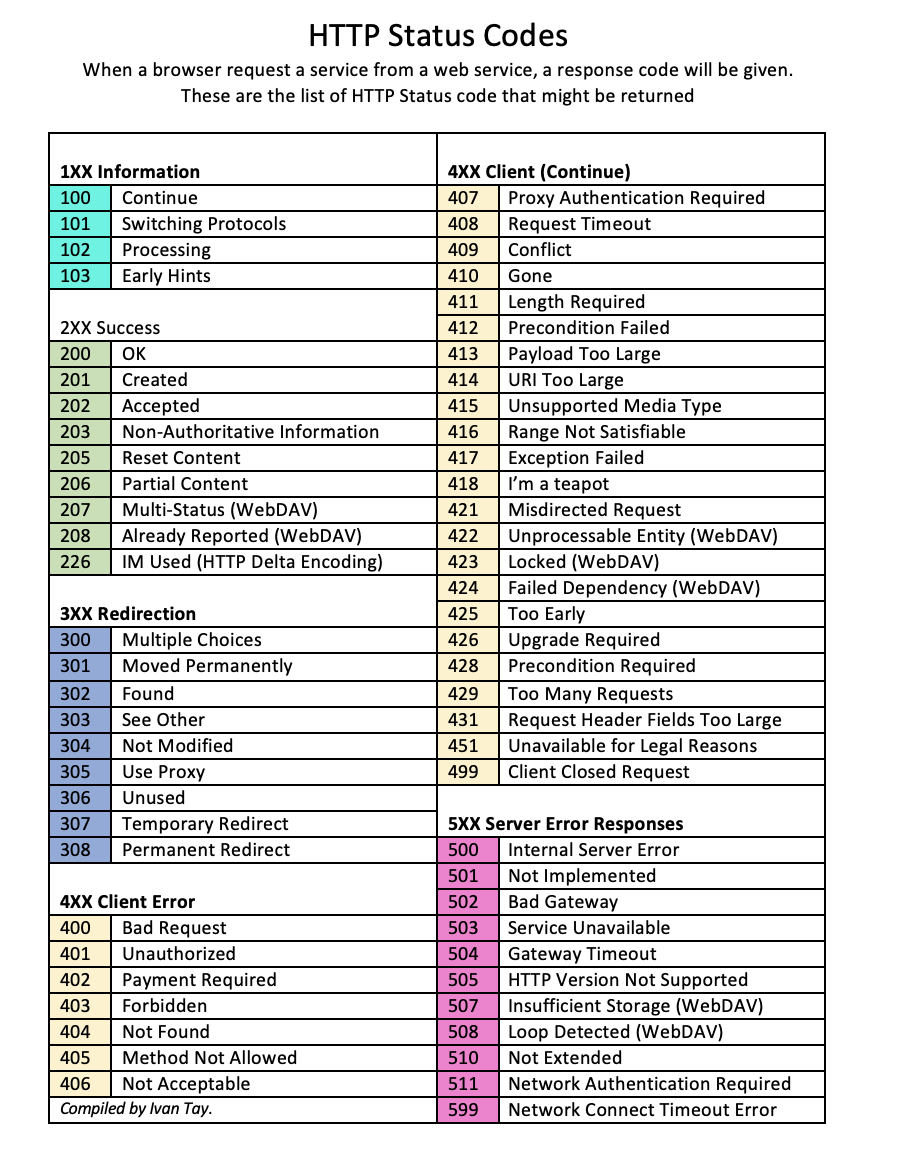
Texto

Descripción generada automáticamente

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

¡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.

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 linea de comandos:

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

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

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

¿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

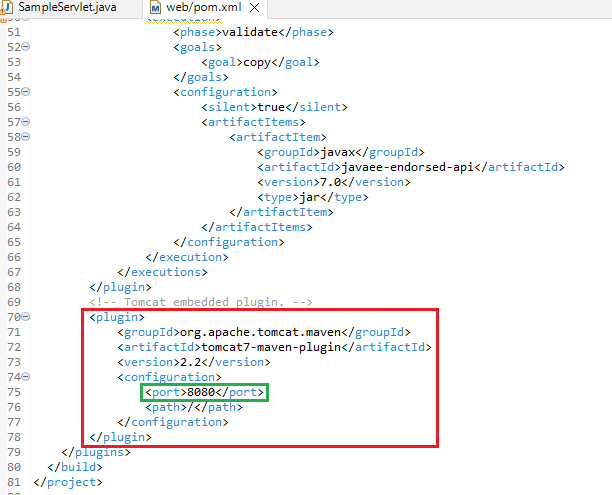
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

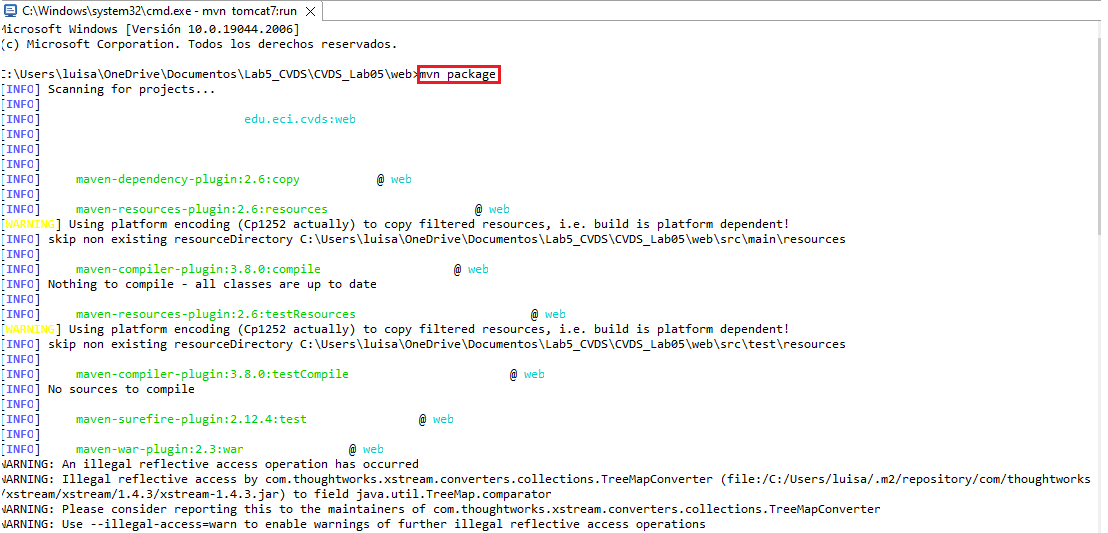


1. Revise en el pom.xml para qué puerto TCP/IP está configurado el servidor embebido de Tomcat (ver sección de plugins).



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

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

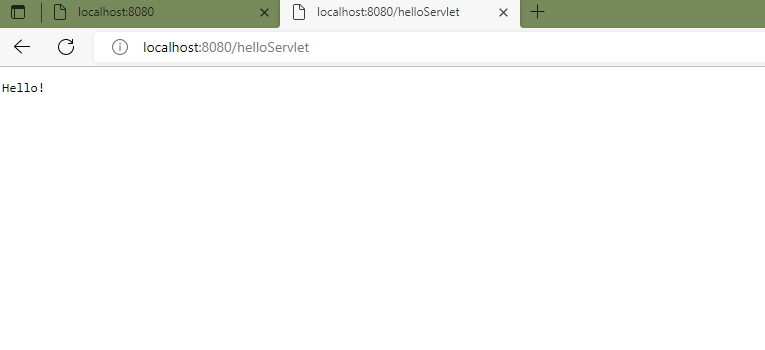


Interfaz de usuario gráfica, Texto, Correo electrónico

Descripción generada automáticamente

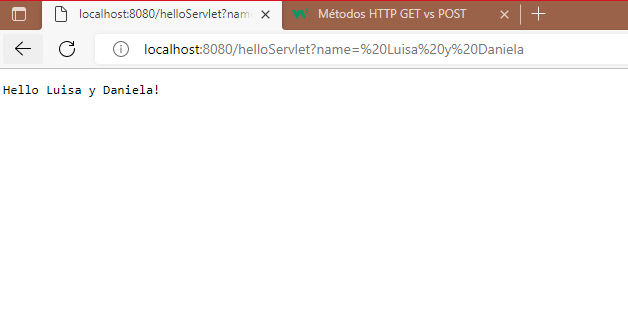
1. Abra un navegador, y en la barra de direcciones ponga la URL con la cual se le enviarán peticiones al ‘SampleServlet’. Tenga en cuenta que la URL tendrá como host ‘localhost’, como puerto, el configurado en el pom.xml y el path debe ser el del Servlet. Debería obtener un mensaje de saludo.

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



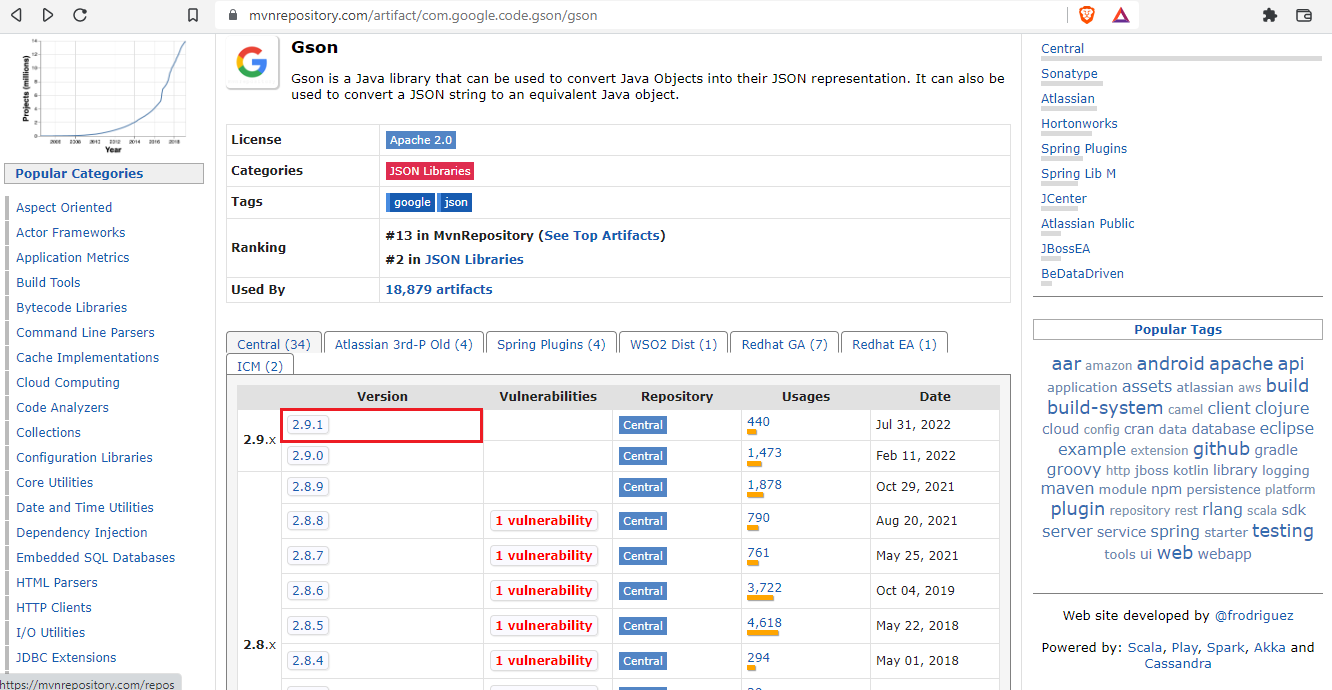
1. Observe que el Servlet ‘SampleServlet’ acepta peticiones GET, y opcionalmente, lee el parámetro ‘name’. Ingrese la misma URL, pero ahora agregando un parámetro GET (si no sabe como hacerlo, revise la documentación en <http://www.w3schools.com/tags/ref_httpmethods.asp>).

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.

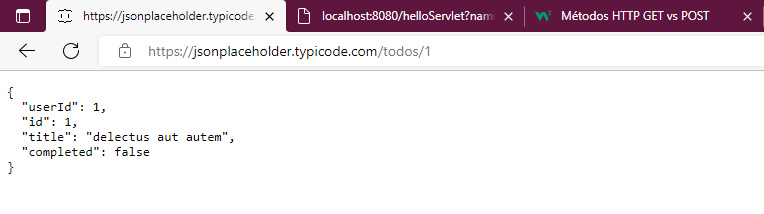




Agregamos la dependencia en pom.xml



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

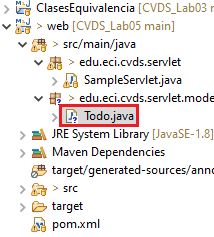
Descripción generada automáticamente

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



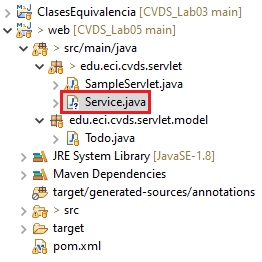
Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto, Correo electrónico

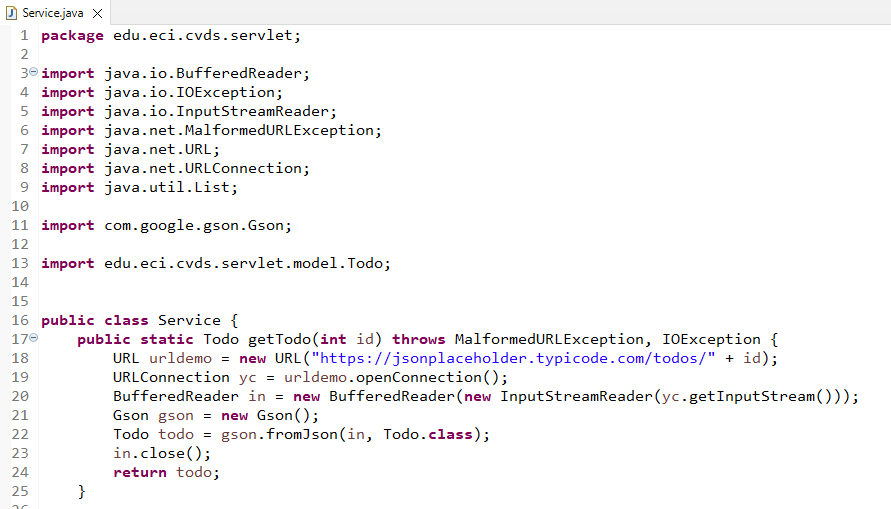
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

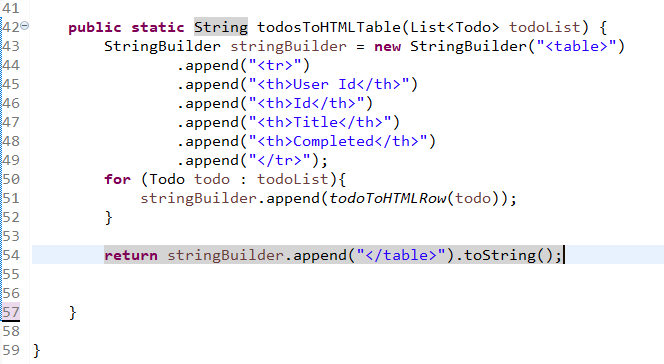
1. Utilice la siguiente clase para consumir el servicio que se encuentra en la dirección url del punto anterior:





Texto

Descripción generada automáticamente

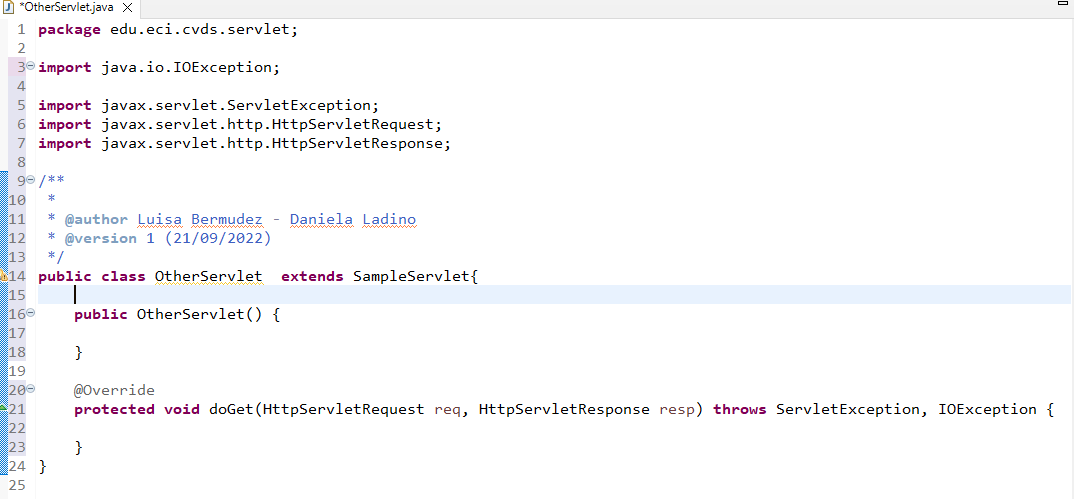


1. Cree una clase que herede de la clase HttpServlet (similar a SampleServlet), y para la misma sobrescriba el método heredado doGet. Incluya la anotación @Override para verificar –en tiempo de compilación- que efectivamente se esté sobreescribiendo un método de las superclases.

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