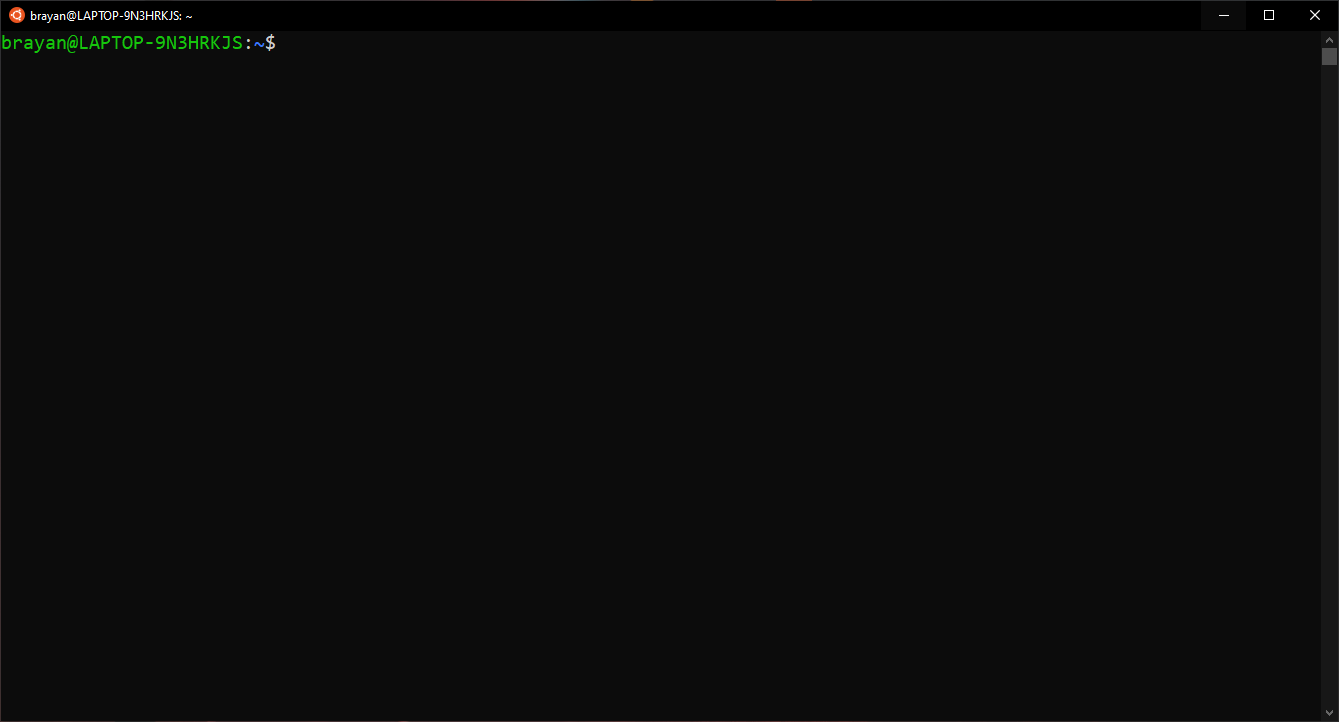
**LABORATORIO 5**

**MVC PRIMEFACES INTRODUCTION - 2021-1**

**PARTE I. - JUGANDO A SER UN CLIENTE HTTP**

1. **Abra una terminal Linux o consola de comandos Windows.**

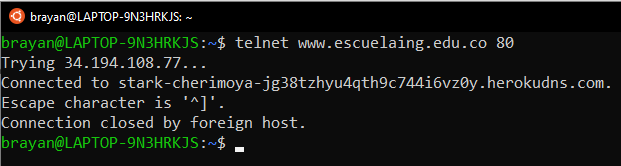


1. **Realice una conexión síncrona TCP/IP a través de Telnet al siguiente servidor:**

* Host: [www.escuelaing.edu.co](http://www.escuelaing.edu.co)
* Puerto: 80

Teniendo en cuenta los parámetros del comando telnet:

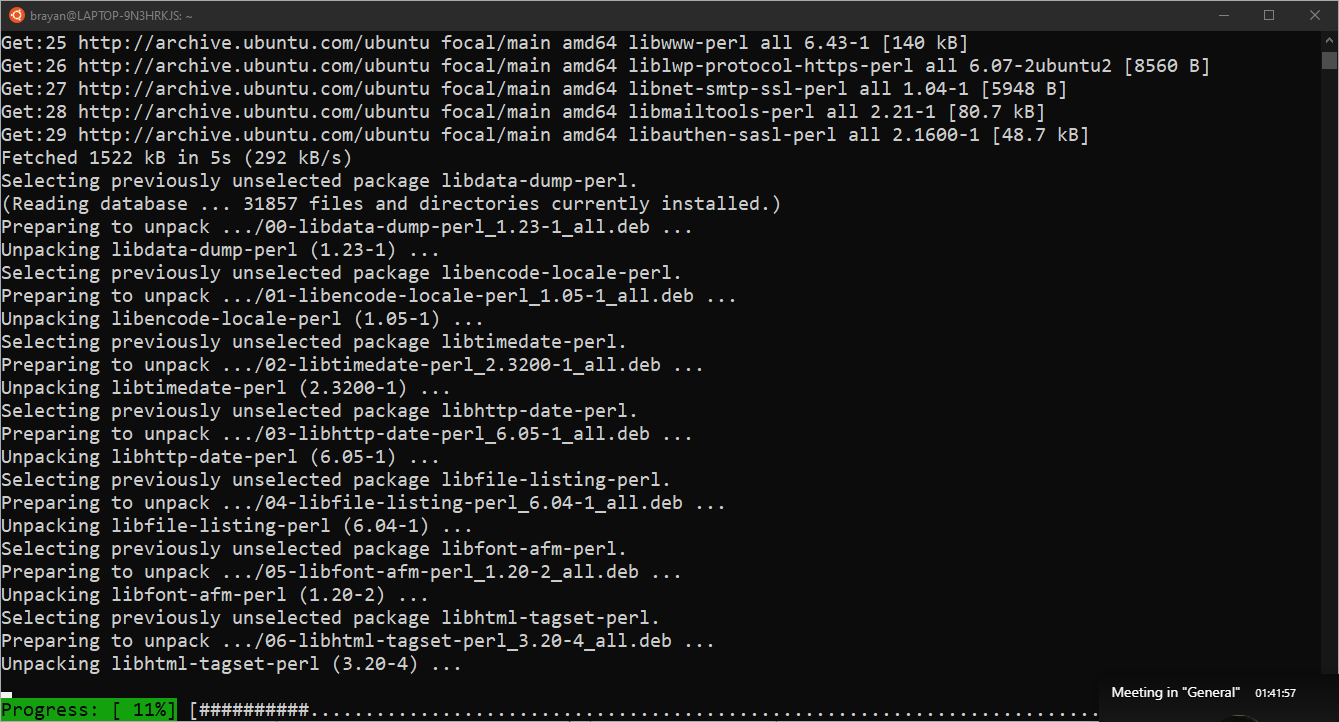
telnet HOST PORT



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**:**

(Si al usar el comando nos dice que este no se reconoce, es posible que primero tengamos que descargarlo, como en nuestro caso)

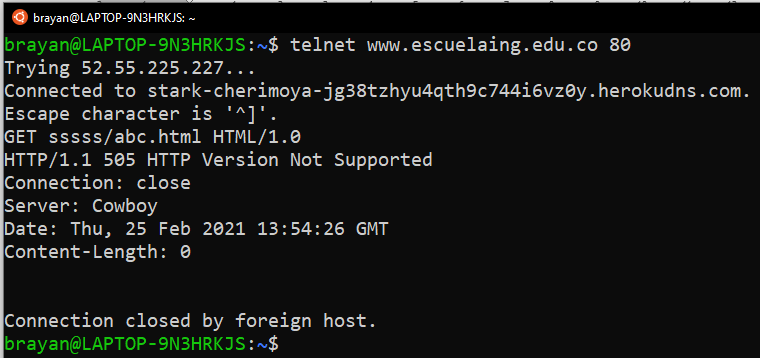


Una vez descargado, ya podremos usar el comando.

Con los datos que nos dan, el comando quedaría:

GET sssss/abc.html HTML/1.0

* Asegúrese de presionar ENTER dos veces después de ingresar el comando**:**



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

Nos sale el código de error 505.

Al investigar el código, obtenemos la siguiente información:



* ¿Qué otros códigos de error existen?, ¿En qué caso se manejarán? **:**

En general, estos códigos de error se identifican por un número de tres cifras.

Así, según el valor del primer dígito, podemos encontrar 5 categorías de código:

* **1xx – Informational Response**

Se emite de forma provisional mientras continúa el procesamiento de la solicitud. Alerta al cliente para que espere una respuesta final.

* **2xx – Succes**

Indica que la acción solicitada por el cliente fue recibida, comprendida y aceptada.

* **3xx – Redirection**

indica que el cliente debe realizar una acción adicional para completar la solicitud.

* **4xx – Client Errors**

Esta clase de código de estado está destinada a situaciones en las que el error parece haber sido causado por el cliente.

* **5xx Server Errors**

Indican casos en los que el servidor se da cuenta de que ha encontrado un error o es incapaz de realizar la solicitud.

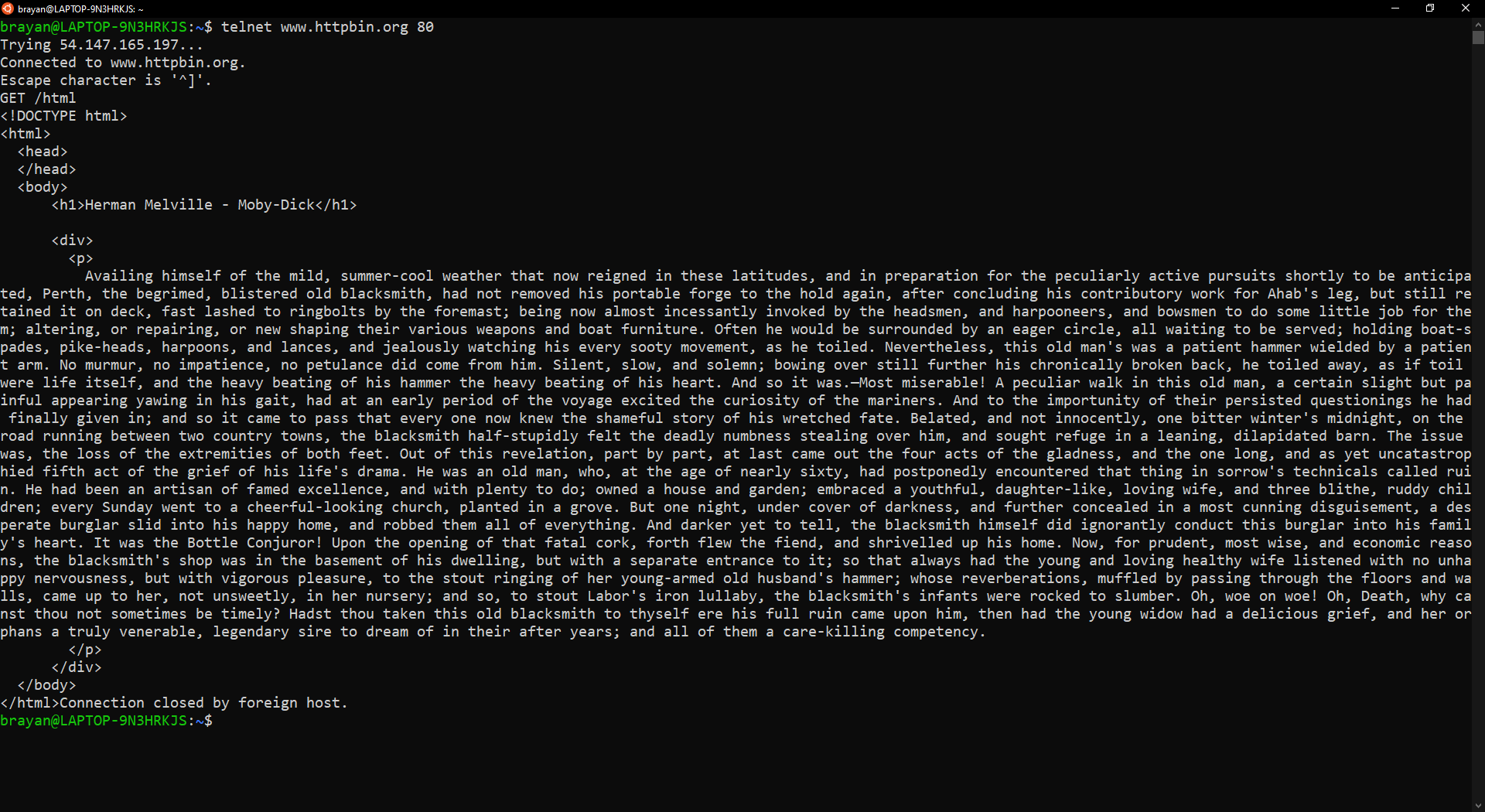
* **Unofficial Codes**

Los códigos no están especificados por ningún estándar.

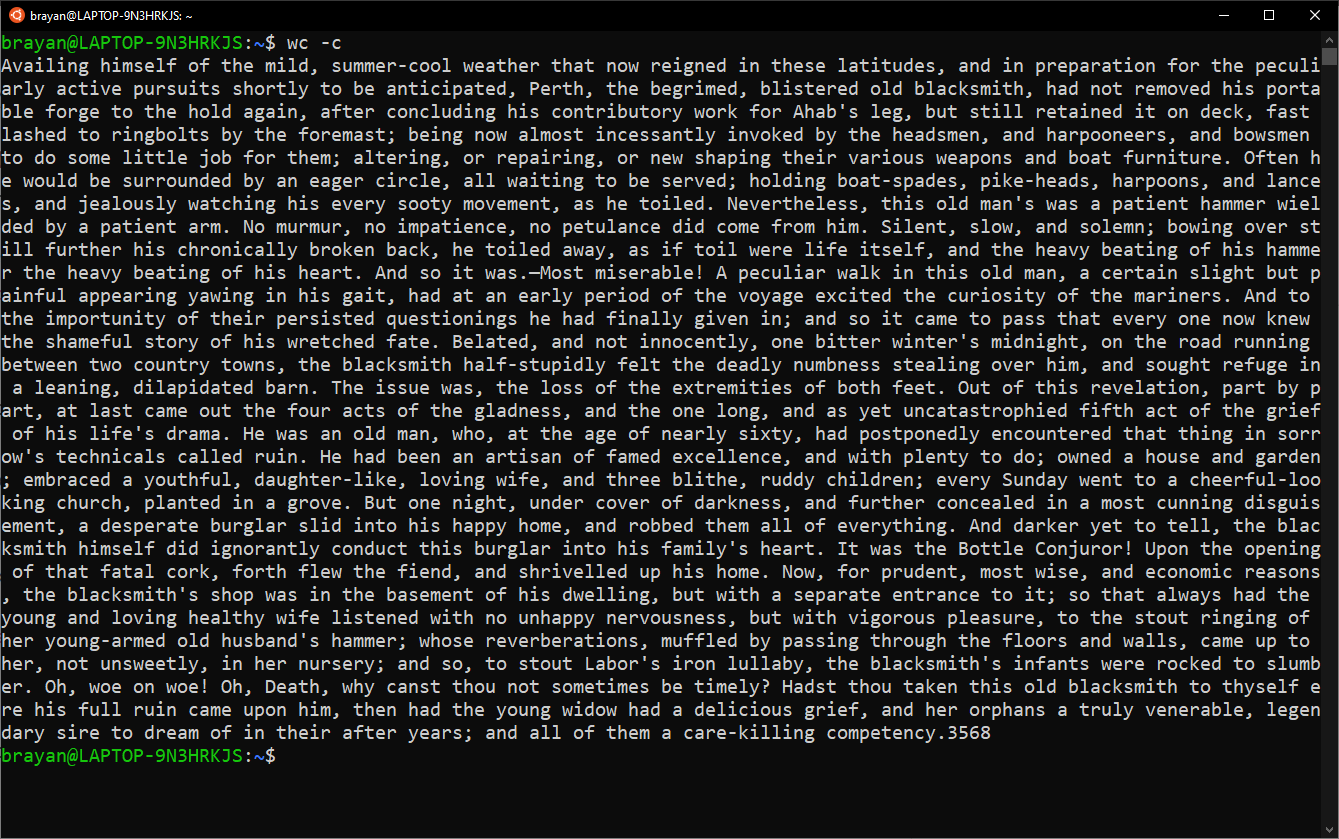
1. **Realice una nueva conexión con telnet, esta vez a:**

* 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?



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.



* Investigue: ¿Cuál es la diferencia entre los verbos GET y POST? ¿Qué otros tipos de peticiones existen?
* **GET**

Al enviar los datos, estos quedan dispuestos en la URL de la página, es decir, los parámetros y sus valores son visibles en la dirección URL.

* **POST**

Al enviar lo datos, estos quedan ocultos, ya que no se encuentran visibles dentro de la URL. De manera que tendremos que utilizar otras herramientas para poder rescatar los datos.

* **Otras 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)

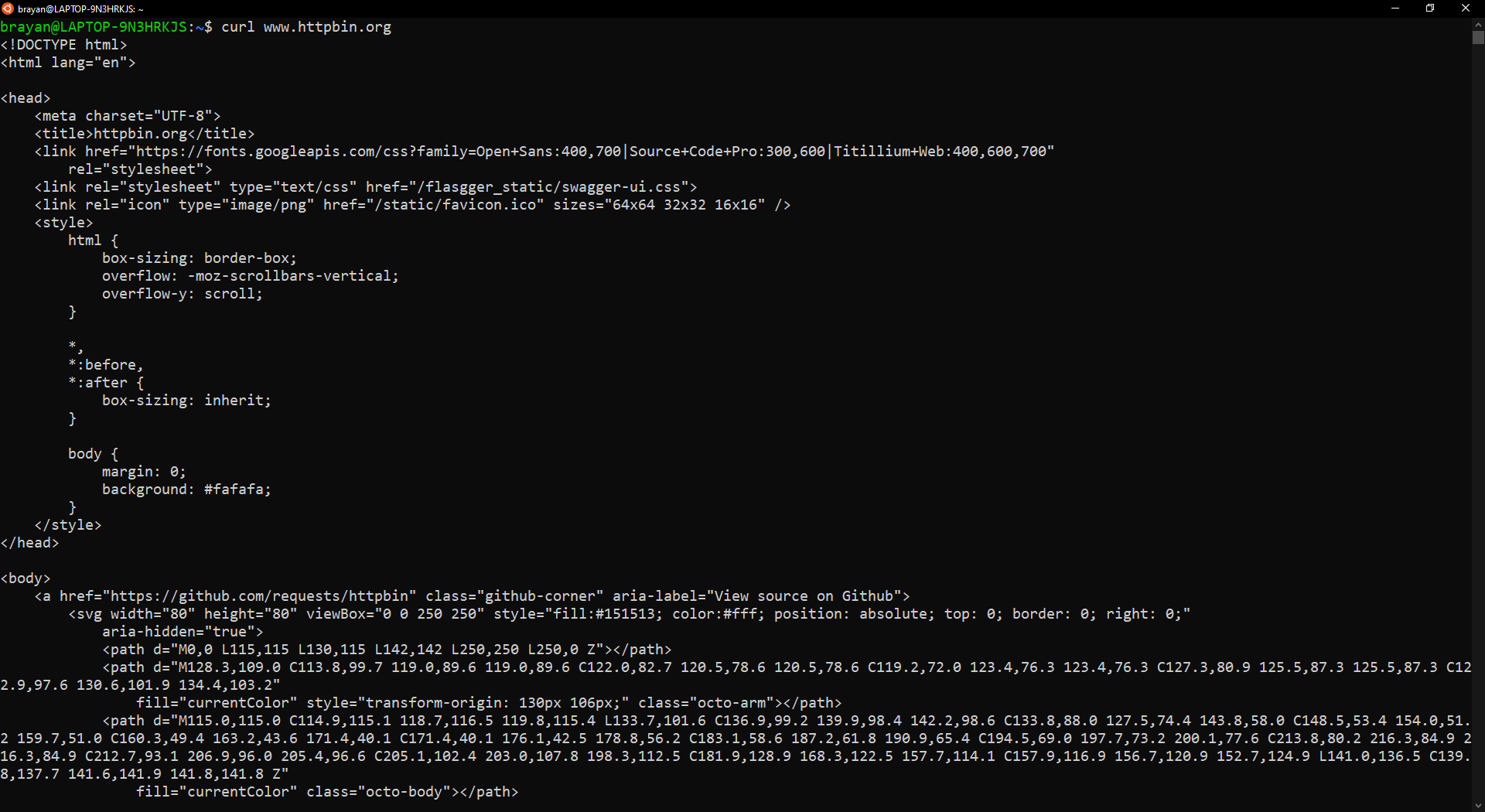
El método TRACE  realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

* [**PATCH**](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PATCH)

El método PATCH  es utilizado para aplicar modificaciones parciales a un recurso.

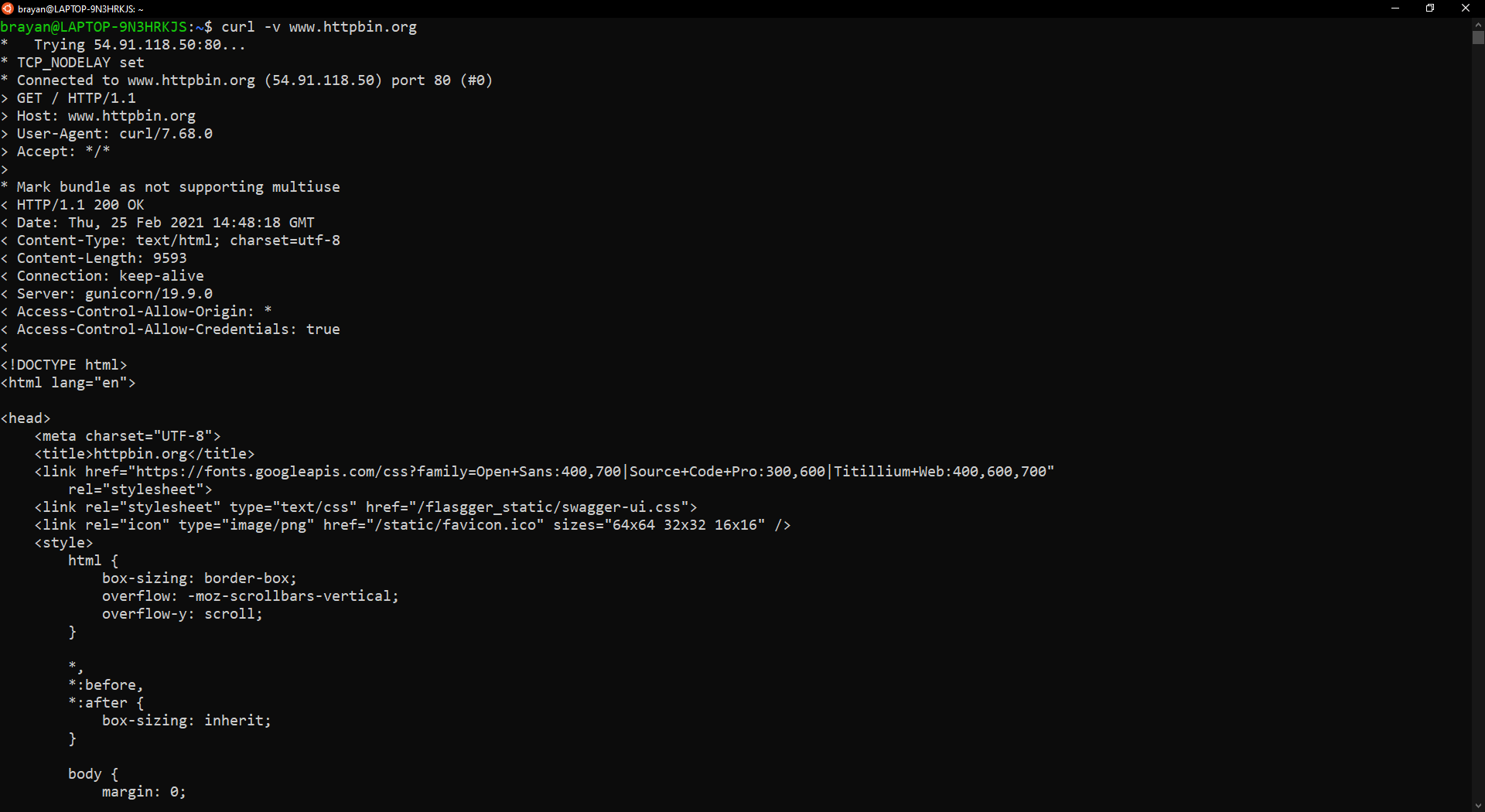
1. **En la práctica no se utiliza telnet para hacer peticiones a sitios web sino el comando curl con ayuda de la línea de comandos:**

curl www.httpbin.org

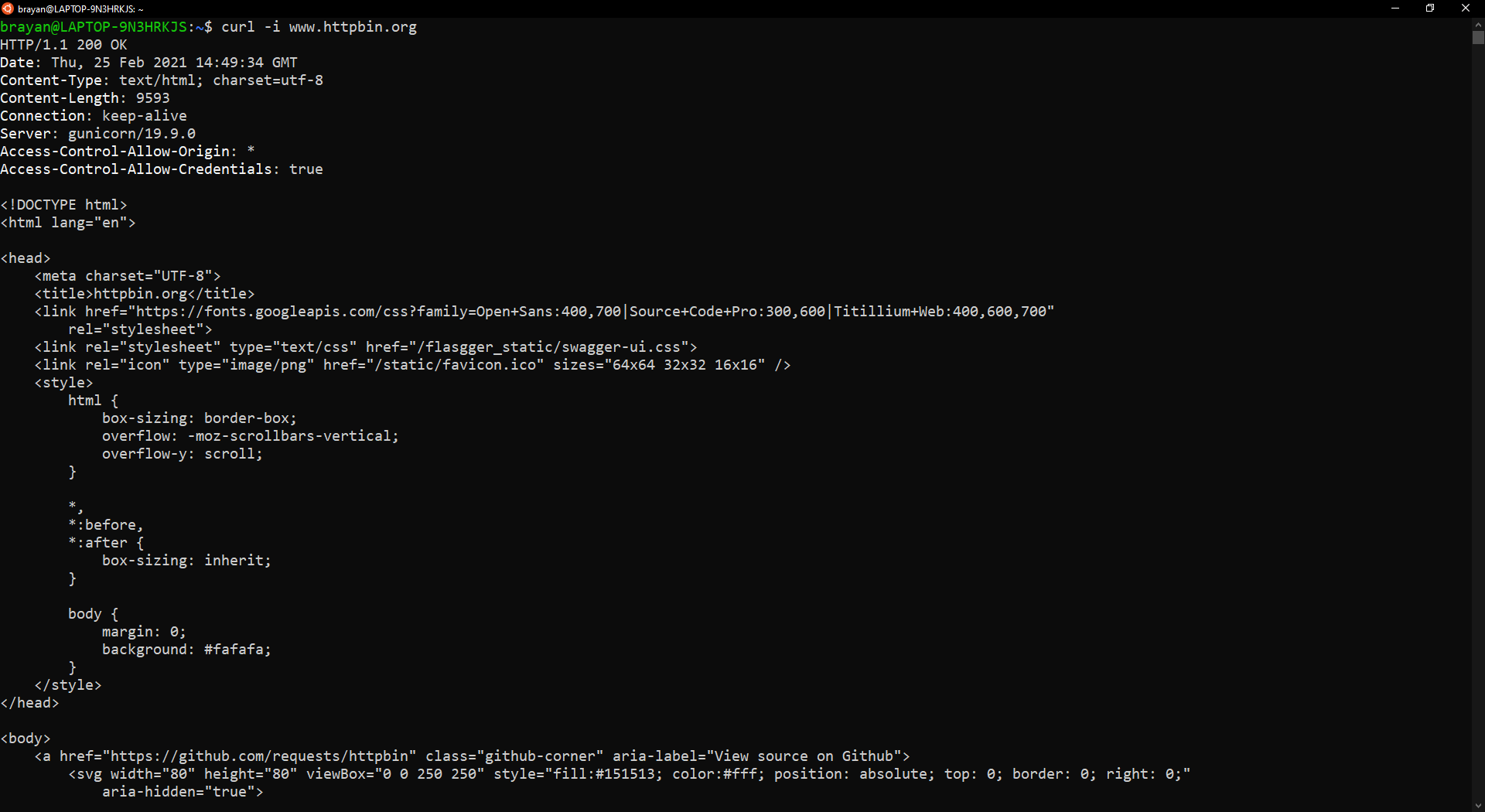


Utilice ahora el parámetro -v y con el parámetro -i:

* curl -v www.httpbin.org

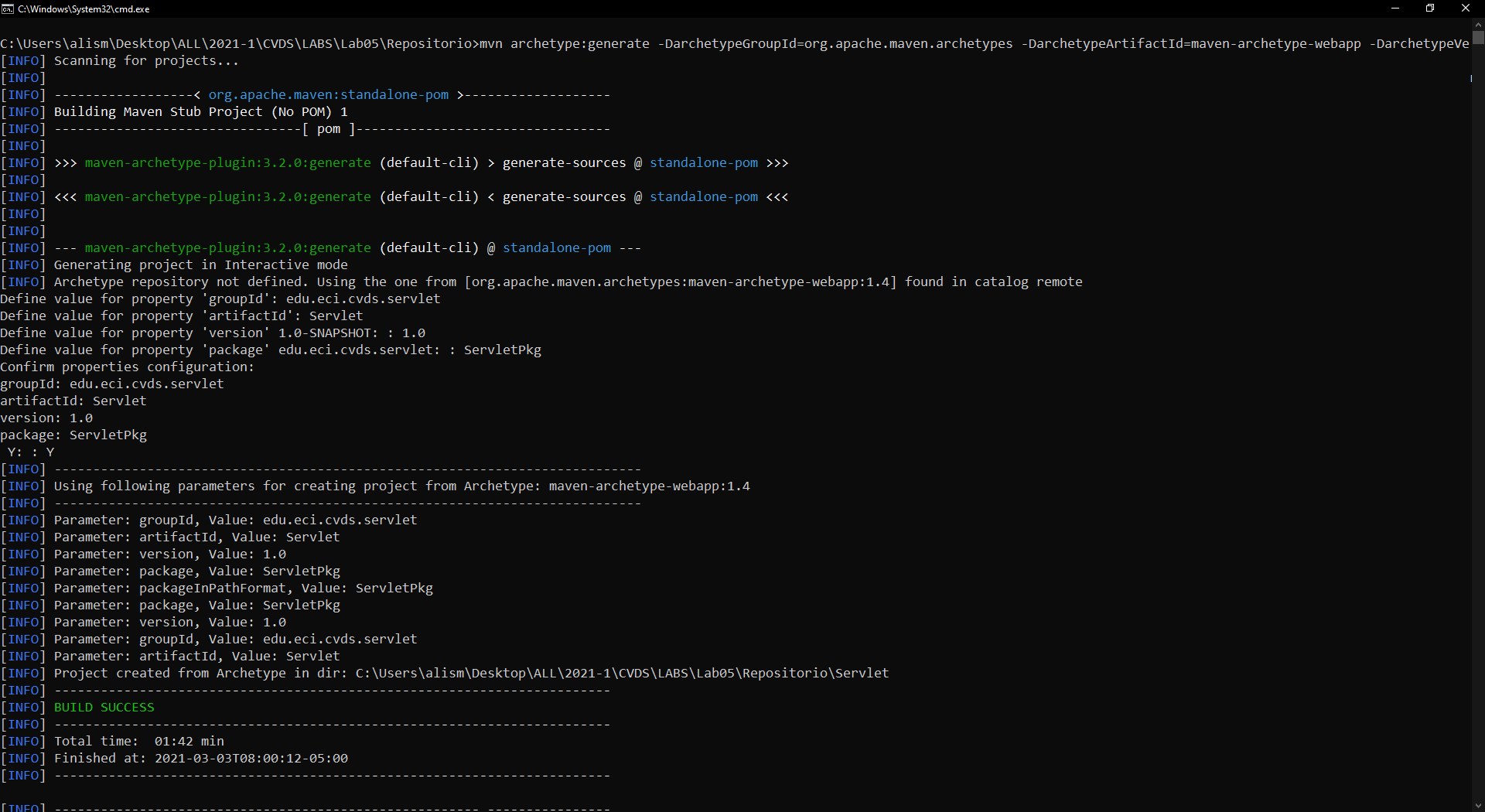


* curl -i www.httpbin.org



**PARTE II. - HACIENDO UNA APLICACIÓN WEB DINÁMICA A BAJO NIVEL.**

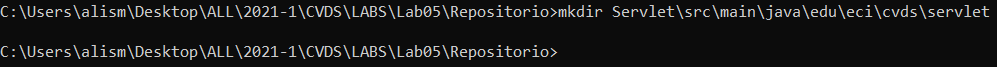
Cree un proyecto Maven nuevo usando el arquetipo de aplicación Web estándar maven-archetype-webapp.

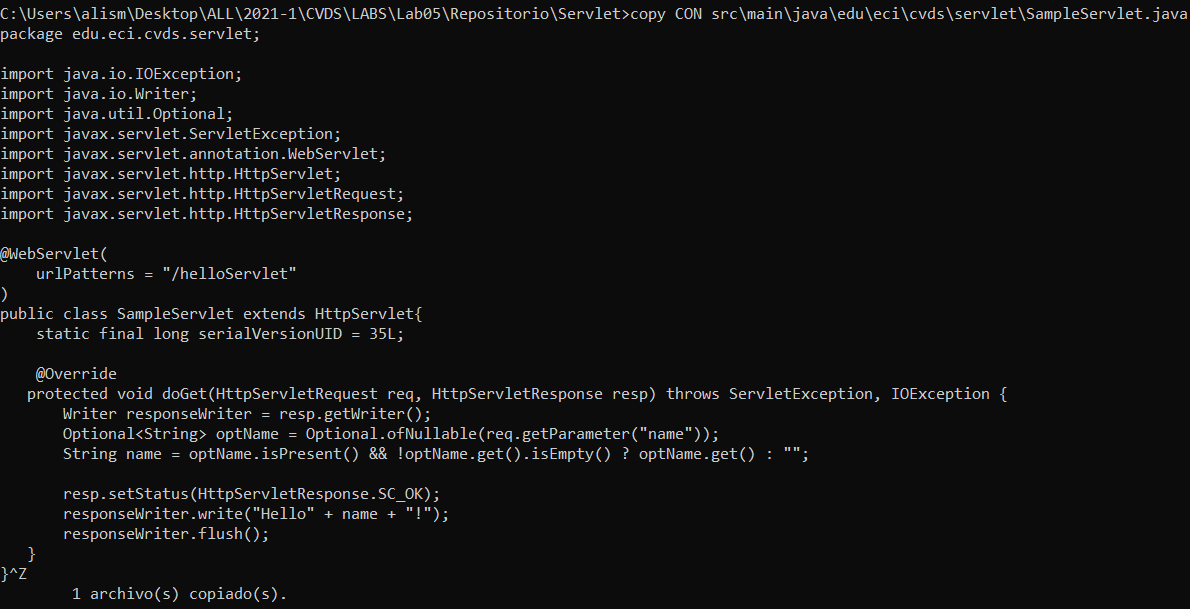


1. **Revise la clase SampleServlet e identifique qué hace**

La clase SampleServlet primeramente define la ruta de la URL por la que va a recibir las peticiones. La petición es manejada por el método doGet(), que lee la posible entrada de valores, para poder brindar una respuesta determinada. Si se envía algún valor en la petición, se imprime un mensaje especial en el cual se incluye el parámetro enviado, pero si llega vacío, se realiza un saludo genérico.

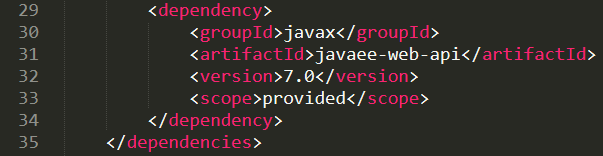
(Añadimos la clase al directorio de trabajo en Maven)





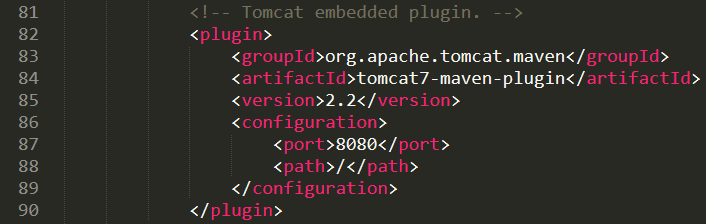
1. **Modificar y actualizar el archivo pom.xml**

Modifique la propiedad "packaging" con el valor "war". Agregue la siguiente dependencia**:**



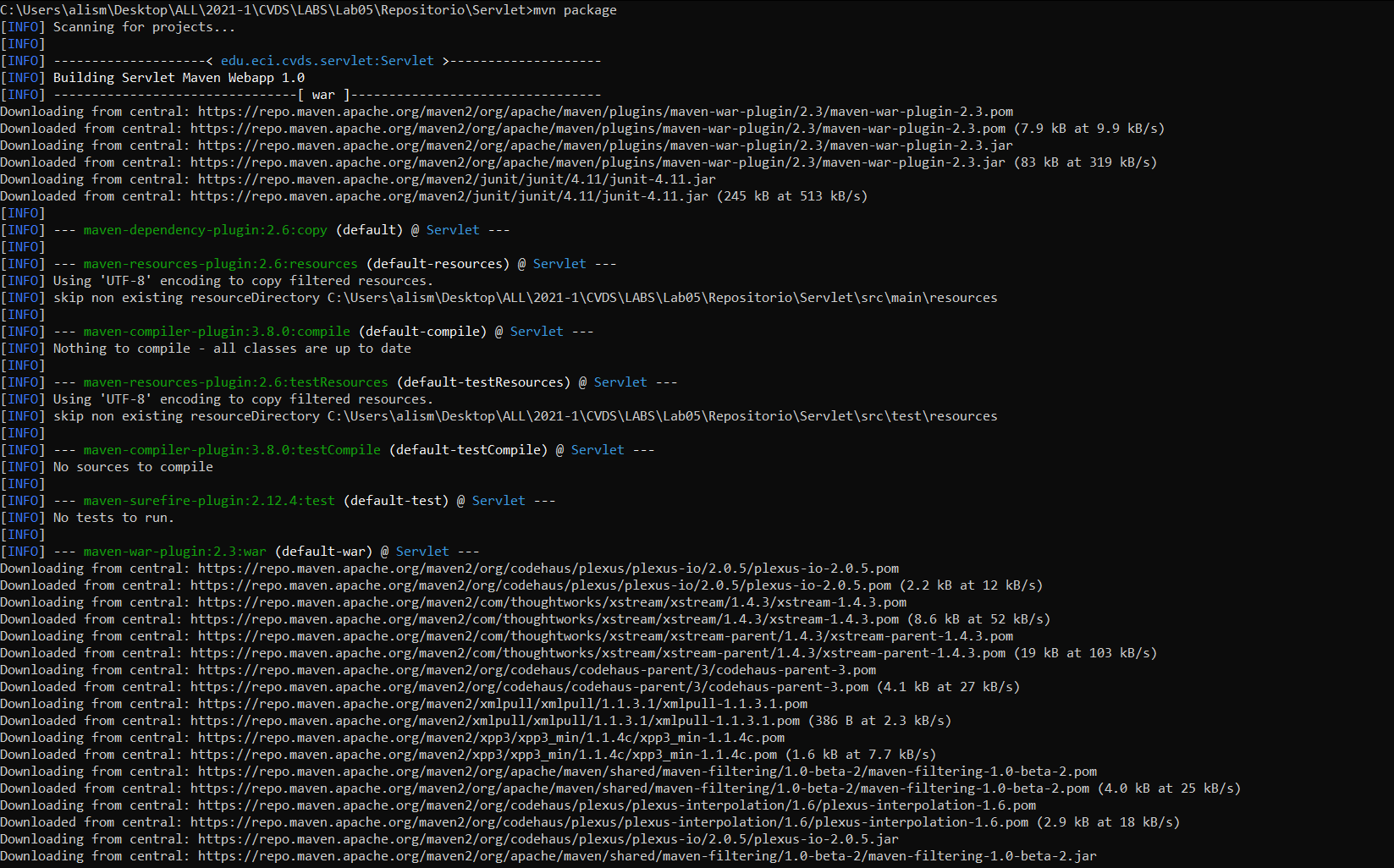
También modificamos la sección <build> como se indica en el laboratorio.

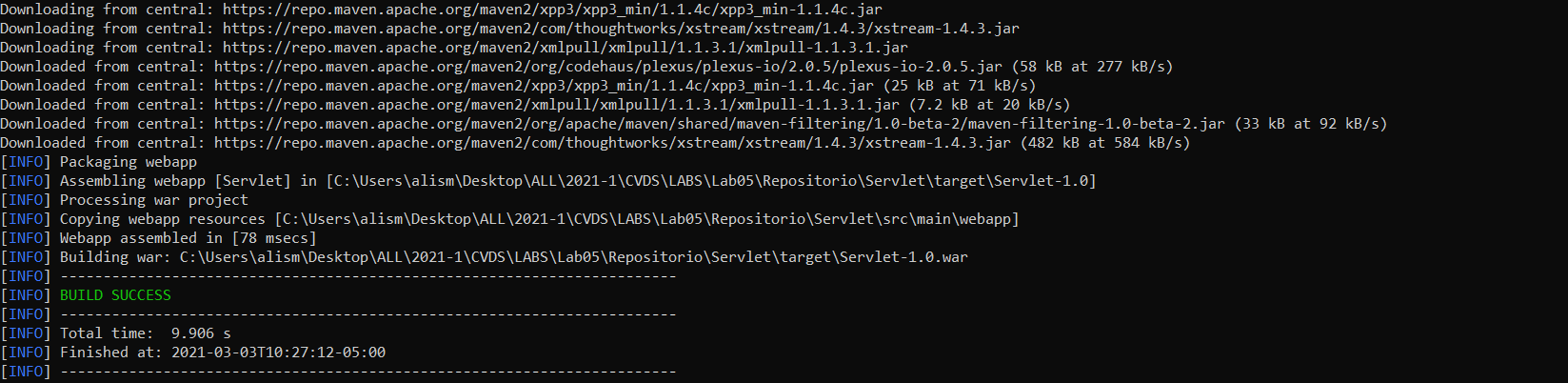
1. **Revisar en el pom.xml para Tomcat**



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

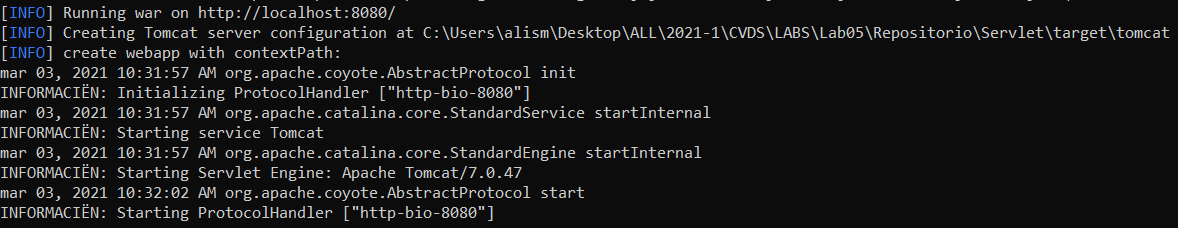
* mvn package





(Luego de ejecutar el comando, pueden que se descarguen varios recursos)

* tomcat7:run



(Aquí también puede que se descarguen algunos recursos más)

Luego de ejecutar el comando, se genera una sesión que queda abierta

1. **Probar el Servlet**

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.



1. **Ingrese la misma URL, pero ahora agregando un parámetro GET**

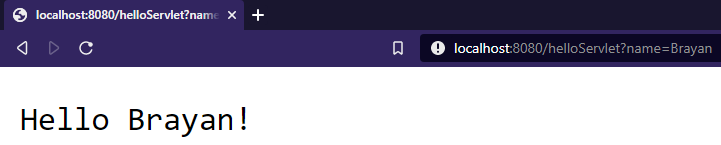
Para realizar el get, completamos la URL con:

?NombreParametro=ValorParametro

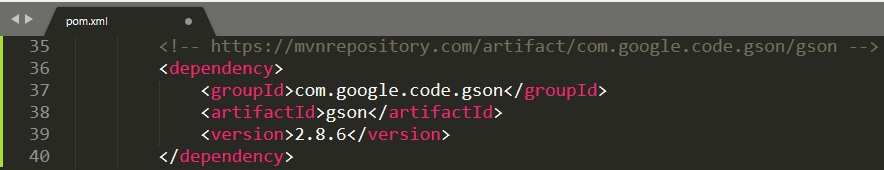
Que para nuestro caso queda como:

?name=Brayan

En caso de que necesitemos ingresar más de un parámetro, solo debemos colocar el carácter **&** al final del último parámetro ingresado, y luego colocar el siguiente parámetro.

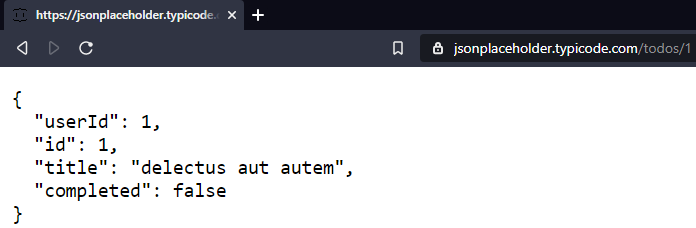


1. **Busque el artefacto gson en el repositorio de maven y agregue la dependencia**

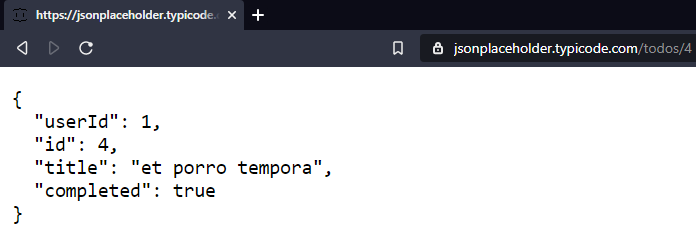


1. **En el navegador revise la dirección https://jsonplaceholder.typicode.com/todos/1. Intente cambiando diferentes números al final del path de la URL**

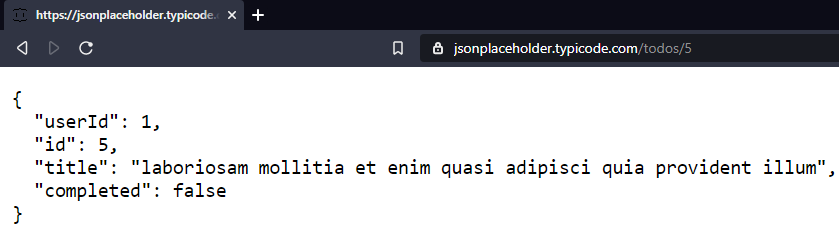
Con 1



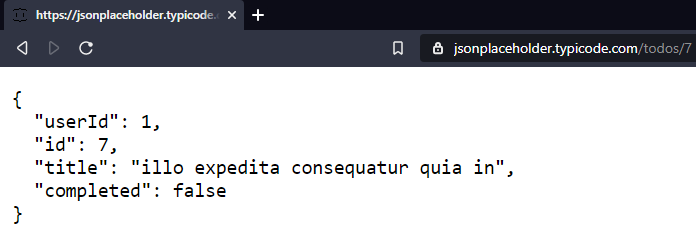
Con 4



Con 5

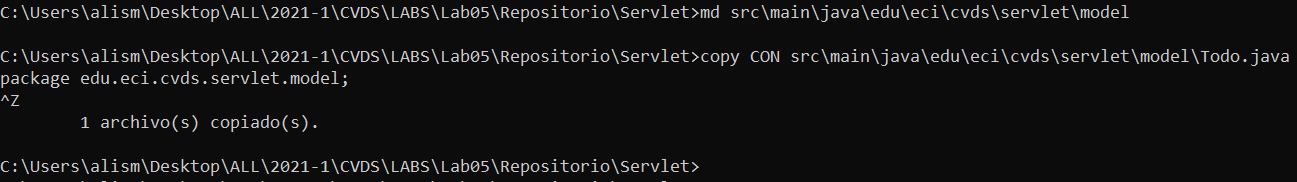


Con 7



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**

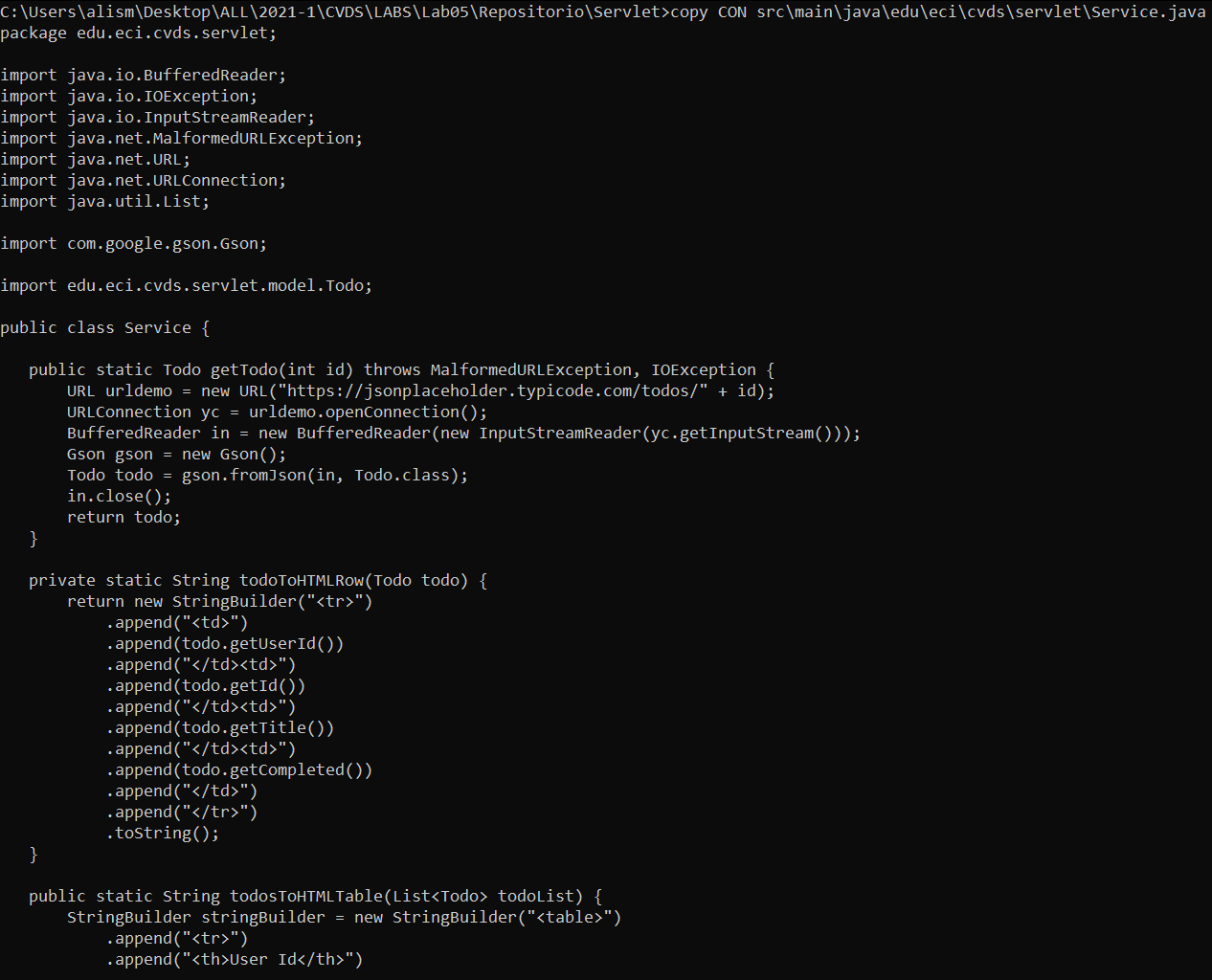
Primero creamos el paquete y el archivo de la clase



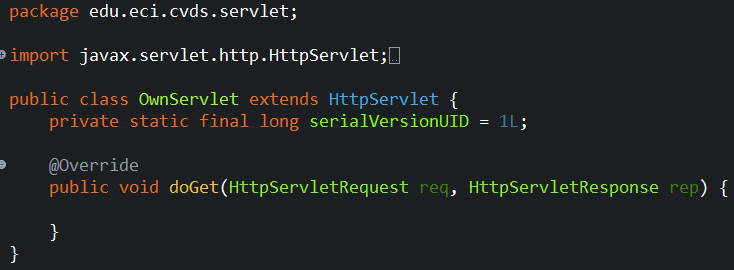
Realizamos el cascarón



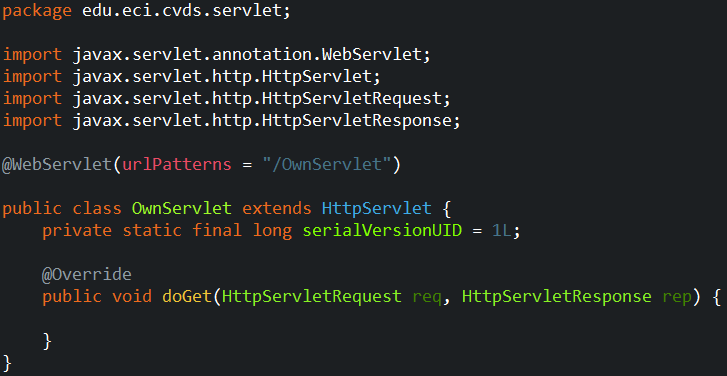
1. **Utilice la clase Service para consumir el servicio que se encuentra en la dirección URL del punto anterior**



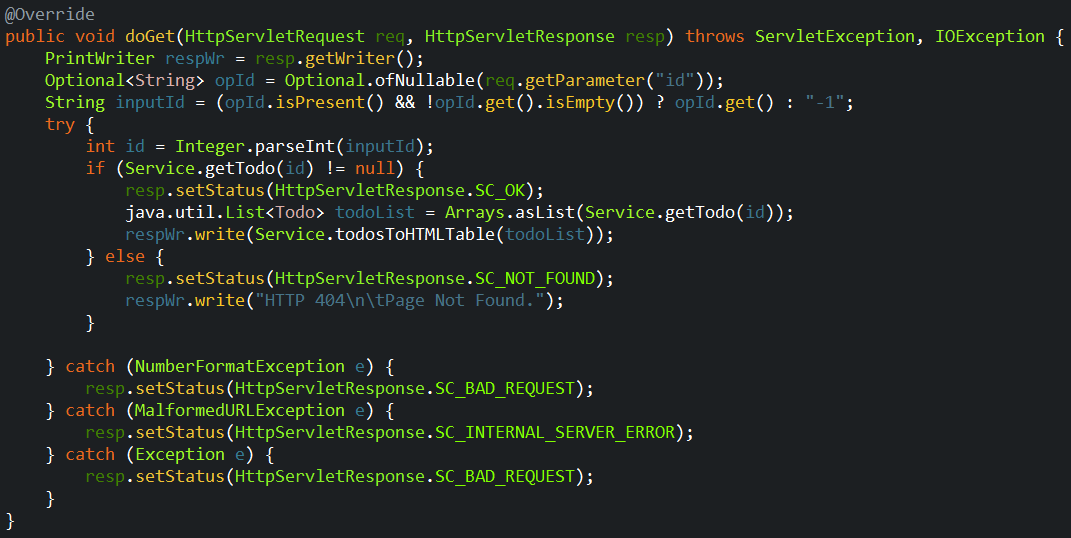
1. **Cree una clase que herede de la clase HttpServlet (similar a SampleServlet), y para la misma sobrescriba el método heredado doGet. Incluya la anotación @Override para verificar –en tiempo de compilación- que efectivamente se esté sobrescribiendo un método de las superclases**

****

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.**

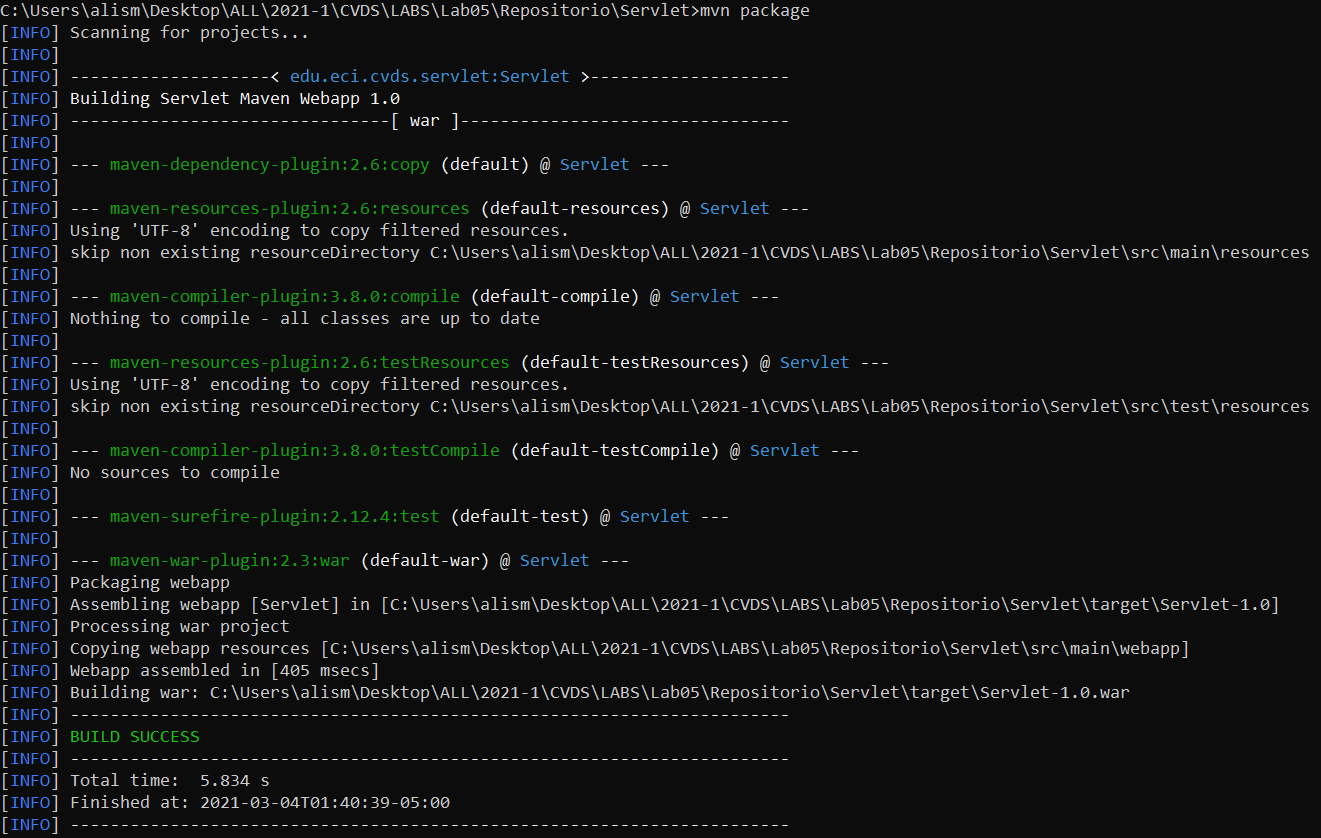


1. **Implemente el método doGet**

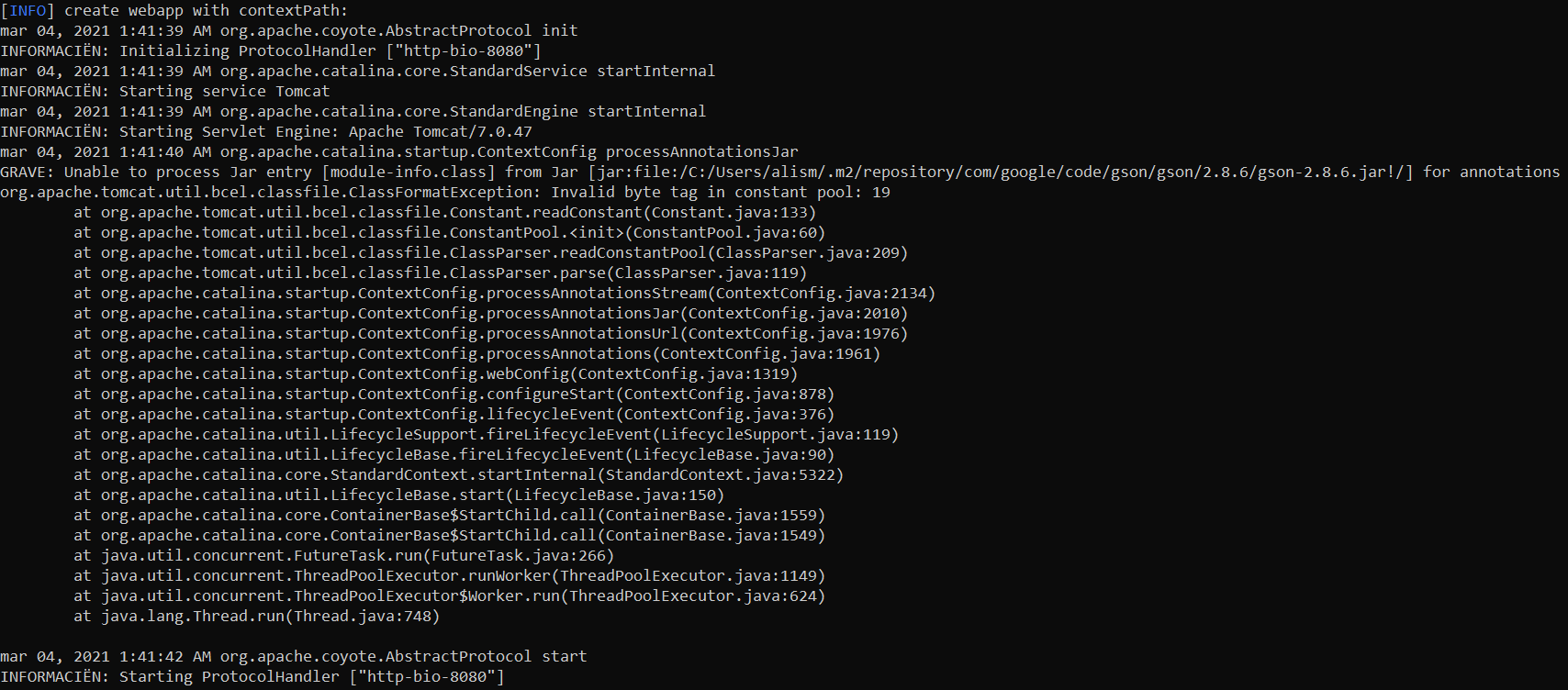


1. **verifique el funcionamiento de la aplicación, recompile y ejecute la aplicación**

Compilamos y empaquetamos de nuevo



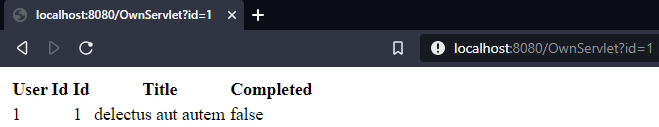
Y volvemos a ejecutar tomcat par poder volver a hacer las comprobaciones



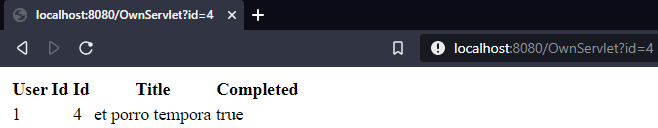
1. **Intente hacer diferentes consultas desde un navegador Web para probar las diferentes funcionalidades**

Al realizar las diferentes consultas sobre los mismos id’s, obtenemos los mismos valores que habíamos obtenido cuando hicimos las consultas en el punto 8.

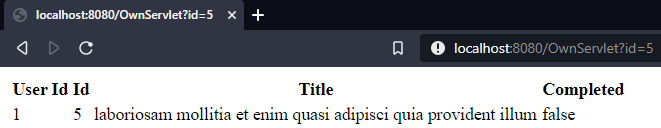
Con 1



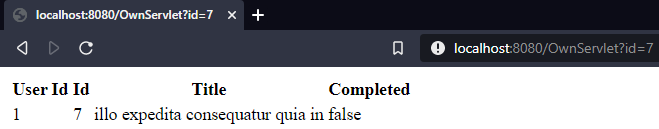
Con 4



Con 5

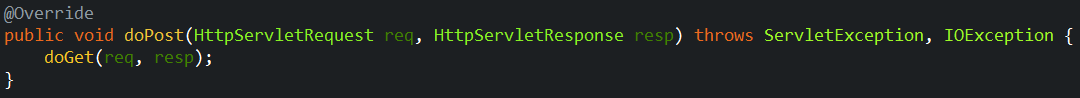


Con 7

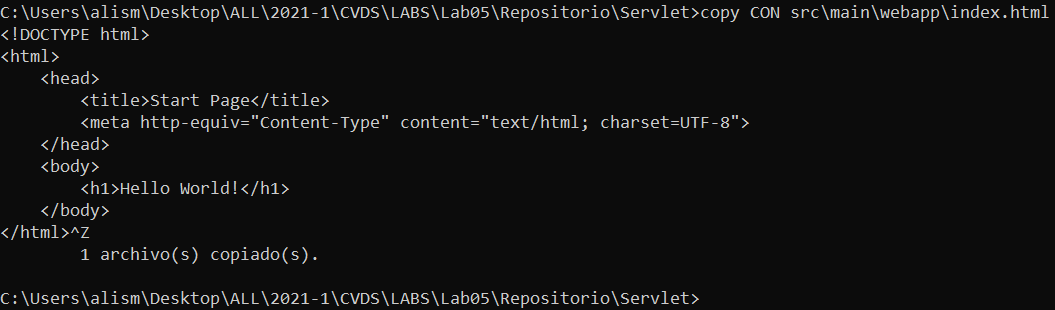


**PARTE III**

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

****

1. **Cree el archivo index.html en el directorio src/main/webapp/index.html**

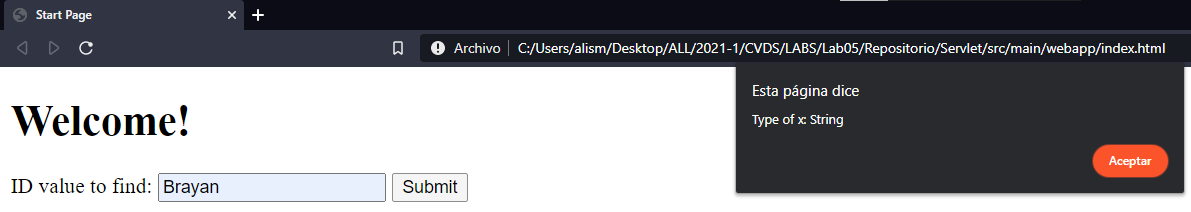
****

1. **En la página anterior, cree un formulario que tenga un campo para ingresar un número y un botón. El formulario debe usar como método ‘POST’, y como acción, la ruta relativa del último servlet creado**

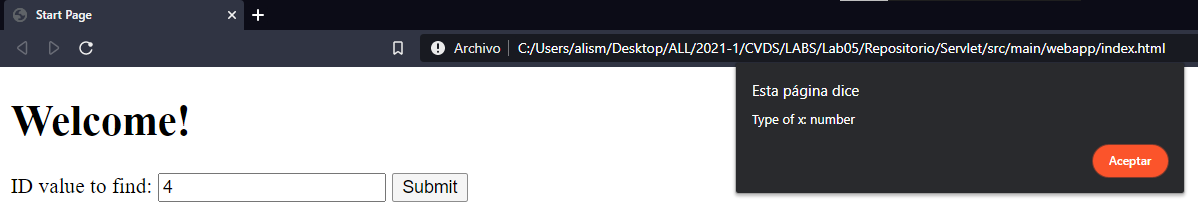
****

1. **Desde el browser valide que el valor ingresado es un valor numérico.**

Cuando se ingresa una cadena

****

Cuando se ingresa un número



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