Lab 5 CVDS

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

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

Revise el RFC del protocolo HTTP, sobre cómo realizar una petición GET. Debe lucir más o menos de esta forma:

GET /with-a-resource.html HTTP/1.0

Host: www.escuelaing.edu.co

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?

Realice una nueva conexión con telnet, esta vez a:

Host: www.httpbin.org

Puerto: 80

Versión HTTP: 1.1

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

CURL:  
¿Cuál esla diferencia entre los verbos GET y POST? ¿Qué otros tipos de peticiones existen?

GET: Se utiliza para solicitar datos del servidor. Cuando se envía una solicitud GET, los parámetros de la solicitud se incluyen en la URL como parte de la cadena de consulta (query string). Esta solicitud es idempotente, lo que significa que realizar la misma solicitud GET varias veces no debería tener efectos diferentes en el servidor, es decir, no debería cambiar el estado del servidor. Se utiliza comúnmente para recuperar recursos (por ejemplo, páginas web, imágenes, archivos) del servidor.

POST: Se utiliza para enviar datos al servidor para su procesamiento. A diferencia de GET, los parámetros de la solicitud POST se envían en el cuerpo del mensaje HTTP, lo que permite enviar una cantidad significativa de datos al servidor. Esta solicitud no es idempotente, lo que significa que realizar la misma solicitud POST varias veces puede tener diferentes efectos en el servidor, como crear un nuevo recurso o actualizar datos existentes. Se utiliza comúnmente para enviar formularios en línea, realizar actualizaciones en la base de datos del servidor, etc.

Otros tipos de peticiones HTTP incluyen:

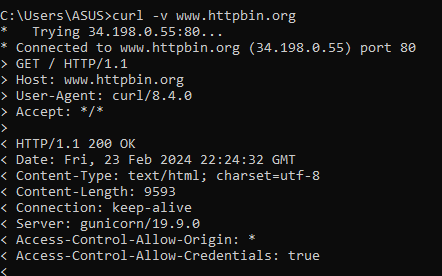
PUT: Se utiliza para enviar datos al servidor para ser almacenados en una ubicación específica. Similar a POST, pero generalmente se utiliza para actualizar recursos existentes en lugar de crear nuevos.

DELETE: Se utiliza para solicitar al servidor que elimine el recurso especificado.

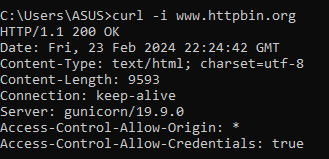
HEAD: Es similar a GET, pero el servidor solo debe responder con los encabezados de la respuesta y no con el cuerpo del mensaje. Se utiliza comúnmente para obtener información sobre un recurso sin solicitar el cuerpo completo del recurso, lo que puede ser útil para verificar la disponibilidad o la fecha de modificación de un recurso.

OPTIONS: Se utiliza para solicitar información sobre las opciones de comunicación disponibles para el recurso de destino o el servidor en sí. Esto puede incluir los métodos de solicitud permitidos, los encabezados admitidos, etc.

curl -v



curl -i



-v (o --verbose): Este parámetro indica a curl que muestre información detallada sobre la comunicación realizada entre el cliente y el servidor. Cuando se utiliza -v, curl mostrará:

Los encabezados de la solicitud HTTP enviada al servidor.

Los encabezados de la respuesta HTTP recibida del servidor.

Información adicional sobre la conexión, como la dirección IP del servidor y los tiempos de conexión.

En resumen, -v activa el modo verbose, proporcionando una salida más detallada que incluye los encabezados de solicitud y respuesta.

-i (o --include): Este parámetro indica a curl que incluya los encabezados de respuesta HTTP en la salida. Cuando se utiliza -i, curl mostrará:

Los encabezados de la respuesta HTTP, incluyendo el código de estado y otros encabezados como Content-Type, Content-Length, etc.

El cuerpo de la respuesta, si lo hay.

### **PARTE II. - HACIENDO UNA APLICACIÓN WEB DINÁMICA USANDO EL PATRÓN MVC**

¿Por qué MVC obtiene ese nombre?

MVC, que significa Modelo-Vista-Controlador, obtiene su nombre de la estructura de diseño que separa una aplicación en tres componentes principales:

Modelo: Representa los datos y la lógica empresarial de la aplicación. El modelo maneja la lógica de la aplicación y la interacción con la base de datos, si es necesario.

Vista: Es la interfaz de usuario que presenta los datos al usuario y maneja la interacción del usuario, como la entrada de datos y los eventos del usuario.

Controlador: Actúa como intermediario entre el modelo y la vista. Responde a las solicitudes del usuario, realiza las operaciones necesarias en el modelo y selecciona la vista adecuada para mostrar los resultados al usuario.

¿Cuáles son las ventajas de usar MVC?

Separación de preocupaciones: MVC permite separar la lógica empresarial, la presentación y el control de la aplicación, lo que facilita la organización del código y la mantenibilidad del proyecto.

Reutilización de código: La separación de los componentes permite reutilizar fácilmente el código en diferentes partes de la aplicación o en otras aplicaciones.

Facilidad de desarrollo en equipo: Al dividir la aplicación en componentes claros y definidos, diferentes desarrolladores pueden trabajar en diferentes partes de la aplicación de manera independiente y coordinada.

Pruebas unitarias más sencillas: Al dividir la aplicación en componentes independientes, es más fácil escribir pruebas unitarias para cada componente, lo que facilita la detección y corrección de errores.

¿Qué diferencia tiene la estructura de directorios de este proyecto comparado con las de proyectos pasados (con solo maven y java EE)?

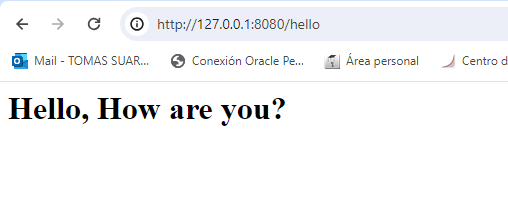
La diferencia radica en la organización de los archivos y carpetas para seguir el patrón MVC

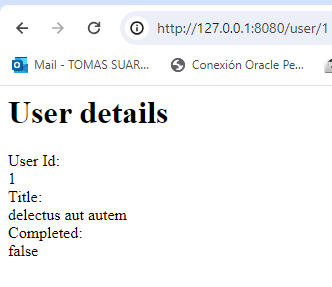
En un proyecto MVC, generalmente se encuentran carpetas separadas para los modelos, las vistas y los controladores. Esto puede resultar en una estructura de directorios más modular y organizada.

En Maven y Java EE, es posible encontrar una estructura de directorios más plana, donde todos los archivos y carpetas relacionados con la aplicación se encuentran en el mismo nivel o en estructuras jerárquicas más simples. La separación de preocupaciones no es tan evidente como en un proyecto MVC.

¿Qué anotaciones usaste y cuál es la diferencia entre ellas?

### **PARTE III. - APLICACIÓN MVC PARA CONSUMO DE SERVICIO RESTful**





¿Qué es RESTful?

En un sistema RESTful, los recursos son representados por URLs (Uniform Resource Locators) y son accesibles a través de operaciones HTTP estándar, como GET, POST, PUT y DELETE. Estos recursos pueden ser cualquier cosa que pueda ser identificada de forma única, como documentos, imágenes, videos, usuarios, etc.

Los principios clave de RESTful incluyen:

Recursos: Los recursos son la unidad central de un sistema RESTful y son identificados por URLs únicas. Cada recurso debe tener una representación, que puede ser en diferentes formatos como JSON, XML, HTML, etc.

Operaciones HTTP: Las operaciones CRUD (Crear, Leer, Actualizar, Borrar) se mapean a los métodos HTTP estándar: GET para leer, POST para crear, PUT para actualizar y DELETE para borrar recursos.

Sin estado: Cada solicitud HTTP debe contener toda la información necesaria para procesarla. El servidor no almacena ningún estado de sesión del cliente entre las solicitudes.

Interfaz uniforme: Los clientes y servidores se comunican a través de una interfaz uniforme estándar, lo que promueve la simplicidad, la escalabilidad y la independencia entre componentes.

Cliente-servidor: El cliente y el servidor son independientes entre sí y pueden evolucionar de forma independiente. Esto permite una mayor escalabilidad y modularidad del sistema.

Sistema sin caché: Las respuestas deben ser etiquetadas como cachéables o no-cachéables. Esto permite a los clientes almacenar en caché las respuestas para mejorar el rendimiento y la eficiencia de la red.

### **PARTE IV. - APLICACIÓN MVC JUEGO**

¿Qué pasa si abro el sitio de juegos en dos navegadores diferentes?

Si se abre el sitio de juegos en dos navegadores diferentes y se juega simultáneamente en ambos, cada navegador mantendrá su propio estado del juego. Esto significa que cada navegador tendrá su propio juego en curso, con su propio número a adivinar y su propio saldo. No habrá interferencia entre los dos juegos, ya que cada sesión del navegador es independiente

Si quisiera qué a cada jugador le aparecieran independientemente sus respectivos saldos. ¿Qué habría que hacer?

Si se quisiera que a cada jugador le aparecieran independientemente sus respectivos saldos, se tendría que implementar una funcionalidad de gestión de sesiones. Esto implicaría asignar una sesión única a cada jugador cuando ingresan al sitio y luego mantener la información específica de cada jugador asociada con su sesión. De esta manera, cada jugador tendría su propio saldo, independientemente de cuántas personas estén jugando al mismo tiempo en el sitio.