# Taller Clientes y Servicios

**Santiago Martínez Martínez**

**Ingeniería de Sistemas**
**Noveno Semestre**
**Escuela Colombiana de Ingeniería Julio Garavito**
**Bogota DC - Colombia**

# Glosario

**Protocolo HTTP: Es un protocolo basado en el principio de arquitectura cliente-servidor. En este caso el usuario enviá una petición y el servidor responde el requerimiento.**

**Servidor HTTP: Hace referencia a un programa que procesa las aplicaciones del lado del servidor, realiza conexiones sincronas o asíncronas con el cliente que realiza peticiones.**

**Cliente-Servidor: Es un modelo de diseño de software en el que se diferencian dos actores principales, uno es el proveedor de servicios, más conocido como servidor y un actor que realiza peticiones o llamados a recursos, que es conocido como cliente.**

# Resumen

**Este documento tiene como objetivo explicar al detalle la implementación realizada para poder diseñar y colocar en funcionamiento un servidor web sin ayuda de Frameworks como Spark o Spring.**

**En este taller para poder realizar la aplicación para crear un servidor HTTP se utilizó Java con el paquete Java.net, el cual ofrece clases para manejar los sockets.**

**Todos los recursos que se muestran en la aplicación se encuentran en la carpeta de resources, la cual tiene como principal fin, poder retornar la información solicitada dependiendo del path que ingrese el usuario, todos los archivos son esencialmente archivos utilizados para realizar el front de las aplicaciones, los cuales son HTML, CSS y JS.**

**Por ultimo, todo las peticiones GET que recibe el servidor son realizadas utilizando funciones lambda, gracias a estas podemos recibir las peticiones y luego añadir una respuesta por parte del servidor.**

# Contents

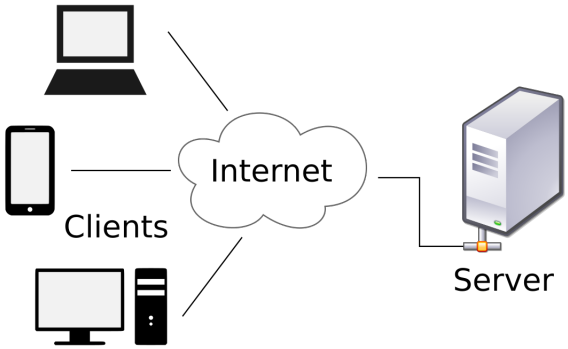


# 1 Introduction

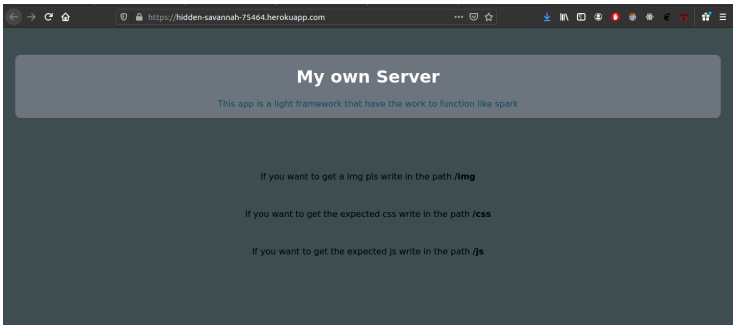
El manejo de la estructura del proyecto se realizo con Maven, ya que es una herramienta que nos ayuda a gestionar los directorios de nuestro proyecto y como base mas importante, nos ayuda a tener una buena administración de las decencias de nuestro proyecto. Como no se utilizó ningún framework, la única dependencia esencial del proyecto es JUnit para nuestras pruebas.

El objetivo de este taller, es poder retornar servicios gracias al que el usuario ingresa un path determinado el cual al momento de ser enviado para que el servidor responda, este nos devuelva el recurso solicitado. En mi caso todo esto lo realizo utilizando la url y no un input en el html, ya que no logré acoplar el javascript con el servidor.

Para poder retornar la pagina de inicio, al igual que los demás recursos se utilizo el paquete File de java, este paquete nos permite crear un objeto File




con el cual podemos leer cada linea del archivo y así con esto retornar cada linea en un String y que por medio del servidor este sea interpretado como un documento html.




## 2 Objetivos

El taller tenia 2 retos que se debían cumplir, el primero es construir un servidor web que soportara múltiples solicitudes. Este debe retornar todos los archivos que son solicitados, ya sean imágenes o documentos html.

El segundo reto era construir un framework parecido a Spark y a Spring el cual se conectará a una base de datos y que este permite acceder a recursos estáticos como documentos js, css o imágenes.

## 3 Solución Requerimientos

Lo primero que se hizo fue construir el servidor con ayuda del paquete java.net, este nos permitió tener un mejor manejo de los sockets del servidor, del socket de cliente y por supuesto de la creación del servidor para responder

las solicitudes. Un requerimiento importante es que este servidor web soporte múltiples solicitudes seguidas, para esto se implementó un ciclo que siempre lea las solicitudes del socket del cliente, es decir, de esta manera el servidor siempre queda activo a solicitudes secuenciales.

Para poder retornas los archivos que son solicitados, lo que se uso fue el paquete java.file, con este paquete convertíamos todos los archivos que se encuentran en la carpeta resources en un objeto File, para poder leer cada linea y adjuntarlo a una variable String, para después por medio de la función lambda, retornar el archivo solicitado.

Para poder cumplir con el requerimiento de que los request al servidor sean utilizando funciones lambda lo que realice, fue utilizar la API de java, la cual nos proporciona un operador binario el cual es BiFunction que soporta request http y permite retornar cualquier tipo de resultado.

# 4 Frontend

La interfaz grafica con la que interactuá el usuario se realizo con html en un solo documento, esto porque al momento de tratar de dividir las responsabilidades en el front, las cuales son estructura, estilo y funcionamiento me vi en errores, entonces tomé la decisión de utilizar las etiquetas de style y script para adjuntar los estilos de las etiquetas y la funcionalidad de cada componente.

```
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Own Server</title>
   <style>
       body {
    background-color: #3e4e50;
}


.container {
 text-align: center;
 align-items: center;
 font-family: 'Hammersmith One', sans-serif;
 margin: 50px auto;
}
```

# 5 Backend

El backend, como ya se comento antes fue escrito con Java y utilizando paquetes necesarios para el control del trafico y los archivos solicitantes.

El manejo de las peticiones las manejo con la clase SeudoSpark, la cual tiene los métodos principales para que el servidor pueda funcionar correctamente, los cuales son el método get(), port() y startServer().

Esta clase maneja la instancia de la clase SeudoSparkServer, la cual implementa un singleton porque solo queremos tener una instancia de esta clase, esta se encarga del manejo de las peticiones y de retornar un template de error en caso de que la petición a la url sea incorrecta.

```java
public class DemoServer {
    Run | Debug
    public static void main(String[] args) throws IOException {
        port(getPort());
        String main = getFile("index.html");
        get("", (req, resp) -> main);
        String img = getFile("img.html");
        get("img", (req , resp) -> img);
        String css = getFile("index.css");
        get("css", (req, resp) -> css);
        String js = getFile("app.js");
        get("js", (req, resp) -> js);
        startServer();
    }
```

# 6 Conexion

```java
public void startServer(int port) throws IOException {
    this.port = port;
    ServerSocket serverSocket = null;
    try {
        serverSocket = new ServerSocket(getPort());
    } catch (IOException e) {
        System.err.println("Could not listen on port: 35000. "+e.toString());
        System.exit(1);
    }

    boolean running = true;
    while (running) {
        Socket clientSocket = null;
        try {
            System.out.println("Listo para recibir ..." + port);
            clientSocket = serverSocket.accept();
        } catch (IOException e) {
            System.err.println("Accept failed.");
            System.exit(1);
        }
```

En este caso, el factor que yo creo fundamental, es la conexión entre el front y el back por medio de la url, esto gracias al socket del cliente que nuestro servidor http siempre atiende, gracias a esta conexión logramos que el programa siempre este en ejecución y podamos realizar los request que creamos necesarios.

# 7 Conclusion

1. Me di cuenta que para implementar un servidor http que realice acciones simples, no es necesario tener un framework que sea robusto y pesado, para esto, podemos de implementar una versión mas ligera y funcional.

2. Las funciones lambda nos ayudan a realizar múltiples acciones que deseamos, esto con recorrer listas, extraer datos condicionados, como también responder solicitudes HTTP.

3. Los canales de conexión entre las diferentes componentes del sistema son de suma importancia, ya que gracias a estos podemos transferir datos.

# References


[1] *MDV.*
`https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_`$globales/Promise$

[2] *Requests.*
`https://javascript.info/promise-chaining`

[3] *Simple HTTP Server.*
`https://dzone.com/articles/simple-http-server-in-java`

[4] *Tutorial.*
`https://es.overleaf.com/learn/latex/Tutorials`