FACULTAD DE INGENIERIA
INGENIERIA WEB II
Servidor HTTP - Uso de CORS - Actividades con Base de Datos
Docente: Ing. Carlos Simes

En esta actividad que arranca de un proyecto base, usaremos el servidor web desarrollado en java y que denominamos IW2.
Este servidor realiza una serie de funciones, las cuales, como actividad integradora final de la asignatura deberán ser complementadas según lo que se solicita al final de este documento.

El servidor tal como está implementado en este proyecto de Netbeans, realiza las siguientes funciones que ya están implementadas:

1. Recibe pedidos **GET** de clientes web a través de un socket. Este socket puede escuchar en cualquier puerto que se indique. En el proyecto escucha en el puerto **8866**.
2. El socket es colocado por una clase administradora del servidor, en un hilo. De esta manera es posible, colocando los pedidos en hilos diferentes, procesar múltiples pedidos de manera simultánea.
3. El pedido GET es recibido por el Socket como un flujo de bytes a través de un stream. Este es procesado para primero verificar que el pedido sea efectivamente una solicitud GET y luego debemos extraer los parámetros en caso que existan para procesarlos.
   Dentro de estos parámetros, encontraremos uno especial que denominamos **'accion'**. Dicho parámetro sirve para indicarle al server que le estamos pidiendo en especial, por ejemplo **accion=ListarProductos**, le indicaría que realice una consulta a una base de datos y nos provea el listado de productos.
4. Una vez que tenemos la acción solicitada y los parámetros, vamos a realizar la acción si es que la misma está implementada, sino se devuelve un mensaje de acción no disponible.
   En este proyecto solo se ha implementado una función listar que consulta una base de datos y devuelve una lista de elementos.
5. Para consultar la base de datos, usaremos patrones de diseño denominados **DAO y DTO** que se explican más adelante.
6. Una vez realizada la consulta a la base de datos, se construye la respuesta a nuestro cliente web, la cual por cuestiones de estandarización será en formato **JSON**.

Como el servidor debe realizar todas estas funciones, a fin de mejorar el diseño y la implementación del mismo, se ha dividido el servidor en varias clases especializadas que realizan las tareas indicadas.

El servidor **IW2** está compuesto por 4 clases básicas más una clase que sirve para probar los servicios.

Además se ha agregado clases basadas en el patrón **DAO** (Data Access Object) y objetos **DTO** (Data Transfer Object) para realizar el acceso a datos. implementando interfaces (**intDAO**) en las que se han especificado los métodos de acceso CRUD (**Create Read Update Delete**).

*Un **DAO** es un objeto de acceso a datos (en inglés, data access object, abreviado DAO). Es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo.*
*Un objeto DAO encapsula las operaciones de acceso y procesamiento de Datos para una entidad en particular. Por ejemplo en una aplicación web que procesa productos, productosDAO será una clase que implementará los métodos de lectura, actualización, creación y borrado de objetos producto en la base de datos. En una mejora de este patrón puede usarse una interface con métodos genéricos, **intDAO**. Esta interfaz se implementará en cada DAO particular, pero al usar la interfaz, nos aseguramos que los DAO tengan todos los mismos métodos de acceso a Base de Datos.*

*Un **DTO** es un objeto de transferencia de datos (en inglés, data transfer object, abreviado DTO) es un objeto que transporta datos entre procesos. La motivación de su uso tiene relación con el hecho que la comunicación entre procesos se realiza generalmente mediante interfaces remotas (por ejemplo, servicios web), donde cada llamada es una operación costosa.*
Como la mayor parte del costo de cada llamada está relacionada con la comunicación de ida y vuelta entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola llamada.
*Para explicarlo más simple, un DTO es simplemente un objeto en memoria del cliente que es una copia de un registro de una tabla en la base de datos. Supongamos que tenemos la aplicación que procesa productos. Si quisiéramos ver un listado de productos, deberíamos hacer un Select a la Base de Datos para obtener ese listado. Si usamos muchas veces ese listado para consultar los productos, es costoso realizar innumerables consultas a la base de datos, que es remota y asiste a todos los clientes de la webapp para consultar casi siempre lo mismo. Entonces podemos crear objetos **productosDTO** que se guarden en la memoria del cliente (un arraylist de productosDTO) equivalentes a los registros de la tabla productos. Salvo que la información de estos productos cambie en la base de datos, podremos acceder a los objetos DTO que cargamos en el cliente y tener la misma información más rápido y sin consultar nuevamente la BD.*

Volviendo a nuestro servidor IW2, el mismo está en proceso de desarrollo y la idea es que ustedes puedan mejorarlo para usarlo con el trabajo final de la asignatura.
Este servidor con explicamos antes, realiza las siguientes funcionalidades:
Recibe mandatos que permiten realizar acciones en la base de datos. El formato implementado actualmente sigue los principios de acceso a los web services
Por ejemplo si se desea obtener una lista de entidades, el mandato es

localhost:xxxx/Listar

En un webservice correctamente implementado, el acceso debería ser

URL del sitio (por ahora usamos localhost:xxxx) / NombreAplicacion /Entidad / Accion / parámetros

Por ejemplo si se deseara acceder a un listado de productos y nuestra aplicación se llama ProductosWs la URL completa debería ser:

localhost:xxxx/ProductosWS/Producto/Listar

Si quisiéramos buscar un producto cuyo **codigo=nnn**, la URL debería ser:

localhost:xxxx/ProductosWS/Producto/Buscar/codigo=nnn

Como estamos implementando todo con el mandato http **GET**, entonces el encabezado que recibe el servidor cuando hacemos el pedido es

GET/ ProductosWS/Producto/Buscar/ codigo=nnn HTTP/1.0

Nombre App **Entidad** **Acción** **Parámetros**

El servidor NO posee todavía un analizador de la URL, pero deberían implementarlo para el trabajo final si desean respetar las convenciones para los webservices.

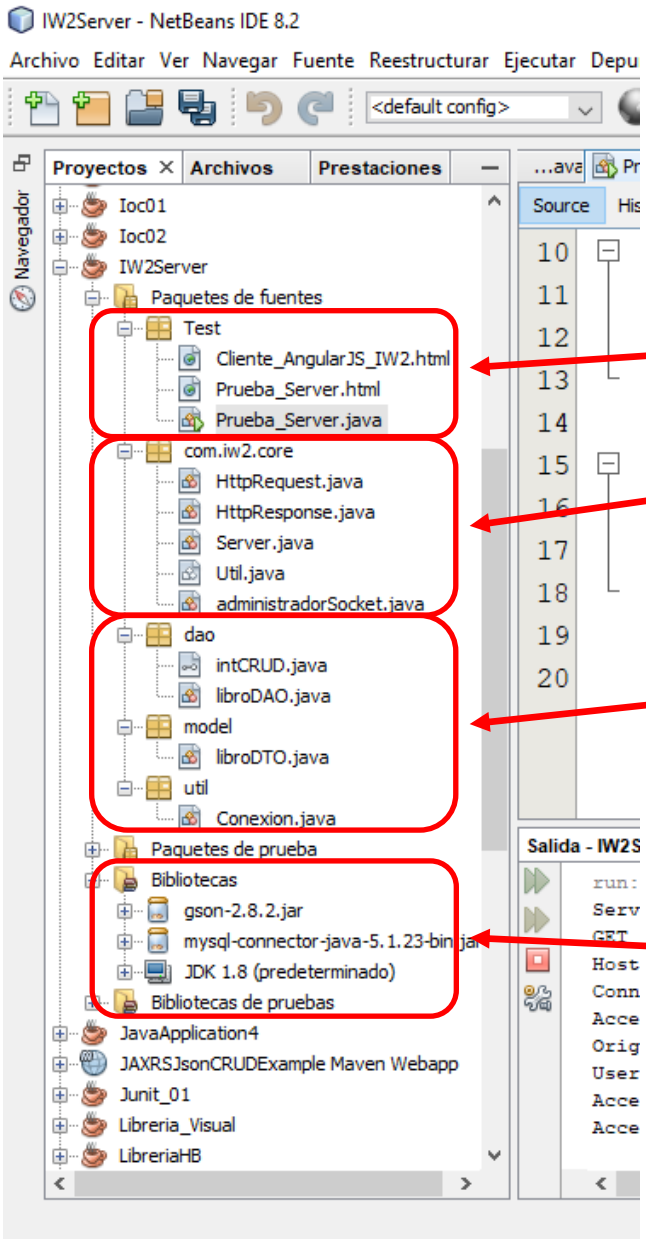
Antes de seguir adelante con el desarrollo de este proyecto, veamos una explicación grafica de los componentes del servidor que vamos a implementar y sus funciones y un video explicativo de como poner en marcha el Servidor:

FUNCIONAMIENTO DEL WEB SERVER IW2

Puesta en Marcha y Funcionamiento del Server IW2

Se ha agregado también una página HTML con **Bootstrap** y **AngularJS** que accede a la base de datos y lee la información contenida en ella.
Cuando ejecuten el proyecto, deberán ver una página similar a esta:

La estructura del proyecto actual es la siguiente:



Paquete con los componentes para probar el server

Paquete con las clases base del Server

Paquete de acceso y manipulación de datos

Librerías para conectar con MySql y para convertir a JSON (librería Gson de Google para java)

Código de las Clases del Proyecto:

Cliente_AngularJS_IW2.html

Esta es una página para probar el servidor, la cual solicita mediante el objeto **$http** y el método get, el listado de libros y los muestra al usuario.

El objeto **$http** es provisto por Angular para facilitarnos la conectividad con nuestro servidor usando el protocolo HTTP. Su uso es extremadamente sencillo. Para usarlo hacemos lo siguiente:

```
var app = angular.module('adminPanel', []);
app.controller('librosController', function($scope,$http) {
```

```
    $http.get("http://localhost:8866/Listar")
    .then(function (response) {$scope.libros = response.data;});
});
```

Aquí lo importante es:
El controlador de la App recibe una **función** a la que le inyectamos el **$scope** para acceder al modelo de datos de la app y el objeto **$http**, que nos provee los **métodos http** para comunicarnos con el servidor.

El objeto $http se usa tan fácil como hacer **$http.get (pedido).**

Además este objeto posee un método **then** que espera hasta que el servidor nos responda el pedido realizado. Dentro de este método, asignamos a nuestro modelo, el resultado del pedido. Listo ! ya obtuvimos la información del servidor. La página completa de prueba será la siguiente:

```
<!DOCTYPE html>
<html lang="en" ng-app="adminPanel">

<head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.js"><
/script>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/c
    ss/
    bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/
    popper.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/
    bootstrap.min.js"></script>
    <title> Cliente Angular JS de la Librería </title>
</head>

<body>
    <div class="container col-md-12" style="background-color: powderblue">
        <h2 align="center"> Sistema de Gestión de Libros </h2>
    </div>

    <div class="container col-md-8 col-md-offset-2" ng-controller="librosController">
        <table class="table table-striped  table-hover table-responsive">
            <th colspan="4" align="center">Listado de Libros Registrados</th>
            <tr>
                <td> ID LIBRO </td>
                <td> TITULO</td>
                <td> AUTOR </td>
                <td> PRECIO </td>
            </tr>
            <tr ng-repeat="data in libros">
                <td>{{data.libro_id}}</td>
                <td>{{data.titulo}}</td>
                <td>{{data.autor}}</td>
                <td>{{data.precio}}</td>
            </tr>
```

```
        </table>
    </div>

    <script>
        var app = angular.module('adminPanel', []);
        app.controller('librosController', function ($scope, $http) {

            $http.get("http://localhost:8866/Listar")

                .then(function (response) { $scope.libros = response.data; });
        });
    </script>

</body>
</html>
```

Prueba_Server.html

Página similar a la anterior, pero sin usar Bootstrap. La solicitud Http.get se realiza por acción del usuario sobre un objeto Select

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Cliente Web en Angular</title>
    <style>
        body {
            font-family: sans-serif;
        }

        li {
            font-size: 0.8em;
        }

        li span {
            font-weight: bold;
        }
    </style>
</head>



<body>
    <div ng-app="apiApp" ng-controller="apiAppCtrl as vm">
        <h1>Cliente REST en Angular</h1>
        <p>
            Selecciona:
            <select ng-model="vm.url" ng-change="vm.resolver()">
                <option value="http://localhost:8866/Listar">Listado de Libros</option>
            </select>
        </p>
        <table border=2>
            <thead>
                <td>Titulo</td>
                <td> Autor </td>
            </thead>
```

```
            <tr ng-repeat="libro in vm.Libros">
                <td>{{libro.titulo}}</td>
                <td>
                    {{libro.autor}}</td>
        </table>
    </div>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.24/angular.min.js">
</script>

    <script>
        angular
            .module('apiApp', [])
            .controller('apiAppCtrl', ['$http', controladorPrincipal]);
        function controladorPrincipal($http) {
            var vm = this;
            vm.resolver = function () {
                $http.get(vm.url).success(function (respuesta) {
                    vm.Libros = respuesta;
                });
            }

        }
    </script>
</body>
</html>
```

Prueba_Server.java

// Clase que inicia el servidor en un puerto determinado, en este caso en el
// 8866

```
package Test;
import com.iw2.core.Server;
/**
 *
 * @author CARLOMAGNO
 */
public class Prueba_Server {
    public static void main(String[] args) {
    Server s1=new Server();
    s1.IniciarServidor(8866);
    }
}
```

HttpRequest.java

// Clase que captura el pedido del cliente, lo procesa y devuelve el método de la
// solicitud, los parámetros recibidos, un parámetro en particular y también
// tiene la capacidad de enviar un archivo que se encuentre en el servidor.
// por ahora solo se implementó envío de archivos tipo: text/html, css, js,
// gif,jpg,pdf,doc,xls

```

package com.iw2.core;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
```

```java
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.StringTokenizer;

/**
  * @author CARLOMAGNO
 */
public class HttpRequest {
  InputStream flujoentrada;
  private static BufferedReader buffer;
  private String[] parametros = null;
  private String[] HttpRequest;
  private String parametro = null;
  private String metodoPedido;
  private String linea;
  private String httpPedido;
  private String header;
  private String NEW_LINE = "\r\n";
  private String accion;
  private String SITE_NAME = "api";

  public HttpRequest(String flujoentrada) throws IOException {
    httpPedido = flujoentrada;
    System.out.println(httpPedido);

    // como httpPedido es un String separados por saltos de linea, generamos un
    // array

    HttpRequest = httpPedido.split("\r\n");
  }
   // Funcion que analiza el pedido y ve si es un GET, POST, etc.
  public String getMetodo() throws IOException {
    // Sout(HttpRequest[0]); Si activamos esta instrucción, podemos ver lo que recibe el
    //server en la consola de Netbeans
    String[] palabras = HttpRequest[0].split("/");
    metodoPedido = palabras[0];
    return metodoPedido;
  }


// Funcion para obtener que se ha solicitado como accion
  public String getAccion() throws IOException {
    accion = null;
    String[] palabras = HttpRequest[0].split("/");
    accion = palabras[1].substring(0, palabras[1].indexOf("HTTP")).trim();
    return accion;
  }
  // Obtenemos los parámetros (variables ) que nos envía el cliente
  public String[] getParametros() throws IOException {
    parametros = null;
    String[] palabras = HttpRequest[0].split("/");
    String Pedido = palabras[1].substring(0, palabras[1].indexOf("HTTP"));
    Sout(Pedido);
    if (Pedido.length() > 0) {
      Sout("Estoy en length >0");
      if (Pedido.indexOf("?") != -1) { // hay `por lo menos un parametro
```

```
        Sout("Estoy en no tengo ?");
        if (Pedido.indexOf("&") == -1) { // Hay un solo parametro
          Sout("Estoy en no tengo &");
          Pedido += "&";

         parametros = Pedido.split("&");
        } else {
          parametros = Pedido.substring(1).split("&");
        }
      } else {
        parametros = null;
        Sout("Estoy en no envíe parámetros ....");
      }
    }
    return parametros;
  }
  // Funcion para obtener los valores de cada parámetro
  public String getValorParametro(String parametro) throws IOException {
    String Busqueda = parametro;
    String Valor = null;
    for (int i = 0; i < parametros.length; i++) {

      if (parametros[i].contains(Busqueda)) {
        Valor = parametros[i].substring(parametros[i].indexOf("=") + 1);
      }
    }
    return Valor;
  }

  public String getHeader() throws IOException {
    header = this.buffer.readLine();
    return header;
  }

  public String getHeaderBody() throws IOException {
    String headerBody = null;
    int posicionDatos;
    while ((linea = buffer.readLine()) != null && (linea.length() != 0)) {
      headerBody += "HTTP-HEADER: " + linea + "\n\r";
    }
    return headerBody;
  }


// Función para enviar archivos al cliente
  public void enviarArchivo(String nombreArchivo, PrintStream ps) throws IOException {
    String filename = "";
    if (nombreArchivo.equals("")) {
      String s = HttpRequest[0];
      StringTokenizer st = new StringTokenizer(s);

      try {

        // Paraeamos el nombre del archivo del comando Get
        if (st.hasMoreElements() && st.nextToken().equalsIgnoreCase("GET") &&
        st.hasMoreElements())
          filename = st.nextToken();
        else
          throw new FileNotFoundException(); // Bad request
```

```
        // Si no pidieron nada, enviamos la página index.html
        if (filename.endsWith("/"))
          filename += "index.html";


        // quitamos la / del nombre del archivo
        while (filename.indexOf("/") == 0)
          filename = filename.substring(1);

        // Reemplazamos la / por \ para archivos basados en windows
        filename = filename.replace('/', File.separator.charAt(0));

        // Como no se debe permitir accesos a subdirectorios elimimamos los caracteres
        // ..
        if (filename.indexOf("..") >= 0 || filename.indexOf(':') >= 0 ||
      filename.indexOf('|')
        >= 0)
          throw new FileNotFoundException();

        if (new File(filename).isDirectory()) {
          filename = filename.replace('\\', '/');
          ps.print("HTTP/1.0 301 Moved Permanently\r\n" + "Location: /"
      + filename + "/\r\n\r\n");
          ps.close();
          return;
        }

      } catch (FileNotFoundException x) {
        ps.print("HTTP/1.0 404 Not Found\r\n" + "Content-
      type: text/html\r\n\r\n" + "<html><head></head><body>"
            + filename + " No se encontró el archivo solicitado</body></html>\n");
        ps.close();
      }
    } else
      filename = nombreArchivo;

    // abrimos el archivo y lo enviamos
    // Open the file (may throw FileNotFoundException)

    System.out.println(filename);
    InputStream f = new FileInputStream(filename);

// Determine the MIME type and print HTTP header
    String mimeType = "text/plain";
    if (filename.endsWith(".html") || filename.endsWith(".htm"))
      mimeType = "text/html";
    else if (filename.endsWith(".jpg") || filename.endsWith(".jpeg"))
      mimeType = "image/jpeg";
    else if (filename.endsWith(".gif"))
      mimeType = "image/gif";
    else if (filename.endsWith(".pdf"))
      mimeType = "application/pdf";
    else if (filename.endsWith(".class"))
      mimeType = "application/octet-stream";
    else if (filename.endsWith(".doc"))
      mimeType = "application/msword";
    System.out.println(mimeType);
    ps.print("HTTP/1.0 200 OK\r\n" + "Content-type: " + mimeType + "\r\n\r\n");

    // Send file contents to client, then close the connection
```

```
    byte[] a = new byte[1048576];
    int n;


    while ((n = f.read(a)) > 0)
      ps.write(a, 0, n);

    ps.close();
  }
  private void Sout(Object object) {

    System.out.println(object);
  }
}
```

Clase **HttpResponse.java**

// permite generar la respuesta para el cliente y enviársela, incluyendo
// entre otras cosas el encabezado de respuesta
// En el encabezado de respuesta (implementado en la función getHeader, se
// ha incluido **encabezado CORS**. **("Access-Control-Allow-Origin:*")**
// Este encabezado le dice al navegador que el servidor admite recibir
// pedidos de **clientes provenientes de orígenes diferentes (diferentes //sitios),** ya que es posible que nuestro servidor implemente servicios que
// puedan ser usados por otras aplicaciones.
// del que reside nuestro servidor. De esta manera, nuestro servidor se
// transforma en un **servidor público**, que puede ser accedido por cualquier
// cliente que conozca la forma de acceso a las funciones del mismo.
// De otro modo, solo los clientes que pertenezcan al mismo dominio del
// servidor, pueden acceder a este y el servidor se transforma en un servidor
// privado, que es la opción por defecto.

```
package com.iw2.core;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.net.Socket;

/**
  * @author CARLOMAGNO
 */

public class HttpResponse {

    PrintWriter salida;
    private HttpRequest entrada;
    private String header;
    private String HTML_START = "<html><title>Servidor IW2 versión 1.0</title><body>";
    private String HTML_BODY = "<div style='background-color:#00B9FF ; width: 100%;font-
weight:bold;font-size: 36px;font-family: Verdana, Geneva, sans-serif;height: 10%; text-
align: center'>Servidor IW2 Ver 1.0</div><br><br><hr>";
```

```
    private String HTML_LEYENDA = "<div style='background-
color: #00FF95; width: 100%;font-weight:bold;font-size: 20px;font-
family: Verdana, Geneva, sans-serif;height: 20%; text-
align: left'>Los Servicios se han iniciado correctamente<br>Se reciben solicitudes GET c
on estructura:<br> /accion/?parametro1=valor1 .... <br>Ejemplo /buscar/producto=tornillo
</div><br><br><hr>";

    private String HTML_END = "</body></html>";
    private String NEW_LINE = "\r\n";

    public HttpResponse(Socket s) throws IOException {
        salida = new PrintWriter(s.getOutputStream()); // permite imprimir lineas

    }
    // Euncion para generar el Encabezado de la Respuesta

    public String getHeader() {
        header = "HTTP/1.0 200 OK" + NEW_LINE;
        header += "Access-Control-Allow-Origin:*" + NEW_LINE;
        header += "Content-Type:text/html;charset=utf-8" + NEW_LINE;
        return header;
    }

    // Funcion para generar la página inicial
    public String getInitPage(String innerHtml) {
        String PaginaInicial = NEW_LINE;
        PaginaInicial += this.HTML_START;
        PaginaInicial += this.HTML_BODY;
        PaginaInicial += this.HTML_LEYENDA;
        PaginaInicial += "<div style='background-color: #F9FF00; width: 100%;font-
weight:bold;font-size: 20px;font-family: Verdana, Geneva, sans-serif;height: 70%; text-
align: left;content-align: center'>";
        if (innerHtml != null) {
            PaginaInicial += innerHtml;
        }
        ;
        PaginaInicial += "</div>" + this.HTML_END;
        return PaginaInicial;
    }
    // Funcion Echo, devuelve un eco con la info recibida
    public String getEcho(String[] parametros) {
        // este método devuelve una página simple de bienvenida
        // donde se muestran los parámetros recibidos
        String paginaEcho = null;
        String responseHeader = this.getHeader();
        String infoRecibida = "";

        infoRecibida = "<table style='width: 50%;background-
color: powderblue' border=2>";
        infoRecibida += "<th colspan=2> Información Recibida desde el Web Browser </th>"
;
        infoRecibida += "<tr><td align=center> parametro </td><td align=center> valor </
td></tr>";

        for (String parametro : parametros) {
            String[] data;
            data = parametro.split("=");
            String filaTabla = "<tr><td>" + data[0] + "</td><td>" + data[1] + "</td></tr
>";
            infoRecibida += filaTabla;
```

```
        }
        infoRecibida += "</table>";
        String initPage = this.getInitPage(infoRecibida);
        paginaEcho = responseHeader + NEW_LINE + initPage;
        return paginaEcho;
    }

// impression de control por el System Out
    public void imprimirSalida(String Mensaje) {
        System.out.println(Mensaje);
        salida.println(Mensaje);

    }

    public void cerrar() {
        salida.close();
    }
    // Declaracion de las constantes de Encabezado
    private static class Headers {
        public static final String SERVER = "IW2 - Server";
        public static final String CONNECTION = "Connection";
        public static final String CONTENT_LENGTH = "Content-Length";
        public static final String CONTENT_TYPE = "Content-Type";
        // agregamos encabezados CORS
        // public static final String ACCESS-CONTROL-ALLOW-ORIGIN="*";
    }

    private static class Status {
        public static final String HTTP_200 = "HTTP/1.1 200 OK";
        public static final String HTTP_404 = "HTTP/1.1 404 Recurso no Encontrado";
    }

}
```

clase **Server.java**

// inicia el servidor y acepta pedidos en el puerto indicado, pasándoles los
// mismos al administrador de sockets. Este los colocará luego en un hilo

```
package com.iw2.core;

import java.net.ServerSocket;
/**
  * @author CARLOMAGNO
 */
public class Server {
    // private static final int PORT=8877;
    public void IniciarServidor(int Puerto) {
        try {
            ServerSocket server = new ServerSocket(Puerto);
            System.out.println("Servidor IW2 funcionado en el Puerto: " + Puerto);
            while (true) {
                new administradorSocket(server.accept()).procesarSocket();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

```
}
```

Clase **administradorSocket.java**

Extiende a la Clase **Thread** (Hilo) para manejar conexiones simultaneas en distintos procesos. Además, usaremos **DAO y DTO** para consultar la base de Datos y obtener información desde allí.

```
package com.iw2.core;

import com.google.gson.Gson;
import static com.iw2.core.Util.Sout;
import dao.libroDAO;
import java.io.BufferedOutputStream;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Date;
import java.util.List;
import java.util.StringTokenizer;
import model.libroDTO;
/**
  * @author CARLOMAGNO
 */
public class administradorSocket extends Thread {
  private final Socket conector;
  private String metodoPedido;
  private String httpPedido;
  private final String NEW_LINE = "\r\n";
  private StringBuffer sb;
  private final String APP_NAME = "api";


  private String URL;

  administradorSocket(Socket insocket) {
    conector = insocket;
  }

  public void procesarSocket() {
    this.start();
  }

  @Override // sobreescribimos el método run del Thread para procesar el socket

  public void run() { // proceso principal del server
    try {
      // capturamos el flujo de entrada
      // creamos un objeto para manipular el pedido. Necesitamos que tenga acceso al
      // socket
      // se lo inyectamos en el constructor

      InputStream flujoentrada = conector.getInputStream();
```

```
      BufferedReader buffer = new BufferedReader(new InputStreamReader(flujoentrada));
      // necesitamos un printStream para enviar archivos
      PrintStream ps = new PrintStream(new BufferedOutputStream(conector.getOutputStream
()));
      // procesamos El Stream y se lo pasamos como String a HttpRequest
      String linea = buffer.readLine();
      String header = linea;
      /** Si la cabecera es nula salimos !!! */
      if (header == null)
        return;

      StringTokenizer tokenizer = new StringTokenizer(header);
      metodoPedido = tokenizer.nextToken();
      httpPedido = tokenizer.nextToken();
      httpPedido = header + "\r\n";

      while (buffer.ready()) {
        /** Leemos todo el pedido HTTP hasta el final.... */

        httpPedido += buffer.readLine() + "\r\n";
      }
      System.out.println(httpPedido);
      /*
       * System.out.println("HTTP-METHOD: " + metodoPedido);
       * System.out.println(httpPedido);
       */

      HttpRequest req = new HttpRequest(httpPedido);
      HttpResponse resp = new HttpResponse(conector);

      // AHORA PARA VER SI TODO ESTA OK VAMOS A GENERAR UN ECHO AL WEB BROWSER
      // ANTES QUE NADA SE ENVIA UN ENCABEZADO DE ESTADO Y AUTORIZACION PARA CORS SINO
      // NO FUNCIONA
      // DESDE OTROS SITIOS EXTERNOS, POR EJEMPLO ALGO QUE HAGAMOS CON ANGULARJS Y
      // QUERRAMOS CONSUMIR
      // EXTERNAMENTE
      // capturamos los parámetros enviados

      String PaginaInicio;
      System.out.println("Accion: " + req.getAccion());

      if (req.getAccion() != null) {
        String hacer = req.getAccion();

        if (hacer.equalsIgnoreCase("Listar")) {

          libroDTO lib = new libroDTO();
          libroDAO ldao = new libroDAO();
          List<libroDTO> listadoLibros;
          listadoLibros = ldao.readAll();
          libroDTO libro = new libroDTO();
          Gson gson = new Gson();
          String listadoJSON = "[";

          for (int t = 0; t < listadoLibros.size(); t++) {
            libro = (libroDTO) listadoLibros.get(t);

            listadoJSON += gson.toJson(libro) + ",";
          }
          listadoJSON = listadoJSON.substring(0, listadoJSON.length() - 1);
```

```
          listadoJSON += "]";
          // resp.enviarRespuestaDatos(200, resp.getInitPage("Hola Mundo !!!"));
          Sout(gson.toJson((libroDTO) lib));



           PaginaInicio = resp.getInitPage(gson.toJson((libroDTO) lib));
          resp.imprimirSalida(resp.getHeader());
          resp.imprimirSalida(listadoJSON);
        } else { // no piden ninguna accion enviamos un archivo, por defecto es index.ht
ml
          if (req.getAccion().equals(" ")) // no pidieron nada enviamos pagina principal
          {
            PaginaInicio = resp.getHeader();
            PaginaInicio += resp.getInitPage("hola desde el servidor IW2");
            resp.imprimirSalida(PaginaInicio);
          } else
                  { // pidieron un archivo, lo enviamos
                        req.enviarArchivo("", ps);
          }
        }
      }
      resp.cerrar();
      conector.close();


    } catch (Exception ex) {
      System.out.println(ex.getMessage());
    }
  }
}
```

Interface **intCRUD** define los métodos Create, Read, RealAll, Update y Delete

```
package dao;

import java.util.List;
/**
 *
 * @author csimes
 */
public interface intCRUD<EntidadDTO> {
    public boolean create(EntidadDTO e);
    public boolean delete(Object clave);

    public boolean update(EntidadDTO e);

    public EntidadDTO read(Object clave);

    public List<EntidadDTO> readAll();

}
```

Clase **libroDAO.java**

// implementa la interfaz intCRUD y los métodos de acceso
// a la base de datos

```java
package dao;

import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import static java.util.Collections.list;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import model.libroDTO;
import util.Conexion;

/**
  * @author csimes
 */
public class libroDAO implements intCRUD<libroDTO> {
    public static final Conexion con = Conexion.crearConexion();
    public static final String SQL_INSERT = "insert into libro(libro_id,titulo,autor,pre
cio) values (?,?,?,?)";
    public static final String SQL_DELETE = "delete from libro where libro_id=?";
    public static final String SQL_UPDATE = "update libro set titulo=?,autor=?,precio=?
where libro_id=?";


    public static final String SQL_READ = "select * from libro where libro_id=?";
    public static final String SQL_READALL = "select * from libro";

    @Override


 public boolean create(libroDTO e) {
        try {
            int control = 0;
            PreparedStatement ps = con.getCnn().prepareCall(SQL_INSERT);
            ps.setInt(1, e.getLibro_id());
            ps.setString(2, e.getTitulo());
            ps.setString(3, e.getAutor());
            ps.setFloat(4, e.getPrecio());
            control = ps.executeUpdate();
            if (control > 0) {
                return true;
            }

        } catch (SQLException ex) {
            Logger.getLogger(libroDAO.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            con.cerrarConexion();
        }
        return false;
    }

    @Override
    public boolean delete(Object clave) {
        try {
            int control = 0;
            PreparedStatement ps = con.getCnn().prepareCall(SQL_DELETE);
            ps.setInt(1, (int) clave);
```

```
            control = ps.executeUpdate();
            if (control > 0) {
                return true;
            }


} catch (SQLException ex) {
            Logger.getLogger(libroDAO.class.getName()).log(Level.SEVERE, null, ex);


} finally {

            con.cerrarConexion();
        }
        return false;

    }

    @Override
    public boolean update(libroDTO e) {
        try {
            int control = 0;
            PreparedStatement ps = con.getCnn().prepareCall(SQL_UPDATE);

            ps.setString(1, e.getTitulo());
            ps.setString(2, e.getAutor());
             ps.setFloat(3, e.getPrecio());
            ps.setInt(4, e.getLibro_id());
            control = ps.executeUpdate();
            if (control > 0) {
                return true;
            }
        } catch (SQLException ex) {
            Logger.getLogger(libroDAO.class.getName()).log(Level.SEVERE, null, ex);
        } finally {

       con.cerrarConexion();
        }
        return false;
    }
    @Override
    public libroDTO read(Object clave) {

        libroDTO libro = null;
        try {

            ResultSet rs = null;
            PreparedStatement ps = con.getCnn().prepareCall(SQL_READ);
            ps.setInt(1, (int) clave);

            rs = ps.executeQuery();

            if (rs.next()) {
                libro = new libroDTO();
                libro.setLibro_id(rs.getInt("libro_id"));
                libro.setTitulo(rs.getString("titulo"));
                libro.setAutor(rs.getString("autor"));
                libro.setPrecio(rs.getFloat("precio"));
                return libro;
            }
```

```
        } catch (SQLException ex) {
            Logger.getLogger(libroDAO.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            con.cerrarConexion();


        }
        return libro;
    }

    @Override
    public List<libroDTO> readAll() {
        libroDTO libro;
        List<libroDTO> lista = new ArrayList<>();

        try {

            ResultSet rs = null;
            PreparedStatement ps = con.getCnn().prepareCall(SQL_READALL);

            rs = ps.executeQuery();

            while (rs.next()) {

                libro = new libroDTO();
                libro.setLibro_id(rs.getInt("libro_id"));
                libro.setTitulo(rs.getString("titulo"));
                libro.setAutor(rs.getString("autor"));
                libro.setPrecio(rs.getFloat("precio"));
                lista.add(libro);

            }
        } catch (SQLException ex) {
            Logger.getLogger(libroDAO.class.getName()).log(Level.SEVERE, null, ex);
        }


        finally {
            con.cerrarConexion();
        }
        return lista;
    }
}
```

Clase libroDTO.java

// POJO de la entidad libro para manipular la info de la misma

```
package model;
/**
  * @author csimes
 */
public class libroDTO {
    private int libro_id;
    private String titulo;
    private String autor;
    private float precio;
```

```
    public int getLibro_id() {
        return libro_id;
    }

    public void setLibro_id(int libro_id) {
        this.libro_id = libro_id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public float getPrecio() {
        return precio;
    }

    public void setPrecio(float precio) {
        this.precio = precio;
    }

    @Override
    public String toString() {
        return "libroDTO{" + "libro_id=" + libro_id + ", titulo=" + titulo + ", autor="
+ autor + ", precio=" + precio + '}';
    }
}
```

Clase conectarDB.java

// permite conectar con cualquier base de datos.
// cambiar los drivers según corresponda, en este caso conecta con MySql
// Si se usa una base de MySql como en este caso se debe incluir en el proyecto
// de netbeans el conector de MySql que permite realizar la conexión.

```
package util;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
```

```java
/**
  * @author csimes
 */


public class Conexion {
    public static Conexion instance; // objeto instancia de tipo conexion para aplicar S
ingleton
    private Connection cnn;

    private Conexion() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            try {
                cnn = DriverManager.getConnection("jdbc:mysql://localhost:3306/libreria"
, "root", "adminroot");
            } catch (SQLException ex) {
                Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE, null, ex);
            }
        } catch (ClassNotFoundException ex) {
            Logger.getLogger(Conexion.class.getName()).log(Level.SEVERE, null, ex);
        }
    }


    // ahora hacemos el Singleton
    public synchronized static Conexion crearConexion() {
        if (instance == null) {

            instance = new Conexion();
        }

        return instance;
    }

    public void cerrarConexion() {
        instance = null;
    }

    public Connection getCnn() {
        return cnn;
    }
}
```

Como explicamos al comienzo, se ha implementado en este servidor, una función de envío de archivos.

Esta funcionalidad permite que por ejemplo, nuestro servidor pueda además de la respuesta generada por código, envíe recursos preexistentes, por ejemplo páginas web más complejas, imágenes, archivos de excel, pdf, etc

Para probar esta funcionalidad, debemos colocar en la raíz de nuestro proyecto algún archivo que queramos solicitar vía web y que debe ser de los tipos soportados, por ejemplo un archivo html, o pdf

Para solicitar el archivo, una vez que el server está funcionando, en la barra del navegador colocamos:
http://localhost:8866/nombrearchivo.ext

Por ejemplo: **http://localhost:8866/index.html**
Ahora empleando todo lo aprendido y basándonos en este server vamos a implementar algunas nuevas funcionalidades.

Segunda Actividad Integradora (fecha de entrega 15-06-2021):

Tomar como base la primera Actividad Integradora, que usaba localStorage.
En este caso usaremos nuestro **Server IW2** al que le agregaremos las funciones necesarias para guardar en una base de datos **MySql** la información que procesa nuestra App y que antes se registraba en localStorage.

La implementación de las funcionalidades solicitadas, debe hacerse en una única actividad que será la **Segunda Actividad Integradora**.

Esta será la segunda nota y servirá para promediarla con la primera calificación y promocionar la parte práctica de la asignatura.
Esta segunda actividad, es necesaria para regularizar la asignatura y como mínimo debe implementar las 3 primeras consignas, para considerarse aprobada para regularizar.
Para promocionar la materia, el proyecto deberá implementar todas las consignas.
La fecha límite de entrega de esta segunda actividad será:**15/06/2021**

Consigna Nº1:

Cree una base de datos en **MySql** e incluya una tabla para guardar la información de la entidad (productos, alumnos, clientes, etc) que usó en su primera actividad integradora.
Implemente en el servidor una función que permita listar dicha entidad.
Para ello usaremos un pedido **GET /ListarEntidad**
Envíe el listado como un archivo JSON y recíbalo en una página usando **AngularJS** como en el ejemplo desarrollado en este tutorial

Consigna Nº2:

Agregue una funcionalidad para buscar un elemento en particular, para ello usaremos un mandato **GET/BuscarEntidad/?Clave=Valor**

Consigna Nº3:

Agregue una funcionalidad para agregar un elemento de la entidad, para ello usaremos un mandato
GET/AgregarEntidad/?atrib1=valor1&atrib2&valor2 .....

Consigna Nº4:

Agregue una funcionalidad para borrar un elemento de la entidad, para ello usaremos un mandato **GET/BorrarEntidad/?clave=valor**

Consigna Nº5:

Implemente las funcionalidades de inserción, actualización y borrado anteriores pero ahora usando el mandato **POST.**
Tips: Cuando se realiza una solicitud Http POST, en la cual enviamos datos como JSON, el formato de la solicitud es:

POST / HTTP/1.1
Content-Type: application/json ← **Tipo de contenido recibido**
User-Agent: PostmanRuntime/7.24.1
Accept: */*
Cache-Control: no-cache
Postman-Token: b4049348-e767-4706-9b3f-36773adb1bca
Host: localhost:8866
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

Content-Length: 318 ← **Largo del Contenido**
← **Línea en Blanco**

```
{
   "id_producto": "123",
   "descripcion": "Tornillo",
   "precio": "25",
   "cantidad": "350"
}
```

← **Información enviada**

A diferencia de una solicitud GET, la solicitud POST coloca **después** del atributo **Content-Length: xxx**, una línea en blanco y luego, el o los parámetros enviados con sus valores.
Si hubiésemos realizado una solicitud POST pero usando un formato estándar en vez de usar JSON (aunque se recomienda usar JSON ya que es un estándar para intercambio de datos), el pedido nos quedaría:

POST / HTTP/1.1
Content-Type: **application/x-www-form-urlencoded** **Datos provienen de un formulario**
User-Agent: PostmanRuntime/7.24.1
Accept: */*
Cache-Control: no-cache
Postman-Token: b4049348-e767-4706-9b3f-36773adb1bca
Host: localhost:8866
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 50 ← **Largo del contenido**
← **Línea en blanco**

id=138&descripcion=arandela&precio=15&cantidad=170

Parámetros

Conociendo estos detalles sobre POST, ahora nos será posible modificar la clase **HttpRequest** del servidor para que detecte que tipo de solicitud está recibiendo

(GET o POST) y obtenga los parámetros correspondientes dependiendo del tipo de solicitud y de la forma (Content-Type) en que los parámetros se reciban.
El uso de la solicitud POST, se prefiere por sobre GET, en casi todas las circunstancias, entre otras cosas, porque las solicitudes POST envían los datos encriptados, a diferencia de GET.
Por cuestiones de seguridad, salvo para las acciones de **listar y buscar,** es altamente recomendable usar solicitudes POST para borrar, actualizar e insertar información en nuestro servidor.