Desarrollo web entorno servidor

Hito individual

Sergio Felipe García

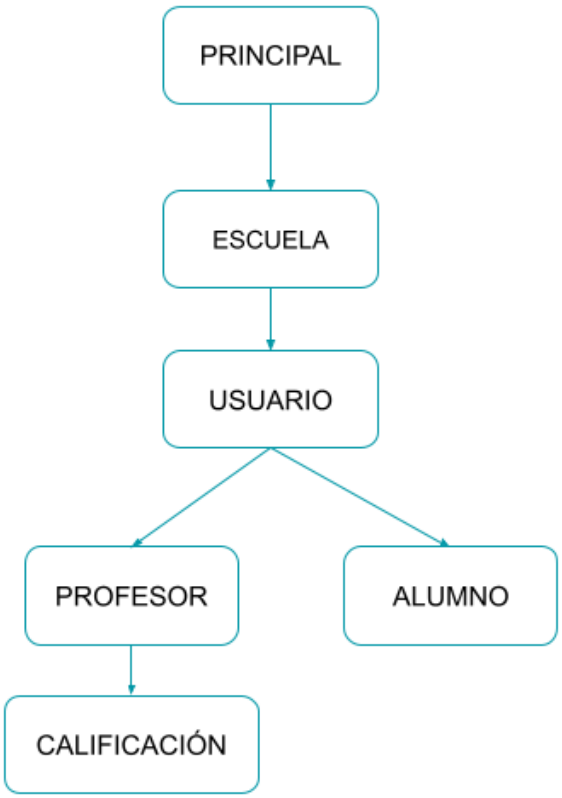
Proyecto Consola

Introducción

La idea es crear un proyecto con similares características y funcionalidades que correrá en 2 entornos diferentes (Consola windows y Navegador web) de esta forma podremos observar las diferentes tecnologías implementadas. Por ejemplo una aplicación de consola no necesita un servidor Tomcat como el utilizado en el siguiente apartado con tecnología de páginas JSP que más adelante explicaré.

Análisis del modelo de clases

Jerarquía de Clases y dependencias



Principal crea instancias de la clase Escuela.

- **Agrega** Profesores y Alumnos a la Escuela
- **Obtiene** un usuario de la escuela según su nombre
- **Solicita** introducir (nombre del alumno y del profesor)
- **Comprueba** si existe un usuario o el profesor y que tipo de usuario es.
- **Proporciona** información de una asignatura y nota
- **Permite** calificar a una lumno por el profesor / añadecalificacionesç

Escuela TreeMap para almacenar usuarios

- **Clave = Login → Valor = Objeto usuario**
- **Método para agregar nuevo profesor** creando instancias de la clase profesor y lo almacena en el Treemap usuario.(login, password,especialidad)
- **Método para agregar nuevo Alumno** instando la clase alumno y lo almacena (login, password)
- **Lista los usuarios** getUsuarios()

Usuario Clase base de la que heredan Profesor y Alumno Se almacenan en el treemap de la Escuela.

- Métodos
  - Obtener nombre de usuario / contraseña. **Login** = Nombre de usuario
  - Establece el nombre del usuario / contraseña

Profesor y Alumno heredan de la clase base Usuario, lo que significa que comparten métodos y propiedades definidos en la clase Usuario.

- Métodos
  - Obtener la lista calificaciones de un alumno
  - Añade las calificaciones de un alumno
  - Obtiene la especialidad del profesor
  - Establece la especialidad del profesor

Calificación utiliza objetos de la clase Profesor.

- Métodos
  - Obtener el profesor que calificó la asignatura
  - Obtener y asignar El nombre de la asignatura
  - Asignar la notas

Tipos de estructuras de datos especiales

TreeMap (en la clase Escuela):

En un mapa TreeMap sus elementos, se ordenan según el tipo de dato de la CLAVE. Por ejemplo si es entero se ordena de mayor a menor. En es caso se utiliza para almacenar

- ***Usuarios***(CLAVE)
- Objetos de tipo ***Profesor y Alumno*** como (Valores) .

Nombre Usuario (Login)	Objeto (Usuarios)	
KEY	Objeto (Profesor)	0Objeto (Alumno)
Key "Profesor1"	Value Profesor1 Object	
Key "Profesor2"	Value Profesor2 Object	
Key "Alumno1"		Value Alumno1 Object

```
// TreeMap llamado 'usuarios' para almacenar usuarios,
// donde la clave es el 'login' (nombre de usuario) y el valor es un objeto 'Usuario'.
this.usuarios = new TreeMap<String, Usuario>();
```

```
// Agregar profesores y alumnos a la escuela.
e.nuevoProfesor("Pepe", "1234", "Matemáticas, física");
e.nuevoProfesor("Juan", "1234", "Informática");
e.nuevoAlumno("Alicia", "2345");
e.nuevoAlumno("Miguel", "2345");
```

ArrayList (en la clase Alumno):

La idea es poder almacenar múltiples calificaciones para cada estudiante. ArrayList tiene diferencias con los arrays normales. Para diferenciarlos vamos a enumerar las principales:

- **ArrayList**:
    - Es una estructura de datos que *permite un tamaño dinámico*. Puedes agregar o eliminar elementos de un ArrayList, y automáticamente se ajustará su tamaño para los nuevos elementos.
    - Es más fácil de usar cuando no conoces cuántos elementos necesitarás almacenar, y no tienes que preocuparte por el tamaño.
    - Sin embargo, puede ser menos eficiente en términos de memoria aunque esta diferencia de memoria puede ser insignificante.
- **Array (Estático)**:
    - Un array estático tiene un *tamaño fijo y se define en el momento de su creación*. No puedes cambiar su tamaño. Si necesitas más espacio, debes crear un nuevo array y copiar los elementos del anterior.
    - Puede ser más incómodo si no sabes cuantos elementos vas a necesitar almacenar, como por ejemplo en este caso las notas.
    - Si hablamos de memoria, es más eficiente si sabes antes el tamaño de los elementos que vas a almacenar.

```
public class Alumno extends Usuario {
    private ArrayList<Calificacion> calificaciones;
```

```
calificaciones = new ArrayList<Calificacion>();
```

ArrayList llamado calificaciones ➜ *ArrayList<TipoDeDato> nombreDelArrayList*
Calificaciones (ArrayList)

- Calificación 1 - Profesor: "Profesor1" - Asignatura: "Matemáticas" - Nota: 90	- Calificacion 2 - Profesor: "Profesor2" - Asignatura: "Historia" - Nota: 85

Conceptos de POO aplicados

- Herencia:
    - En las clases Profesor y Alumno, que heredan de la clase base Usuario. También la clase Calificación, que depende de la clase Profesor.
    - Profesor y Alumno heredan propiedades y métodos de la clase Usuario, que evita la duplicación de código. La clase Calificación depende del Profesor para representar quién calificó una asignatura.
- Abstracción:
    - Aplica en las clases Usuario, Profesor, Alumno y Calificación, donde hay conceptos abstractos como usuarios, profesores, alumnos y calificaciones.
    - Sirve para representar objetos del mundo real en el programa
- Encapsulamiento:
    - El encapsulamiento está en las clases (Usuario, Profesor, Alumno, Calificación). Los atributos son privados. *private String login;* y los métodos públicos para leer o modificar los atributos. **public void** setLogin(String login)
    - Así se protegen los atributos y se controla el acceso a ellos. 4
- Polimorfismo:
    - En la clase Alumno, usa el método calificar que acepta un objeto de tipo Profesor como argumento.

```
public void calificar(Profesor p, String asignatura, int nota) {
this.calificaciones.add(new Calificacion(p, asignatura, nota))
```

    - Así el método actuará dependiendo del tipo de objeto que se le pasa como argumento.

Tecnologías utilizadas.

Eclipse:

Utilizaré Eclipse es un entorno de desarrollo (IDE) es utilizado para el desarrollo de software en varios lenguajes, incluido Java que es el lenguaje seleccionado para realizar esta práctica. Eclipse es compatible con Windows.

Eclipse proporciona herramientas de depuración que facilitan la identificación y corrección de errores en el código, aún así en algunos casos he introducido algunas sentencias que me permiten seguir el flujo del código a través de la consola que proporciona Eclipse.

Java

Como lenguaje de programación Java, que es una plataforma de desarrollo de aplicaciones diseñada para aplicaciones independientes y de escritorio.

Las aplicaciones Java SE pueden ejecutarse en diferentes sistemas operativos sin necesidad de modificaciones, en caso de usar la máquina virtual Java (JVM).

Además Java incluye una amplia biblioteca estándar que proporciona funciones y clases para tareas comunes, como manipulación de archivos, redes, E/S, gráficos y mucho más.

Proyecto Web

Tecnologías utilizadas.

Con el fin de cerrar este apartado y que se vean en mi documento las diferencias entre una aplicación que corre sobre la consola y otra que corre sobre un navegador Web voy a introducir aquí las tecnologías que he añadido para poder ejecutar mi proyecto web.

TOMCAT 10

Tomcat 10 es un servidor web, que está adecuado para el desarrollo y la implementación de aplicaciones **web Java**. Es decir, se utiliza para alojar aplicaciones web basadas en Java. Tomcat 10 es compatible con Java EE 9 a diferencia de la aplicación explicada anteriormente de consola que utiliza Java SE que es la edición estándar.

Tomcat 10 soporta Servlets y páginas web JSP.

Java Servlet

Básicamente, es una clase Java que se utiliza para extender la funcionalidad de un servidor web y responder a solicitudes HTTP. Los servlets son parte de la tecnología Java EE (Enterprise Edition) y **se ejecutan en el servidor web** en este caso en Apache Tomcat.

Los servlets se utilizan para manejar solicitudes entrantes desde los clientes web, como navegadores. Pueden procesar datos enviados por el cliente y generar respuestas dinámicas, que pueden ser páginas **HTML**, XML, JSON u otros tipos de contenido. **En la práctica realizada** podremos ver cómo se generan contenidos dinámicamente para mostrar las notas de los alumnos en diferentes apartados de la aplicación.

Otro aspecto importante que se ha manejado en esta práctica con los servlets ha sido, **mantener el estado de la sesión del usuario**.

Las páginas JSP (JavaServer Pages)

Son una tecnología utilizada en el desarrollo web que permite la creación de contenido web dinámico **mediante la incorporación de código Java directamente en páginas HTML** . Son un estándar de la plataforma Java EE (Enterprise Edition)..

Las páginas JSP permiten a los desarrolladores insertar fragmentos de código Java directamente en las páginas web. Este código Java se delimita generalmente con etiquetas `<% ... %>`.

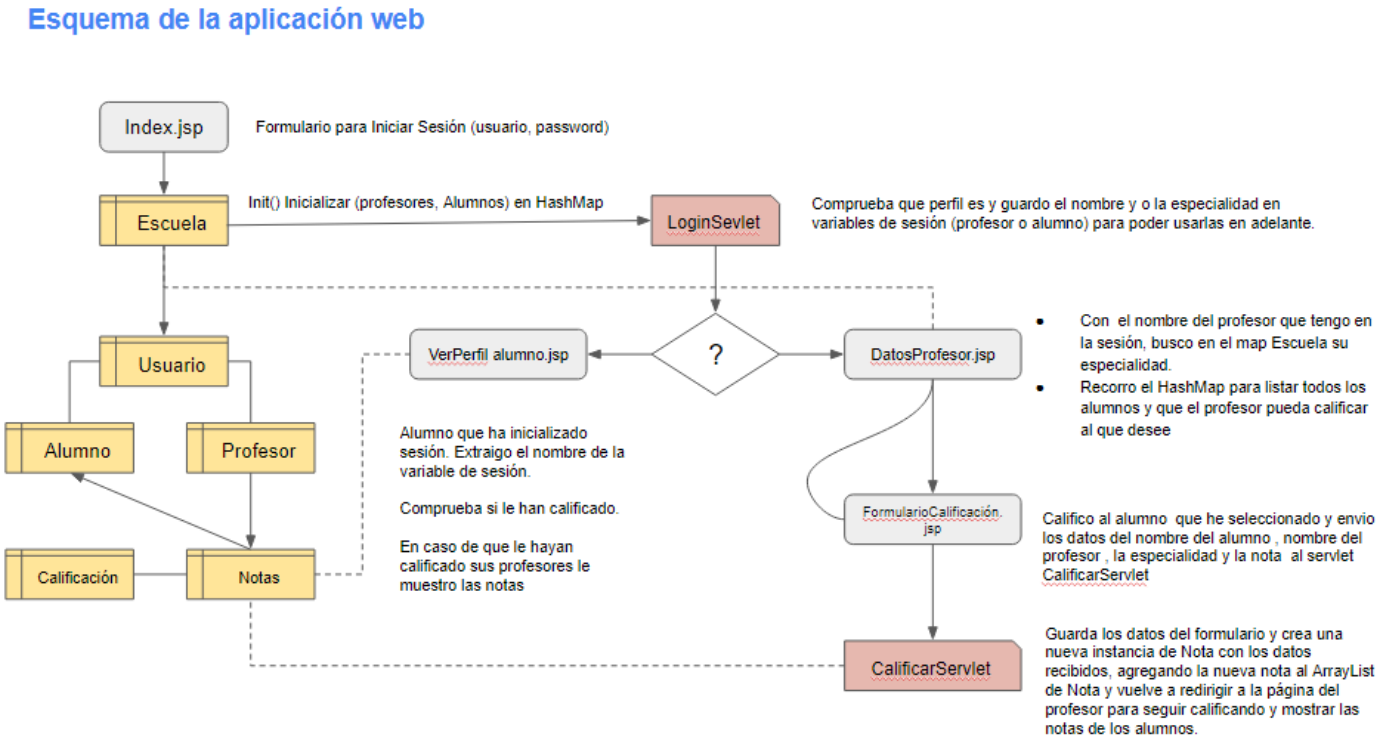
Las páginas JSP permiten usar el patrón de diseño Modelo-Vista-Controlador (MVC) para separar la lógica de negocio de la presentación.

Para ejecutar páginas JSP, se utilizan servidores como Apache Tomcat, que procesan las páginas JSP, ejecutan el código Java incrustado y generan páginas web dinámicas en tiempo de ejecución.

Ejecución del proyecto.

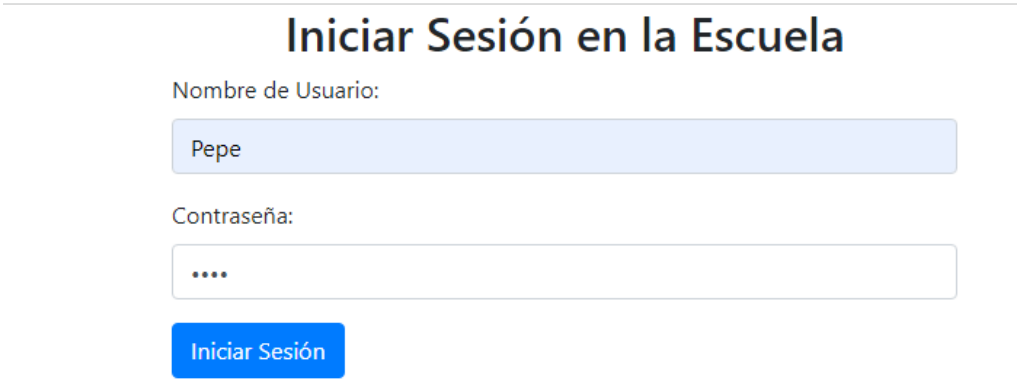
Aunque más adelante mostraré pantallazos, para explicar el funcionamiento de la aplicación Web que he desarrollado, adjunto este esquema que me ha servido para idear los pasos que tenía que dar antes de desarrollar el proyecto con la esperanza de que ayude a una mejor comprensión.

Esquema básico



Login en la Aplicación (index.jsp)

Para poder operar con la aplicación he desarrollado un formulario que solicita el nombre del usuario y la contraseña. (Vista).



Cuando el usuario introduce los datos y hace clic en iniciar sesión, procedo a controlar el acceso.

LoginServlet (Controlador)

Este servlet contiene el método especial init(). Lo utilizo para tareas de inicialización. Este método se llama automáticamente una sola vez durante la inicialización del servlet y se utiliza para realizar configuraciones y preparativos.

```java
package servlets;

import jakarta.servlet.ServletException;

/** * Servlet implementation class LoginServlet */

public class LoginServlet extends HttpServlet {
    private Escuela escuela;

    public void init() throws ServletException { // INIT OJO - se llama una sola vez cuando el servlet se crea por primera vez o se inicializa.
        escuela = new Escuela(); // Inicializa el objeto Escuela

        getServletContext().setAttribute("miEscuela", escuela); // Almacena la instancia de Escuela en el contexto de la aplicación

        // Así, puedo  acceder a esta instancia de Escuela desde cualquier servlet o página JSP
        // en tu aplicación utilizando getServletContext().getAttribute("miEscuela") Por si acaso.

    }

    // Proceso los datos del formulario de Login
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
            String login = request.getParameter("login");
            String password = request.getParameter("password");


        // Utiliza el objeto Escuela para verificar las credenciales
        HashMap<String, Usuario> usuarios = escuela.getUsuarios();
        Usuario usuario = usuarios.get(login);

        if (usuario != null && usuario.getPassword().equals(password)) {
            // Verifica si el usuario es un profesor
            if (usuario instanceof Profesor) {

            // Guarda el usuario en la sesión
                HttpSession session = request.getSession();
                session.setAttribute("usuario", usuario);
             // Obtiene la especialidad del profesor
                String especialidadProfesor = ((Profesor) usuario).getEspecialidad();
                session.setAttribute("especialidad", especialidadProfesor);

```

En este caso cargamos los *Alumnos y Profesores* que van a poder acceder a la aplicación. Me ayuda la clase Escuela a crearlos y a introducirlos en un Hasmap que está definido en la clase Usuarios.

```java
package servlets;

import java.util.HashMap;

public class Escuela {
    private HashMap<String, Usuario> usuarios;

    public Escuela() {
        super();
        this.usuarios = new HashMap<String, Usuario>();
        // Agrega aquí la lógica para crear profesores y alumnos predeterminados
        nuevoProfesor("Pepe", "1234", "Matemáticas, física");
        nuevoProfesor("Juan", "1234", "Informática");
        nuevoAlumno("Alicia", "2345");
        nuevoAlumno("Miguel", "2345");
    }

    // El método getUsuarios() es un método de acceso en la clase Escuela
    // que se utiliza para obtener el HashMap de usuarios. Su función es
    // proporcionar acceso a la colección de usuarios almacenada en la instancia de la clase Escuela.

    public HashMap<String, Usuario> getUsuarios() {
        return usuarios;
    }

    public void nuevoProfesor(String login, String password, String especialidad) {
        Profesor profesor = new Profesor(login, password, especialidad);
        usuarios.put(login, profesor);
    }

    public void nuevoAlumno(String login, String password) {
        Alumno alumno = new Alumno(login, password);
        usuarios.put(login, alumno);
    }
}
```

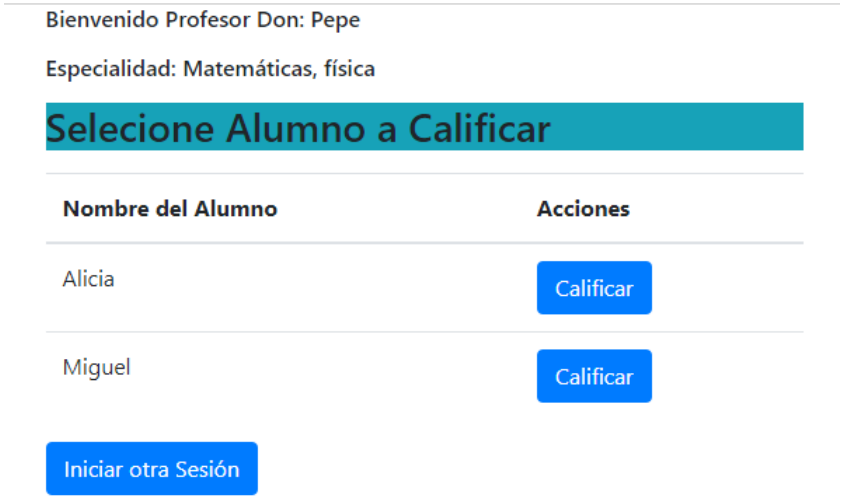
Aquí muestro el código de la clase Usuario, la cual maneja los datos de (*login*) que es el nombre del usuario que inicia sesión y su contraseña (*password*) que se inicializa en la Clase Escuela.

```java
package servlets;

public class Usuario {
    private String login;
    private String password;

    public Usuario(String login, String password) {
        super();
        this.login = login;
        this.password = password;
    }

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Perfil del Profesor (DatosProfesor.jsp) -Vista

En *LoginServlet* (controlador) comprobamos si el usuario que ha iniciado sesión es una instancia de *profesor* o de *alumno*. Si es un profesor mostramos su perfil con su nombre y sus especialidades. Estos datos están establecidos en la sesión para poder ser recuperados.

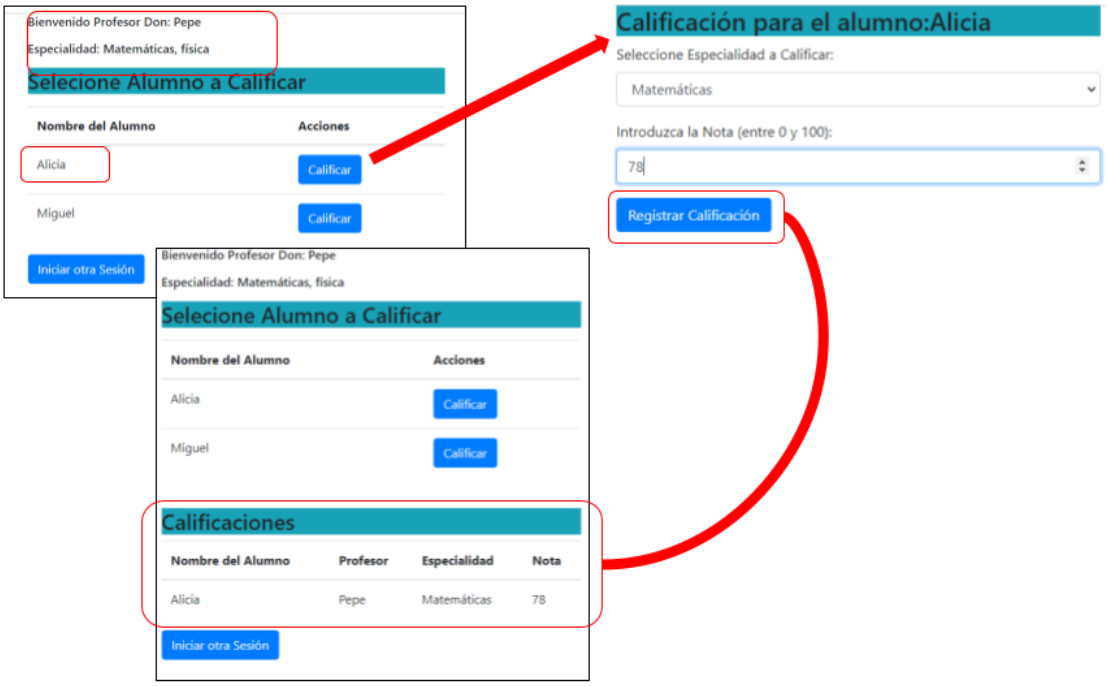
Además, mostramos una tabla con los Alumnos de la escuela para que pueda calificarlos. El botón calificar redirige a la página *FormularioCalificacion.jsp* con el nombre del alumno al que desea calificar el profesor.



FormularioCalificación.jsp

Básicamente este formulario muestra el nombre del alumno que obtenemos en el paso anterior cuando hacemos clic en el botón calificar. Incluyo este valor como un parámetro oculto del formulario:

```
<input type="hidden" name="alumno" value="${param.alumno}">
```



Además, en concreto el profesor "*Pepe*", se había inicializado en la clase escuela con 2 Especialidades o asignaturas, por lo que para poder mostrarlo adecuadamente en el formulario y que el profesor seleccione la que más le convenga he tenido que tratar dicho String.

```
<%
String especialidadProfesor = (String) session.getAttribute("especialidad");
// La especialidad la habiamos establecido en logingServlet para la sesión
        if (especialidadProfesor != null) {
                // Dividir el string en un array de especialidades
                String[] especialidadesArray = especialidadProfesor.split(",");
%>
```

1. Primero metiendo cada especialidad en un array

2. y luego recorriendo dicho array para mostrar tantas opciones como especialidades tiene el profesor.

```
<select name="especialidad" class="form-control">
            <% for (String especialidad : especialidadesArray) { %>
                        <option value="<%= especialidad.trim() %>"><%= especialidad.trim() %></option>
            <% } %>
</select>
```

El profesor podrá registrar tantas calificaciones como desee para las especialidades que él tiene.

CalificarServlet

El servlet recoge los datos introducidos en el formulario anterior.

```
package servlets;


import jakarta.servlet.ServletException;

@WebServlet("/CalificarServlet2")
public class CalificarServlet extends HttpServlet {
    private ArrayList<Nota> notas = new ArrayList<>(); // ArrayList para almacenar las notas

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Obtener los datos del formulario
        String nombreAlumno = request.getParameter("alumno");
        String nombreProfesor = request.getParameter("profesor");
        String especialidad = request.getParameter("especialidad");
        int nota = Integer.parseInt(request.getParameter("nota"));

        System.out.println("obtengo los datos del formulario");

        // Crear una nueva instancia de Nota con los datos recibidos
        Nota nuevaNota = new Nota(nombreAlumno, nombreProfesor, especialidad, nota);

        // Agregar la nueva nota al ArrayList
        notas.add(nuevaNota);

     // Mensaje para ver el resultado de los datos que estamos gurdando
        System.out.println("Nota guardada: Alumno: " + nombreAlumno + ", Profesor: " + nombreProfesor + ", Especialidad: " + especialidad + ", Nota: " + nota);

        // Guardamos el array en la sesion por que lo vamos a utlizar en mas sitios

        HttpSession session = request.getSession();
        session.setAttribute("notas", notas);

        // Redirigir a la página donde deseas mostrar las notas
        response.sendRedirect("DatosProfesor.jsp");
    }
}
```

El método `*doPost*()` me permite procesar solicitudes HTTP

- `*request*`: Es un objeto de la clase `HttpServletRequest` maneja solicitud entrante, como parámetros o datos del formulario.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Obtener los datos del formulario
String nombreAlumno = request.getParameter("alumno");
String nombreProfesor = request.getParameter("profesor");
String especialidad = request.getParameter("especialidad");
int nota = Integer.parseInt(request.getParameter("nota"));
```

Un problema que he encontrado a la hora de mostrar las calificaciones de los alumnos ha sido, que no se ocurría ninguna forma para que los datos de las notas persistiesen para ser mostrados en las distintas secciones. Al no haber estudiado el manejo de fichero JSON o de Texto o BBDD con Java, he optado por guardarlos de forma ordenada en un Array Notas.

```
// Crear una nueva instancia de Nota con los datos recibidos
Nota nuevaNota = new Nota(nombreAlumno, nombreProfesor, especialidad, nota);
```

Para esto he creado una nueva clase llamada Nota.java la cual mantengo durante toda la sesión.

```
1  package servlets;
2  public class Nota {
3      private String nombreAlumno;
4      private String nombreProfesor;
5      private String especialidad;
6      private int nota;
7
8      public Nota(String nombreAlumno, String nombreProfesor, String especialidad, int nota) {
9          this.nombreAlumno = nombreAlumno;
10         this.nombreProfesor = nombreProfesor;
11         this.especialidad = especialidad;
12         this.nota = nota;
13     }
14
15     // Agrega los getters y setters según sea necesario
16
17     public String getNombreAlumno() {
18         return nombreAlumno;
19     }
20
21     public String getNombreProfesor() {
22         return nombreProfesor;
23     }
24
25     public String getEspecialidad() {
26         return especialidad;
27     }
28
29     public int getNota() {
30         return nota;
31     }
32 }
```

De esta forma puedo mostrar al profesor las calificaciones que ha puesto a sus alumnos. A continuación muestro una imagen con el resultado.

Bienvenido Profesor Don: Pepe

Especialidad: Matemáticas, física

Selecione Alumno a Calificar

Nombre del Alumno	Acciones
Alicia	Calificar
Miguel	Calificar

Calificaciones

Nombre del Alumno	Profesor	Especialidad	Nota
Alicia	Pepe	Matemáticas	78

Iniciar otra Sesión

Perfil del alumno (VerperfilAlumno.jsp)

En esta página se muestra el perfil del alumno cuando hemos iniciado sesión como un alumno, para poder enviar los datos del nombre del alumno la asignatura y la calificación utilice un ArrayList para guardar las notas del alumno calificado.

```
ArrayList<Nota> notas = (ArrayList<Nota>) session.getAttribute("notas");
```

Luego utilice un condicional para que cuando tuviera la nota me mostrase la tabla con los datos del alumno calificado pero me aparecia un problema el cual era que me salia las notas de todos los alumnos, para corregir el error utilice otro condicional para mostrar las notas del usuario que está conectado en la sesión.

```
<!-- Los profesores guardan las notas en un array y ahora verificar si hay notas en el array -->
<%
    ArrayList<Nota> notas = (ArrayList<Nota>) session.getAttribute("notas");
    if (notas != null && !notas.isEmpty()) {
%>
    <!-- Si no esta vacio quiero mostrar las notas aquí -->

    <br>
    <h3 class="bg-info" align center>Calificaciones</h3>

    <table class="table" style="max-width: 600px;">
        <thead>
            <tr>
                <th>Nombre del Alumno</th>
                <th>Asignatura</th>
                <th>Nota</th>
            </tr>
        </thead>
        <tbody>
            <%
            // Pero no quiero mostrar todas solo quiero mostrar las notas del usuario que esta conectado en la sesión
            // la clase usuario tiene varios atributos solo necesito el login que es donde almacena el nombre

            Usuario usuarioSesion = (Usuario) session.getAttribute("usuario"); // solo quiero
            String nombreAlumnoSesion = usuarioSesion.getLogin();
            for (Nota nota : notas) {
                if (nota.getNombreAlumno().equals(nombreAlumnoSesion)) { // Compara el alumno de las notas con el nombre del alumno en sesión
            %>
                <tr>
                    <td><%= nota.getNombreAlumno() %></td>
                    <td><%= nota.getEspecialidad() %></td>
                    <td><%= nota.getNota() %></td>
                </tr>
            <%
                }
            }
            %>
        </tbody>
    </table>
<%
    }
%>
```

De esta forma puedo mostrar al alumno sus calificaciones.A continuación muestro una imagen con el resultado.

Datos del Alumno

Nombre: Alicia

Calificaciones

Nombre del Alumno	Asignatura	Nota
Alicia	Matemáticas	78
Alicia	Matemáticas	78

Cerrar Sesión