

# Arquitectura y Diseño de Sistemas Web y C/S



## **Práctica Evaluación Final**

Alejandro Rodríguez Blanco 03206933C

David Bachiller Vela 03221211S

Víctor Sanavia Valdeolivas 03202543T

D. Roberto Barchino Plata

## Contenido

Introducción .....	3
Presentación de la prueba práctica .....	3
Objetivos de la práctica. ....	3
Requisitos y herramientas/tecnologías utilizadas.....	4
Análisis del Problema .....	4
Descripción del problema a abordar. ....	4
Detalles sobre la aplicación web de gestión de cine.....	5
Esquema de información y modelo E/R propuesto.....	5
Implementación .....	8
Uso de HTML5, CSS3 y JavaScript (jQuery). ....	9
Programación en JAVA con el patrón MVC.....	32
Autenticación, perfiles de usuario y sesiones. ....	71
Almacenamiento de datos en Apache Derby. ....	83
Manual de Usuario .....	92
Instrucciones para la instalación y ejecución. ....	92
Guía detallada de las funcionalidades.....	92
Gestión/Administración del Cine.....	92
Gestión de Películas. ....	93
Gestión de Salas. ....	95
Gestión de Entradas. ....	96
Gestión de Reservas. ....	97
Gestión de Informes.....	98
Funcionalidades para los Clientes. ....	98
Registro.....	98
Reserva de entradas.....	99
Comentarios. ....	104
Conclusiones .....	106
Referencias .....	106

## Introducción

El proyecto aborda la creación de una aplicación Web cliente/servidor, siguiendo el patrón de diseño Modelo Vista Controlador (MVC), con el propósito de gestionar de manera eficiente y una aplicación de cines.

La aplicación se compone de dos partes funcionales esenciales. En primer lugar, se aborda la administración y gestión interna del cine, que permite al administrador de la plataforma realizar acciones sobre las entidades de nuestra web. En segundo lugar, se presenta la parte orientada a los clientes, ofreciendo acceso a la web de la compañía con información detallada sobre películas y permitiendo a los usuarios registrados realizar funciones como la reserva de entradas y la adición de comentarios.

El esquema de información se modela mediante un enfoque de Entidad/Relación (E/R), detallando la estructura de datos que la aplicación gestionará. Además, se emplearán tecnologías clave como HTML5, CSS3, JavaScript (jQuery), y la implementación del backend se llevará a cabo en JAVA, aplicando el patrón MVC para garantizar una arquitectura organizada y modular.

A lo largo de este documento, se explorarán los detalles de la implementación, desde el diseño de la base de datos en Apache Derby hasta la programación de las funcionalidades requeridas, incluyendo aspectos críticos como la autenticación, perfiles de usuario, y sesiones. Se proporcionará un manual detallado para guiar a los usuarios en la instalación y uso de la aplicación, así como una defensa de la práctica durante la evaluación final.

## Presentación de la prueba práctica.

### Objetivos de la práctica.

Como objetivos de la práctica, tendríamos lo siguiente:

- **Usabilidad y experiencia de usuario:** Nos tenemos que asegurar de que la interfaz de usuario es intuitiva y sencilla de usar tanto para los roles de usuario como para los de administrador, además de proporcionar una navegación fluida entre cada una de ellas
- **Seguridad:** Implementar medidas de seguridad adecuadas con el fin de proteger la información del usuario y asegurarnos de la integridad de todos los datos presentes en nuestra base de datos. También es preciso proteger nuestro sistema frente a inyecciones SQL
- **Funcionalidades del usuario:** Confirmar que todos los usuarios presentes en el sistema puedan visualizar el contenido de las películas, sus detalles, comprar las entradas, gestionar sus reservas y realizar el pago correspondiente con una tarjeta de crédito
- **Gestión de películas:** Comprobar que es posible la modificación, creación, eliminación y consulta de todas las películas desde la interfaz del administrador
- **Gestión de reservas:** Permitir la visualización y modificación de reservas en la interfaz de administrador

- **Gestión de informes:** Tenemos que ser capaces de leer los datos y/o estadísticas que se generan en una película y hacer que estos informes se entiendan bien para mejorar la toma de decisiones
- **Gestión de usuarios:** Asegurarnos de que el administrador tenga acceso a toda la funcionalidad del sistema e implementar el registro de usuarios y su correcto inicio de sesión en la aplicación
- **Optimización del rendimiento:** Desarrollar códigos eficientes con el fin de que los tiempos de carga en nuestra aplicación sea el menor posible.
- **Documentación:** Proporcionar una correcta descripción de la estructura de nuestra aplicación web para futuras mejoras de la misma
- **Uso del patrón MVC (Modelo-Vista-Controlador):** Coordinar todos los elementos para una correcta optimización de la aplicación

### Requisitos y herramientas/tecnologías utilizadas.

En cuanto a los requisitos, tenemos los siguientes requisitos funcionales y no funcionales:

- **Requisitos funcionales**
  - Registro de usuarios
  - Gestión de películas
  - Búsqueda y visualización de detalles de películas
  - Reserva de entradas para las distintas películas
  - Generación de informes y estadísticas
  - Gestión de salas
  - Autenticación de usuarios
- **Requisitos no funcionales**
  - Proteger la aplicación frente a ataques
  - Interfaz intuitiva y fácil de usar
  - Documentación completa y clara

Las herramientas o tecnologías usadas son las siguientes:

- **Java:** Con este lenguaje de programación desarrollaremos el código para los servlets y los JSP
- **HTML:** Usaremos este lenguaje para programar el cuerpo de las páginas web en los JSP
- **CSS:** Con el objetivo de dar un buen diseño a todas nuestras páginas y que sean más claras y animadas
- **JavaScript:** Se usará en especial para realizar el chequeo de todos los formularios de las páginas JSP

## Análisis del Problema

### Descripción del problema a abordar.

Los distintos problemas que abordaremos en las distintas áreas serán los siguientes:

- **Gestión de películas**

- Los administradores podrán añadir películas especificando todos los campos necesarios para ello
  - Posibilidad de consultar, modificar y eliminar películas o su información
- **Gestión de salas**
  - Asignaciones de salas a películas y configurar sus horarios
- **Reservas de entradas**
  - Los usuarios podrán elegir la película que deseen y reservar una entrada para la película o películas deseadas
  - Los administradores podrán gestionar todas esas reservas
- **Informes**
  - **Filtrar las** películas por género y salas
  - Hacer que la visualización de resultados sea clara e intuitiva
- **Gestión de usuarios**
  - Distinguir entre usuario y administrador
  - Implementar contraseñas para cada usuario con el fin de proporcionar seguridad
- **Seguridad**
  - Proteger la base de datos
  - Almacenamiento seguro de contraseñas

#### Detalles sobre la aplicación web de gestión de cine.

La aplicación de cines tendrá las siguientes características:

- Autenticación y validación de sesiones
- Gestión de Películas (borrar, crear, modificar y consultar)
- Gestión de Salas (borrar, crear, modificar y consultar)
- Gestión de Entradas (borrar, crear, modificar y consultar)
- Gestión de Reservas (consultar)
- Gestión de informes (listado de películas filtrando por género y salas)

#### Esquema de información y modelo E/R propuesto.

Para poder elaborar el modelo entidad/relación que haga referencia a la manera que se relacionan los diferentes objetos y personas de nuestra aplicación, hemos empleado la herramienta Pgmodeler, esta se puede encontrar en la siguiente [web](#).

Al analizar detenidamente el texto y haciendo uso de lo expuesto en el enunciado, hemos encontrado las siguientes entidades (con sus respectivos atributos):

**Película:** entidad que representa la información fundamental de una película proyectada en el cine.

- Atributos:
  - nombrePelícula (Clave Primaria)
  - sinopsis
  - paginaOficial
  - tituloOriginal

- genero
- nacionalidad
- duracion
- anho
- distribuidora
- director
- clasificacionEdad
- otrosDatos
- actores
- url\_image
- url\_video

**Sala:** entidad que define las características de una sala de proyección.

- Atributos:
  - nombreSala (Clave Primaria)
  - filas
  - columnas

**Entrada:** entidad que registra las entradas a las proyecciones.

- Atributos:
  - idEntrada (Clave Primaria)
  - fecha
  - hora
  - fila
  - columna

**Usuario:** entidad que almacena la información de los usuarios registrados en el sistema.

- Atributos:
  - nombre
  - apellidos
  - contrasenha
  - email (Clave Primaria)
  - fechaNacimiento

**Reserva:** entidad que representa las reservas de entradas realizadas por los usuarios.

- Atributos:
  - numeroRef (Clave Primaria)

**Comentario:** entidad que almacena los comentarios de los usuarios sobre las películas.

- Atributos:
  - texto
  - valoracion
  - fechaComentario

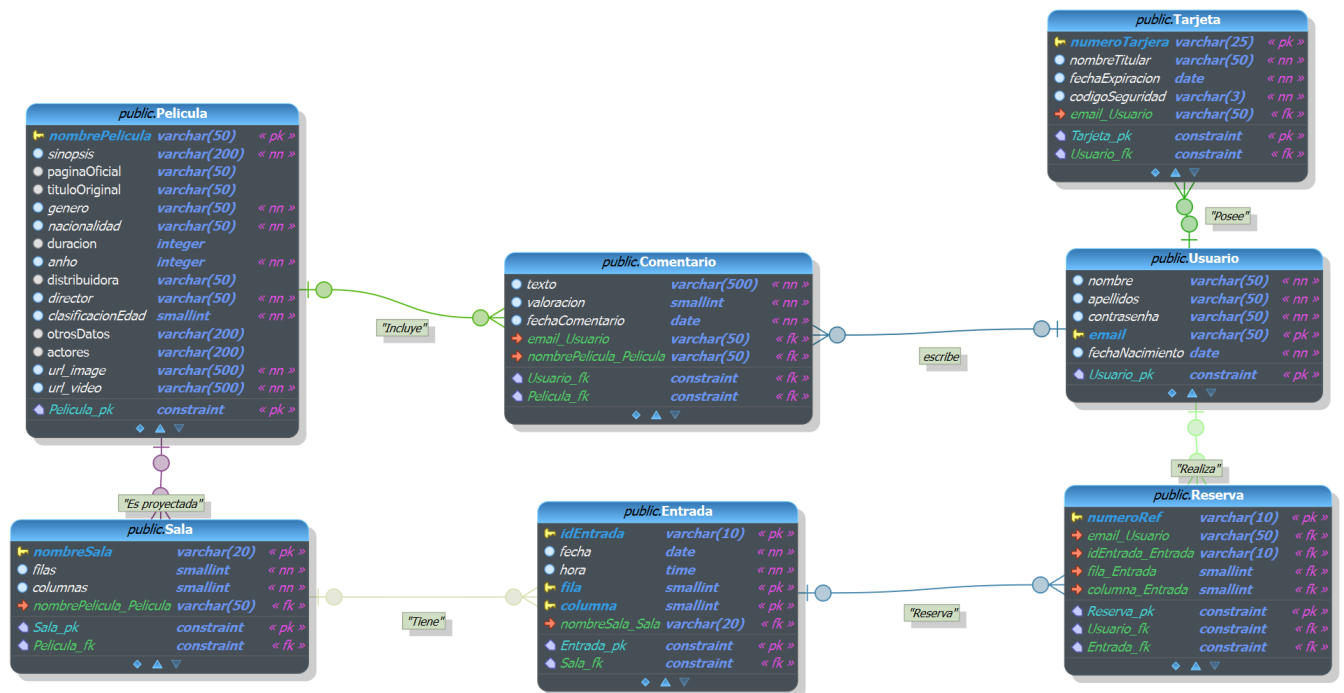
**Tarjeta:** entidad que guarda la información de las tarjetas de crédito asociadas a los usuarios.

- Atributos:
  - numeroTarjeta (Clave Primaria)
  - nombreTitular
  - fechaExpiracion
  - codigoSeguridad

Estudiadas las entidades con sus respectivos atributos, vamos a presentar las relaciones que hemos creído convenientes:

- La relación entre "Pelicula" y "Sala" se establece mediante la columna nombrePelicula\_Pelicula, indicando qué películas se están proyectando en qué salas.
- La relación entre "Sala" y "Entrada" se establece a través de la clave foránea nombreSala\_Sala, indicando a qué sala pertenece cada entrada.
- La relación entre "Usuario" y "Reserva" se establece mediante la clave foránea email\_Usuario, vinculando las reservas con los usuarios correspondientes.
- La relación entre "Entrada" y "Reserva" se establece mediante la clave foránea idEntrada\_Entrada, fila\_Entrada, columna\_Entrada, indicando a qué entrada está vinculada cada reserva.
- La relación entre "Usuario" y "Comentario" se establece mediante la clave foránea email\_Usuario, vinculando los comentarios con los usuarios correspondientes.
- La relación entre "Pelicula" y "Comentario" se establece mediante la clave foránea nombrePelicula\_Pelicula, vinculando los comentarios con las películas correspondientes.
- La relación entre "Usuario" y "Tarjeta" se establece mediante la clave foránea email\_Usuario, vinculando las tarjetas con los usuarios correspondientes.

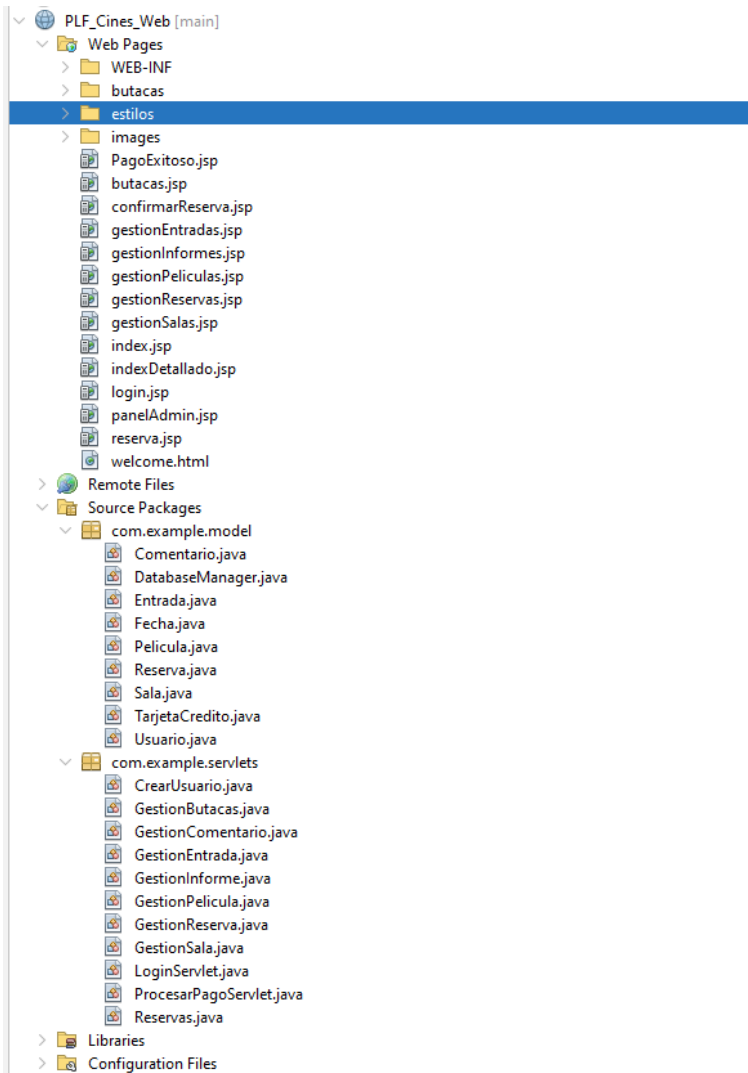
Con estas entidades y relaciones hemos elaborado un modelo como el siguiente:



## Implementación

Lo primero que hay que ver es la estructura del proyecto:





Se puede ver claramente el patrón modelo vista controlador, por un lado, tenemos todos los archivos.jsp que van a actuar como vista, lanzando peticiones al controlador que son todas las clases del paquete **com.examples.servlets**.

Luego en los servlets devolvemos la información de esas peticiones si es necesario, o gestionamos información de las sesiones, o por último, la opción que más hemos llevado a cabo que es el contacto con el modelo que es nuestra base de datos, toda esa gestión de la base de datos, se ha llevado a cabo en la clase **DataBaseManager.java** dentro de paquete **com.example.model**, acciones para poder almacenar usuarios películas, salas, para poder consultar estas pasándole un nombre, para poder modificarlas, para poder borrarlas, una función que devuelva un arraylist con todas las películas, con todas las salas, etc.

Uso de HTML5, CSS3 y JavaScript (jQuery).

Uso de HTML5:

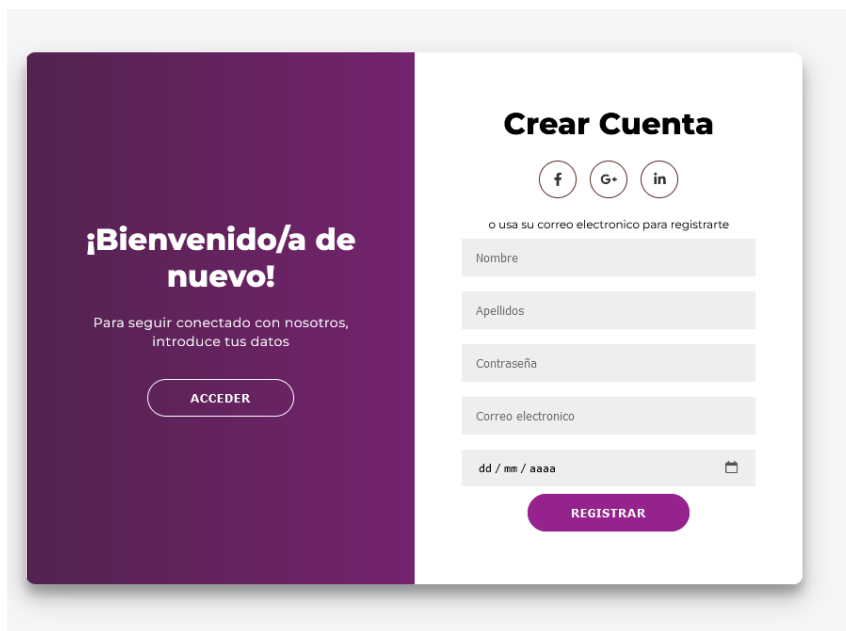
El uso de html en este proyecto ha sido dentro de los archivos.jsp, que permiten la integración de código java, gestión de sesiones y manejos de bases de datos con html5, lo que nos permitía optimizar en gran forma el código.

En este caso el uso ha sido en el apartado de la vista dentro del patrón MVC, tanto para el panel de administrador, como para el registro de los usuarios, como para la propia plataforma de cines y hacer reservas y comentarios.

**Pasamos a ver por encima el uso principal del html5, en los diferentes paneles:**

### Panel de Registro e Inicio de sesión:

El usuario lo que ve es esto:



```
<div class="form-container sign-up-container">
    <form action="CrearUsuario" method="post">
        <h1>Crear Cuenta</h1>
        <div class="social-container">
            <a href="#" class="social"><i class="fab fa-facebook-f"></i></a>
            <a href="#" class="social"><i class="fab fa-google-plus-g"></i></a>
            <a href="#" class="social"><i class="fab fa-linkedin-in"></i></a>
        </div>
        <span>o usa su correo electrónico para registrarte</span>
        <input type="text" placeholder="Nombre" id="Name" name="Name"/>
        <span id="error-nombre" class="error"></span>
        <input type="text" placeholder="Apellidos" id="Apellidos" name="Apellidos"/>
        <span id="error-apellido" class="error"></span>
        <input type="password" placeholder="Contraseña" id="pswd" name="pswd"/>
        <span id="error-contrasena" class="error"></span>
        <input type="email" placeholder="Correo electrónico" id="mail" name="mail"/>
        <span id="error-email" class="error"></span>
    </form>
</div>
```

```

        <input type="date" placeholder="Fecha-nacimiento" id="Fecha-nacimiento"
name="Fecha-nacimiento"/>

        <span id="error-fecha" class="error"></span>

        <button type="submit" id="Registrar">Registrar</button>

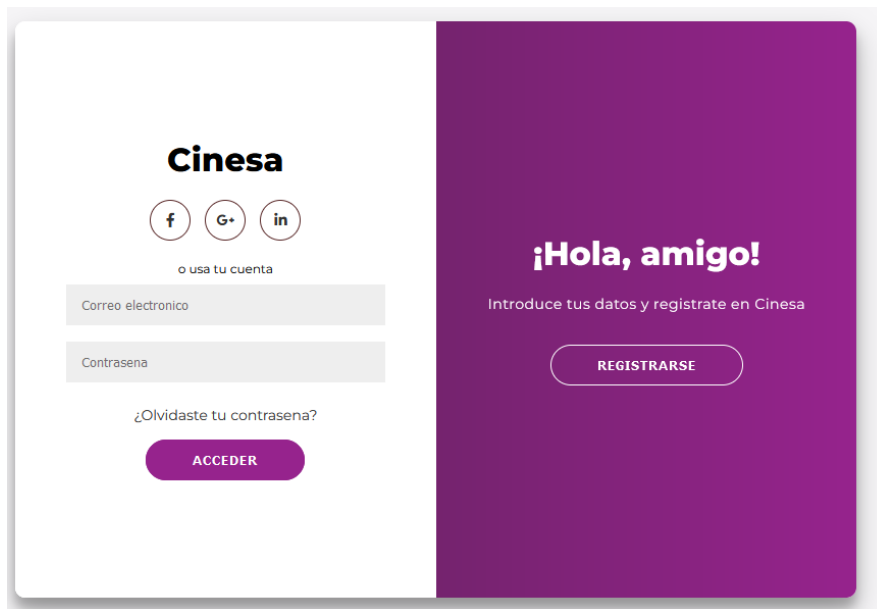
    </form>

</div>

```

Este es un formulario HTML para crear una cuenta de usuario. Cuando el usuario hace clic en el botón "Registrar", los datos del formulario se envían a un servlet llamado "CrearUsuario" utilizando el método POST.

El formulario incluye campos para el nombre, apellidos, contraseña, correo electrónico y fecha de nacimiento del usuario. Cada campo de entrada tiene un elemento `<span>` asociado con un ID específico (por ejemplo, "error-nombre", "error-apellido", etc.) que se puede usar para mostrar mensajes de error relacionados con ese campo.



```

<div class="form-container sign-in-container">

    <form action="AccederUsuario" method="post">

        <h1>Cinesa</h1>

        <div class="social-container">

            <a href="#" class="social"><i class="fab fa-facebook-f"></i></a>

            <a href="#" class="social"><i class="fab fa-google-plus-g"></i></a>

            <a href="#" class="social"><i class="fab fa-linkedin-in"></i></a>

        </div>

        <span>o usa tu cuenta</span>

        <input type="email" placeholder="Correo electrónico" id = "mail-2"
name="mail-2"/>

        <span id="error-email-acceso" class="error"></span>

```

```
<input type="password" placeholder="Contraseña" id = "pswd-2" name="pswd-2"/>

<span id="error-contrasena-acceso" class="error"></span>

<a href="#">¿Olvidaste tu contraseña?</a>

<button type="submit" id="Acceder">Acceder</button>

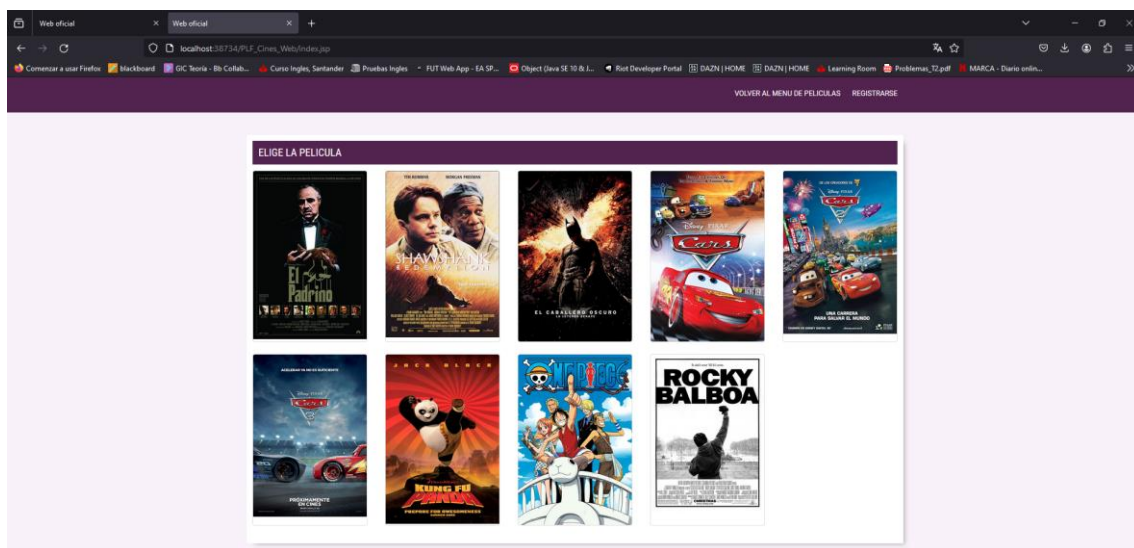
</form>

</div>
```

Este es un formulario HTML para iniciar sesión en una cuenta de usuario. Cuando el usuario hace clic en el botón "Acceder", los datos del formulario se envían a un servlet llamado "AccederUsuario" utilizando el método POST.

El formulario incluye campos para el correo electrónico y la contraseña del usuario. Cada campo de entrada tiene un elemento `<span>` asociado con un ID específico (por ejemplo, "error-email-acceso", "error-contrasena-acceso") que se puede usar para mostrar mensajes de error relacionados con ese campo.

### Panel de la aplicación principal index.jsp:



```
<div class="container">

    <div class="white-box">

        <h3 class="titulo-cine">ELIGE LA PELICULA</h3>

        <div class="row">

            <% // Llamamos al metodo getAllPelículas para iterar sobre todas las películas
                List<Película> listaPelículas DatabaseManager.getInstance().getAllPelículas();
            %>

            <% for(Película película : listaPelículas){ %>

                <div class="col-6 col-sm-4 col-md-3 col-xl-2dot4 mb20">
```

```

        <div class="card card-event info-overlay">
            <div class="img has-background">

                <a href="indexDetallado.jsp?id=<%= pelicula.getNombre() %>"
class="event-pop-link">

                    <div class="event-pop-info">
                        <p class="price" style="font-size:1.1em; padding:
5px;"><button type="submit" style="background: none; border: none; padding: 0; color: inherit; cursor:
pointer;"><%= pelicula.getNombre() %></button></p>
                        <span style="font-size: 0.90em; background-color: #53234f;"
class="badge badge-primary">VER FICHA Y PASES</span><br /><br />
                    </div>
                </a>

                <a href="indexDetallado.jsp?id=<%= pelicula.getNombre() %>"></a>

            </div>
        </div>
    </div>
<% } %>
</div>

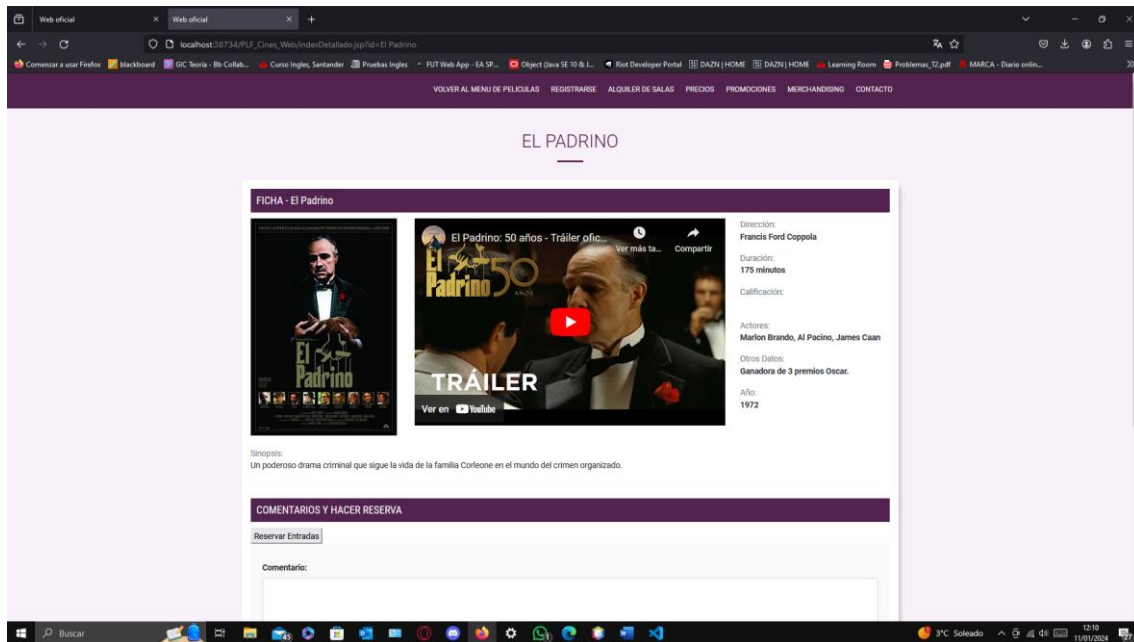
</div>
</div>

```

Primero, se obtiene una lista de objetos Pelicula llamando al método getAllPelículas() del DatabaseManager.

Luego, se itera sobre esta lista con un bucle for. Para cada película, se genera un bloque de HTML que incluye un enlace a una página de detalles de la película (indexDetallado.jsp), pasando el nombre de la película como un parámetro en la URL.

Dentro de este bloque, se muestra el nombre de la película en un botón y se proporciona un enlace para ver más detalles y pases de la película. También se muestra una imagen de la película, con la URL de la imagen obtenida llamando al método getUrl\_image() del objeto Pelicula.



Sección dentro de cada película indexDetallado.jsp

```
<% String nombrePelicula = request.getParameter("id");
    Pelicula pelicula =
DatabaseManager.getInstance().getPeliculaPorNombre(nombrePelicula);
    session.setAttribute("pelicula",pelicula);
    %>

<% if (pelicula != null) { %>

    <div class="row clearfix">
        <h1 class="text-center title-1"><%= pelicula.getNombre() %></h1>
        <hr class="mx-auto small text-hr" style="margin-bottom: 30px
!important">

        <div style="clear:both">
            <hr>
        </div>
    </div>

    <div class="white-box" style="padding:15px">
        <h3 class="titulo-cine">FICHA - <%= pelicula.getNombre() %></h3>
        <div class="row mb20">
            <div class="col-3 d-none d-md-block">
```

```
<a href="<%= pelicula.getUrl_image() %>" data-
toggle="lightbox"></a>

</div>

<div class="col-6 d-none d-md-block"><iframe width="100%"
height="350" src="<%= pelicula.getUrl_video() %>" title="YouTube video player" frameborder="0"
allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture;
web-share" allowfullscreen></iframe></div>

<div class="col-3 d-none d-md-block" style="padding-left: 10px;">
    <ul class="list-unstyled">
        <li><p class="text-muted">Dirección:<br><strong class="text-
dark"> <%= pelicula.getDirector() %></strong></p></li>

        <li><p class="text-muted">Duración:<br><strong class="text-
dark"><%= pelicula.getDuracion() %> minutos</strong></p></li>

        <li><p class="text-muted">Calificación:<br><strong
class="text-dark"><img title="No recomendada para menores de <%= pelicula.getClasificacionEdad()
%>" style="width:20px; margin-right: 5px;"></strong></p></li>

        <li><p class="text-muted">Actores:<br><strong class="text-
dark"><%= pelicula.getActores() %></strong></p></li>

        <li><p class="text-muted">Otros Datos:<br><strong
class="text-dark"><%= pelicula.getOtrosDatos() %></strong></p></li>

        <li><p class="text-muted">Año:<br><strong class="text-
dark"><%= pelicula.getAño() %></strong></p></li>

    </ul>
</div>

<div class="col-4 d-md-none">

    <span data-toggle="modal" data-target="#trailer" class="btn btn-
primary" style="padding-top: 10px; margin-top: 10px; display: block; padding-bottom:
10px;">TRAILER <i class="fa fa-play" style="margin-left: 2px;"></i></span>

</div>

</div>

<div class="row mb30">
    <div class="col-sm-12">
```

```

<span class="text-muted">Sinopsis:</span>

<p><%= pelicula.getSinopsis() %></p>

</div>

</div>

```

Primero, se verifica si el objeto pelicula no es nulo. Si no es nulo, se muestra la información de la película.

El nombre de la película se muestra en un encabezado <h1> y también en un encabezado <h3> precedido por la palabra "FICHA".

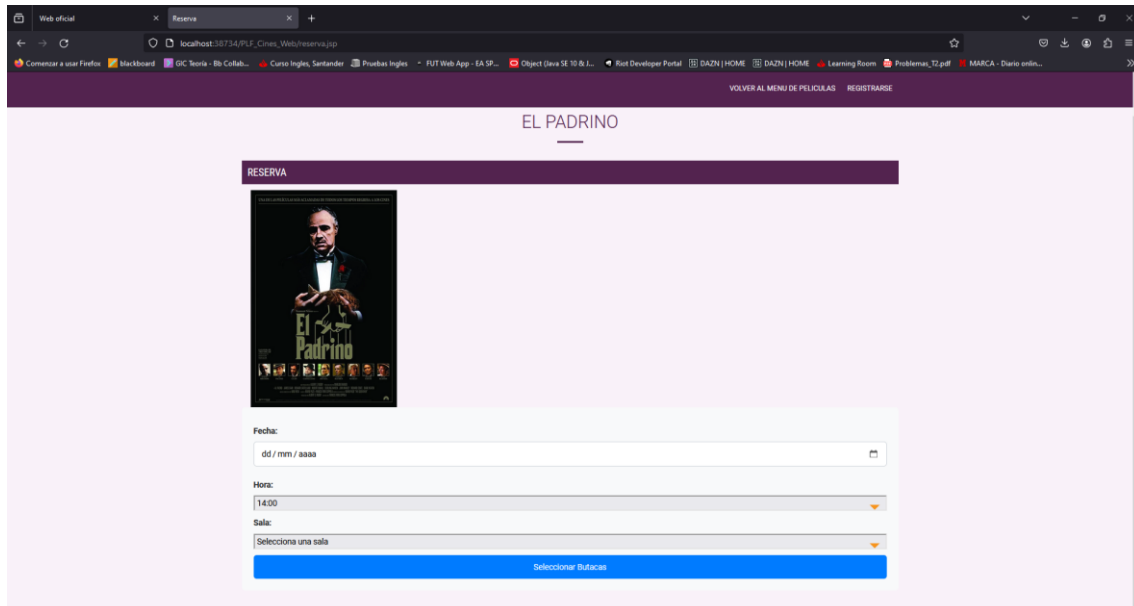
Se muestra una imagen de la película, con la URL de la imagen obtenida llamando al método getUrl\_image() del objeto Pelicula. Esta imagen está vinculada a la URL de la imagen, por lo que al hacer clic en ella se abrirá la imagen en una nueva ventana o pestaña.

También se muestra un video de la película, con la URL del video obtenida llamando al método getUrl\_video() del objeto Pelicula.

Se muestra información adicional sobre la película en una lista sin orden, incluyendo el director, la duración, la clasificación de edad, los actores, otros datos y el año.

Finalmente, se muestra la sinopsis de la película, obtenida llamando al método getSinopsis() del objeto Pelicula.

### Panel de gestión de reservas reserva.jsp:



```

<h3 class="titulo-cine">RESERVA</h3>

<div class="col-3 d-none d-md-block">

```



```

        <a href="<%= peliculaSeleccionada.getUrl_image() %>" data-
toggle="lightbox"></a>

    </div>

    <div class ="formreservas">

    <form action="Reservas" method="post" onsubmit="return validarFormulario()"
>

        <label for="fecha">Fecha:</label>
        <input type ="date" name="fecha" id="fecha" required>
        <br>

        <label for="hora">Hora:</label>
        <select name="hora" id="hora" required></select>
        <br>

        <br>
        <label for="sala">Sala:</label>
        <select name="salaSeleccionada" id="salaSeleccionada">
            <% List<Sala> salas = new ArrayList<>();
            try {
                salas = DatabaseManager.getAllSalas(); // Obtener todas las
salas

            } catch (Exception e) {
                e.printStackTrace();
            }
            %>
            <option value="" disabled selected>Selecciona una sala</option>
            <%
                for (Sala sala : salas) {
                    if
(sala.getNombre_pelicula().equals(peliculaSeleccionada.getNombre())) {
                        %>
                        <option value="<%= sala.getNombreSala() %>"><%=
sala.getNombreSala() %></option>
                        <%
                    }
                }
            %>

```

```

        </select>
        <br>
        <br>

        <button type="submit">Seleccionar Butacas</button>
    </form>
</div>
</div>
</div>
</div>
<script>
    // Obtener la duración de la película y el tiempo entre pases desde el servidor
    var duracionPelicula = <%= peliculaSeleccionada.getDuracion() %>;
    var tiempoEntrePases = 15; // minutos

    // Establecer la hora de inicio y fin del cine
    var horaInicioCine = new Date();
    horaInicioCine.setHours(14, 0, 0, 0);

    var horaFinCine = new Date();
    horaFinCine.setHours(23, 59, 0, 0);

    // Calcular los horarios de los pases redondeando a las horas enteras
    var horarios = [];
    var sdf = new Intl.DateTimeFormat('en-US', {hour: 'numeric', minute: 'numeric',
hour12: false});

    // Calcular el número de pases en el día
    var numPases = Math.floor((9 * 60 / duracionPelicula));
    for (var i = 0; i < numPases; i++) {
        // Formatear la hora
        var horaRedondeada = new Date(horaInicioCine);
        horaRedondeada.setMinutes(Math.round(horaRedondeada.getMinutes() / 15) * 15);

        // Verificar si la hora está dentro del horario del cine
        if (horaRedondeada >= horaInicioCine && horaRedondeada <= horaFinCine) {
            horarios.push(sdf.format(horaRedondeada));
        }

        // Calcular el siguiente pase
        horaInicioCine.setMinutes(horaInicioCine.getMinutes() + duracionPelicula +
tiempoEntrePases);
    }
}

```

```
}

// Limpiar la lista de horarios
var selectHora = document.getElementById("hora");
selectHora.innerHTML = "";

// Añadir opciones a la lista de horarios
for (var i = 0; i < horarios.length; i++) {
    var option = document.createElement("option");
    option.value = horarios[i];
    option.textContent = horarios[i];

    // Agregar la opción a la lista de horarios
    selectHora.appendChild(option);
}

function validarFormulario() {
    // Validar la fecha
    var fechaInput = document.getElementById("fecha").value;
    var fechaSeleccionada = new Date(fechaInput);
    var fechaActual = new Date();

    if (fechaSeleccionada < fechaActual) {
        alert("La fecha no puede ser menor que el día actual");
        return false;
    }

    // El formulario es válido
    return true;
}

</script>

</body>
```

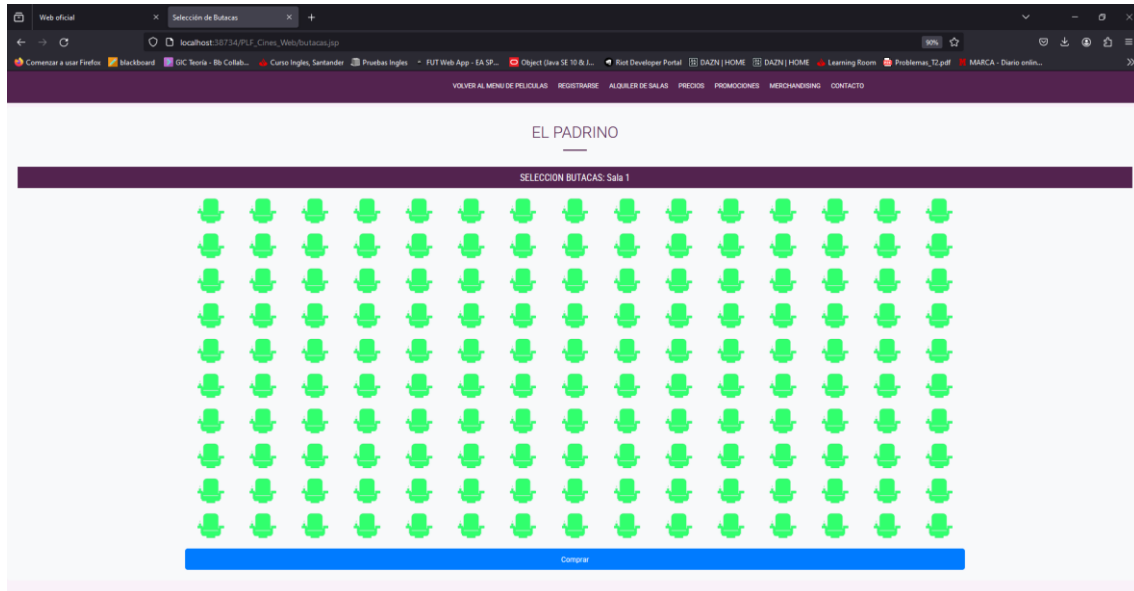
Primero, se muestra una imagen de la película seleccionada. Luego, se presenta un formulario que permite a los usuarios seleccionar la fecha, la hora y la sala para la película.

La fecha se selecciona a través de un campo de entrada de fecha. La hora se selecciona a través de un menú desplegable que se llena dinámicamente con JavaScript basado en la duración de la película y el horario de funcionamiento del cine.

Las salas se seleccionan a través de un menú desplegable que se llena dinámicamente con JSP. Se obtiene una lista de todas las salas de la base de datos y se añaden al menú desplegable las salas que están mostrando la película seleccionada.

El formulario se envía a un servlet llamado "Reservas" utilizando el método POST. Antes de enviar el formulario, se valida la fecha seleccionada con JavaScript para asegurarse de que no sea anterior a la fecha actual.

Panel de gestión de butacas **butacas.jsp**:



```
<h3 class="titulo-cine">SELECCION BUTACAS: <%= salaSelec.getNombreSala() %></h3>

<div class = "tablabutaca">
  <form action="GestionButacas" method="post">
    <table style='border-collapse: collapse;'>
      <%
        for (int i = 0; i < salaSelec.getFilas(); i++) {
      %>
      <tr>
        <%
          for (int j = 0; j < salaSelec.getColumnas(); j++) {
            int fila = i + 1;
            int columna = j + 1;
            boolean ocupadaPorOtroCliente = false;

            // Verificar si la butaca está ocupada por otras entradas
            for (Entrada entrada : entradas) {
              System.out.println(entrada.getFecha().equals(fecha));
              if
(entrada.getNombreSala().equals(salaSelec.getNombreSala()) &&
                entrada.getFila() == fila &&
                entrada.getColumna() == columna &&
```

```
entrada.getFecha().toLocaleDate().equals(fecha.toLocalDat
e()) &&

entrada.getHora().equals(hora))
{
    ocupadaPorOtroCliente = true;

    break;
}
}
%>
<td>
    <%
        if (!ocupadaPorOtroCliente) {
            // Verificar si la butaca está libre u ocupada
            boolean ocupada = false;

            if (ocupada) {
                //Si esta ocupada , establecer imagen
butaca_seleccionada

            }
            <img class='butaca ocupada'
                src='butacas/butaca_seleccionada.png'
                alt='Butaca <%= fila %>-<%= columna %>'
                data-fila='<%= fila %>' data-columna='<%= columna %>'>
            <%
        } else {
            //Si esta libre , establecer imagen butaca_libre

            <img class='butaca'
                src='butacas/butaca_libre.png'
                alt='Butaca <%= fila %>-<%= columna %>'
                data-fila='<%= fila %>' data-columna='<%= columna %>'>
            <%
        }
    } else {
        //Si esta ocupada previamente, establecer imagen butaca_ocupada
        <img class='butaca ocupadaPorClientes'
            src='butacas/butaca_ocupada.png'
            alt='Butaca <%= fila %>-<%= columna %>'
```

```

      data-fila='<%= fila %>' data-columna='<%= columna %>'>

      <%
      }
      %>

    </td>

    <%
    }
    }
    %>
  </tr>

</table>

<!-- Campo oculto para almacenar butacas seleccionadas -->
<input type="hidden" name="butacasSeleccionadas" id="butacasSeleccionadas">

<button type="submit" name="comprarButacas">Comprar</button>
</form>
</div>

```

Primero, se muestra el nombre de la sala. Luego, se presenta una tabla que representa los asientos en la sala. Cada fila de la tabla representa una fila de asientos y cada celda representa un asiento.

Para cada asiento, se verifica si está ocupado por otro cliente. Si es así, se muestra una imagen de un asiento ocupado. Si no, se verifica si el asiento está ocupado. Si es así, se muestra una imagen de un asiento seleccionado. Si no, se muestra una imagen de un asiento libre.

Los asientos seleccionados se almacenan en un campo oculto en el formulario. Cuando el usuario hace clic en el botón "Comprar", el formulario se envía a un servlet llamado "GestionButacas" utilizando el método POST.

Panel de pago **confirmarReserva.jsp**:

### Informacion del pago



Nombre del titular  
 David Bachiller

Numero de tarjeta  
 5200 8282 8282 8210

Fecha de caducidad (mm/yy)  
 02/28

Código de seguridad  
 198

Procesar pago

```
<div class="form-container">
  <div class="field-container">
    <label for="name">Nombre del titular</label>
    <input id="name" name="titular" maxlength="20" type="text" required>
  </div>
  <div class="field-container">
    <label for="cardnumber">Numero de tarjeta</label><span
id="generatecard">generar numero random</span>
    <input id="cardnumber" name="numeroT" type="text"
pattern="\d{4}\s?\d{4}\s?\d{4}\s?\d{4}" inputmode="numeric">
    <svg id="ccicon" class="ccicon" width="750" height="471" viewBox="0 0 750
471" version="1.1" xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">

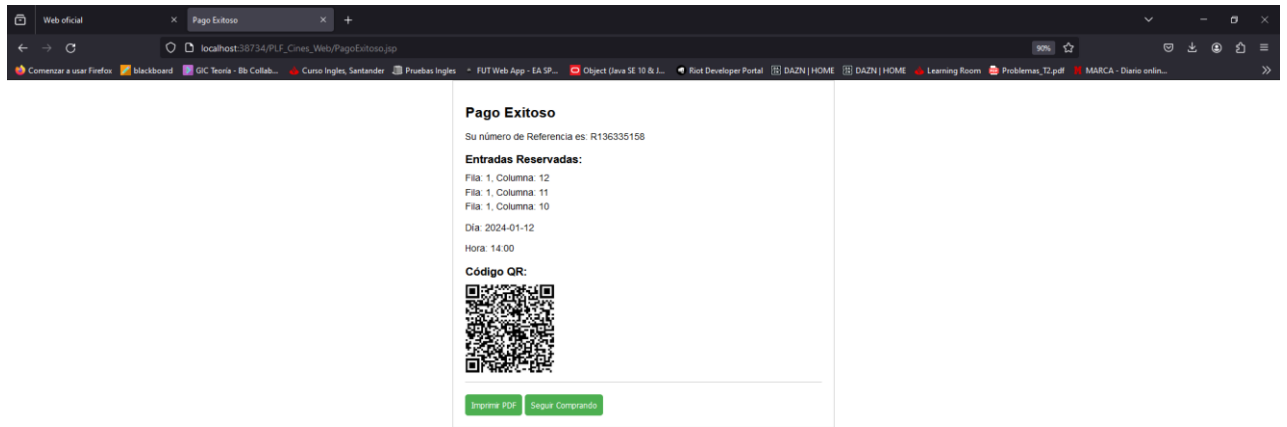
    </svg>
  </div>
  <div class="field-container">
    <label for="expirationdate">Fecha de caducidad (mm/yy)</label>
    <input id="expirationdate" name="fechaCaducidad" type="text" pattern="[0-
9]{2}/[0-9]{2}" inputmode="numeric" required="">
  </div>
  <div class="field-container">
    <label for="securitycode">Codigo de seguridad</label>
    <input id="securitycode" name="codigoSeguridad" type="text" pattern="[0-
9]*" inputmode="numeric" required>
  </div>
  <button type="submit">Procesar pago</button>
</div>
</form>
```

El formulario incluye campos para el nombre del titular de la tarjeta, el número de la tarjeta, la fecha de caducidad de la tarjeta y el código de seguridad de la tarjeta.

- El campo "Nombre del titular" acepta hasta 20 caracteres y es obligatorio.
- El campo "Número de tarjeta" debe seguir el patrón de 4 grupos de 4 dígitos, que pueden estar separados por espacios. También hay un enlace para generar un número de tarjeta aleatorio, pero actualmente no hace nada ya que su funcionalidad no está implementada.
- El campo "Fecha de caducidad" debe seguir el patrón mm/yy y es obligatorio.
- El campo "Código de seguridad" solo acepta números y es obligatorio.

Cuando el usuario hace clic en el botón "Procesar pago", los datos del formulario se envían al servidor para su procesamiento.

## Panel de confirmación de pago y reserva **PagoExitoso.jsp**:



```
<h2>Pago Exitoso</h2>

<p>Su número de Referencia es: <%= session.getAttribute("numRef") %></p>

<h3>Entradas Reservadas:</h3>

<!-- Mostrar información de cada entrada --%>

<% String butacasJSON = (String) session.getAttribute("butacasSeleccionadas");

// Eliminar corchetes y comillas para obtener pares clave-valor separados por comas
String cleanString = butacasJSON.replaceAll("[\\[\\]\\{\\}\\\" ]", "");

// Dividir la cadena en pares clave-valor
String[] keyValuePairs = cleanString.split(",");

// Crear una lista de mapas para almacenar las butacas seleccionadas
List<Map<String, Integer>> butacasList = new ArrayList<>();

// Iterar sobre los pares clave-valor y agregar a la lista de mapas
for (int i = 0; i < keyValuePairs.length; i += 2) {
    String[] filaKC = keyValuePairs[i].split(":");
    String[] columnaKC = keyValuePairs[i + 1].split(":");
    Map<String, Integer> butaca = Map.of("fila", Integer.parseInt(filaKC[1]), "columna",
Integer.parseInt(columnaKC[1]));
    butacasList.add(butaca);
}

%>

<ul>

<% for (Map<String, Integer> butaca : butacasList) { %>

    <li>Fila: <%= butaca.get("fila") %>, Columna: <%= butaca.get("columna") %></li>

    <% } %>

</ul>
```



```
<p>Día: <%= session.getAttribute("fecha") %></p>

<p>Hora: <%= session.getAttribute("hora") %></p>

<div class="section">

    <!-- Mostrar el código QR generado en el JavaScript -->

    <h3>Código QR:</h3>

    <%

        String datosReserva = "Numero de Referencia: " + session.getAttribute("numRef") +

                                "Día: " + session.getAttribute("fecha") +

                                "Hora: " + session.getAttribute("hora") +

                                "Butacas Reservadas: " + butacasJSON;

    %>

    <div id="codigoQR"></div>

</div>

<button onclick="imprimirPDF()">Imprimir PDF</button>

<a href="index.jsp"><button type="button">Seguir Comprando</button></a>

<script>

    var qrcode = new QRCode("codigoQR", {

        text: '<%= datosReserva %>',

        width: 200,

        height: 200,

        correctLevel: QRCode.CorrectLevel.H

    });

    function imprimirPDF() {

        const options = {

            margin: 1,

            filename: 'pago_exitoso.pdf',

            image: {type: 'jpeg', quality: 0.98},

            html2canvas: {

                scale: 2,

                letterRendering: true,

            },

            jsPDF: {unit: 'in', format: 'a3', orientation: 'portrait'},

        };

        const content = document.body;

        html2pdf().from(content).set(options).save();
```

```
}  
</script>
```

La página muestra un mensaje de éxito y el número de referencia de la transacción recuperado de la sesión.

Luego recupera una cadena JSON de asientos seleccionados de la sesión, limpia la cadena y la divide en pares clave-valor. Estos pares se utilizan para crear una lista de mapas, cada uno representando un asiento seleccionado con sus números de fila y columna.

Los asientos seleccionados se muestran en formato de lista.

La fecha y hora seleccionadas para la película se recuperan de la sesión y se muestran.

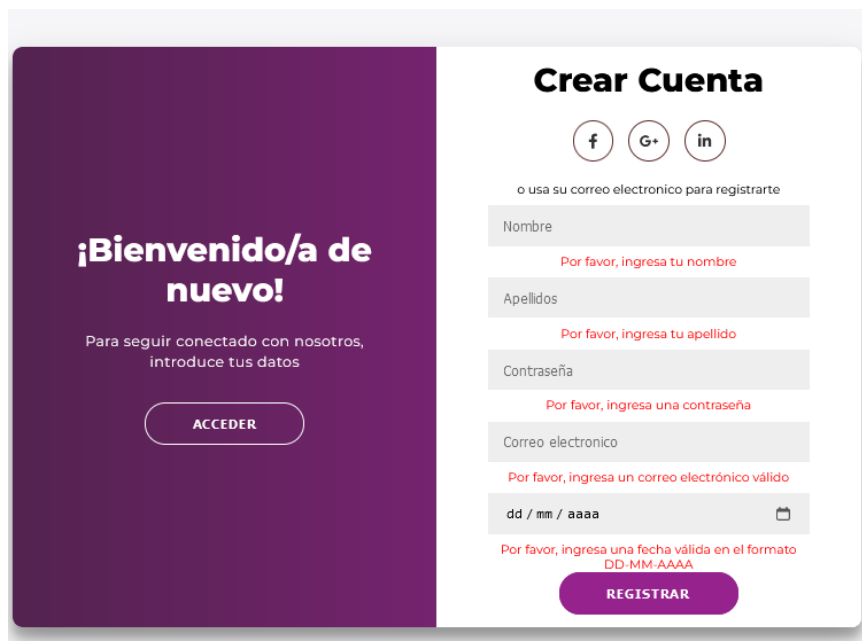
Se genera un código QR dinámicamente utilizándola librería qrcode. El código QR contiene el número de referencia, la fecha, la hora y los asientos seleccionados.

Se proporciona un botón para imprimir la página como PDF. Esto se logra utilizando la biblioteca JavaScript html2pdf. La función imprimirPDF() se llama cuando se hace clic en el botón, que convierte la página actual en un PDF y la guarda con el nombre de archivo 'pago\_exitoso.pdf'.




También se proporciona un botón para volver a la página de inicio para seguir comprando.

### Uso de JavaScript:

Principalmente hemos usado javascript para el chequeo de formularios, tanto el de panel de registro de usuarios como el panel del administrador, para no poder rellenar datos absurdos, pasamos a ver el panel de registro:



**Crear Cuenta**

o usa su correo electrónico para registrarte

Nombre

Por favor, ingresa tu nombre

Apellidos


Por favor, ingresa tu apellido

Contraseña

Por favor, ingresa una contraseña

Correo electrónico

Por favor, ingresa un correo electrónico válido

dd / mm / aaaa 

Por favor, ingresa una fecha válida en el formato DD-MM-AAAA

**REGISTRAR**

```
document.addEventListener('DOMContentLoaded', function() {  
    document.getElementById('Registrar').addEventListener('click', function(e) {  
        var nombre = document.getElementById('Name').value;
```

```
var apellido = document.getElementById('Apellidos').value;
var emailRegistro = document.getElementById('mail').value;
var contrasena = document.getElementById('pswd').value;
var fechaNacimiento = document.getElementById('Fecha-nacimiento').value;
var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
var fechaPattern = /^[d{4}-d{2}-d{2}$/; // Define un patrón de fecha (AAAA-MM-DD)

// Verificar si el nombre está lleno
if (nombre.trim() === '') {
    document.getElementById('error-nombre').textContent = 'Por favor, ingresa tu
nombre';
    e.preventDefault();
} else {
    document.getElementById('error-nombre').textContent = '';
}

// Verificar si el apellido está lleno
if (apellido.trim() === '') {
    document.getElementById('error-apellido').textContent = 'Por favor, ingresa tu
apellido';
    e.preventDefault();
} else {
    document.getElementById('error-apellido').textContent = '';
}

// Verificar si el correo electrónico de registro es válido
if (!emailRegistro.match(emailPattern)) {
    document.getElementById('error-email').textContent = 'Por favor, ingresa un correo
electrónico válido';
    e.preventDefault();
} else {
    document.getElementById('error-email').textContent = '';
}

// Verificar si la contraseña está llena
if (contrasena.trim() === '') {
    document.getElementById('error-contrasena').textContent = 'Por favor, ingresa una
contraseña';
    e.preventDefault();
} else {
    document.getElementById('error-contrasena').textContent = '';
}
```

```
// Verificar si la fecha cumple con el formato esperado
if (!fechaNacimiento.match(fechaPattern)) {
    document.getElementById('error-fecha').textContent = 'Por favor, ingresa una fecha
válida en el formato DD-MM-AAAA';
    e.preventDefault();
} else {
    document.getElementById('error-fecha').textContent = '';
}
});

document.getElementById('Acceder').addEventListener('click', function(e) {
    var emailAcceso = document.getElementById('mail-2').value;
    var contrasena = document.getElementById('pswd-2').value;
    var emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    // Verificar si el correo electrónico de acceso es válido
    if (!emailAcceso.match(emailPattern)) {
        document.getElementById('error-email-acceso').textContent = 'Por favor, ingresa un
correo electrónico válido';
        e.preventDefault();
    } else {
        document.getElementById('error-email-acceso').textContent = '';
    }

    // Verificar si la contraseña está lleno
    if (contrasena.trim() === '') {
        document.getElementById('error-contrasena-acceso').textContent = 'Por favor, ingresa
una contraseña';
        e.preventDefault();
    } else {
        document.getElementById('error-contrasena-acceso').textContent = '';
    }
});

document.getElementById('signIn').addEventListener('click', function() {
    document.getElementById('container').classList.remove('right-panel-active');
});

document.getElementById('signUp').addEventListener('click', function() {
    document.getElementById('container').classList.add('right-panel-active');
});
});
```

Se encarga de la validación de los formularios de registro y acceso, y de la interacción con los botones de inicio de sesión y registro.

- Cuando se hace clic en el botón 'Registrar', se recogen los valores de los campos del formulario de registro. Se verifica que cada campo esté lleno y que los campos de correo electrónico y fecha de nacimiento cumplan con los patrones requeridos. Si un campo no cumple con las validaciones, se muestra un mensaje de error y se evita la acción predeterminada del evento de clic (probablemente la presentación del formulario).
- Cuando se hace clic en el botón 'Acceder', se recogen los valores de los campos del formulario de acceso. Se verifica que el correo electrónico y la contraseña estén llenos y que el correo electrónico cumpla con el patrón requerido. Si un campo no cumple con las validaciones, se muestra un mensaje de error y se evita la acción predeterminada del evento de clic.
- Cuando se hace clic en el botón 'signIn', se elimina la clase 'right-panel-active' del elemento con id 'container'. Esto probablemente cambia la vista para mostrar el formulario de inicio de sesión.
- Cuando se hace clic en el botón 'signUp', se agrega la clase 'right-panel-active' al elemento con id 'container'. Esto probablemente cambia la vista para mostrar el formulario de registro.

Panel de guardar películas:

Género

Por favor, ingresa generos validos. Los siguientes generos no son validos:

Nacionalidad

Por favor, ingresa una nacionalidad valida

Duración (minutos)

Año:

Distribuidora

Director

ClasificacionEdad

Por favor, ingresa una clasificacion de edad valida

Otros Datos

```
document.getElementById('botonCrearPelicula').addEventListener('click', function(e) {
    //Cogeremos todos los datos del formulario
    var anho = parseInt(document.getElementById('anho').value, 10);
    var genero = document.getElementById('genero').value.toLowerCase();
    var nacionalidad = document.getElementById('nacionalidad').value;
    var clasificacionEdad = parseInt(document.getElementById('clasificacionEdad').value,
10);

    var duracion = parseInt(document.getElementById('duracion').value, 10);
    var listaGeneros = ['accion', 'animacion', 'aventura', 'ciencia ficcion', 'comedia',
'documental', 'drama', 'fantasia', 'historica', 'musical', 'romantica', 'suspense', 'terror'];
    var listaNacionalidades = ['Espanola', 'Francesa', 'Inglesa', 'Americana', 'Turca',
'Italiana'];
    var listaClasificaciones = [0, 7, 12, 16, 18];

    // Una vez que todos los atributos estén seleccionados, haremos el checking de los
mismos para ver si son validos
    // Verificar si el anno introducido esta entre 1980 y 2024
    if (anho < 1980 || anho > 2024) {
        document.getElementById('mensajeErrorAnho').textContent = 'Por favor, ingresa un
anho entre 1980 y 2024';
        document.getElementById('mensajeErrorAnho').style.color = "red";
        e.preventDefault();
    } else {
        document.getElementById('mensajeErrorAnho').textContent = '';
    }

    //
    //Verificar que los generos introducidos son validos
```

```
var generos = genero.split(',').map(g => g.trim());
var generosInvalidos = generos.filter(g => !listaGeneros.includes(g));

if (generosInvalidos.length > 0) {
    //Se verifica que no contiene el genero
    document.getElementById('mensajeErrorGenero').textContent = 'Por favor, ingresa
generos validos. Los siguientes generos no son validos: ' + generosInvalidos.join(', ');
    document.getElementById('mensajeErrorGenero').style.color = "red";
    e.preventDefault();
} else {
    document.getElementById('mensajeErrorGenero').textContent = '';
}

//Verificar si la nacionalidad introducida esta en la lista
if (!listaNacionalidades.includes(nacionalidad)) {
    document.getElementById('mensajeErrorNacionalidad').textContent = 'Por favor,
ingresa una nacionalidad valida';
    document.getElementById('mensajeErrorNacionalidad').style.color = "red";
    e.preventDefault();
} else {
    document.getElementById('mensajeErrorNacionalidad').textContent = '';
}

//Verificar si la clasificacion de edad introducida esta en la lista
if (!listaClasificaciones.includes(clasificacionEdad)) {
    document.getElementById('mensajeErrorClasificacionEdad').textContent = 'Por favor,
ingresa una clasificacion de edad valida';
    document.getElementById('mensajeErrorClasificacionEdad').style.color = "red";
    e.preventDefault();
} else {
    document.getElementById('mensajeErrorClasificacionEdad').textContent = '';
}

//Verificar si la duracion esta entre 0 y 773 minutos
if (duracion < 0 || duracion > 773) {
    document.getElementById('mensajeErrorDuracion').textContent = 'Por favor, ingresa
una duracion entre 0 y 773 minutos';
    document.getElementById('mensajeErrorDuracion').style.color = "red";
    e.preventDefault();
} else {
    document.getElementById('mensajeErrorDuracion').textContent = '';
}
});
```

Este script recoge los valores de varios campos de un formulario y realiza una serie de comprobaciones para validar estos valores. Si un valor no cumple con las condiciones de validación, se muestra un mensaje de error y se evita la acción predeterminada del evento de clic (probablemente la presentación del formulario).

Las comprobaciones de validación son las siguientes:

- El año (anho) debe estar entre 1980 y 2024.
- Los géneros (genero) deben estar en la lista de géneros permitidos (listaGeneros).
- La nacionalidad debe estar en la lista de nacionalidades permitidas (listaNacionalidades).
- La clasificación de edad (clasificacionEdad) debe estar en la lista de clasificaciones de edad permitidas (listaClasificaciones).
- La duración debe estar entre 0 y 773 minutos.

Si un valor no cumple con estas condiciones, se muestra un mensaje de error en rojo y se evita la presentación del formulario. Si un valor cumple con las condiciones, se borra cualquier mensaje de error anterior.

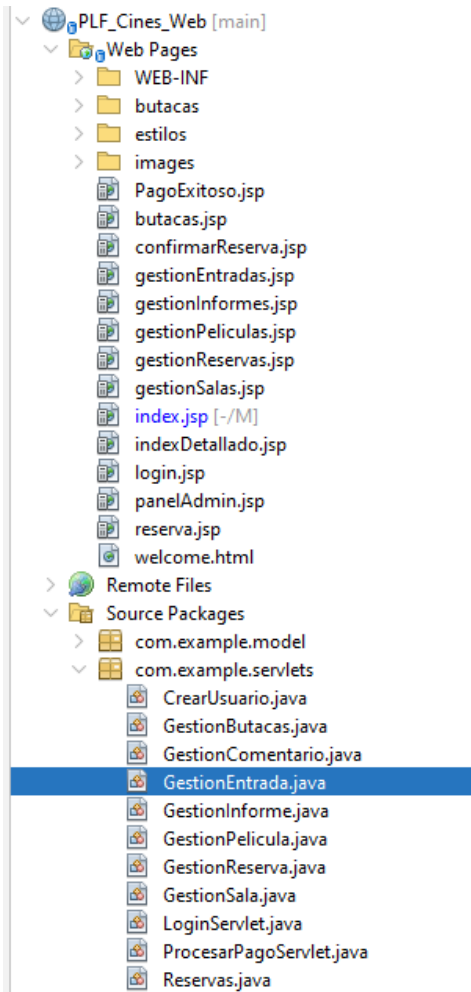
## Programación en JAVA con el patrón MVC.

Como se ha comentado en la introducción de este apartado, nosotros hemos implementado en todo momento el patrón MVC, por un lado, tenemos todos los archivos.jsp que van a actuar como vista, lanzando peticiones al controlador, que son todas las clases del paquete **com.examples.servlets**.

Luego en los servlets devolvemos la información de esas peticiones si es necesario, o gestionamos información de las sesiones, o por último, la opción que más hemos llevado a cabo que es el contacto con el modelo que es nuestra base de datos, toda esa gestión de la base de datos, se ha llevado a cabo en la clase **DataBaseManager.java** dentro de paquete **com.example.model**, acciones para poder almacenar usuarios películas, salas, para poder consultar estas pasándole un nombre, para poder modificarlas, para poder borrarlas, una función que devuelva un arraylist con todas las películas, con todas las salas, etc.

Gestión del administrador:





En todo momento enviamos un método post de la vista al controlador para que este almacene esos datos en el modelo.

Vamos a ver ejemplo de gestión de películas, paso a paso para que se entienda como se aplica el patrón, ya que en el resto de los casos es igual, tanta gestión de salas, gestión de entradas, de informes, etc.

Lo primero es ver la gestión del modelo y es el cómo almacenamos e Utilizamos toda la información de la base de datos.

#### Clase **DataBaseManager.java**

Lo primero que hacemos es crear la estructura básica con los datos que queramos definiendo el constructor y los getters y setters:

```
private String nombre;  
  
private String sinopsis;  
  
private String paginaOficial;  
  
private String tituloOriginal;
```



```
private String genero;

private String nacionalidad;

private int duracion;

private int año;

private String distribuidora;

private String director;

private int clasificacionEdad;

private String otrosDatos;

private String actores;

private String url_image;

private String url_video;


    public Pelicula(String nombre, String sinopsis, String paginaOficial, String tituloOriginal,
String genero, String nacionalidad, int duracion, int año, String distribuidora, String
director, int clasificacionEdad, String otrosDatos, String actores, String url_image, String
url_video) {

        this.nombre = nombre;

        this.sinopsis = sinopsis;

        this.paginaOficial = paginaOficial;

        this.tituloOriginal = tituloOriginal;

        this.genero = genero;

        this.nacionalidad = nacionalidad;

        this.duracion = duracion;

        this.año = año;

        this.distribuidora = distribuidora;

        this.director = director;

        this.clasificacionEdad = clasificacionEdad;

        this.otrosDatos = otrosDatos;

        this.actores = actores;
```

```
this.url_image = url_image;

this.url_video = url_video;

}
```

Creamos la gestión de la película, acciones como guardar película, modificar película, borrar película , getAllPelículas, las cuales usaremos a lo largo de toda la aplicación:

```
public static void guardarPelícula(Película película) throws SQLException {

    abrirConexion();

    System.out.println("GuardarPelícula");

    try {

        if (película != null) {

            String sql = "INSERT INTO pelicula (nombrepelicula, sinopsis, paginaoficial, titulooriginal,
            genero, nacionalidad, duracion, anho, distribuidora, director, clasificacionEdad, otrosdatos, actores,
            url_image, url_video) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";

            try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

                preparedStatement.setString(1, película.getNombre());

                preparedStatement.setString(2, película.getSinopsis());

                preparedStatement.setString(3, película.getPaginaOficial());

                preparedStatement.setString(4, película.getTituloOriginal());

                preparedStatement.setString(5, película.getGenero());

                preparedStatement.setString(6, película.getNacionalidad());

                preparedStatement.setInt(7, película.getDuracion());

                preparedStatement.setInt(8, película.getAño());

                preparedStatement.setString(9, película.getDistribuidora());

                preparedStatement.setString(10, película.getDirector());

                preparedStatement.setInt(11, película.getClasificacionEdad());

                preparedStatement.setString(12, película.getOtrosDatos());

                preparedStatement.setString(13, película.getActores());

                preparedStatement.setString(14, película.getUrl_image());

                preparedStatement.setString(15, película.getUrl_video());

            }

        }

    }

}
```





```
        resultSet.getInt("duracion"),

        resultSet.getInt("anho"),

        resultSet.getString("distribuidora"),

        resultSet.getString("director"),

        resultSet.getInt("clasificacionEdad"),

        resultSet.getString("otrosdatos"),

        resultSet.getString("actores"),

        resultSet.getString("url_image"),

        resultSet.getString("url_video")

    );

    peliculas.add(pelicula);

}

}

}

} finally {

    cerrarConexion();

}

return peliculas;

}

public static Pelicula getPeliculaPorNombre(String nombre) throws SQLException {

    abrirConexion();

    try {

        String sql = "SELECT * FROM pelicula WHERE nombrepelicula = ?";

        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setString(1, nombre);

            try (ResultSet resultSet = preparedStatement.executeQuery()) {

                if (resultSet.next()) {
```

```
        return new Pelicula(

            resultSet.getString("nombrepelicula"),

            resultSet.getString("sinopsis"),

            resultSet.getString("paginaOficial"),

            resultSet.getString("titulooriginal"),

            resultSet.getString("genero"),

            resultSet.getString("nacionalidad"),

            resultSet.getInt("duracion"),

            resultSet.getInt("anho"),

            resultSet.getString("distribuidora"),

            resultSet.getString("director"),

            resultSet.getInt("clasificacionEdad"),

            resultSet.getString("otrosdatos"),

            resultSet.getString("actores"),

            resultSet.getString("url_image"),

            resultSet.getString("url_video")

        );

    }

}

} finally {

    cerrarConexion();

}

return null;

}

public static void borrarPelicula(Pelicula pelicula) throws SQLException {

    abrirConexion();
```

```
try {

    String sql = "DELETE FROM pelicula WHERE nombrepelicula = ?";

    try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

        preparedStatement.setString(1, pelicula.getNombre());

        preparedStatement.executeUpdate();

    }

} finally {

    cerrarConexion();

}

}

public static void modificarPelicula(String nombreActual, Pelicula nuevaPelicula) throws SQLException {

    abrirConexion();

    try {

        String sql = "UPDATE pelicula SET nombrepelicula=?, sinopsis=?, paginaoficial=?, titulooriginal=?,  
genero=?, nacionalidad=?, duracion=?, anho=?, distribuidora=?, director=?, clasificacionedad=?, otrosdatos=?,  
actores=?, url_image=?, url_video=? WHERE nombrepelicula=?";

        try (PreparedStatement preparedStatement = connection.prepareStatement(sql)) {

            preparedStatement.setString(1, nuevaPelicula.getNombre());

            preparedStatement.setString(2, nuevaPelicula.getSinopsis());

            preparedStatement.setString(3, nuevaPelicula.getPaginaOficial());

            preparedStatement.setString(4, nuevaPelicula.getTituloOriginal());

            preparedStatement.setString(5, nuevaPelicula.getGenero());

            preparedStatement.setString(6, nuevaPelicula.getNacionalidad());

            preparedStatement.setInt(7, nuevaPelicula.getDuracion());

            preparedStatement.setInt(8, nuevaPelicula.getAño());

            preparedStatement.setString(9, nuevaPelicula.getDistribuidora());

            preparedStatement.setString(10, nuevaPelicula.getDirector());

            preparedStatement.setInt(11, nuevaPelicula.getClasificacionEdad());
```

```
        preparedStatement.setString(12, nuevaPelicula.getOtrosDatos());

        preparedStatement.setString(13, nuevaPelicula.getActores());

        preparedStatement.setString(14, nuevaPelicula.getUrl_image());

        preparedStatement.setString(15, nuevaPelicula.getUrl_video());

        preparedStatement.setString(16, nombreActual); // Condición para actualizar la película
específica

        preparedStatement.executeUpdate();

    }

    } finally {

        cerrarConexion();

    }

}
```

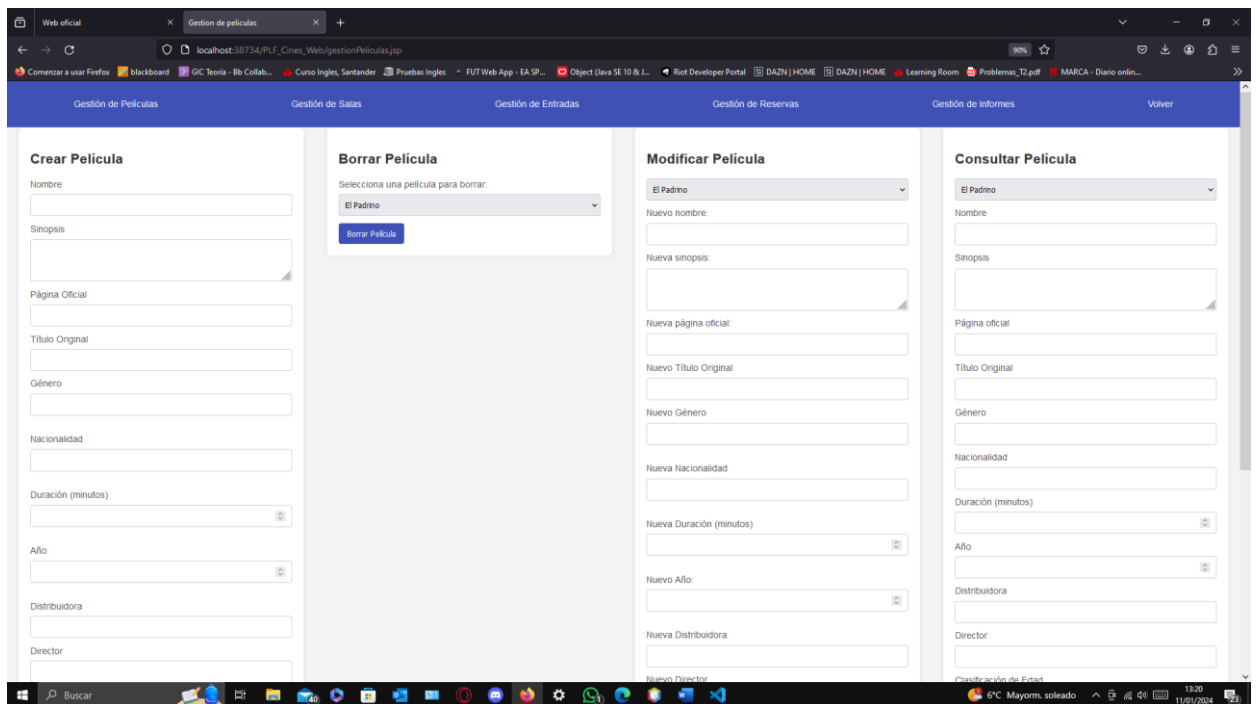
- **guardarPelicula(Pelicula pelicula):** Este método guarda una película en la base de datos. Abre una conexión, prepara una sentencia SQL INSERT, establece los parámetros de la sentencia a las propiedades del objeto película, ejecuta la sentencia y finalmente cierra la conexión.
- **getAllPeliculas():** Este método recupera todas las películas de la base de datos. Abre una conexión, prepara una sentencia SQL SELECT, ejecuta la sentencia, itera sobre el conjunto de resultados para crear una lista de objetos de película, y finalmente cierra la conexión.
- **getPeliculaPorNombre(String nombre):** Este método recupera una película de la base de datos por su nombre. Abre una conexión, prepara una sentencia SQL SELECT con una cláusula WHERE, establece el parámetro de la sentencia al nombre proporcionado, ejecuta la sentencia, crea un objeto de película a partir del primer resultado (si lo hay), y finalmente cierra la conexión.
- **borrarPelicula(Pelicula pelicula):** Este método elimina una película de la base de datos. Abre una conexión, prepara una sentencia SQL DELETE, establece el parámetro de la sentencia al nombre del objeto película, ejecuta la sentencia y finalmente cierra la conexión.



- `modificarPelicula(String nombreActual, Pelicula nuevaPelicula)`: Este método actualiza una película en la base de datos. Abre una conexión, prepara una sentencia SQL UPDATE, establece los parámetros de la sentencia a las propiedades del nuevo objeto película y al nombre actual, ejecuta la sentencia y finalmente cierra la conexión.

Una vez creado todos los métodos que nos permiten trabajar con el modelo, vamos a pasar a ver la gestión de la vista con el controlador.

Vista:



Para saber que acción estamos llevando a cabo, ya sea crear película, borrarla, modificarla o consultar hemos creado dentro de la clase.jsp unos inputs ocultos que nos permiten saber en cada momento que acción estamos llevando a cabo:

```
<!-- Campos que esta ocultos para saber que accion esta realizando el servlet y no crear un
servlet exclusivo para cada accion de

    borrar de insertar, modificar o mostrar contenido-->

<input type="hidden" name="accion" value="crear">
```

Dentro de cada acción enviamos la información al servlet mediante un doPost

```
<form action="GestionPelicula" method="post">

    <h2>Borrar Película</h2>

    <label>Selecciona una película para borrar:</label>
```

```
<select name="peliculaABorrar">

    <% List<Pelicula> peliculas = new ArrayList<>();

    try {

        peliculas = DatabaseManager.getAllPeliculas();

    } catch (Exception e) {

        e.printStackTrace();

    }

    for (Pelicula pelicula : peliculas) { %>

        <option value="<%= pelicula.getNombre() %>"><%= pelicula.getNombre() %></option>

        <% } %>

    </select><br>

    <!-- Campos que esta ocultos para saber que accion esta realizando el servlet y no
crear un servlet exclusivo para cada accion de

    borrar de insertar, modificar o mostrar contenido-->

    <input type="hidden" name="accion" value="borrar">

    <button type="submit">Borrar Película</button>

</form>
```

Ahora vamos a pasar a ver el servlet:

```
@WebServlet("/GestionPelicula")

public class GestionPelicula extends HttpServlet {

    @Override

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {

        int duracion = 0;

        int año = 0;

        int clasificacionEdad = 0;

        // Obtén el parámetro de acción desde el formulario
```

```
String accion = request.getParameter("accion");

// Realiza diferentes acciones según el valor de 'accion'

if ("crear".equals(accion)) {

    // Obtén los parámetros del formulario de registro

    String nombre = request.getParameter("nombre");

    String sinopsis = request.getParameter("sinopsis");

    String pagina_oficial = request.getParameter("paginaOficial");

    String titulo_original = request.getParameter("tituloOriginal");

    String genero = request.getParameter("genero");

    String nacionalidad = request.getParameter("nacionalidad");

    try {

        duracion = Integer.parseInt(request.getParameter("duracion"));

    } catch (NumberFormatException e) {

        System.out.println(e);

    }

    try {

        año = Integer.parseInt(request.getParameter("anho"));

    } catch (NumberFormatException e) {

        System.out.println(e);

    }

    String distribuidora = request.getParameter("distribuidora");

    String director = request.getParameter("director");

    try {

        clasificacionEdad = Integer.parseInt(request.getParameter("clasificacionEdad"));

    } catch (NumberFormatException e) {

        System.out.println(e);

    }

}
```

```
String otrosDatos = request.getParameter("otrosDatos");

String actores = request.getParameter("actores");

String url_image = request.getParameter("imagen");

String url_video = request.getParameter("video");

try {

    Pelicula pelicula = new Pelicula(nombre, sinopsis, pagina_oficial, titulo_original, genero,
nacionalidad, duracion, año, distribuidora, director, clasificacionEdad, otrosDatos, actores, url_image,
url_video);

    // Guardar el usuario en la base de datos

    DatabaseManager.getInstance().guardarPelicula(pelicula);

    response.sendRedirect("gestionPelículas.jsp"); // Redirigir a la página principal

} catch (SQLException e) {

    response.getWriter().println("Error al crear la película en el servlet.");

}

} else if ("borrar".equals(accion)) {

    // Obten el nombre de la película a borrar desde la solicitud

    String nombrePeliculaABorrar = request.getParameter("peliculaABorrar");

    try {

        // Obtén la película por su nombre

        Pelicula pelicula = DatabaseManager.getPeliculaPorNombre(nombrePeliculaABorrar);

        if (pelicula != null) {

            // Borra la película de la base de datos

            DatabaseManager.getInstance().borrarPelicula(pelicula);

            response.sendRedirect("gestionPelículas.jsp"); // Redirigir a la página principal

            System.out.println("Película borrada con éxito");

        } else {

            response.getWriter().println("No se encontró la película a borrar.");

        }

    } catch (SQLException e) {
```

```
e.printStackTrace();

response.getWriter().println("Error al borrar la película.");

}

} else if ("modificar".equals(accion)) {

// Obtén el nombre de la película a modificar desde la solicitud

String nombrePeliculaAModificar = request.getParameter("peliculaAModificar");

try {

    // Obtén la película por su nombre

    Pelicula pelicula =
DatabaseManager.getInstance().getPeliculaPorNombre(nombrePeliculaAModificar);

    System.out.println(nombrePeliculaAModificar);

    if (pelicula != null) {

        // Modifica la película con los nuevos valores

        pelicula.setNombre(request.getParameter("nuevoNombre"));

        pelicula.setSinopsis(request.getParameter("nuevaSinopsis"));

        pelicula.setPaginaOficial(request.getParameter("nuevaPaginaOficial"));

        pelicula.setTituloOriginal(request.getParameter("nuevoTituloOriginal"));

        pelicula.setGenero(request.getParameter("nuevoGenero"));

        pelicula.setNacionalidad(request.getParameter("nuevaNacionalidad"));

        pelicula.setDuracion(Integer.parseInt(request.getParameter("nuevaDuracion")));

        pelicula.setAño(Integer.parseInt(request.getParameter("nuevoAño")));

        pelicula.setDistribuidora(request.getParameter("nuevaDistribuidora"));

        pelicula.setDirector(request.getParameter("nuevoDirector"));

        pelicula.setClasificacionEdad(Integer.parseInt(request.getParameter("nuevaClasificacionEda
d"))));

        pelicula.setOtrosDatos(request.getParameter("nuevosDatos"));

        pelicula.setActores(request.getParameter("nuevosActores"));

        pelicula.setUrl_image(request.getParameter("nuevaImagen"));

        pelicula.setUrl_video(request.getParameter("nuevoVideo"));
```

```
// Guarda la película modificada en la base de datos

DatabaseManager.modificarPelicula(nombrePeliculaAModificar, pelicula);

response.sendRedirect("gestionPeliculas.jsp");

} else {

    response.getWriter().println("No se encontró la película a modificar.");

}

} catch (SQLException e) {

    e.printStackTrace();

    response.getWriter().println("Error al modificar la película.");

}

} else if ("Consultar".equals(accion)) {

    String nombrePeliculaAConsultar = request.getParameter("peliculaAConsultar");

    try {

        // Obtén la película por su nombre

        Pelicula pelicula = DatabaseManager.getPeliculaPorNombre(nombrePeliculaAConsultar);

        if (pelicula != null) {

            // Setea los atributos de la película en el request para que puedan ser accesibles en el
JSP

            request.setAttribute("nombreConsultar", pelicula.getNombre());

            request.setAttribute("sinopsisConsultar", pelicula.getSinopsis());

            request.setAttribute("paginaOficialConsultar", pelicula.getPaginaOficial());

            request.setAttribute("tituloOriginalConsultar", pelicula.getTituloOriginal());

            request.setAttribute("generoConsultar", pelicula.getGenero());

            request.setAttribute("nacionalidadConsultar", pelicula.getNacionalidad());

            request.setAttribute("duracionConsultar", pelicula.getDuracion());

            request.setAttribute("AnhoConsultar", pelicula.getAño());

            request.setAttribute("distribuidoraConsultar", pelicula.getDistribuidora());

            request.setAttribute("directorConsultar", pelicula.getDirector());
```

```
request.setAttribute("clasificacionEdadConsultar", pelicula.getClasificacionEdad());

request.setAttribute("datosConsultar", pelicula.getOtrosDatos());

request.setAttribute("actoresConsultar", pelicula.getActores());

request.setAttribute("ImagenConsultar", pelicula.getUrl_image());

request.setAttribute("VideoConsultar", pelicula.getUrl_video());


// Redirige a la página del formulario con los campos ya poblados

request.getRequestDispatcher("gestionPelículas.jsp").forward(request, response);

} else {

    response.getWriter().println("No se encontró la película a Consultar.");

}

} catch (SQLException e) {

    e.printStackTrace();

    response.getWriter().println("Error al consultar la película.");

}

} else {

    response.getWriter().println("Acción no reconocida");

}

}

}
```

Este código es un servlet en Java que maneja las solicitudes HTTP POST para la gestión de películas. El servlet se mapea a la URL `/GestionPelícula` mediante la anotación `@WebServlet`.

El método `doPost(HttpServletRequest request, HttpServletResponse response)` se sobrescribe para manejar las solicitudes POST. Este método se invoca automáticamente cuando el servidor recibe una solicitud POST para la URL `/GestionPelícula`.

Dentro del método `doPost`, se obtiene el parámetro de acción desde la solicitud. Este parámetro determina qué acción se debe realizar: crear, borrar, modificar o consultar una película.

Si la acción es "crear", se obtienen todos los parámetros necesarios para crear una nueva película desde la solicitud. Estos parámetros se utilizan para crear un nuevo objeto `Película`, que se guarda en la base de datos utilizando el método `guardarPelícula` del `DatabaseManager`. Si ocurre un error al guardar la película, se envía un mensaje de error al cliente.

Si la acción es "borrar", se obtiene el nombre de la película a borrar desde la solicitud. Se busca la película en la base de datos utilizando el método `getPelículaPorNombre` del `DatabaseManager`. Si la película existe, se borra de la base de datos utilizando el método `borrarPelícula` del `DatabaseManager`. Si la película no existe, se envía un mensaje de error al cliente.

Si la acción es "modificar", se obtiene el nombre de la película a modificar desde la solicitud. Se busca la película en la base de datos utilizando el método `getPelículaPorNombre` del `DatabaseManager`. Si la película existe, se modifican sus atributos con los nuevos valores obtenidos desde la solicitud, y se guarda la película modificada en la base de datos utilizando el método `modificarPelícula` del `DatabaseManager`. Si la película no existe, se envía un mensaje de error al cliente.

Si la acción es "Consultar", se obtiene el nombre de la película a consultar desde la solicitud. Se busca la película en la base de datos utilizando el método `getPelículaPorNombre` del `DatabaseManager`. Si la película existe, se establecen sus atributos en el objeto `request` para que puedan ser accesibles en la página JSP, y se redirige a la página del formulario con los campos ya rellenos. Si la película no existe, se envía un mensaje de error al cliente.

Si la acción proporcionada en la solicitud no es ninguna de las anteriores, se envía un mensaje de error al cliente diciendo "Acción no reconocida".

De esta forma gestionamos las películas con el patrón MVC, y de la misma forma lo hacemos con las salas, las entradas y las reservas, creamos los 5 métodos en la clase `databaseManager.java`, luego en el jsp enviamos los datos al servlet reconociendo la acción que hemos realizado y ya dentro del servlet recogemos y almacenamos esos datos.

Visto el patrón MVC para la gestión del cine, vamos a ver como funciona este patrón para la vista de usuario.

Al igual que con el administrador, la clase que se encarga de gestionar todo el modelo es `DatabaseManager.java`. Por lo que omitiremos su explicación y veremos el controlador y la vista para generar la experiencia de usuario.

Primero explicare el controlador, es decir los servlets empleados para producir esta lógica:

- `Reservas.java`:

```
package com.example.servlets;

import com.example.model.DatabaseManager;
import com.example.model.Sala;
import java.io.IOException;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.sql.SQLException;
import java.util.logging.Level;
```



```
import java.util.logging.Logger;

@WebServlet(name = "Reservas", urlPatterns = {"/Reservas"})
public class Reservas extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        HttpSession session = request.getSession();

        // Obtener los parámetros del formulario
        String peliculaId = request.getParameter("peliculaSeleccionada");
        String fecha = request.getParameter("fecha");
        String hora = request.getParameter("hora");

        String sala = request.getParameter("salaSeleccionada");
        System.out.println(sala);
        try {
            Sala salaSelec = DatabaseManager.getInstance().getSalaPorNombre(sala);
            System.out.println(salaSelec.getNombreSala() + "EN EL SERVLET");
            // Almacenar los datos en la sesión
            session.setAttribute("peliculaId", peliculaId);
            session.setAttribute("hora", hora);
            session.setAttribute("fecha", fecha);
            session.setAttribute("sala", salaSelec);
            // Redireccionar a butacas.jsp

            response.sendRedirect(request.getContextPath() + "/butacas.jsp");
        } catch (SQLException ex) {
            Logger.getLogger(Reservas.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

Solicita al DatabaseManager la sala para obtener los datos respectivos a la sala, filas y columnas, necesarios para mostrar en butacas.jsp la distribución de estas.

Luego tenemos GestionButacas.java, que se encarga de añadir a la sesión los datos de las butacas seleccionadas y redireccionar para continuar con el pago:

```
package com.example.servlets;

import com.example.model.Sala;

import java.io.IOException;

import jakarta.servlet.ServletException;
```

```
import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.servlet.http.HttpSession;

@WebServlet("/GestionButacas")

public class GestionButacas extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

        HttpSession session = request.getSession();

        Sala sala = (Sala) session.getAttribute("sala");

        // Obtener la cadena JSON de la solicitud

        String butacasSeleccionadasJSON = request.getParameter("butacasSeleccionadas");

        session.setAttribute("butacasSeleccionadas",butacasSeleccionadasJSON);

        response.sendRedirect(request.getContextPath() + "/confirmarReserva.jsp");

    }

}
```

El último servlet empleado es el de ProcesarPago.java, este se encarga de validar la tarjeta, y de crear las entradas asociadas a su reserva, que termina redireccionando a PagoExitoso.jsp para ver los datos de la reserva. El código queda:

```
package com.example.servlets;

import com.example.model.DatabaseManager;

import com.example.model.Entrada;

import com.example.model.Fecha;

import com.example.model.Reserva;

import com.example.model.Sala;

import com.example.model.TarjetaCredito;
```

```
import com.example.model.Usuario;

import java.io.IOException;

import java.io.PrintWriter;


import java.sql.SQLException;


import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.time.LocalDateTime;
import java.util.Random;
import java.util.logging.Level;
import java.util.logging.Logger;


@WebServlet("/ProcesarPagoServlet")
public class ProcesarPagoServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;


    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        response.setContentType("text/html;charset=UTF-8");

        PrintWriter out = response.getWriter();


        try {

            // Obtener los parámetros del formulario

            String numeroTarjetaConEspacios = request.getParameter("numeroT");
```

```
String numeroTarjeta = numeroTarjetaConEspacios.replace(" ", "");

String nombre_titular = request.getParameter("titular");
String fechaExpiracion = request.getParameter("fechaCaducidad");

String codigoSeguridad = request.getParameter("codigoSeguridad");

// Obtener la información del usuario de la sesión
HttpSession session = request.getSession();
Usuario usuarioActual = (Usuario) session.getAttribute("usuario");
String emailUsuario = usuarioActual.getCorreo();

TarjetaCredito tarjeta = new TarjetaCredito(numeroTarjeta, nombre_titular,
fechaExpiracion, codigoSeguridad, emailUsuario);

DatabaseManager.getInstance().guardarTarjeta(tarjeta);

// Validar la tarjeta en la base de datos
if (DatabaseManager.getInstance().validarTarjeta(emailUsuario, numeroTarjeta,
fechaExpiracion, codigoSeguridad)) {
    // Los datos de la tarjeta son válidos

String numRef = "";
int numeroAleatorioRef = new Random().nextInt(900000000) + 1000000000;
numRef = "R" + Integer.toString(numeroAleatorioRef); // Crear una nueva entrada

session.setAttribute("numRef", numRef);

//CREACION DE RESERVA
Sala sala = (Sala) session.getAttribute("sala");
```

```
// Obtener la cadena JSON de la solicitud

String butacasSeleccionadasJSON = (String)
session.getAttribute("butacasSeleccionadas");

// Eliminar corchetes y comillas para obtener pares clave-valor separados por comas
String cleanString = butacasSeleccionadasJSON.replaceAll("[\\[\\]\\{\\}\\\" ]", "");

System.out.println(cleanString);

// Dividir la cadena en pares clave-valor
String[] keyValuePairs = cleanString.split(",");

//IdEntrada para la entradas seleccionadas por el mismo usuario.
String idEntrada = "";
int numeroAleatorio = new Random().nextInt(900000000) + 100000000;
idEntrada = "E" + Integer.toString(numeroAleatorio);// Crear una nueva entrada

// Obtener información adicional de la sesión
String fechaStr = (String) session.getAttribute("fecha");
Fecha fecha = new Fecha(fechaStr);

String horaStr = (String) session.getAttribute("hora");
LocalTime hora = LocalTime.parse(horaStr);

int fila = 0;
int columna = 0;

String nombreSala = sala.getNombreSala();

for (int i = 0; i < keyValuePairs.length; i += 2) {
    // Obtener la fila y columna
```

```
//Extraemos la fila
String[] filaKC = keyValuePairs[i].split(":");
fila = Integer.parseInt(filaKC[1]);

//Extraemos la columna
String[] columnaKC = keyValuePairs[i + 1].split(":");
columna = Integer.parseInt(columnaKC[1]);

Entrada entrada = new Entrada(idEntrada, fecha, hora, fila, columna, nombreSala);
Reserva reserva = new Reserva(numRef, emailUsuario, idEntrada, fila, columna);

try {
    // Guardar la entrada en la base de datos
    DatabaseManager.getInstance().guardarEntrada(entrada);
    DatabaseManager.getInstance().guardarReserva(reserva);

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

response.sendRedirect(request.getContextPath() + "/PagoExitoso.jsp");

} else {
    // Los datos de la tarjeta no son válidos
    out.println("<h2>Error en el Pago: Datos de Tarjeta Incorrectos</h2>");

}

} catch (Exception e) {
    e.printStackTrace();
    out.println("<h2>Error en el Pago</h2>");
}
```

```
}
```

```
}
```

Visto cómo funciona nuestro Controlador, podemos ver lo que ve el usuario gracias a esto. Usaremos, al igual que antes, páginas .jsp.

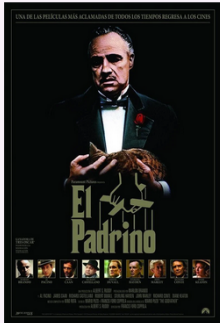
Para la vista tenemos: reserva.jsp, butacas.jsp y PagoExitoso.jsp

Reserva.jsp tiene una apariencia al público como sigue:

VOLVER AL MENU DE PELICULAS
REGISTRARSE

## EL PADRINO

RESERVA



Fecha:

Hora:

Sala:

Seleccionar Butacas

El código empleado para mostrar esto es:

```
<%@ page contentType="text/html" pageEncoding="UTF-8"%>
<%@ page import="java.util.List" %>
<%@ page import="com.example.model.Pelicula" %>
<%@ page import="com.example.model.Sala" %>
<%@ page import="com.example.model.DatabaseManager" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.text.SimpleDateFormat" %>
<%@ page import="java.util.Calendar" %>
<%@ page import="java.util.Date" %>
```

```
<!DOCTYPE html>

<html>

  <head>

    <title>Reserva</title>

    <meta charset="utf-8">

    <link href="estilos/bootstrap.css" rel="stylesheet">
    <link href="estilos/style.css" rel="stylesheet">
    <link href="estilos/owl.carousel.css" rel="stylesheet">

    <link href="estilos/fontawesome.min.css" rel="stylesheet"/>

    <link href="estilos/multicines.css" rel="stylesheet" />
  </head>

  <body>

    <div id="wrapper" class="d-flex flex-column">

      <div class="header">

        <nav class="navbar fixed-top navbar-site navbar-light bg-light navbar-expand-lg"
role="navigation" >

          <div class="container">

            <div class="navbar-identity">

              <button data-target=".navbar-collapse" data-toggle="collapse" class="navbar-toggler
pull-right" type="button">

                <svg xmlns="https://www.w3.org/2000/svg" viewBox="0 0 30 30" width="30"
height="30" focusable="false"><title>Menu</title><path stroke="currentColor" stroke-width="2" stroke-
linecap="round" stroke-miterlimit="10" d="M4 7h22M4 15h22M4 23h22"/></svg>

              </button>

            </div>

            <div class="navbar-collapse collapse" style="height: auto;">

              <ul class="nav navbar-nav ml-auto navbar-right">
```



```
<li class="nav-item"><a href="index.jsp" class="nav-link">VOLVER AL MENU DE
PELICULAS</a></li>

<li class="nav-item"><a href="login.jsp" class="nav-link">REGISTRARSE</a></li>

</ul>

</div>

<!-- /.nav-collapse -->

</div>

<!-- /.container-fluid -->

</nav>

</div>

<!-- /.header -->

<%
Película películaSeleccionada = (Película) session.getAttribute("película");
%>

<div class="main-container inner-page flex-fill " style="padding-top: 30px !important;">

<div style="padding-top: 30px" class="container">

<div align="center" class="row clearfix">

<h1 class="text-center title-1"><%= películaSeleccionada.getNombre() %></h1>

<hr class="mx-auto small text-hr" style="margin-bottom: 30px !important">

<div style="clear:both">

<hr>

</div>

</div>

<h3 class="titulo-cine">RESERVA</h3>

<div class="col-3 d-none d-md-block">

<a href="<%= películaSeleccionada.getUrl_image() %>" data-toggle="lightbox"></a>

</div>

<div class="formreservas">
```

```
<form action="Reservas" method="post" onsubmit="return validarFormulario()" >

<label for="fecha">Fecha:</label>

<input type="date" name="fecha" id="fecha" required>

<br>

<label for="hora">Hora:</label>

<select name="hora" id="hora" required></select>

<br>

<br>

<label for="sala">Sala:</label>

<select name="salaSeleccionada" id="salaSeleccionada">

    <% List<Sala> salas = new ArrayList<>();

    try {

        salas = DatabaseManager.getAllSalas(); // Obtener todas las salas

    } catch (Exception e) {

        e.printStackTrace();

    }

    %>

    <option value="" disabled selected>Selecciona una sala</option>

    <%

        for (Sala sala : salas) {

            if (sala.getNombre_pelicula().equals(peliculaSeleccionada.getNombre())) {

                %>

                <option value="<%= sala.getNombreSala() %>"><%= sala.getNombreSala() %></option>

                <%

                    }

                }

            %>

        }

    %>

</select>

<br>
```

```
<br>

<button type="submit">Seleccionar Butacas</button>

</form>

</div>

</div>

</div>

</div>

</div>

<script>

    // Obtener la duración de la película y el tiempo entre pases desde el servidor
    var duracionPelicula = <%= peliculaSeleccionada.getDuracion() %>;

    var tiempoEntrePases = 15; // minutos

    // Establecer la hora de inicio y fin del cine
    var horaInicioCine = new Date();

    horaInicioCine.setHours(14, 0, 0, 0);

    var horaFinCine = new Date();

    horaFinCine.setHours(23, 59, 0, 0);

    // Calcular los horarios de los pases redondeando a las horas enteras
    var horarios = [];

    var sdf = new Intl.DateTimeFormat('en-US', {hour: 'numeric', minute: 'numeric', hour12: false});

    // Calcular el número de pases en el día
    var numPases = Math.floor((9 * 60 / duracionPelicula));

    for (var i = 0; i < numPases; i++) {

        // Formatear la hora
        var horaRedondeada = new Date(horaInicioCine);

        horaRedondeada.setMinutes(Math.round(horaRedondeada.getMinutes() / 15) * 15);

        // Verificar si la hora está dentro del horario del cine
        if (horaRedondeada >= horaInicioCine && horaRedondeada <= horaFinCine) {
```

```
        horarios.push(sdf.format(horaRedondeada));
    }

    // Calcular el siguiente pase
    horalInicioCine.setMinutes(horalInicioCine.getMinutes() + duracionPelicula + tiempoEntrePases);
}

// Limpiar la lista de horarios
var selectHora = document.getElementById("hora");
selectHora.innerHTML = "";

// Añadir opciones a la lista de horarios
for (var i = 0; i < horarios.length; i++) {
    var option = document.createElement("option");
    option.value = horarios[i];
    option.textContent = horarios[i];

    // Agregar la opción a la lista de horarios
    selectHora.appendChild(option);
}

function validarFormulario() {
    // Validar la fecha
    var fechaInput = document.getElementById("fecha").value;
    var fechaSeleccionada = new Date(fechaInput);
    var fechaActual = new Date();

    if (fechaSeleccionada < fechaActual) {
        alert("La fecha no puede ser menor que el día actual");
        return false;
    }

    // El formulario es válido
    return true;
}
```

```
</script>

</body>

</html>
```

La principal interacción que hay en el modelo son con los inputs, estos al rellenar el formulario, mediante el método POST, envía los datos al servlet para gestionar la información. Por otro lado, el JavaScript se encarga de generar las horas en función de las horas límite y comprobar las entradas de los inputs.

Al pulsar el botón seleccionar butacas, nos redirecciona al servlet, enviando los datos del formulario y que dará paso a butacas.jsp con los datos establecidos en el servlet.



Se crean butacas en función de las filas y columnas que tenga la sala, apareciendo en azul, y , en caso de que ya estuvieran seleccionadas en rojo. En verde aparecen las ya seleccionables. Las butacas seleccionadas se almacenan en un archivo JSON, que será enviado al servlet GestionButacas, para meterlo en una sesión y poder usarlo a la hora de generar entradas.

```
<%@ page contentType="text/html; charset=UTF-8"%>

<%@ page import="java.util.List" %>

<%@ page import="com.example.model.Sala" %>

<%@ page import="com.example.model.Entrada" %>

<%@ page import="com.example.model.Pelicula" %>

<%@ page import="com.example.model.DatabaseManager" %>
```

```
<%@ page import="com.example.model.Fecha" %>

<%@ page import="java.time.LocalTime" %>

<%@ page import="java.util.ArrayList" %>

<%

    List<Entrada> entradas = new ArrayList<>();
try {
    Sala salaSelec = (Sala) session.getAttribute("sala");
    String fechaStr = (String) session.getAttribute("fecha");

    Fecha fecha = new Fecha(fechaStr);
    System.out.println(fecha);

    entradas = DatabaseManager.getAllEntradas();

    String horaStr = (String) session.getAttribute("hora");
    LocalTime hora = LocalTime.parse(horaStr);

    Pelicula peliculaSeleccionada = (Pelicula) session.getAttribute("pelicula");

%>
<!DOCTYPE html>
<html>

    <head>

        <meta charset="UTF-8">

        <title>Selección de Butacas</title>

        <script src="https://code.jquery.com/jquery-3.6.4.min.js"></script>

        <script async src="https://www.googletagmanager.com/gtag/js?id=UA-155274620-1"></script>

        <link href="estilos/bootstrap.css" rel="stylesheet">

        <link href="estilos/style.css" rel="stylesheet">

        <link href="estilos/owl.carousel.css" rel="stylesheet">
```

```
<link href="estilos/fontawesome.min.css" rel="stylesheet"/>

<link href="estilos/multicines.css" rel="stylesheet" />
</head>
<body>
  <div id="wrapper" class="d-flex flex-column">
    <div class="header">
      <nav class="navbar fixed-top navbar-site navbar-light bg-light navbar-expand-lg"
role="navigation" >
        <div class="container">
          <div class="navbar-identity">

            <button data-target=".navbar-collapse" data-toggle="collapse" class="navbar-toggler pull-
right" type="button">
              <svg xmlns="https://www.w3.org/2000/svg" viewBox="0 0 30 30" width="30"
height="30" focusable="false"><title>Menu</title><path stroke="currentColor" stroke-width="2" stroke-
linecap="round" stroke-miterlimit="10" d="M4 7h22M4 15h22M4 23h22"/></svg>
            </button>
          </div>
          <div class="navbar-collapse collapse" style="height: auto;">
            <ul class="nav navbar-nav ml-auto navbar-right">
              <li class="nav-item"><a href="index.jsp" class="nav-link">VOLVER AL MENU DE
PELICULAS</a></li>
              <li class="nav-item"><a href="login.jsp" class="nav-link">REGISTRARSE</a></li>
            </ul>
          </div>
        </div>
      </nav>
    </div>
    <div align="center" style="padding-top: 30px" class="formreservas">
      <div class="row clearfix">
        <h1 class="text-center title-1"><%= peliculaSeleccionada.getNombre() %></h1>
        <hr class="mx-auto small text-hr" style="margin-bottom: 30px !important">
      </div>
    </div>
  </div>
</body>
</html>
```

```
<div style="clear:both">

    <hr>

</div>

</div>

<h3 class="titulo-cine">SELECCION BUTACAS: <%= salaSelec.getNombreSala() %></h3>

<div class = "tablabutaca">

<form action="GestionButacas" method="post">

    <table style='border-collapse: collapse;'>

        <%

        for (int i = 0; i < salaSelec.getFilas(); i++) {

            %>

            <tr>

                <%

                for (int j = 0; j < salaSelec.getColumnas(); j++) {

                    int fila = i + 1;

                    int columna = j + 1;

                    boolean ocupadaPorOtroCliente = false;

                    // Verificar si la butaca está ocupada por otras entradas

                    for (Entrada entrada : entradas) {

                        System.out.println(entrada.getFecha().equals(fecha));

                        if (entrada.getNombreSala().equals(salaSelec.getNombreSala()) &&

                            entrada.getFila() == fila &&

                            entrada.getColumna() == columna &&

                            entrada.getFecha().toLocalDate().equals(fecha.toLocalDate()) &&

                            entrada.getHora().equals(hora))

                            {

                                ocupadaPorOtroCliente = true;

                                break;

                            }

                        }

                    }

                }

            }

        }

    }

}
```



```
%>

<td>

  <%

    if (!ocupadaPorOtroCliente) {

      // Verificar si la butaca está libre u ocupada

      boolean ocupada = false;

      if (ocupada) {

        //Si esta ocupada , establecer imagen butaca_seleccionada

      %>

      <img class='butaca ocupada'

        src='butacas/butaca_seleccionada.png'

        alt='Butaca <%= fila %>-<%= columna %>'

        data-fila='<%= fila %>' data-columna='<%= columna %>'>

      <%

    } else {

      //Si esta libre , establecer imagen butaca_libre

      %>

      <img class='butaca'

        src='butacas/butaca_libre.png'

        alt='Butaca <%= fila %>-<%= columna %>'

        data-fila='<%= fila %>' data-columna='<%= columna %>'>

      <%

    }

  } else {

    //Si esta ocupada previamente, establecer imagen butaca_ocupada

    %>

    <img class='butaca ocupadaPorClientes'

      src='butacas/butaca_ocupada.png'

      alt='Butaca <%= fila %>-<%= columna %>'

      data-fila='<%= fila %>' data-columna='<%= columna %>'>

    <%
```

```
        }
        %>

    </td>

    <%
    }
    }
    %>
</tr>

</table>

<!-- Campo oculto para almacenar butacas seleccionadas -->
<input type="hidden" name="butacasSeleccionadas" id="butacasSeleccionadas">

    <button type="submit" name="comprarButacas">Comprar</button>
</form>
</div>
</div>
</div>
<script>
$(document).ready(function () {
    var butacasSeleccionadas = [];

    // Manejar el clic en una butaca
    $('.butaca').click(function () {
        var fila = $(this).data('fila');
        var columna = $(this).data('columna');

        // Verificar si la butaca está ocupada por otros clientes
        var ocupadaPorClientes = $(this).hasClass('ocupadaPorClientes');

        if (!ocupadaPorClientes) {
```

```
// Verificar si la butaca está libre u ocupada
var ocupada = $(this).hasClass('ocupada');

if (!ocupada) {
    // Si la butaca está libre, cambiar a ocupada
    $(this).attr('src', 'butacas/butaca_seleccionada.png');
    $(this).addClass('ocupada');

    // Agregar la butaca a la lista de seleccionadas
    butacasSeleccionadas.push({fila: fila, columna: columna});
} else {
    // Si la butaca está ocupada, cambiar a libre
    $(this).attr('src', 'butacas/butaca_libre.png');
    $(this).removeClass('ocupada');

    // Eliminar la butaca de la lista de seleccionadas
    butacasSeleccionadas = butacasSeleccionadas.filter(function (butaca) {
        return butaca.fila !== fila || butaca.columna !== columna;
    });
}

// Actualizar el campo oculto con las butacas seleccionadas
$('#butacasSeleccionadas').val(JSON.stringify(butacasSeleccionadas));
});
});
</script>
</body>

</html>
<%
} catch (Exception e) {
    e.printStackTrace();
}
%>
```

Al hacer clic en el botón comprar, las butacas seleccionadas se envían al servlet, que lo almacenará en la sesión y redireccionará para introducir la tarjeta. Como la tarjeta ya ha sido explicada previamente, voy a proceder, una vez aceptada la tarjeta, a explicar PagoExitoso.jsp.

Este se ve:

## Pago Exitoso


Su número de Referencia es: R154822734

### Entradas Reservadas:

Fila: 1, Columna: 3  
Fila: 1, Columna: 4  
Fila: 1, Columna: 5

Día: 2024-01-31  
Hora: 14:00

### Código QR:



[Imprimir PDF](#) [Seguir Comprando](#)

Apareciendo el resumen con los datos seleccionados y la referencia de la reserva, además, para al llegar al cine verificar, se incluye la generación de un código QR para comprobar los datos (por el momento solo almacena los datos de la reserva). Para después si se desea, imprimir en pdf la pagina web, para, en caso de no disponer de internet se pueda mantener en el dispositivo.

El código que se encarga de esto es el siguiente:

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="java.util.Map" %>
<%@ page import="java.util.ArrayList" %>
<%@ page import="java.util.Arrays" %>
<!DOCTYPE html>
```

```
<html>

<head>

    <title>Pago Exitoso</title>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/html2pdf.js/0.10.1/html2pdf.bundle.min.js"
integrity="sha512-
GsLIZN/3F2ErC5ifS5QtgpiJtWd43JWSulgh7mbzZ8zBps+dvLusV+eNQATqgA/HdeKFVgA5v3S/clrLF7Qnlg==
" crossorigin="anonymous" referrerpolicy="no-referrer"></script>

    <script src="https://cdnjs.cloudflare.com/ajax/libs/qrcodejs/1.0.0/qrcode.min.js"
crossorigin="anonymous" referrerpolicy="no-referrer"></script>

    <link rel="stylesheet" href="estilos/pagoexitoso.css">
</head>

<body>

    <h2>Pago Exitoso</h2>

    <p>Su número de Referencia es: <%= session.getAttribute("numRef") %></p>

    <h3>Entradas Reservadas:</h3>

    <%-- Mostrar información de cada entrada --%>

    <% String butacasJSON = (String) session.getAttribute("butacasSeleccionadas");

    // Eliminar corchetes y comillas para obtener pares clave-valor separados por comas
    String cleanString = butacasJSON.replaceAll("[\\[\\]\\{\\}\\\"' ]", "");

    // Dividir la cadena en pares clave-valor
    String[] keyValuePairs = cleanString.split(",");

    // Crear una lista de mapas para almacenar las butacas seleccionadas
    List<Map<String, Integer>> butacasList = new ArrayList<>();

    // Iterar sobre los pares clave-valor y agregar a la lista de mapas
    for (int i = 0; i < keyValuePairs.length; i += 2) {

        String[] filaKC = keyValuePairs[i].split(":");

        String[] columnaKC = keyValuePairs[i + 1].split(":");
```

```
        Map<String, Integer> butaca = Map.of("fila", Integer.parseInt(filaKC[1]), "columna",
Integer.parseInt(columnaKC[1]));

        butacasList.add(butaca);

    }

%>

<ul>

    <% for (Map<String, Integer> butaca : butacasList) { %>

        <li>Fila: <%= butaca.get("fila") %>, Columna: <%= butaca.get("columna") %></li>

        <% } %>

    </ul>

<p>Día: <%= session.getAttribute("fecha") %></p>

<p>Hora: <%= session.getAttribute("hora") %></p>

<div class="section">

    <!-- Mostrar el código QR generado en el JavaScript -->

    <h3>Código QR:</h3>

    <%

        String datosReserva = "Numero de Referencia: " + session.getAttribute("numRef") +

            "Dia: " + session.getAttribute("fecha") +

            "Hora: " + session.getAttribute("hora") +

            "Butacas Reservadas: " + butacasJSON;

    %>

    <div id="codigoQR"></div>

</div>

<button onclick="imprimirPDF()">Imprimir PDF</button>

<a href="index.jsp"><button type="button">Seguir Comprando</button></a>

<script>

    var qrcode = new QRCode("codigoQR", {
```

```
text: '<%= datosReserva %>',
width: 200,
height: 200,

correctLevel: QRCode.CorrectLevel.H
});

function imprimirPDF() {
  const options = {
    margin: 1,
    filename: 'pago_exitoso.pdf',
    image: {type: 'jpeg', quality: 0.98},
    html2canvas: {
      scale: 2,
      letterRendering: true,
    },
    jsPDF: {unit: 'in', format: 'a4', orientation: 'portrait'},
  };

  const content = document.body;

  html2pdf().from(content).set(options).save();
}

</script>

</body>
</html>
```

Este solo muestra los datos almacenados en la sesión que han sido enviados por el servlet.

### Autenticación, perfiles de usuario y sesiones.

En este caso se ha empleado manejo de sesiones de usuario en todo momento para recoger los datos que sean necesarios, nuestra aplicación esta programada de forma que se puede hacer comentarios solo cuando el usuario esta registrado al igual que reserva de butacas y pago solo

cuando estamos registrados, y para llevar un control de los registros empelamos gestión de sesiones.

Vamos a ver como recogemos datos del usuario para hacer comentarios y para recoger sus datos en la tarjeta de crédito:

```
try {

    if (correo.equals("admin@gmail.com") && contrasena.equals("admin")) {

        // Usuario admin autenticado correctamente

        response.sendRedirect("panelAdmin.jsp");

    } else {

        // Obtener el usuario por correo

        Usuario usuario = DatabaseManager.getInstance().getUsuarioPorCorreo(correo);

        if (usuario != null) {

            if (usuario.getContraseña().equals(contrasena)) {

                HttpSession session = request.getSession();

                session.setAttribute("usuario", usuario);

                // Usuario autenticado correctamente (no admin)

                // Aquí puedes redirigir a una página de bienvenida o realizar otras acciones

                //response.getWriter().println("Acceso autorizado. ¡Bienvenido, " + usuario.getNombre() + "!");

                response.sendRedirect("welcome.html");

            }else{

                response.getWriter().println("Usuario y/o contraseña incorrectos.");

            }

        } else {

            // Usuario no encontrado o contraseña incorrecta

            response.getWriter().println("No hay ningun usuario registrado en el sistema.");

        }

    }

}
```



```
} catch (SQLException e) {  
  
    e.printStackTrace();  
  
    response.getWriter().println("Error al acceder.");  
  
}
```

Primero, verifica si las credenciales proporcionadas corresponden a las del administrador (correo: "admin@gmail.com", contraseña: "admin"). Si es así, redirige al usuario a la página "panelAdmin.jsp".

Si las credenciales no son las del administrador, intenta obtener un usuario de la base de datos que coincida con el correo proporcionado. Esto se hace utilizando el método `getUsuarioPorCorreo` del `DatabaseManager`.

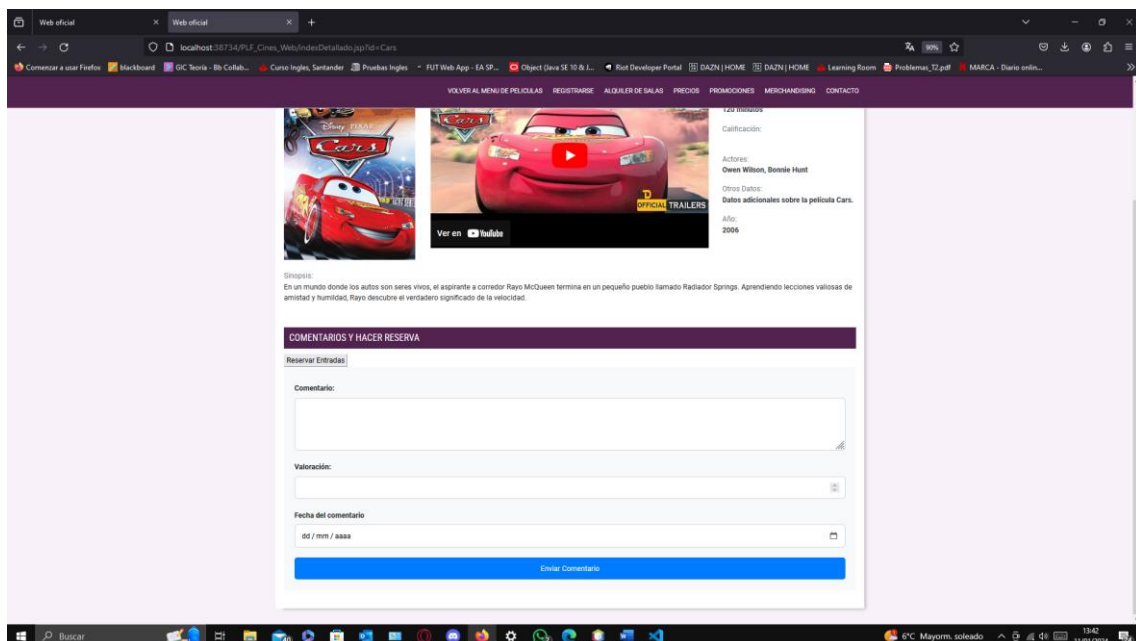
Si se encuentra un usuario con el correo proporcionado, verifica si la contraseña proporcionada coincide con la del usuario. Si es así, inicia una nueva sesión HTTP, guarda el objeto usuario en la sesión y redirige al usuario a la página "welcome.html". Si la contraseña no coincide, envía un mensaje al cliente diciendo "Usuario y/o contraseña incorrectos".

Si no se encuentra un usuario con el correo proporcionado, envía un mensaje al cliente diciendo "No hay ningún usuario registrado en el sistema".

En el momento que se inicia la sesión y se validan los datos creamos la sesión del usuario para recoger sus datos

Si ocurre una excepción `SQLException` durante el proceso, imprime la traza de la pila y envía un mensaje al cliente diciendo "Error al acceder".

Así se vería el panel para poder hacer un comentario y una reserva, pero solo cuando se ha iniciado sesión:



```
<% if (session.getAttribute("usuario") != null) { %>
```

```
<%  
  
    // Obtener el usuario de la sesión  
  
    Usuario usuario = (Usuario) session.getAttribute("usuario");  
  
    String correoUsuario = (usuario != null) ? usuario.getCorreo() : "";  
  
    %>  
  
    <a href="reserva.jsp"><button type="button">Reservar  
Entradas</button></a>  
  
    <div class="comentarios">  
  
        <!-- Sección para hacer comentario -->  
  
        <form action="GestionComentarios" method="post">  
  
            <!-- Otros campos del formulario -->  
  
            <label for="textoComentario">Comentario:</label>  
  
            <textarea id="textoComentario" name="textoComentario"  
required></textarea>  
  
            <label for="valoracionComentario">Valoración:</label>  
  
            <input type="number" id="valoracionComentario"  
name="valoracionComentario" required>  
  
            <label for="fecha">Fecha del comentario</label>  
  
            <input type="date" id="fecha" name="fecha" required><br>  
  
            <input type="hidden" name="accion" value="escribirComentario">  
  
            <input type="hidden" name="emailDelUsuario" value="<%=  
correoUsuario %>">  
  
            <input type="hidden" name="nombrePelicula" value="<%=  
nombrePelicula %>">  
  
            <button type="submit">Enviar Comentario</button>
```

```
</form>

</div>

<% } %>

<%
```

Primero, verifica si hay un atributo "usuario" en la sesión HTTP. Si existe, significa que el usuario está autenticado.

Luego, obtiene el objeto "usuario" de la sesión y extrae el correo del usuario. Si el objeto "usuario" es nulo, el correo del usuario se establece en una cadena vacía.

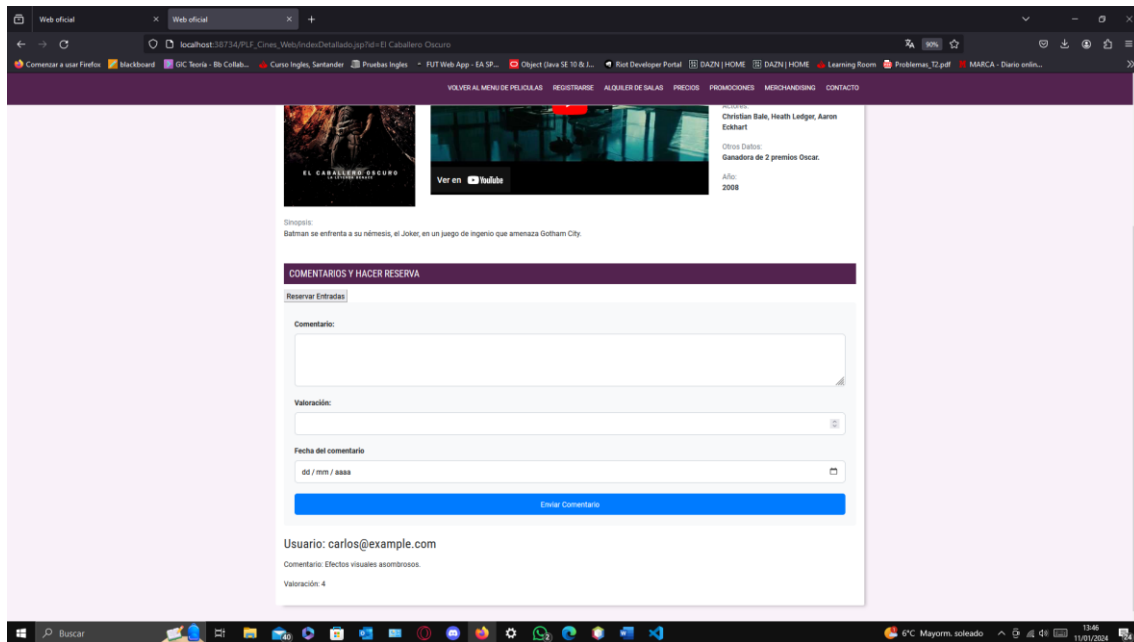
Después de eso, muestra un botón "Reservar Entradas" que, cuando se hace clic, redirige al usuario a la página "reserva.jsp".

A continuación, muestra un formulario para que el usuario pueda escribir un comentario. El formulario incluye campos para el texto del comentario, la valoración del comentario y la fecha del comentario. Todos estos campos son obligatorios, como se indica con el atributo "required".

El formulario también incluye tres campos ocultos. El campo "accion" se establece en "escribirComentario", que se utiliza en el servidor para determinar qué acción realizar. El campo "emailDelUsuario" se establece en el correo del usuario, y el campo "nombrePelicula" se establece en el nombre de la película. Estos campos ocultos se utilizan para enviar información adicional al servidor cuando se envía el formulario.

Finalmente, el formulario incluye un botón "Enviar Comentario" que, cuando se hace clic, envía el formulario al servidor para su procesamiento.

Para hacer que se vean los comentarios que los usuarios han ido poniendo sobre esa película usaremos las funciones ya creadas en la base de datos:



```
List<Comentario> comentarios = DatabaseManager.getComentariosPorNombrePelicula(nombrePelicula);

System.out.println("Comentarios obtenidos: " + comentarios); %>

<% if (comentarios != null) { %>
<% for (Comentario comentario : comentarios) { %>

<div>

    <h2>Usuario: <%= comentario.getEmail_user() %></h2>

    <p>Comentario: <%= comentario.getTexto() %></p>

    <p>Valoración: <%= comentario.getValoracion() %></p>

</div>

<% } %>

<% } else { %>

<p>No hay comentarios para esta película.</p>

<% } %>
```

Primero, se obtienen los comentarios de la base de datos utilizando el método `getComentariosPorNombrePelicula` del `DatabaseManager`. Este método devuelve una lista de objetos `Comentario` que corresponden a la película cuyo nombre se pasa como argumento. Luego, se imprime en la consola del servidor el número de comentarios obtenidos.

Después, se verifica si la lista de comentarios no es nula. Si no es nula, se recorre la lista de comentarios y se muestra cada comentario en un `div`. Para cada comentario, se muestra el correo

electrónico del usuario que hizo el comentario, el texto del comentario y la valoración del comentario.

Si la lista de comentarios es nula, se muestra un mensaje diciendo "No hay comentarios para esta película".

Falta comentar las sesiones empleadas en la reserva, estas se han usado para mostrar, en la vista, los datos que ha ido siendo seleccionados por el usuario. De manera que tenemos:

- Butacas.jsp:

```
<%
    List<Entrada> entradas = new ArrayList<>();

    try {
        Sala salaSelec = (Sala) session.getAttribute("sala");
        String fechaStr = (String) session.getAttribute("fecha");
        Fecha fecha = new Fecha(fechaStr);

        System.out.println(fecha);

        entradas = DatabaseManager.getAllEntradas();

        String horaStr = (String) session.getAttribute("hora");
        LocalTime hora = LocalTime.parse(horaStr);

        Pelicula peliculaSeleccionada = (Pelicula) session.getAttribute("pelicula");
    }
    %>
```

- Reserva.jsp:

```
<%  
Película películaSeleccionada = (Película) session.getAttribute("película");  
%>
```

- PagoExitoso.jsp:

```
<% String butacasJSON = (String) session.getAttribute("butacasSeleccionadas");

// Eliminar corchetes y comillas para obtener pares clave-valor separados por comas
String cleanString = butacasJSON.replaceAll("[\\[\\]\\{\\}\\\"'"]", "");

// Dividir la cadena en pares clave-valor
String[] keyValuePairs = cleanString.split(",");
```

```
// Crear una lista de mapas para almacenar las butacas seleccionadas
List<Map<String, Integer>> butacasList = new ArrayList<>();

// Iterar sobre los pares clave-valor y agregar a la lista de mapas
for (int i = 0; i < keyValuePairs.length; i += 2) {
    String[] filaKC = keyValuePairs[i].split(":");
    String[] columnaKC = keyValuePairs[i + 1].split(":");
    Map<String, Integer> butaca = Map.of("fila", Integer.parseInt(filaKC[1]), "columna",
Integer.parseInt(columnaKC[1]));
    butacasList.add(butaca);
}
%>
<ul>
    <% for (Map<String, Integer> butaca : butacasList) { %>
        <li>Fila: <%= butaca.get("fila") %>, Columna: <%= butaca.get("columna") %></li>
        <% } %>
    </ul>

<p>Día: <%= session.getAttribute("fecha") %></p>
<p>Hora: <%= session.getAttribute("hora") %></p>
<div class="section">
    <!-- Mostrar el código QR generado en el JavaScript -->

    <h3>Código QR:</h3>
    <%
        String datosReserva = "Numero de Referencia: " + session.getAttribute("numRef") +
            "Día: " + session.getAttribute("fecha") +
            "Hora: " + session.getAttribute("hora") +
            "Butacas Reservadas: " + butacasJSON;

    %>
```

Y en los servlets encontramos el uso de sesiones para almacenar y ver los datos introducidos a lo largo de los formularios rellenados por el usuario, previo registro:

- GestionButacas.java:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

    HttpSession session = request.getSession();

    Sala sala = (Sala) session.getAttribute("sala");

    // Obtener la cadena JSON de la solicitud
    String butacasSeleccionadasJSON = request.getParameter("butacasSeleccionadas");
    session.setAttribute("butacasSeleccionadas",butacasSeleccionadasJSON);
    response.sendRedirect(request.getContextPath() + "/confirmarReserva.jsp");

}
```

- ProcesarPagoServlet.java:

```
String numeroTarjetaConEspacios = request.getParameter("numeroT");

String numeroTarjeta = numeroTarjetaConEspacios.replace(" ", "");

String nombre_titular = request.getParameter("titular");
String fechaExpiracion = request.getParameter("fechaCaducidad");

String codigoSeguridad = request.getParameter("codigoSeguridad");

// Obtener la información del usuario de la sesión
HttpSession session = request.getSession();
Usuario usuarioActual = (Usuario) session.getAttribute("usuario");
String emailUsuario = usuarioActual.getCorreo();

TarjetaCredito tarjeta = new TarjetaCredito(numeroTarjeta, nombre_titular, fechaExpiracion,
codigoSeguridad, emailUsuario);

DatabaseManager.getInstance().guardarTarjeta(tarjeta);

// Validar la tarjeta en la base de datos
```

```
if (DatabaseManager.getInstance().validarTarjeta(emailUsuario, numeroTarjeta, fechaExpiracion,
codigoSeguridad)) {

    // Los datos de la tarjeta son válidos

    String numRef = "";

    int numeroAleatorioRef = new Random().nextInt(900000000) + 1000000000;

    numRef = "R" + Integer.toString(numeroAleatorioRef);// Crear una nueva entrada

    session.setAttribute("numRef", numRef);

    //CREACION DE RESERVA

    Sala sala = (Sala) session.getAttribute("sala");

    // Obtener la cadena JSON de la solicitud

    String butacasSeleccionadasJSON = (String) session.getAttribute("butacasSeleccionadas");

    // Eliminar corchetes y comillas para obtener pares clave-valor separados por comas

    String cleanString = butacasSeleccionadasJSON.replaceAll("[\\[\\]\\{\\}\\\" ]", "");

    System.out.println(cleanString);

    // Dividir la cadena en pares clave-valor

    String[] keyValuePairs = cleanString.split(",");

    //IdEntrada para la entradas seleccionadas por el mismo usuario.

    String idEntrada = "";

    int numeroAleatorio = new Random().nextInt(900000000) + 1000000000;

    idEntrada = "E" + Integer.toString(numeroAleatorio);// Crear una nueva entrada

    // Obtener información adicional de la sesión

    String fechaStr = (String) session.getAttribute("fecha");

    Fecha fecha = new Fecha(fechaStr);
```



```
String horaStr = (String) session.getAttribute("hora");

LocalTime hora = LocalTime.parse(horaStr);

int fila = 0;

int columna = 0;

String nombreSala = sala.getNombreSala();

for (int i = 0; i < keyValuePairs.length; i += 2) {

    // Obtener la fila y columna

    //Extraemos la fila

    String[] filaKC = keyValuePairs[i].split(":");

    fila = Integer.parseInt(filaKC[1]);

    //Extraemos la columna

    String[] columnaKC = keyValuePairs[i + 1].split(":");

    columna = Integer.parseInt(columnaKC[1]);

    Entrada entrada = new Entrada(idEntrada, fecha, hora, fila, columna, nombreSala);

    Reserva reserva = new Reserva(numRef, emailUsuario, idEntrada, fila, columna);

    try {

        // Guardar la entrada en la base de datos

        DatabaseManager.getInstance().guardarEntrada(entrada);

        DatabaseManager.getInstance().guardarReserva(reserva);

    } catch (SQLException ex) {

        Logger.getLogger(GestionButacas.class.getName()).log(Level.SEVERE, null, ex);

    }

}

response.sendRedirect(request.getContextPath() + "/PagoExitoso.jsp");
```

- Reservas.java:

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)

    throws ServletException, IOException {
```

```
HttpSession session = request.getSession();

// Obtener los parámetros del formulario
String peliculaId = request.getParameter("peliculaSeleccionada");
String fecha = request.getParameter("fecha");
String hora = request.getParameter("hora");

String sala = request.getParameter("salaSeleccionada");
System.out.println(sala);

try {
    Sala salaSelec = DatabaseManager.getInstance().getSalaPorNombre(sala);
    System.out.println(salaSelec.getNombreSala() + "EN EL SERVLET");
    // Almacenar los datos en la sesión
    session.setAttribute("peliculaId", peliculaId);
    session.setAttribute("hora", hora);
    session.setAttribute("fecha", fecha);
    session.setAttribute("sala", salaSelec);
    // Redireccionar a butacas.jsp

    response.sendRedirect(request.getContextPath() + "/butacas.jsp");
} catch (SQLException ex) {
    Logger.getLogger(Reservas.class.getName()).log(Level.SEVERE, null, ex);
}

}
```

## Gestión de Sesión del administrador

Por último, falta la gestión del panel de administrador, para evitar conflictos de seguridad y que copiando la url y entrando a otro navegador podamos acceder al contenido del usuario vamos a implementar dentro de los diferentes paneles del administrador una gestión de sesiones:

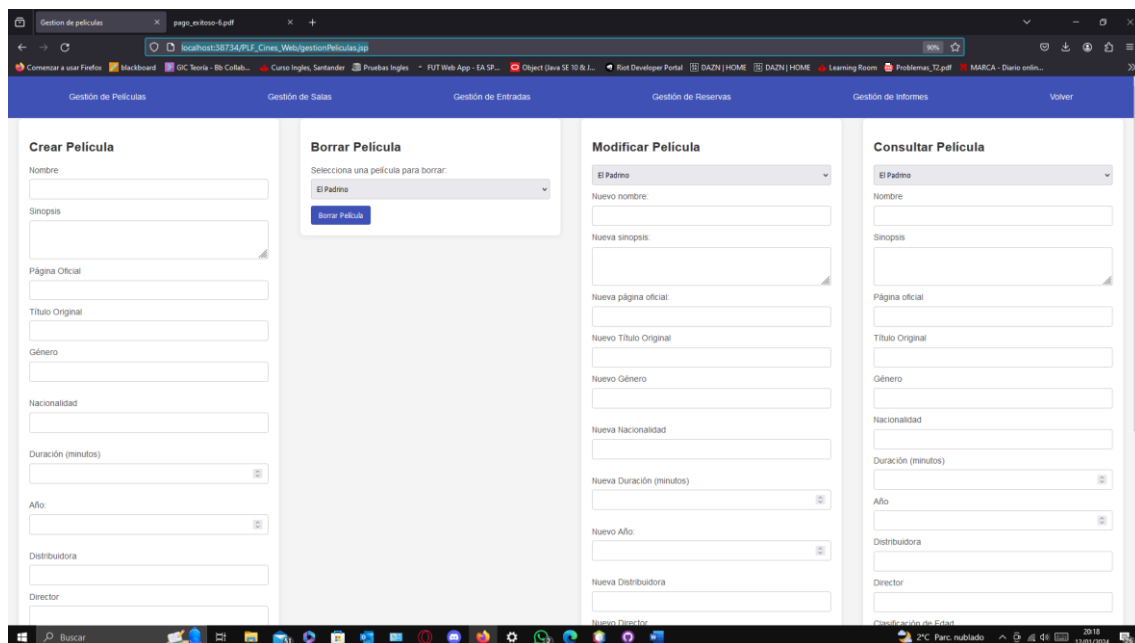
```
String correo = request.getParameter("mail-2");
```

```
String contrasena = request.getParameter("pswd-2");
```

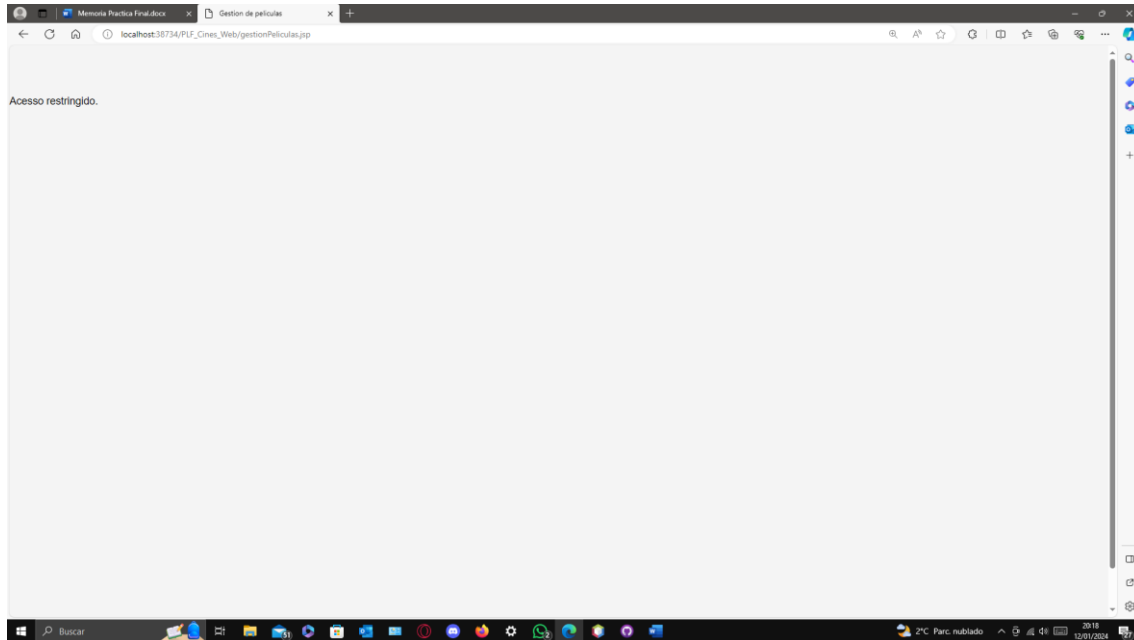
```
try {  
    if (correo.equals("admin@gmail.com") && contrasena.equals("admin")) {  
        // Usuario admin autenticado correctamente  
        Usuario usuario = new Usuario("Admin", "Admin", contrasena, correo, null);  
        HttpSession session = request.getSession();  
        session.setAttribute("usuario", usuario);  
        response.sendRedirect("panelAdmin.jsp");  
    }  
}
```

Este es el url de la gestión de películas del administrador, pero podría ser para cualquier acción del administrador:

[http://localhost:38734/PLF\\_Cines\\_Web/gestionPelículas.jsp](http://localhost:38734/PLF_Cines_Web/gestionPelículas.jsp)



Ahora vamos a otro navegador a introducir ese mismo el url para intentar pillar una vulnerabilidad del sistema



Esto es lo que nos encontramos.

Esto es lo que se ha hecho:

```
<% if (session.getAttribute("usuario") != null) { %>
```

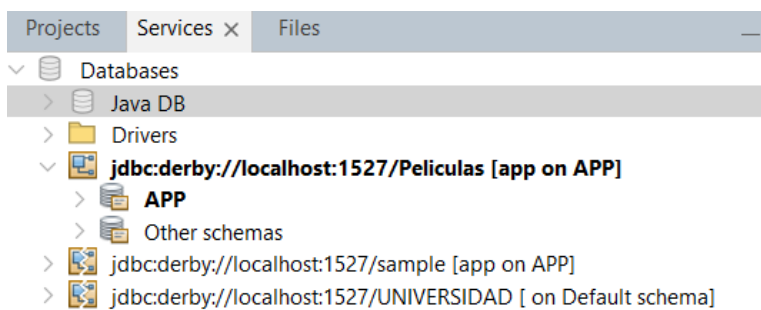
```
<% } else { %>
```

```
    <p>Acesso restringido.</p>
```

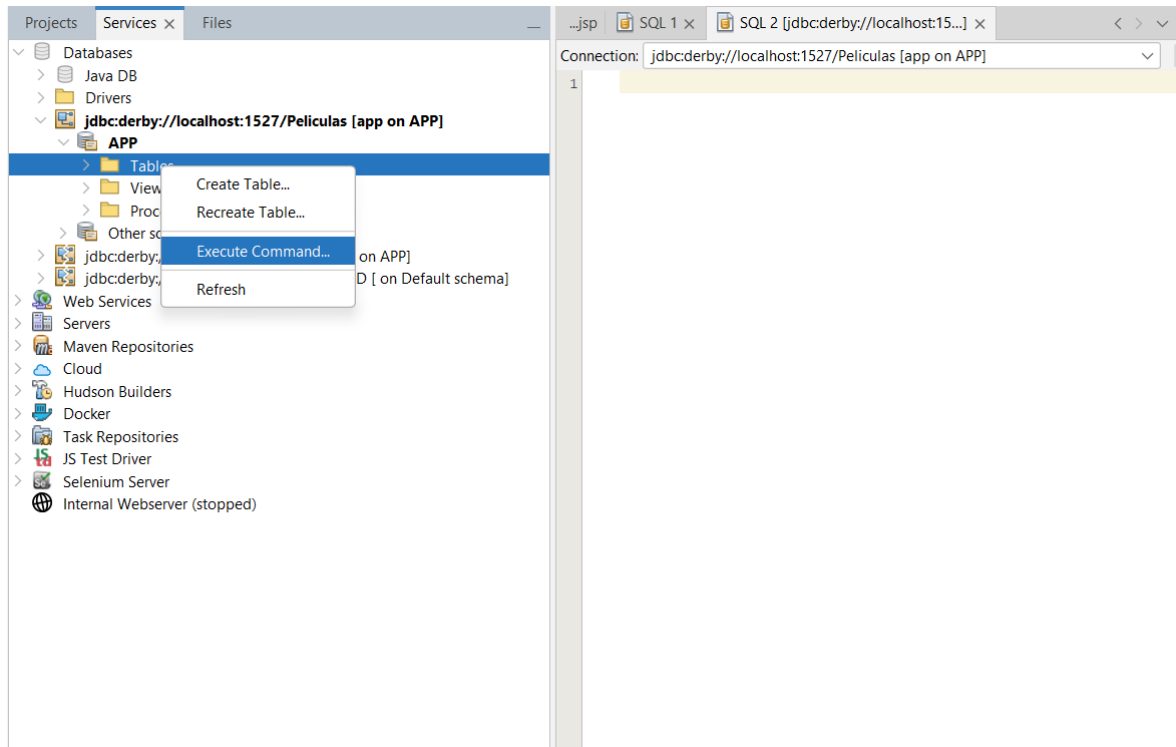
```
<% } %>
```

### Almacenamiento de datos en Apache Derby.

El empleo de Apache Derby ha sido esencial, pues nos ha servido para la creación de la base de datos empleada en nuestro MVC, en la sección de Modelo. El almacenamiento de datos se ha realizado, primero creando una nueva base de datos, a la que hemos llamado Películas.



Una vez hecho esto, hemos procedido a insertar todas nuestras tablas, con sus relaciones. Esto se hace muy simple usando comandos SQL para la creación de tablas y restricciones.



Y el código que hemos introducido ha sido el siguiente:

```
-- object: Pelicula | type: TABLE --
CREATE TABLE Pelicula (
  nombrePelicula varchar(50) NOT NULL,
  sinopsis varchar(500) NOT NULL,
  paginaOficial varchar(50),
  tituloOriginal varchar(50),
  genero varchar(50) NOT NULL,
  nacionalidad varchar(50) NOT NULL,
  duracion integer NOT NULL,
  anho integer NOT NULL,
  distribuidora varchar(50),
  director varchar(50) NOT NULL,
  clasificacionEdad smallint NOT NULL,
  otrosDatos varchar(200) NOT NULL,
  actores varchar(200) NOT NULL,
  url_image varchar(500) NOT NULL,
  url_video varchar(500) NOT NULL,
  CONSTRAINT PK_Pelicula PRIMARY KEY (nombrePelicula)
);

-- object: Sala | type: TABLE --
CREATE TABLE Sala (
  nombreSala varchar(20) NOT NULL,
  filas smallint NOT NULL,
  columnas smallint NOT NULL,
  nombrePelicula_Pelicula varchar(50), -- Nueva columna
  CONSTRAINT PK_Sala PRIMARY KEY (nombreSala)
```

```
);

-- object: Entrada | type: TABLE --
CREATE TABLE Entrada (
    idEntrada varchar(10) NOT NULL,
    fecha date NOT NULL,
    hora time NOT NULL,
    fila smallint NOT NULL,
    columna smallint NOT NULL,
    nombreSala_Sala varchar(20),
    CONSTRAINT PK_Entrada PRIMARY KEY (idEntrada, fila, columna)
);

-- object: Usuario | type: TABLE --
CREATE TABLE Usuario (
    nombre varchar(50) NOT NULL,
    apellidos varchar(50) NOT NULL,
    contrasena varchar(50) NOT NULL,
    email varchar(50) NOT NULL,
    fechaNacimiento date NOT NULL,
    CONSTRAINT PK_Usuario PRIMARY KEY (email)
);

-- object: Reserva | type: TABLE --
CREATE TABLE Reserva (
    numeroRef varchar(20) NOT NULL,
    email_Usuario varchar(50),
    idEntrada_Entrada varchar(10),
    fila_Entrada smallint NOT NULL,
    columna_Entrada smallint NOT NULL,
    CONSTRAINT PK_Reserva PRIMARY KEY (numeroRef),
    CONSTRAINT EntradaReserva_fk FOREIGN KEY (idEntrada_Entrada,
fila_Entrada, columna_Entrada)
REFERENCES Entrada (idEntrada, fila, columna)
);

-- object: Comentario | type: TABLE --
CREATE TABLE Comentario (
    texto varchar(500) NOT NULL,
    valoracion smallint NOT NULL,
    fechaComentario date NOT NULL,
    email_Usuario varchar(50) NOT NULL,
    nombrePelicula_Pelicula varchar(50)
);

--object: Tarjeta | type: TABLE --
CREATE TABLE Tarjeta (
    numeroTarjeta varchar(25) NOT NULL,
    nombreTitular varchar(50) NOT NULL,
    fechaExpiracion varchar(10) NOT NULL,
```

```
codigoSeguridad varchar(3) NOT NULL,  
email_Usuario varchar(50) NOT NULL,  
CONSTRAINT PK_Tarjeta PRIMARY KEY (numeroTarjeta),  
CONSTRAINT FK_Tarjeta_Usuario FOREIGN KEY (email_Usuario) REFERENCES  
Usuario (email)  
);  
  
-- object: SalaPelicula_fk | type: CONSTRAINT --  
ALTER TABLE Sala ADD CONSTRAINT SalaPelicula_fk FOREIGN KEY  
(nombrePelicula_Pelicula)  
REFERENCES Pelicula (nombrePelicula);  
  
-- object: SalaEntrada_fk | type: CONSTRAINT --  
ALTER TABLE Entrada ADD CONSTRAINT SalaEntrada_fk FOREIGN KEY  
(nombreSala_Sala)  
REFERENCES Sala (nombreSala);  
  
-- object: UsuarioReserva_fk | type: CONSTRAINT --  
ALTER TABLE Reserva ADD CONSTRAINT UsuarioReserva_fk FOREIGN KEY  
(email_Usuario)  
REFERENCES Usuario (email);  
  
-- object: EntradaReserva_fk | type: CONSTRAINT --  
--ALTER TABLE Reserva ADD CONSTRAINT EntradaReserva_fk FOREIGN KEY  
(idEntrada_Entrada, fila_Entrada, columna_Entrada)  
--REFERENCES Entrada (idEntrada, fila, columna);  
  
-- object: ComentarioUsuario_fk | type: CONSTRAINT --  
ALTER TABLE Comentario ADD CONSTRAINT ComentarioUsuario_fk FOREIGN KEY  
(email_Usuario)  
REFERENCES Usuario (email);  
  
-- object: ComentarioPelicula_fk | type: CONSTRAINT --  
ALTER TABLE Comentario ADD CONSTRAINT ComentarioPelicula_fk FOREIGN KEY  
(nombrePelicula_Pelicula)  
REFERENCES Pelicula (nombrePelicula);  
  
--IMPORTS--  
  
-- Películas  
  
INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,  
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,  
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)  
VALUES ('El Padrino', 'Un poderoso drama criminal que sigue la vida de la  
familia Corleone en el mundo del crimen organizado.',  
'https://example.com/godfather', 'Il Padrino', 'Crimen, Drama', 'Estados  
Unidos', 175, 1972, 'Paramount Pictures', 'Francis Ford Coppola', 18,
```

```
'Ganadora de 3 premios Oscar.', 'Marlon Brando, Al Pacino, James
Caan','images/ElPadrino.jpeg',
'https://www.youtube.com/embed/iOyQx7MXaz0?si=nuCJfNniJoPzuQTb');

INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)
VALUES ('Cadena Perpetua', 'La historia de un hombre inocente encarcelado
que encuentra esperanza y amistad en Shawshank.',
'https://example.com/shawshank', 'Shawshank Redemption', 'Drama',
'Estados Unidos', 142, 1994, 'Columbia Pictures', 'Frank Darabont', 15,
'Basada en la novela de Stephen King.', 'Tim Robbins, Morgan Freeman',
'images/CadenaPerpetua.jpeg',
'https://www.youtube.com/embed/4u87tmlj4oE?si=n318897_-rDPVLzP');

INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)
VALUES ('El Caballero Oscuro', 'Batman se enfrenta a su némesis, el
Joker, en un juego de ingenio que amenaza Gotham City.',
'https://example.com/darkknight', 'The Dark Knight', 'Acción, Crimen,
Drama', 'Estados Unidos', 152, 2008, 'Warner Bros.', 'Christopher Nolan',
16, 'Ganadora de 2 premios Oscar.', 'Christian Bale, Heath Ledger, Aaron
Eckhart', 'images/CaballeroOscuro.jpeg',
'https://www.youtube.com/embed/9MEuGQtM9wA?si=A5YSZGVuwDcTK83o');

INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)
VALUES ('Cars', 'En un mundo donde los autos son seres vivos, el
aspirante a corredor Rayo McQueen termina en un pequeño pueblo llamado
Radiador Springs. Aprendiendo lecciones valiosas de amistad y humildad,
Rayo descubre el verdadero significado de la velocidad.',
'https://www.paginaoficial.com/cars', 'Cars', 'Animación',
'Estadounidense', 120, 2006, 'Pixar', 'John Lasseter', 3, 'Datos
adicionales sobre la película Cars.', 'Owen Wilson, Bonnie Hunt',
'images/Cars.png',
'https://www.youtube.com/embed/nuQDFYpPUh4?si=owxKwSIVF6QzUeh1');

INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)
VALUES ('Cars 2', 'Rayo McQueen y su amigo Mate se ven envueltos en una
misión internacional de espionaje cuando son reclutados para participar
en el Grand Prix Mundial. Intrigas, giros y vueltas esperan a nuestros
héroes mientras luchan contra una misteriosa amenaza.',
'https://www.paginaoficial.com/cars2', 'Cars 2', 'Animación',
'Estadounidense', 130, 2011, 'Pixar', 'John Lasseter', 3, 'Datos
adicionales sobre la película Cars 2.', 'Owen Wilson, Larry the Cable
```



```
Guy', 'images/Cars2.png',  
'https://www.youtube.com/embed/oGpIMRq103k?si=UUS0oOjMZg_RPptk');
```

```
INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,  
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,  
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)  
VALUES ('Cars 3', 'Con una nueva generación de corredores amenazando su  
posición, Rayo McQueen se embarca en un emocionante viaje de  
redescubrimiento y superación. Con la ayuda de entrenadores y nuevos  
amigos, Rayo intenta volver al ruedo y dejar su marca.',  
'https://www.paginaoficial.com/cars3', 'Cars 3', 'Animación',  
'Estadounidense', 110, 2017, 'Pixar', 'Brian Fee', 3, 'Datos adicionales  
sobre la película Cars 3.', 'Owen Wilson, Cristela Alonzo',  
'images/Cars3.png',  
'https://www.youtube.com/embed/wtmW9rSRIzU?si=MttED7g2UgQk5A08');
```

```
INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,  
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,  
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)  
VALUES ('Kung Fu Panda', 'Po, un torpe oso panda, sueña con convertirse  
en un hábil guerrero de kung fu. Su vida da un giro cuando es elegido  
accidentalmente para cumplir una antigua profecía. Po se embarca en una  
aventura cómica y emocionante para aprender las artes marciales y salvar  
el Valle de la Paz.', 'https://www.paginaoficial.com/kungfupanda', 'Kung  
Fu Panda', 'Animación', 'Estadounidense', 95, 2008, 'DreamWorks  
Animation', 'Mark Osborne, John Stevenson', 3, 'Datos adicionales sobre  
la película Kung Fu Panda.', 'Jack Black, Angelina Jolie',  
'images/KunfuPanda.png', 'https://www.youtube.com/embed/vRhMIpFu-  
Zw?si=Hwgor3iZaTcqtzm4');
```

```
INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,  
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,  
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)  
VALUES ('One Piece', 'Monkey D. Luffy y su valiente tripulación emprenden  
un viaje épico en busca del legendario tesoro conocido como "One Piece".  
Enfrentándose a peligros, enemigos y desafíos, Luffy se esfuerza por  
convertirse en el Rey de los Piratas.',  
'https://www.paginaoficial.com/onepiece', 'One Piece', 'Animación',  
'Japonesa', 120, 2022, 'Toei Animation', 'Eiichiro Oda', 12, 'Datos  
adicionales sobre la película One Piece.', 'Mayumi Tanaka, Kazuya Nakai',  
'images/OnePiece.png',  
'https://www.youtube.com/embed/CWrJQzm0diI?si=tzF0AIbtEib0kGCV');
```

```
INSERT INTO Pelicula (nombrePelicula, sinopsis, paginaOficial,  
tituloOriginal, genero, nacionalidad, duracion, anho, distribuidora,  
director, clasificacionEdad, otrosDatos, actores, url_image, url_video)  
VALUES ('Rocky', 'Rocky Balboa, un boxeador de Filadelfia con pocas  
oportunidades, recibe la oportunidad de enfrentarse al campeón de peso  
pesado Apollo Creed. Aunque inicialmente es considerado como un oponente
```

```
fácil, Rocky se prepara para la pelea de su vida, demostrando coraje y
determinación.', 'https://www.paginaoficial.com/rocky', 'Rocky', 'Drama',
'Estadounidense', 120, 1976, 'United Artists', 'John G. Avildsen', 12,
'Datos adicionales sobre la película Rocky.', 'Sylvester Stallone, Talia
Shire, Burgess Meredith', 'images/Rocky.png',
'https://www.youtube.com/embed/Vb-Bx2YkIPA?si=UZigoaMfxKRR6Hy8');
```

```
-- Puedes agregar más películas según sea necesario...
```

```
-- Salas
```

```
INSERT INTO Sala (nombreSala, filas, columnas, nombrePelicula_Pelicula)
VALUES ('Sala 1', 10, 15, 'El Padrino');
```

```
INSERT INTO Sala (nombreSala, filas, columnas, nombrePelicula_Pelicula)
VALUES ('Sala 2', 8, 12, 'Cadena Perpetua');
```

```
INSERT INTO Sala (nombreSala, filas, columnas, nombrePelicula_Pelicula)
VALUES ('Sala 3', 12, 20, 'El Caballero Oscuro');
```

```
-- Entradas
```

```
INSERT INTO Entrada (idEntrada, fecha, hora, fila, columna,
nombreSala_Sala)
VALUES ('E1', '2023-01-01', '12:00:00', 5, 10, 'Sala 1');
```

```
INSERT INTO Entrada (idEntrada, fecha, hora, fila, columna,
nombreSala_Sala)
VALUES ('E2', '2023-01-02', '15:30:00', 3, 8, 'Sala 2');
```

```
INSERT INTO Entrada (idEntrada, fecha, hora, fila, columna,
nombreSala_Sala)
VALUES ('E3', '2023-01-03', '18:45:00', 8, 15, 'Sala 3');
```

```
-- Usuarios
```

```
INSERT INTO Usuario (nombre, apellidos, contrasenha, email,
fechaNacimiento)
VALUES ('Rafael', 'Gonzalez', '1234', 'rafael@example.com', '1990-05-
15');
```

```
INSERT INTO Usuario (nombre, apellidos, contrasenha, email,
fechaNacimiento)
VALUES ('Maria', 'Lopez', '12345', 'maria@example.com', '1985-09-22');
```

```
INSERT INTO Usuario (nombre, apellidos, contrasenha, email,
fechaNacimiento)
VALUES ('Carlos', 'Martinez', '12345', 'carlos@example.com', '1995-12-
10');
```

```
-- Reservas
```

```
INSERT INTO Reserva (numeroRef, email_Usuario, idEntrada_Entrada,
fila_Entrada, columna_Entrada)
VALUES ('R1', 'rafael@example.com', 'E1', 5, 10);

INSERT INTO Reserva (numeroRef, email_Usuario, idEntrada_Entrada,
fila_Entrada, columna_Entrada)
VALUES ('R2', 'maria@example.com', 'E2', 3, 8);

INSERT INTO Reserva (numeroRef, email_Usuario, idEntrada_Entrada,
fila_Entrada, columna_Entrada)
VALUES ('R3', 'carlos@example.com', 'E3', 8, 15);

-- Comentarios
INSERT INTO Comentario (texto, valoracion, fechaComentario,
email_Usuario, nombrePelicula_Pelicula)
VALUES ('Una obra maestra, actuaciones increíbles.', 5, '2023-01-05',
'rafael@example.com', 'El Padrino');

INSERT INTO Comentario (texto, valoracion, fechaComentario,
email_Usuario, nombrePelicula_Pelicula)
VALUES ('Emocionante de principio a fin.', 4, '2023-01-06',
'maria@example.com', 'Cadena Perpetua');

INSERT INTO Comentario (texto, valoracion, fechaComentario,
email_Usuario, nombrePelicula_Pelicula)
VALUES ('Efectos visuales asombrosos.', 4, '2023-01-07',
'carlos@example.com', 'El Caballero Oscuro');

-- Tarjetas
INSERT INTO Tarjeta (numeroTarjeta, nombreTitular, fechaExpiracion,
codigoSeguridad, email_Usuario)
VALUES ('1111222233334444', 'Rafael Gonzalez', '2024-08-31', '456',
'rafael@example.com');

INSERT INTO Tarjeta (numeroTarjeta, nombreTitular, fechaExpiracion,
codigoSeguridad, email_Usuario)
VALUES ('4444555566667777', 'Maria Lopez', '2023-11-30', '987',
'maria@example.com');

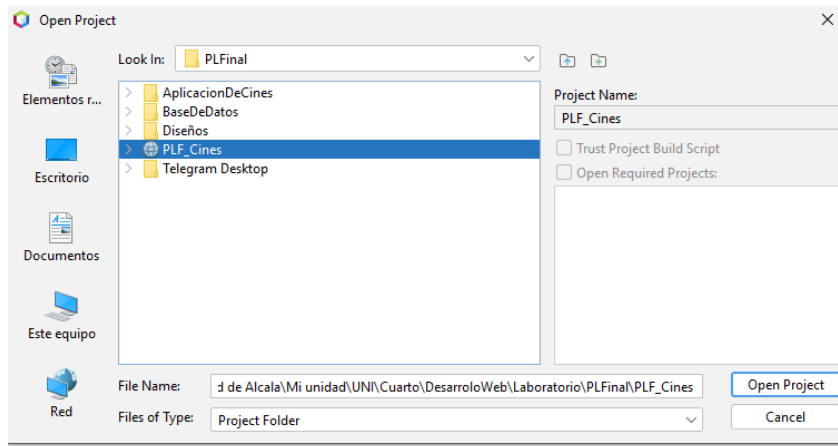
INSERT INTO Tarjeta (numeroTarjeta, nombreTitular, fechaExpiracion,
codigoSeguridad, email_Usuario)
VALUES ('7777888899990000', 'Carlos Martinez', '2023-09-30', '345',
'carlos@example.com');
```

## Manual de Usuario

En este apartado nos pondremos en el lugar de un usuario que nunca ha usado la web, para orientarle a lo largo de las funcionalidades que tiene la página.

### Instrucciones para la instalación y ejecución.

Una vez descomprimido el zip de la descarga, el archivo que nos interesa se llama PLF\_Cines, por lo que entramos en NetBeans que es el entorno en el que se ha llevado a cabo la práctica y seleccionamos la opción de **open Project**.



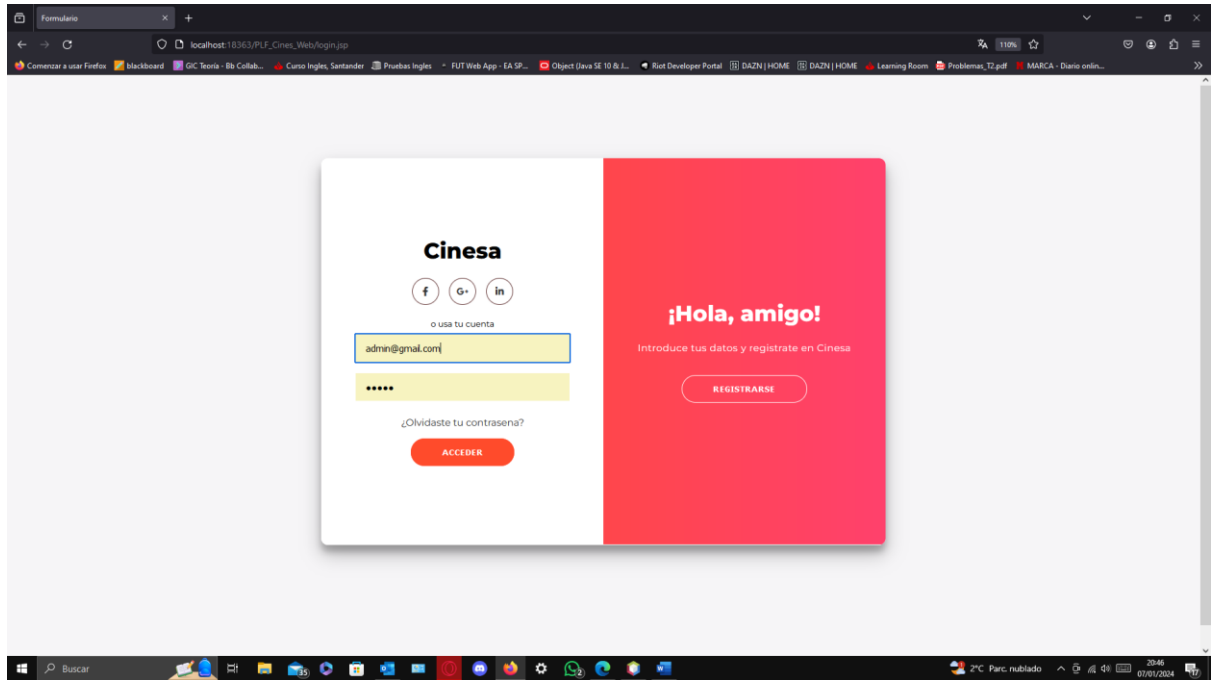
Seguidamente en nuestra base de datos en este caso apache Derby insertamos el código sql que viene en la carpeta Base de datos, para rellenar las tablas y poder comenzar a trabajar, hay que tener en cuenta que es un proyecto web por lo que cada vez que hacemos un cambio se auto compila, le damos a la opción de **clean and build** y si todo está bien pasamos a la ejecución del programa.

## Guía detallada de las funcionalidades.

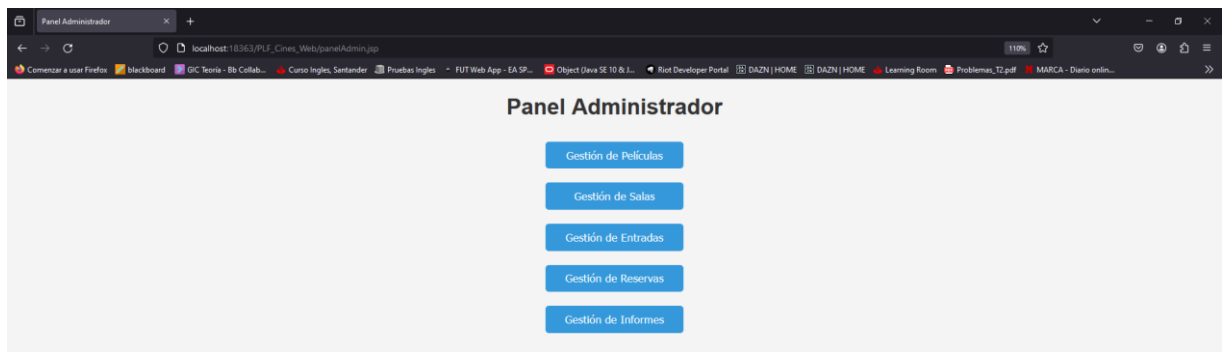
### Gestión/Administración del Cine.

El administrador es una especie de super usuario creado en el sistema, pero que no está dentro de la base de datos para evitar si nos hackean o consiguen acceso a la base de datos, que no nos puedan controlar nuestro panel de administración.

Pasamos a ver la parte visual, para poder acceder a la administración tenemos que iniciar sesión como administradores, por lo que nos dirigimos a la opción de registrarse y escribimos dentro del panel del login [admin@gmail.com](mailto:admin@gmail.com) admin.



Y accedemos:

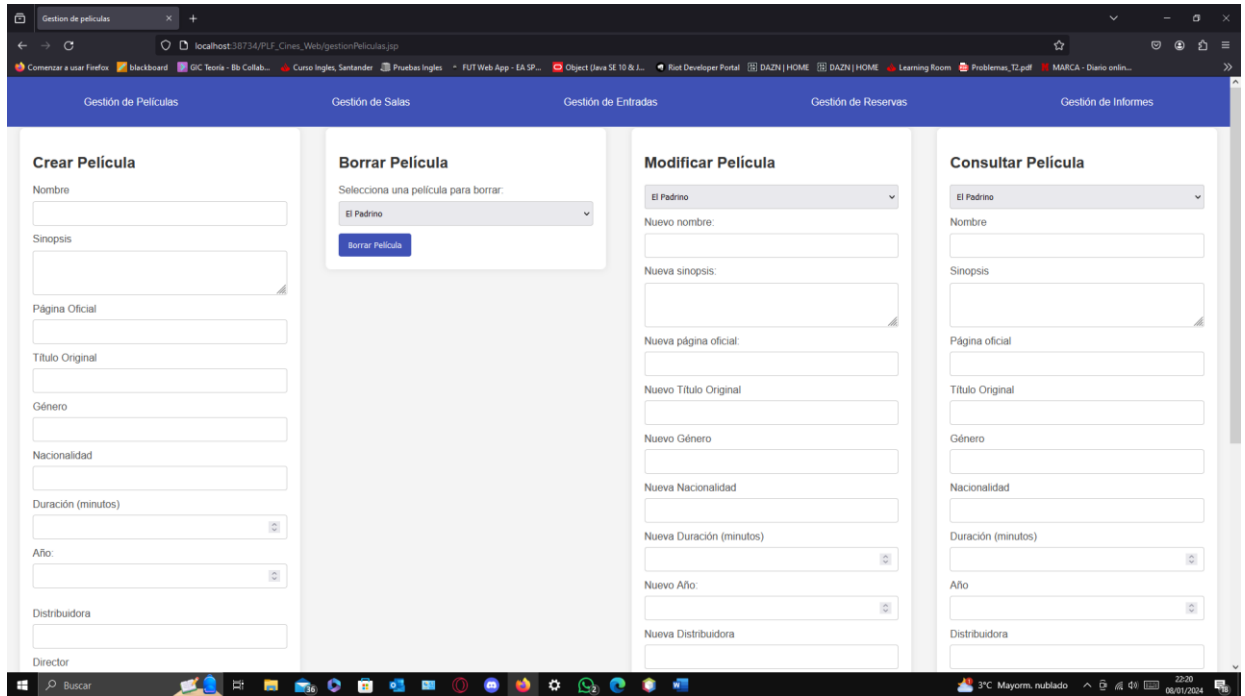


Ahora el administrador decide qué operación realizar.

### Gestión de Películas.

Todo el panel del administrador se ha hecho llevando a cabo el patrón de diseño MVC, la implementación y el código interno explicado está en la sección de implementación, pasamos a ver el panel de gestión de películas:

Tenemos 4 posibles operaciones, insertar película, borrar película, modificar película y consultar película:



Información para el usuario: En el campo Url imagen se ha de poner `images/"NombreImagen".png`, esto es debido a que en el proyecto tenemos una carpeta llamada `images` donde guardamos todas las portadas de las películas.

### Crear Películas:

Hay un cheking creado con javascript en el caso de que se creen datos que no existen

### Borrar Película:

Sale una lista desplegable con todas las películas, el administrador selecciona la que desee y la borra.

### Modificar película:

Se selecciona la película y se cambia los campos que se desee.

### Consultar película:

Se selecciona de la lista desplegable la película y se muestran los datos en los campos.

### Consultar Película

El Padrino ▼

Nombre

El Caballero Oscuro

Sinopsis

Batman se enfrenta a su némesis, el Joker, en un juego de ingenio que amenaza Gotham City.

Página oficial

https://example.com/darkknight

Título Original

The Dark Knight

Género

Acción, Crimen, Drama

Nacionalidad

Estados Unidos

Duración (minutos)

152

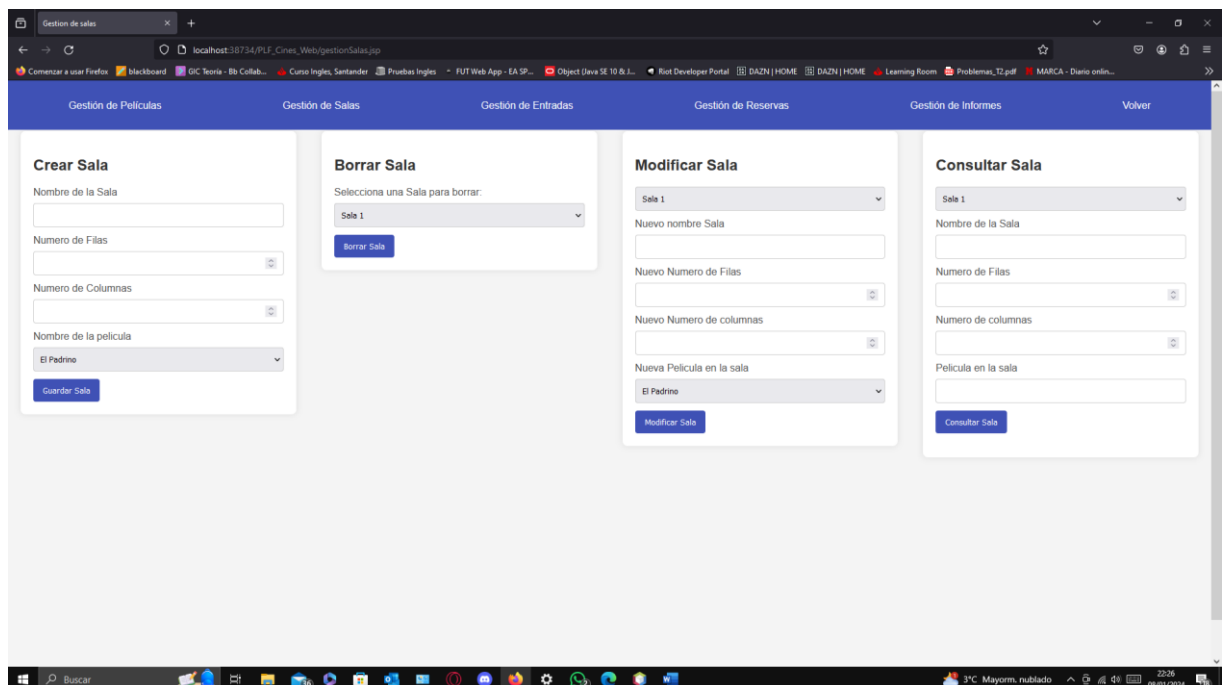
Año

2008

Distribuidora

### Gestión de Salas.

En este apartado principalmente crearemos las salas y añadiremos a esas salas películas, tenemos las mismas 4 operaciones, crear, borrar, modificar y consultar:



### Crear Sala:

Hay un cheking creado con javascript en el caso de que se creen datos que no existen, el ultimo campo nos permite abrir una lista desplegable con todas las películas y seleccionar la que queramos en esa sala.

### Borrar Sala:

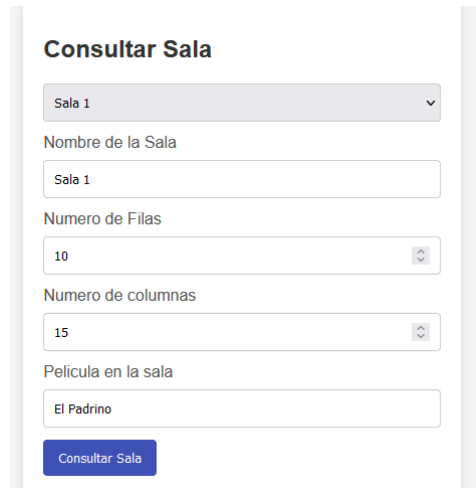
Sale una lista desplegable con todas las salas, el administrador selecciona la que desee y la borra.

### Modificar Sala:

Se selecciona la sala y se cambia los campos que se desee.

### Consultar Sala:

Se selecciona de la lista desplegable la sala y se muestran los datos en los campos.



**Consultar Sala**

Sala 1

Nombre de la Sala

Sala 1

Numero de Filas

10

Numero de columnas

15

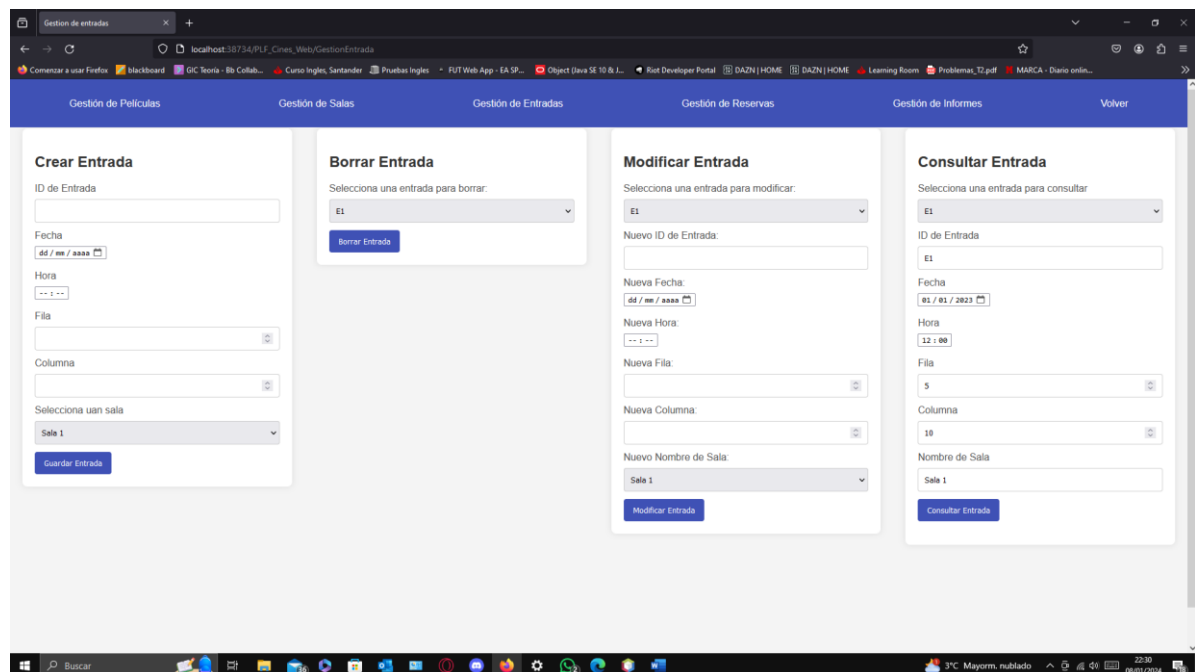
Película en la sala

El Padrino

Consultar Sala

### Gestión de Entradas.

En este apartado el administrador puede crear entradas en una determinada sala , un determinado día y a una determinada hora , luego ya quedara esa butaca o esas butacas seleccionadas.



Gestión de entradas

localhost:38734/PLF\_Cines/Web/GestionEntrada

Comenzar a usar Firefox | Blackboard | GIC Teoría - Bn Collab... | Curso Inglés, Santander | Pruebas Inglés | PUT Web App - EA SP... | Object Java SE 10 & L... | Rust Developer Portal | DAZN | HOME | DAZN | HOME | Learning Room | Problemas\_T2.pdf | MARCA - Diario online...

Gestión de Películas | Gestión de Salas | **Gestión de Entradas** | Gestión de Reservas | Gestión de Informes | Volver

**Crear Entrada**

ID de Entrada

Fecha

dd / mm / aaaa

Hora

-- : --

Fila

Columna

Selecciona una sala

Sala 1

Guardar Entrada

**Borrar Entrada**

Selecciona una entrada para borrar:

E1

Borrar Entrada

**Modificar Entrada**

Selecciona una entrada para modificar:

E1

Nuevo ID de Entrada:

Nueva Fecha:

dd / mm / aaaa

Nueva Hora:

-- : --

Nueva Fila:

Nueva Columna:

Nuevo Nombre de Sala:

Sala 1

Modificar Entrada

**Consultar Entrada**

Selecciona una entrada para consultar:

E1

ID de Entrada

E1

Fecha

dd / mm / aaaa

Hora

-- : --

Fila

Columna

Nombre de Sala

Sala 1

Consultar Entrada

23:30 06/01/2024



**Crear Entrada:**

Hay un cheking creado con javascript en el caso de que se creen datos que no existen, el ultimo campo nos permite abrir una lista desplegable con todas las salas y seleccionar la que queramos en esa entrada.

**Borrar entrada:**

Sale una lista desplegable con todas las entradas, el administrador selecciona la que desee y la borra.

**Modificar entrada:**

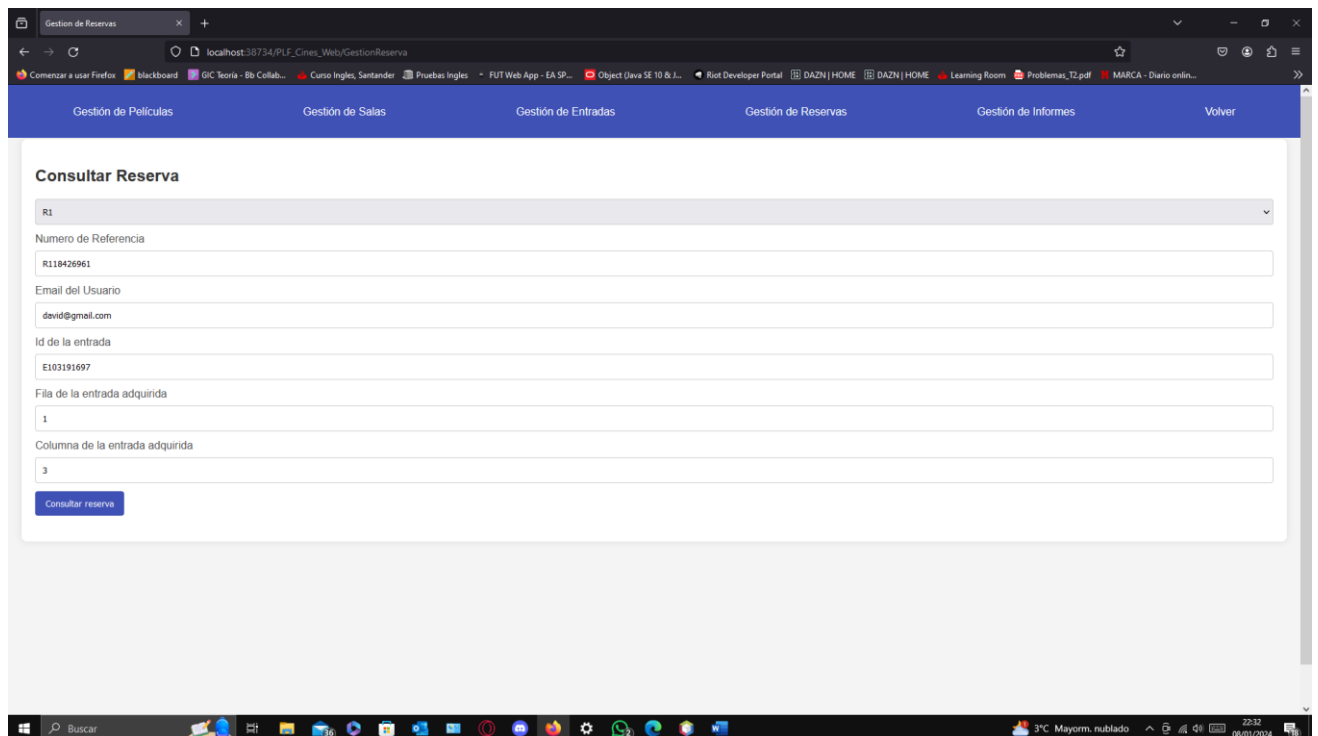
Se selecciona la entrada y se cambia los campos que se desee.

**Consultar entrada:**

Se selecciona de la lista desplegable la entrada y se muestran los datos en los campos.

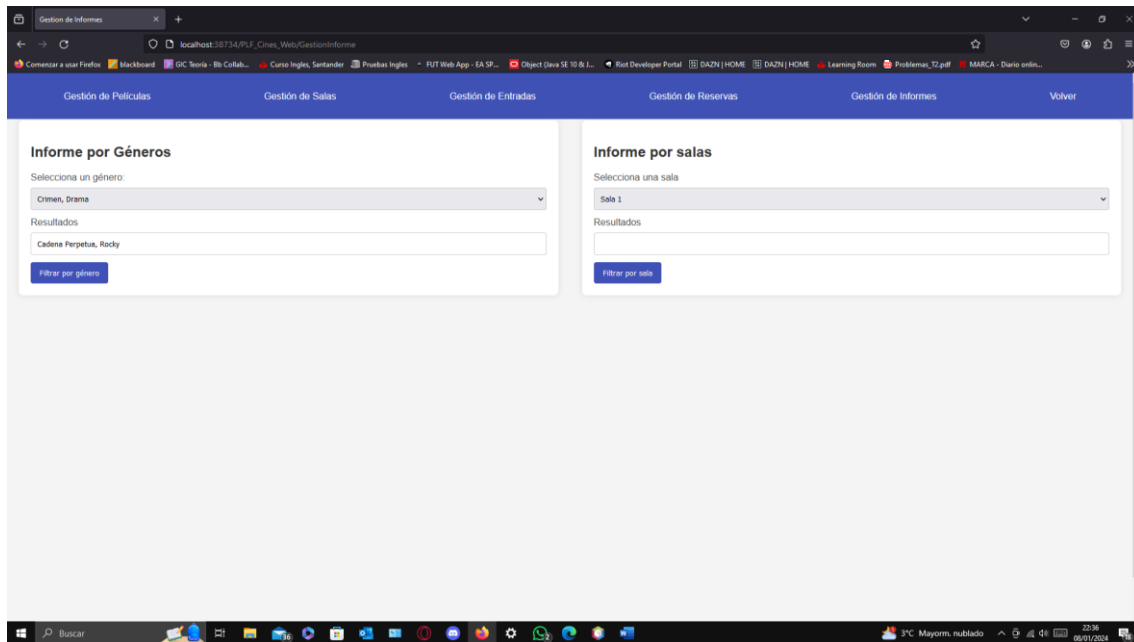
**Gestión de Reservas.**

En este campo a diferencia de los tres anteriores solo le sirve al administrador para consultar reservas, para que cuando los clientes lleguen al cine con su numero de reserva, solo tener que seleccionar el numero de la reserva y mirar los datos.



## Gestión de Informes.

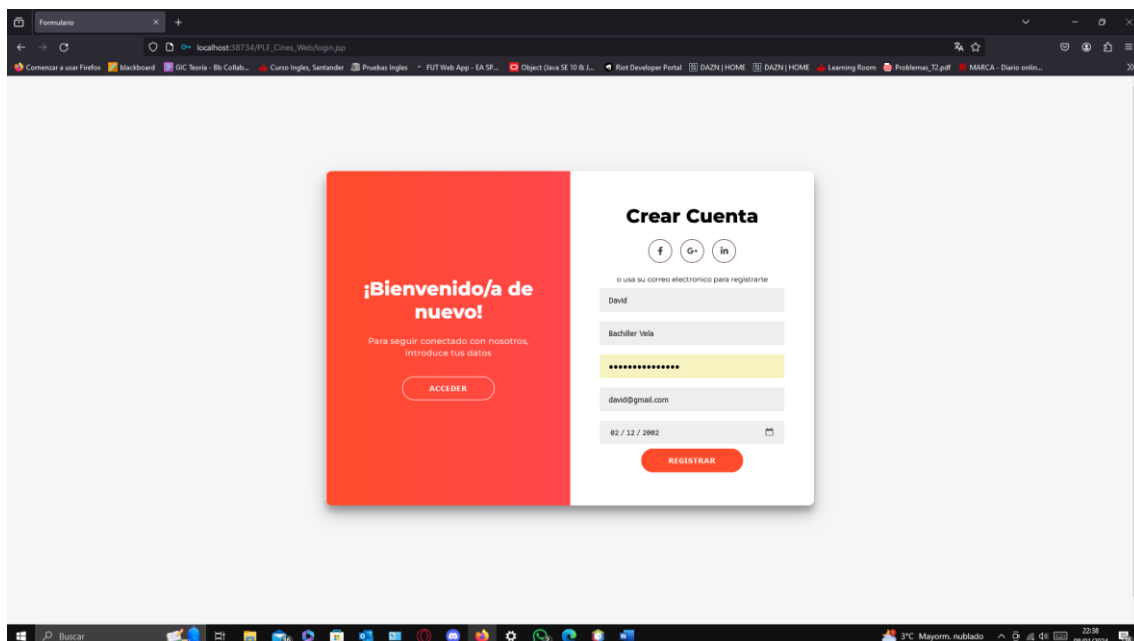
Al igual que el anterior es solo un panel de consulta en este caso ahí informes relacionados con el género y de las películas y las salas, que pueden servir como mero informativo o para en un futuro sacar estadísticas.



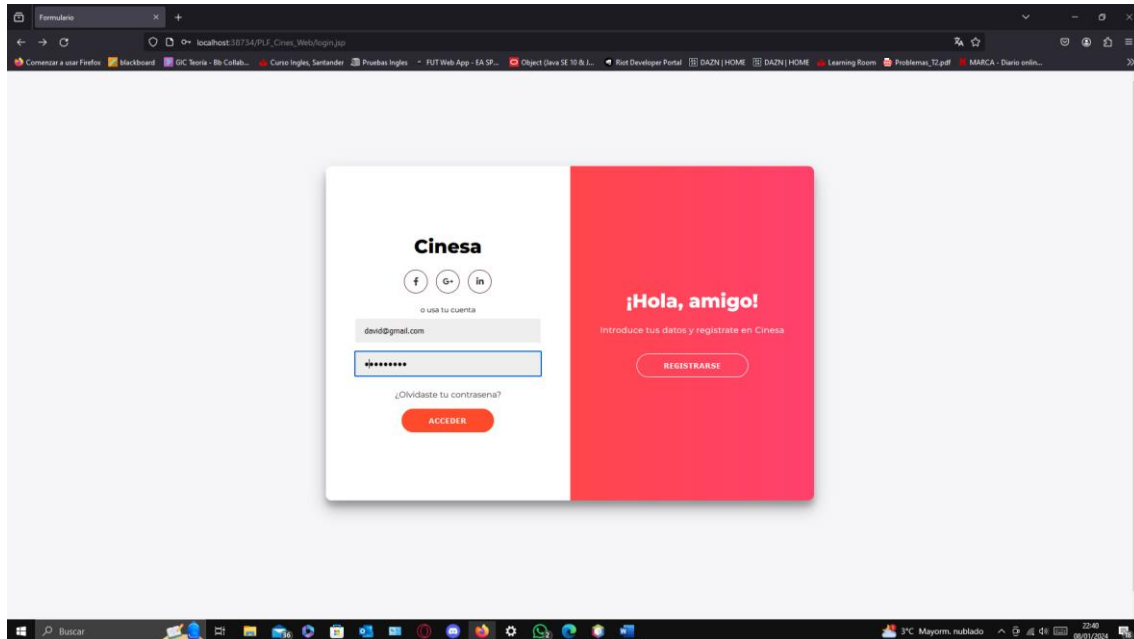
## Funcionalidades para los Clientes.

### Registro.

Dentro del registro tenemos la posibilidad de registrarnos o de iniciar sesión si ya estamos registrados, simplemente rellenamos los datos, para darnos de alta en el sistema.



El panel de login nos permite iniciar sesión y si todo se ha creado correctamente mediante gestión de sesiones accedemos al sistema, y ya podremos hacer comentarios y hacer una reserva.



Reserva de entradas.

Una vez iniciada sesión podemos realizar reservas sobre las distintas películas de nuestro cine pues ahora en cada una no aparece el botón “reservar entradas”.



Al pulsar en él, nos redireccionará para seleccionar las salas, hora y fecha cuando queremos ver la película.

Primero necesitamos seleccionar la fecha:

## EL PADRINO

### RESERVA



Calendar for January 2024. The date 11 is selected.

lun	mar	mié	jue	vie	sáb	dom
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

Limpiar

dd / mm / aaaa

Hora:

14:00

Sala:

Selecciona una sala

Seleccionar Butacas

Las horas:

Hora:

14:00

14:00

17:15

Y las salas:

Selecciona una sala

Selecciona una sala

Sala 1

Una vez seleccionado esto, al pulsar el botón “Seleccionar Butacas”, nos aparecerá una página con las butacas de la sala. En rojo aquellas que no pueden volver a ser seleccionadas por estar ocupadas por otros clientes, en verde las disponibles.






































































































































































EL PADRINO

SELECCION BUTACAS: Sala 1

Comprar

El usuario seleccionara unas butacas:

Comprar

Y pulsara el botón comprar para proceder al pago, donde tendremos que introducir los datos de la tarjeta.

## Información del pago



Nombre del titular

Rafael Gonzalez

Numero de tarjeta

generar numero random

3566 0020 2036 0505



Fecha de caducidad (mm/yy)

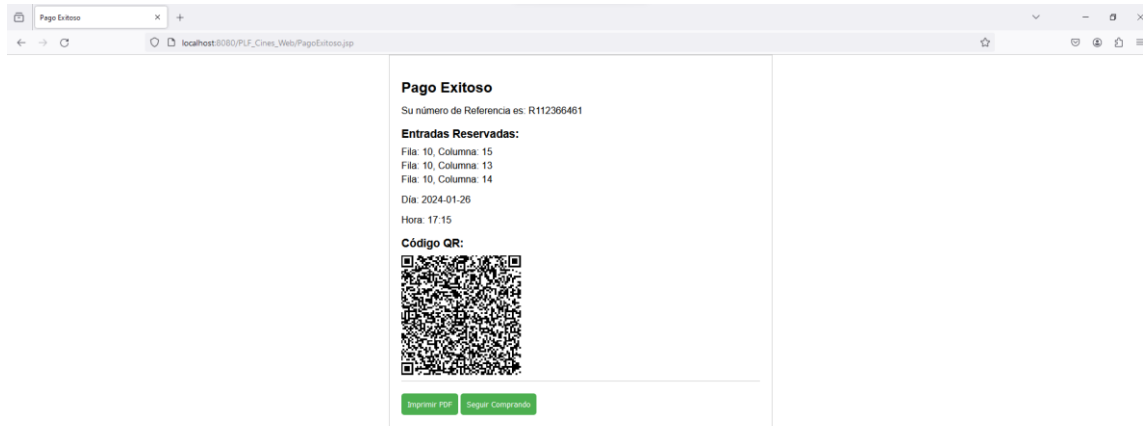
12/26

Codigo de seguridad

123

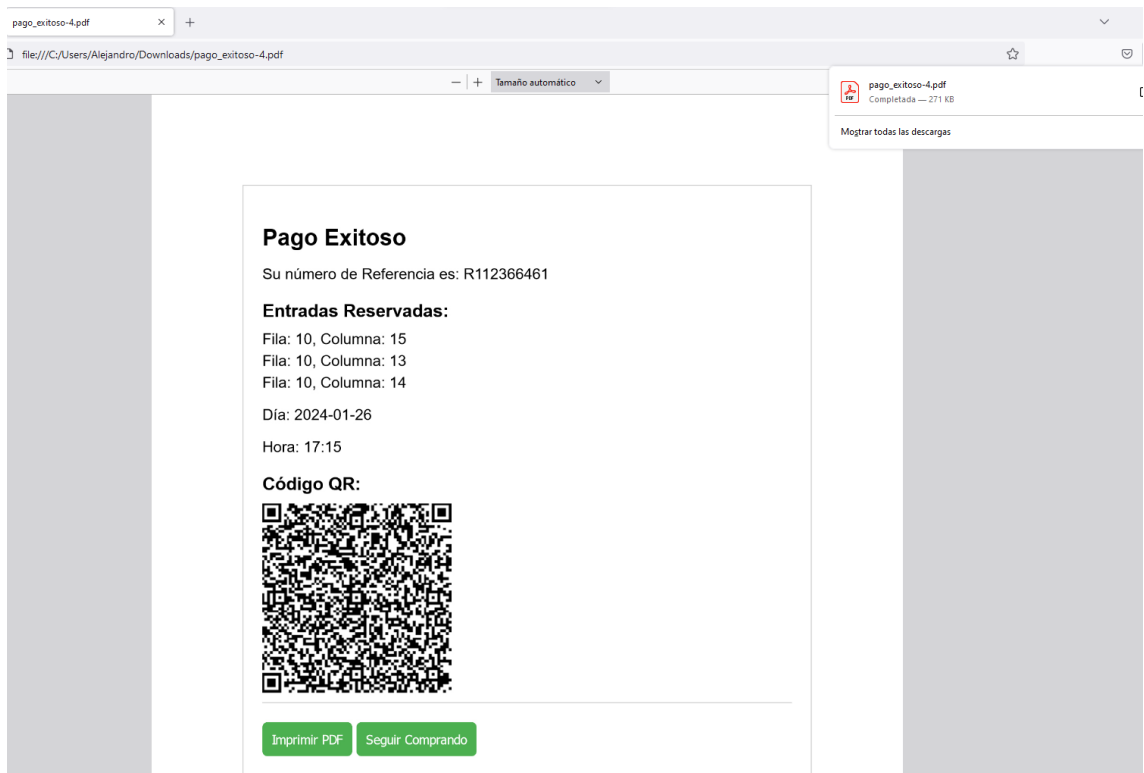
Procesar pago

Se puede introducir uno propio o generar un random, pero si ya esta creada recomiendo introducirla manual. AL pulsar en “procesar pago” nos aparecerán los datos de nuestra reserva, junto al QR que almacena los datos de esta para que puedan ser comprobados por el cine en cuestión.

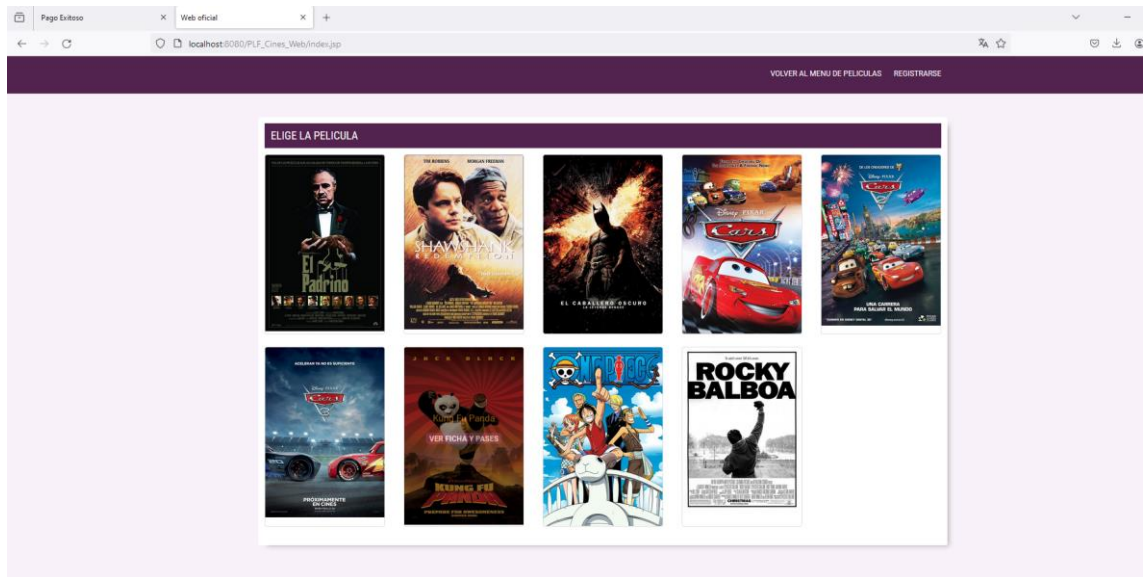


Aquí tenemos dos opciones, una, que es generar el pdf de la página o volver para poder seguir comprando.

- Pulsando “imprimir pdf”:



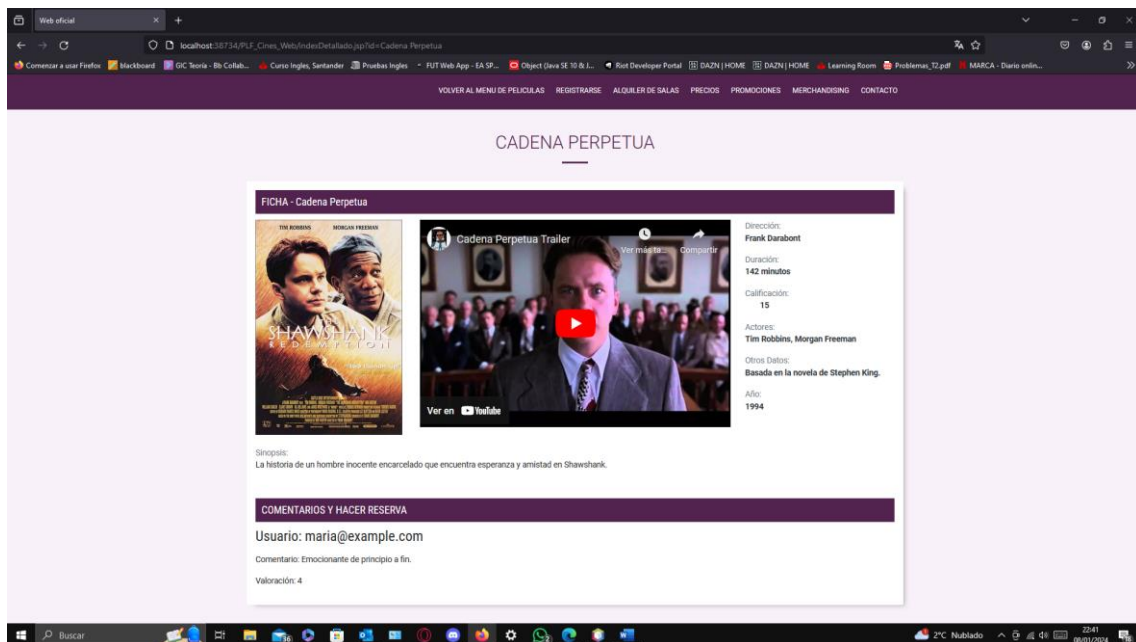
- Pulsando “Seguir Comprando”:



Aquí la guía vuelve al principio, el usuario sigue logueado, por lo que puede interactuar con la página de manera habitual.

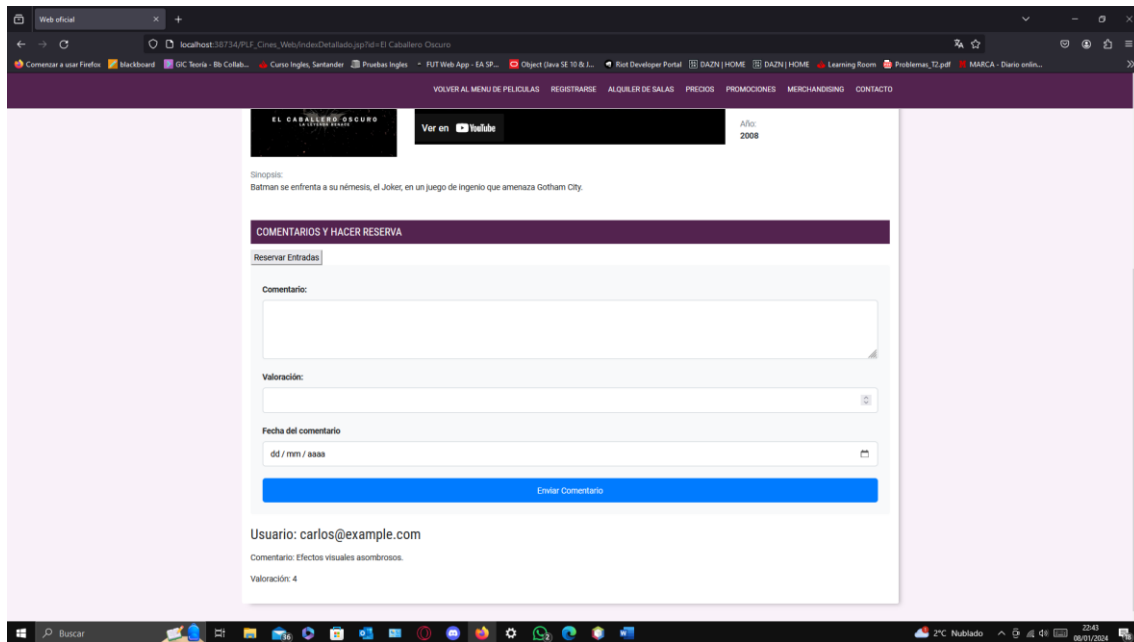
### Comentarios.

Si accedemos a la aplicación sin estar registrados podremos ver los comentarios de las películas, pero no podremos hacer comentarios y se verá tal que así:



Solo veremos los comentarios de la gente, ahora para poder hacer comentarios nos registramos:





Web oficial

localhost:38734/PLF\_Cine/Web/IndiceDetallado.jsp?id=El Caballero Oscuro

VOLVER AL MENÚ DE PELÍCULAS REGISTRARSE ALQUILER DE SALAS PRECIOS PROMOCIONES MERCHANDISING CONTACTO

EL CABALLERO OSCURO

Ver en YouTube

Año: 2008

Shoppis:  
Batman se enfrenta a su némesis, el Joker, en un juego de ingenio que amenaza Gotham City.

COMENTARIOS Y HACER RESERVA

Reservar Entradas

Comentario:

Valoración:

Fecha del comentario  
dd / mm / aaaa

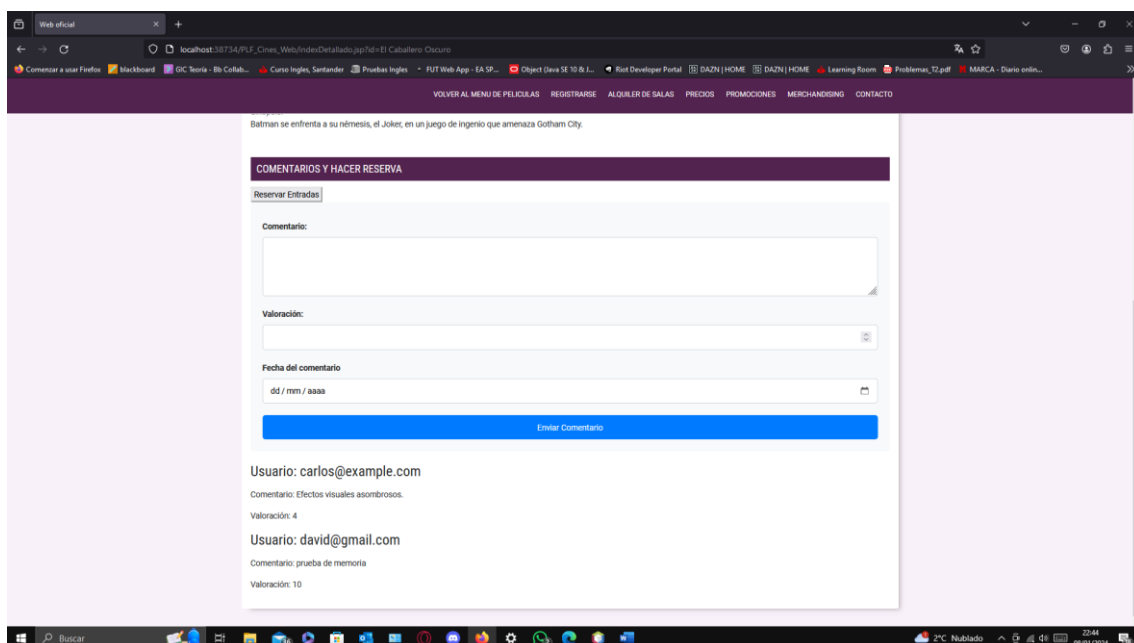
Enviar Comentario

Usuario: carlos@example.com

Comentario: Efectos visuales asombrosos.

Valoración: 4

Ahora si podemos hacer un comentario, vamos a hacer uno de prueba para el caballero oscuro:



Web oficial

localhost:38734/PLF\_Cine/Web/IndiceDetallado.jsp?id=El Caballero Oscuro

VOLVER AL MENÚ DE PELÍCULAS REGISTRARSE ALQUILER DE SALAS PRECIOS PROMOCIONES MERCHANDISING CONTACTO

EL CABALLERO OSCURO

Ver en YouTube

Año: 2008

Shoppis:  
Batman se enfrenta a su némesis, el Joker, en un juego de ingenio que amenaza Gotham City.

COMENTARIOS Y HACER RESERVA

Reservar Entradas

Comentario:

Valoración:

Fecha del comentario  
dd / mm / aaaa

Enviar Comentario

Usuario: carlos@example.com

Comentario: Efectos visuales asombrosos.

Valoración: 4

Usuario: david@gmail.com

Comentario: prueba de memoria

Valoración: 10

Y si consultamos la base de datos vemos también que se ha almacenado mediante gestión de sesiones el correo del usuario y el nombre de la película.

SELECT * FROM APP.COMENTA...					
Max. rows: 100 Fetched Rows: 7					
#	TEXTO	VALORACION	FECHA_COMENTARIO	EMAIL_USUARIO	NOMBRE_PELICULA_PELICULA
1	Una obra maestra, actuaciones increíbles.	5	2023-01-05	rafael@example.com	El Padrino
2	Emocionante de principio a fin.	4	2023-01-06	maria@example.com	Cadena Perpetua
3	Efectos visuales asombrosos.	4	2023-01-07	carlos@example.com	El Caballero Oscuro
4	Larga pero intensa perfecta para una noche ...	10	2024-01-05	david@gmail.com	El Padrino
5	Siempre sera mi pelicula favorita de todos lo...	10	2024-01-05	david@gmail.com	Cars
6	Super divertida y entretenida de ver	10	2024-01-07	david@gmail.com	One Piece
7	prueba de memoria	10	2024-01-08	david@gmail.com	El Caballero Oscuro

## Conclusiones

La práctica ha sido un breve acercamiento al mundo de la web de la mano del patrón MVC. Donde hemos encontrado dificultades, sobre todo para comprender bien el funcionamiento del mismo. Pero, sobre todo, a la hora de establecer el área de trabajo, es decir, la interfaz, la base de datos, y el repositorio para poder trabajar en equipo. Una vez solventado esto, el trabajo siguió de manera eficiente y se fueron consiguiendo los objetivos de manera rápida.

Cabe añadir que se han adquirido unos amplios conocimientos de HTML5, CSS3, JavaScript además del uso de librerías fundamentales para la lógica y la presentación de nuestra web, como jquery, html2pdf o qrcode.

En resumen, este proyecto ha proporcionado una experiencia completa en el desarrollo de aplicaciones web, integrando una estructura organizada con tecnologías modernas y características fundamentales. La implementación del patrón MVC ha sido la piedra angular, ofreciendo una base sólida y modular para futuras expansiones y mejoras en la aplicación.

## Referencias

- [1] D. Roberto Barchino Plata. (2023). Arquitectura y diseño de Sistemas Web y C/S. Universidad de Alcalá.
- [2] Apache Derby Documentation. [Online]. Available: <https://db.apache.org/derby/>
- [3] SQL Tutorial. [Online]. Available: <https://www.1keydata.com/es/sql/>
- [4] QRCode.js Documentation. [Online]. Available: <https://davidshimjs.github.io/qrcodejs/>
- [5] Video Tutorial: Introduction to MVC Pattern. [Online]. Available: <https://www.youtube.com/watch?v=1nTT2IJ-gTg>