

Memoria práctica 1

Aplicaciones Distribuidas en Internet e Interfaces de Usuario



Sergio Garcia Puertas



I. INTRODUCCIÓN

El objetivo principal de esta práctica es desarrollar una página web que esté desplegada en un servidor, en nuestro caso hemos utilizado Glassfish, y que, consultando a una base de datos, en nuestro caso XAMPP, mediante llamadas asíncronas, dibuje diferentes gráficos con los datos que se obtienen.

II. MANUAL DE USUARIO

Este manual de usuario está pensado para que una persona ejecute el proyecto de manera local en su ordenador.

A. Instalación

La opción más sencilla es descargarse la máquina virtual donde ya están todos los programas necesarios instalados para ejecutar el proyecto y utilizar la página web. Con tal de poder utilizarla, se debe de instalar utilizando la aplicación *Oracle VM VirtualBox* [1].

Una vez esté la máquina virtual instalada hay que ajustar 3 parámetros de la configuración con tal de evitar que se nos pueda colgar en cualquier momento.

- 1) *Sistema > Placa base*: Darle el máximo de memoria ram disponible en la zona verde que aparece.
- 2) *Sistema > Procesador*: Del total de CPUs que se le puede asignar de la zona verde, darle 1 menos (a no ser que como máximo se le pueda dar 1).
- 3) *Pantalla > Pantalla*: Asignarle la máxima memoria de video posible.

En este punto ya se podrá ejecutar la máquina virtual y no debería dar ningún problema.

B. Ejecución

Primero de todo se debe de abrir la Oracle VM VirtualBox y ahí nos aparecerá la máquina virtual del proyecto, de nombre "MV2020". Al seleccionarla pulsamos el botón de Iniciar y esperamos a que cargue. Al iniciarse pedirá que se introduzca una contraseña, esta es: dm1lt1m.

Una vez dentro, lo primero que hay que hacer es abrir el programa XAMPP. Para esto hay que abrir un terminal y ejecutar el siguiente comando:

```
sudo / opt/ lampp/ manager-linux-x64.run
```

A continuación se abrirá el panel de control y nos deberemos de ir a la pestaña "Manage Servers". En esta pestaña hay que iniciar el servidor de MySQL (MySQL Database) y el servidor de Apache (Apache Web Server).

Ahora que ya está la base de datos en marcha, debemos abrir Netbeans, cargar el proyecto "ADIIUp1" que hay en el escritorio y ejecutarlo.

Al ejecutarlo se abrirá el navegador y únicamente veremos una barra de navegación y la imagen de la figura 1 repetida.



Fig. 1: Imagen de cargando

Esto se debe a que han de cargar tres gráficos que obtienen los datos haciendo una petición a la base de datos. Entonces hasta que se obtengan los datos y se puedan dibujar las gráficas se muestra el gif de *cargando*.

La página a la que se accede por defecto es la parte pública. Si se desea acceder a la página privada, es necesario acceder con un usuario. Dándole al botón de "Registrarse" de la barra de navegación nos aparecerá el desplegable de la figura 2.

Registrate

Usuario

Contraseña

Confirma la contraseña

Entrar Cancelar

Fig. 2: Ventana para registrarse

Una vez se introducen los datos, si son válidos, al darle al botón de “entrar” se crea el usuario y se inicia sesión de manera automática.

Si las contraseñas no son iguales, aparece la advertencia de la figura 3.

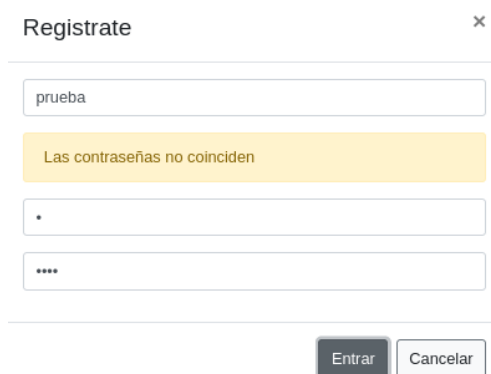


Fig. 3: Contraseñas no coinciden

Si se va a registrar un usuario que ya existe, aparece la advertencia de la figura 4.



Fig. 4: Usuario ya existe

Por otro lado, si ya se tiene un usuario y se quiere acceder con este, hay que seleccionar el botón de “Iniciar Sesión” que aparece en la barra de navegación e introducir la información. Aparecerá el desplegable de la figura 5.

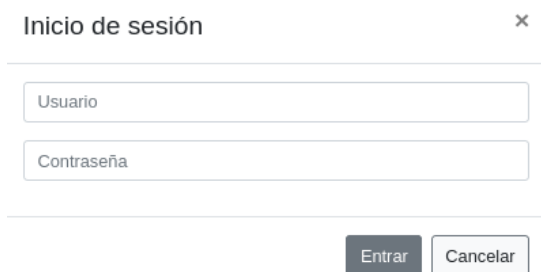


Fig. 5: Iniciar sesión

Si introducimos unos datos de un usuario que no existe, aparecerá la advertencia de la figura 6.

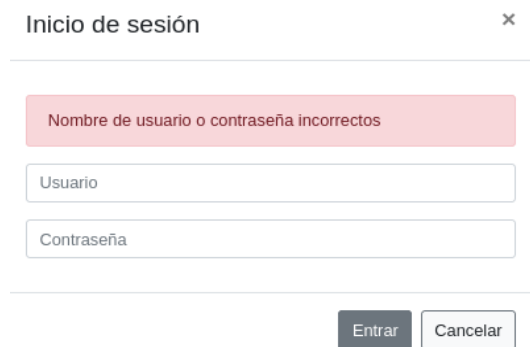


Fig. 6: Usuario incorrecto

Una vez se haya iniciado sesión, aparecerá el nombre de usuario en la barra de navegación. Si se desea ir a la página privada, habrá que hacer click en la flecha roja que aparece en la parte superior derecha (figura 7).



Fig. 7: Ir a la página privada

Al acceder a la parte privada únicamente se verán dos elementos. Uno que contiene checkboxes y otro que contiene información (aunque aparecerán todos los campos vacíos).

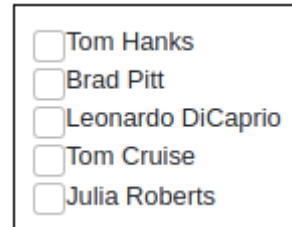


Fig. 8: Elemento con checkboxes

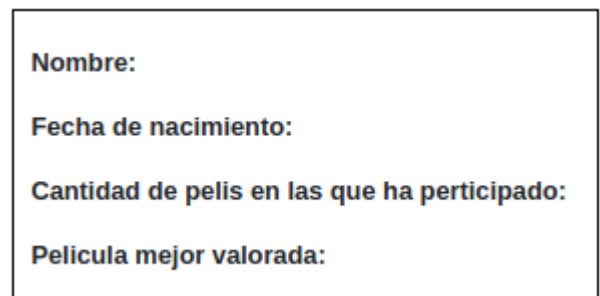


Fig. 9: Elemento con información

Una vez clickemos en una de les checkboxes, se enviará una petición a la base de datos, y cuando obtengamos esa información, se rellenarán los campos que aparecen en la figura 9 y se pintará una gráfica en función de la cantidad de

películas en las que haya participado cada actor/actriz de los que tengamos seleccionados.

Si se desea, se puede volver a la parte pública clickando en el botón de arriba a la izquierda (figura 10).



Fig. 10: Ir a la página pública

Por último, independientemente de en qué página nos encontremos, podremos cerrar sesión apretando el botón de la derecha de la barra de navegación “Cerrar”. Si se clicka estando en la página privada se redirige a la página pública.

III. DISEÑO

Antes de empezar a codificar hicimos un diseño previo de la página web utilizando el software Adobe XD [2]. En las imágenes siguientes (figuras 11 y 12) podemos ver el resultado de este diseño preliminar.

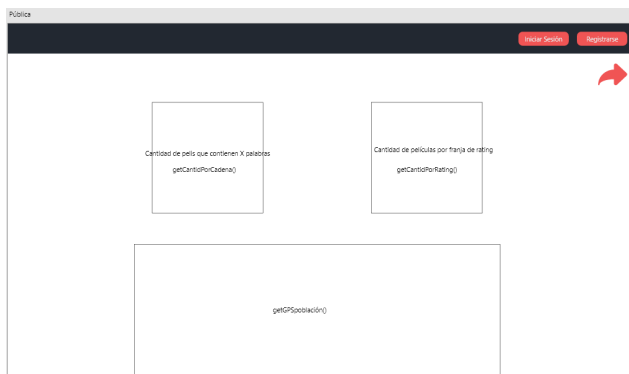


Fig. 11: Diseño previo - parte pública

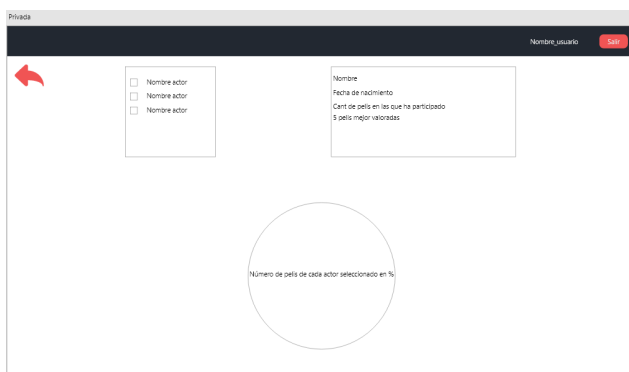


Fig. 12: Diseño previo - parte privada

La página web consta de 2 páginas, una pública y una privada. Ambas páginas tienen un elemento común, una barra de navegación superior.

En caso de no haber iniciado sesión encontraremos en ella dos botones. Uno para iniciar sesión con una cuenta ya existente y otro para crear una cuenta nueva.

En caso de haber iniciado sesión desaparecerán los botones de inicio de sesión y registro y aparecerán en su lugar nuestro nombre de usuario junto a un botón para cerrar la sesión. Para la implementación de los estilos de la página web se ha hecho uso de la librería bootstrap en su versión 4.4.1 [3].

A. Parte pública

Está formada por dos gráficas y un mapa, además de un botón en la parte superior derecha que nos permite acceder a la parte privada de la web.

La primera gráfica muestra la cantidad de películas de la base de datos que contienen una determinada palabra, la selección de palabras ha sido realizada de forma arbitraria.

La segunda gráfica muestra la cantidad de películas que tienen una determinada nota divididas por rangos. Es decir, nos muestra el porcentaje de películas que tienen entre un 0 y un 2, entre un 2 y un 4, etc.

Finalmente, el mapa consiste en un mapa de España en el que se han indicado las capitales de las comunidades autónomas. Al situar el puntero sobre cada una de ellas nos mostrará su nombre completo además de sus coordenadas. En la figura 13 podemos ver el resultado final.

B. Parte privada

Los elementos más importantes de esta página son el recuadro de información y la gráfica interactiva. La página contiene un recuadro con checkboxes en las que podremos seleccionar uno o varios actores cuyos datos se mostrarán en el recuadro de información.

La gráfica mostrará, en forma de porcentajes, una comparación del número de películas en las que ha participado cada actor/actriz.

Además de estos elementos, encontraremos un botón en la parte superior izquierda que nos permitirá volver a la parte pública. En la figura 14 podemos ver el resultado final.

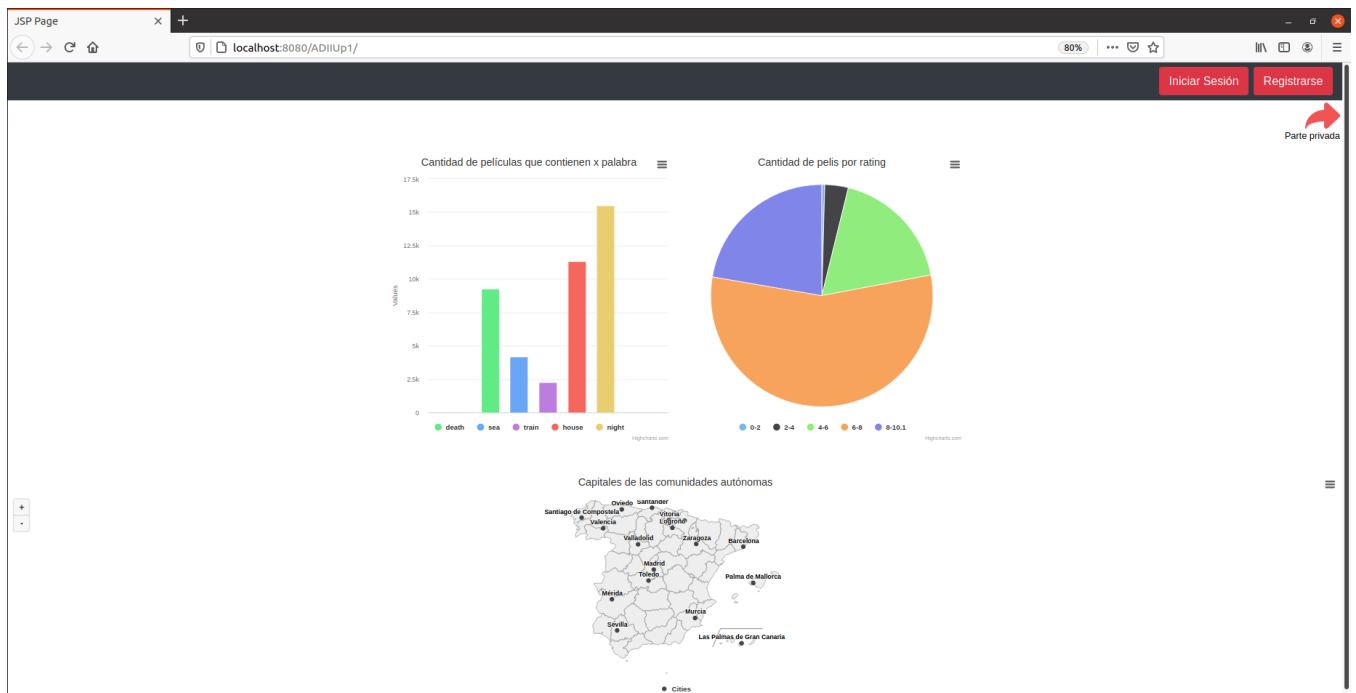


Fig. 13: Resultado final - parte pública

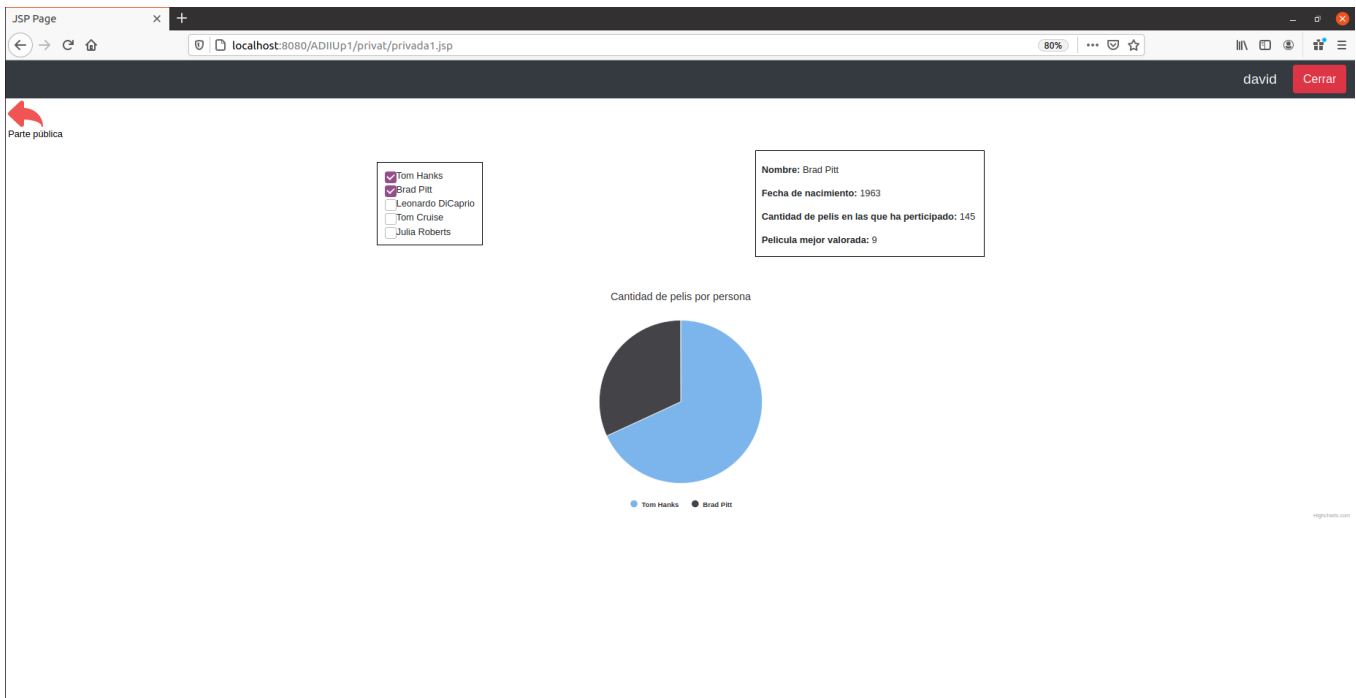


Fig. 14: Resultado final - parte privada

IV. ARQUITECTURA

A. Funcionamiento

En la figura 15 podemos ver el esquema de la arquitectura que se ha utilizado para realizar esta práctica.

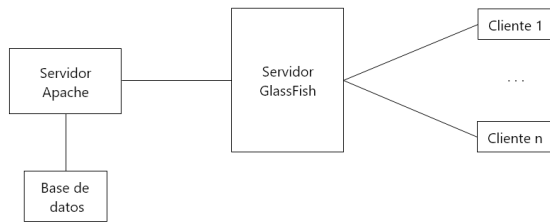


Fig. 15: Arquitectura

Al ejecutar el proyecto de netbeans, se despliega el servidor de GlassFish así como el proyecto en concreto. Una vez el servidor se está ejecutando podremos acceder a la página web accediendo a la url correspondiente, en este caso: <http://localhost:8080/ADIIUp1>. Por defecto al acceder a esta url se cargará el archivo llamado *index.jsp*.

Cuando un usuario accede a una url, el servidor envía al cliente la información de los archivos respectivos a esa url. Por ejemplo, si la url <http://ejemplo.com/indice.html> hace referencia a un único archivo html, cuando un cliente acceda a esta url el servidor le devolverá la información de este archivo. De tal manera que el cliente pueda visualizar en su dispositivo el contenido html.

El servidor GlassFish es capaz de interpretar código jsp, por tanto a la hora de enviar la información de un archivo, si se encuentra código jsp, en vez de enviarlo lo ejecuta.

El cliente no puede comunicarse con la base de datos, sólo puede el servidor. Es por esto que una de las funciones que puede realizar el cliente, es hacer peticiones asíncronas (mediante ajax) al servidor web con tal de obtener información de la base de datos (este proceso está mejor especificado en el apartado de *Parte Servidor* a continuación).

Una vez el servidor recibe esta petición, se comunica con el servidor Apache y le envía la operación que debe realizar sobre la base de datos. Entonces el servidor Apache responde al servidor Web, que procesa la información recibida y la envía al cliente si es necesario.

Para esta práctica, el cliente necesita acceder a la base de datos por dos motivos diferentes. Uno es para obtener la información que necesita para crear las gráficas y los mapas. El otro caso es cuando quiere iniciar sesión o registrarse.

B. Parte cliente

La parte del cliente está estructurada en diversos directorios. El directorio raíz de nuestra página contiene las páginas html y jsp correspondientes a la parte pública, además de varios subdirectorios. Estos son:

- 1) *privat*: Contiene la página jsp referente a la parte privada de la página web. En caso de que hubiese varias páginas privadas que utilizasen el mismo nivel de permisos, estarían en este mismo directorio.
- 2) *js*: Contiene todos los archivos javascript de la página web.
- 3) *CSS*: Contiene todos los archivos de estilo CSS de la página web.
- 4) *img*: Contiene imágenes y recursos utilizados.

A continuación se explican detalladamente los ficheros a los que hace referencia cada parte de la página web y cómo se han implementado las diversas funcionalidades.

1) *index.jsp*: contiene el cuerpo de la parte pública de la página web. Aquí encontramos dos gráficas y un mapa.

El proceso de carga de datos de las gráficas y el mapa es el siguiente: Al cargarse la página se envían varias peticiones ajax a un servlet pidiendo parte de la información necesaria para la gráfica. A medida que las peticiones ajax se resuelven, se ejecutan sus funciones de respuesta, que incrementan un contador. Una vez llega la última, el contador llega a su valor máximo y se llama a la función que dibuja la gráfica.

Además de incrementar el contador, las funciones de respuesta también guardan los datos en caché. De esta manera cuando el usuario recargue la página no se volverán a realizar las peticiones, sino que se leerán los datos de la caché.

Para dibujar las gráficas se ha hecho uso de la librería Highcharts [4]. Los archivos que contienen las funciones para dibujar y solicitar los datos de las gráficas son *container1.js* y *container2.js* localizados en la carpeta *js*.

El proceso de carga de datos del mapa es similar al de las gráficas. En este caso toda la información se solicita en una única petición al servidor, por lo que no es necesario ningún contador.

Para dibujar el mapa se ha utilizado la librería Highcharts maps o Highmaps [5]. Dado que ya utilizamos Highcharts en la página, esta librería ha tenido que ser instalada como un módulo de Highcharts en lugar de como una librería por sí sola con el objetivo de evitar conflictos. El archivo que contiene las funciones para dibujar y solicitar los datos del mapa es *container3.js* localizado en la carpeta *js*.

Durante el tiempo anterior a que se carguen los datos de las gráficas y el mapa, se muestra un gif de carga que ocupa su lugar para que el usuario sea consciente de que se está cargando la información.

2) *privada1.jsp*: contiene el cuerpo de la parte privada de la página web. Al marcar una de las checkboxes se invoca a un controlador ubicado en el archivo *container_priv.js*. Este comprobará si la información correspondiente al actor marcado se encuentra en la caché del cliente. Si es así, llamará a una función encargada de rellenar el recuadro de información del actor y a otra función encargada de dibujar la gráfica.

En caso de no tener la información hará una petición ajax al servlet, que consultará la información del actor en la base de datos. Una vez el servlet obtenga la información del actor, esta se guardará en caché para evitar volver a consultarla en el futuro.

Al desmarcar una checkbox se limpiará el cuadro de información y volverá a llamar a la función encargada de dibujar la gráfica. Esta función comprueba el estado de cada una de las checkboxes y utiliza solamente la información de los actores marcados.

Cabe destacar que durante el proceso de carga de la gráfica y mientras se espera a que se actualice, se muestra un gif de carga para informar al usuario de que se está esperando información del servidor.

3) *capcalera.jsp*: es un archivo que se incluye al inicio de *index.jsp* y *privadal.jsp* con el objetivo de controlar el acceso del usuario a la parte privada. También tiene el contenido de la barra de navegación superior y, por tanto, contiene los botones para redirigir al usuario a los programas de inicio/cierre de sesión y de creación de usuario. Esta funcionalidad se ha implementado de la siguiente forma:

Al acceder a la página web, en primer lugar, se leen los datos del usuario almacenados en la sesión (en caso de no haber iniciado sesión estos serán NULL). A continuación, se comprueba si estamos accediendo a una página dentro de la carpeta *privat*.

En caso afirmativo, si los datos del usuario indican que no tiene permiso para acceder a la parte privada, se le redirigirá a la parte pública. En caso contrario se enviará al cliente el código html de la cabecera.

Si el usuario ya ha iniciado sesión, la cabecera contendrá su identificador y el botón de cierre de sesión. En caso contrario, contendrá los botones para que inicie sesión o se cree una cuenta.

Cabe destacar que el código *jsp* comprueba que el nivel de permisos del usuario sea de 1 o superior, el cual es el nivel por defecto al crear una cuenta. Este sistema de permisos permitiría fácilmente su expansión de forma que pueda haber varias páginas para las cuales es necesario un nivel de privilegios distinto para acceder.

Los botones de inicio de sesión y registro nos dan acceso a unos formularios que podremos completar para acceder a la página *login.jsp* o *signup.jsp*. El botón de cierre de sesión nos llevará directamente a la página *logout.jsp*. Estas tres páginas se explicarán más adelante.

Toda la estética, animaciones y gestión de errores durante los procesos de inicio de sesión y registro se han implementado mediante una combinación de bootstrap y código javascript con jquery [6] ubicado en el archivo *signinup.js*.

4) *login.jsp*: es el encargado de realizar (o no) el inicio de sesión de un usuario. A esta página se envían, desde el formulario de inicio de sesión, el nombre de usuario y contraseña con los que se está intentando iniciar sesión. Estos

se envían al método *getUserAccess()* el cual nos devolverá el nivel de permisos que tiene el usuario.

En caso de que ese par de usuario-contraseña no se encuentre en la base de datos, la método devolverá como nivel: -1. Si el nivel devuelto por el método es mayor o igual a 0 se guardará la información del usuario en la sesión del servidor y se redirigirá al usuario de vuelta a la página desde la cual había venido.

En caso de que el usuario no exista o la contraseña sea incorrecta, se devolverá al usuario a la página desde la que había venido, en la que se le mostrará un mensaje de error.

Cabe destacar, que ya que todo esto se realiza con código *jsp*, que se ejecuta en el servidor, en lugar de realizar una petición HTTP al servlet, la llamada al método se realiza de forma directa dentro del código *jsp*.

5) *signup.jsp*: es el encargado de la introducción de nuevos usuarios en la base de datos. En primer lugar, recoge el nombre de usuario y la contraseña que se le han enviado desde el formulario de registro. A continuación, llama al método *checkUser()* que comprueba si existe un usuario en la base de datos con el mismo nombre que el usuario que se pretende crear.

En caso afirmativo se llama a *createUser()* pasándole por parámetros el nombre de usuario y la contraseña. Se introduce en la base de datos una entrada para el nuevo usuario y a continuación, se introducen los datos del usuario en la sesión y se le redirige de vuelta a la página de la que vino.

En caso de que el usuario que se intenta crear ya exista en la base de datos, se redirige al usuario a la página de la que vino, en la que se le mostrará un mensaje de error.

6) *logout.jsp*: es una página que elimina los datos de la sesión y después redirige al usuario de vuelta a la página desde la que ha venido.

C. Parte servidor

En la parte del servidor encontramos dos paquetes de java. Primero está el paquete *meuservlets*, que contiene un único servlet llamado *bdpeliculas.java*.

Un servlet es una clase java que es capaz de atender a peticiones HTTP y responder [7]. Para crear un servlet propio necesitamos crear una clase java nueva y que ésta herede de la clase *HttpServlet*. Además, se debe definir un nombre (*name*) y un patrón de url (*urlPatterns*). En nuestro caso, el nombre es “bdpeliculas” y el patrón de url es “/bdpeliculas”.

El atributo *name* debe ser único para cada servlet y sirve para mapear la instancia de java con la url definida [8]. El atributo *urlPatterns* define la url con la que se va a poder realizar peticiones al servlet. En nuestro caso, la url completa es: <http://localhost:8080/ADIIUp1/bdpeliculas>.

Para poder enviar información en la petición HTTP, se deben usar parámetros al final de la url. El inicio de los parámetros se indica con un “?”. El formato de los parámetros es “clave”=“valor”. Si se desea enviar varios parámetros estos estarán separados por un “&”.

La principal función de nuestro servlet es proporcionar al usuario información de la base de datos, ya que este no puede acceder directamente a la misma.

La base de datos que hemos utilizado tiene el siguiente esquema.

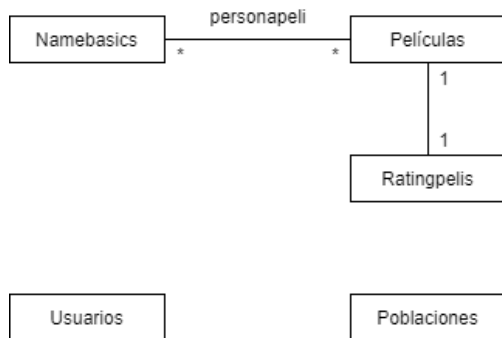


Fig. 16: Modelo de datos

Como se puede observar, la base de datos contiene cinco clases distintas, aunque no todas están relacionadas. En concreto, las clases *Usuarios* y *Poblaciones* no están relacionadas con ninguna otra clase. A continuación podemos encontrar una descripción de cada una de las tablas que forman la base de datos.

1) *Namebasics*: contiene un identificador, el nombre, la fecha de nacimiento y la fecha de defunción de una serie de actores/actrices. En caso de no haber fallecido, la fecha de defunción coge el valor de -1.

2) *Películas*: contiene un identificador y el título original de una serie de películas.

3) *Personapeli*: tabla que surge a partir de la relación entre *Namebasics* y *Películas* y que contiene, para cada relación, el identificador del actor/actriz y de la película en la que ha participado.

4) *Ratingpelis*: contiene el identificador de la película, la cantidad de votos que le han dado y la valoración media.

5) *Poblaciones*: contiene una lista de poblaciones en la que se especifican el código del país según la norma ISO 3166-1 alfa-2 [9], el nombre de cada población, su latitud y su longitud.

6) *Usuarios*: tabla en la que se almacenan los usuarios registrados en la página web. Contiene el nombre de usuario, contraseña y nivel de privilegios.

El servlet tiene dos métodos que escuchan a peticiones HTTP. El método `doGet()` atiende a peticiones de tipo GET y el método `doPost()` atiende a peticiones de tipo POST. Ambos métodos llaman al método `processRequest()`. Este método define el contenido de la `HTTPResponse` como `"text/html; charset=UTF-8"`, llama al método `gestionPetición()`, que es el que obtiene la respuesta adecuada en función de la petición, y por último envía la respuesta al usuario.

El método `gestionPetición()` lo primero que hace es obtener el valor del parámetro "op", que indica la operación que desea realizar el usuario, y también obtiene el valor del parámetro "par", que es la información que se necesita para realizar la operación indicada.

Entonces, mediante *else if* hay definidos una serie de posibles casos según el valor del parámetro "op". De esta manera, llamará al método para realizar la operación correspondiente, y le asignará el resultado a la variable "res", que es la que se le devuelve al método `processRequest()` y que por tanto es la que contiene la información que se le enviará de vuelta al usuario.

Las diferentes operaciones que se pueden realizar están divididas en archivos java que se encuentran en el paquete *peradb*. A continuación se explicará qué métodos hay en cada uno de estos archivos.

1) *DBActionsGlobal*: Esta clase la hemos creado nosotros y contiene el método que se utiliza para conseguir todos los datos necesarios de un actor/actriz con tal de poder mostrar toda la información en la página privada. Este método lo hemos añadido nosotros y, dado que coge información de diferentes tablas, no encajaba en ninguna de las clases que ya venían definidas por el profesor.

a) *getInfoTabla()*: Recibe como parámetro el nombre de un actor/actriz y realiza una consulta a la base de datos para obtener su nombre, fecha de nacimiento, cantidad de películas en las que ha participado y la película con mejor valoración entre las que ha participado. La consulta es la siguiente:

```

SELECT primaryname, birthyear, COUNT(*) cant,
MAX(ratio) maxima from namebasics JOIN personapeli ON
namebasics.nconst = personapeli.nconst JOIN peliculas ON
personapeli.tconst = peliculas.tconst JOIN ratingpelis ON
peliculas.tconst = ratingpelis.tconst WHERE primaryname =
'[cadena]';, donde [cadena] es el nombre del actor/actriz que
recibe el método por parámetro.
  
```

La información se devuelve en un String que sigue el formato {"nombre" : "primaryname", "fecnac" : "birthyear", "cantpelis" : cant, "bestpeli" : maxima}.

2) *DBActionsNameBasics*: Esta clase contiene métodos que consultan únicamente la tabla "namebasics" de la base de datos.

a) *getTodosPorEdad()*: Devuelve un listado en formato JSON con todos los actores/actrices que tengan la edad que se le pasa al método por parámetro, o que hayan fallecido con esa edad.

b) *getCantidadPorEdad()*: Funciona igual que el método anterior, pero en vez de devolver un listado con la información de los diferentes actores/actrices, devuelve la cantidad de actores/actrices que cumplen las dos condiciones mencionadas.

c) *getCantidadPorFranja()*: Funciona igual que el método anterior, pero en vez de comprobar una edad en concreto, comprueba una franja de edad.

3) *DBActionsPeliculas*: Esta clase contiene métodos que consultan únicamente la tabla “peliculas” de la base de datos.

a) *getPelisPorCadena()*: Devuelve un listado en formato JSON con todas las películas que contienen un string que se le pasa al método como parámetro.

b) *getCantidPorCadena()*: Funciona igual que el método anterior, pero devuelve la cantidad de películas que contienen el string en vez de un listado. Originalmente la consulta solo era capaz de reconocer películas que contenían el string en el interior del título. Modificamos la consulta para que sea capaz de reconocer también películas en las que la cadena se encuentra al inicio, al final o cuyo título sea únicamente la cadena.

4) *DBActionsPersonaPeli*: Esta clase contiene métodos que obtienen información para la cual es necesaria consultar un conjunto de tablas. Concretamente “namebasics”, “personapeli” y “peliculas”.

a) *getPelisDePersona()*: Devuelve un listado en formato JSON de todas las películas en las que ha participado un actor/actriz que se le pasa al método por parámetro.

b) *getPersonasDePeli()*: Devuelve un listado en formato JSON de todas las personas que han participado en una película que se le pasa al método por parámetro.

5) *DBActionsPoblaciones*: Esta clase contiene un único método que consulta la tabla “poblaciones” y se utiliza para obtener información de las poblaciones para el mapa de la parte pública de la página web.

a) *getGPSPoblacion()*: dada una serie de poblaciones españolas separadas por “-”, que recibe el método por parámetro, devuelve un listado en formato JSON con el nombre, latitud y longitud de cada población. Modificamos la función original de forma que la estructura del JSON se adaptara a nuestras necesidades al trabajar con Highmaps.

6) *DBActionsRatingPelis*: Esta clase contiene métodos que consultan las tablas “película” y “ratingpelis” de forma que se pueda obtener información sobre el rating de una película partiendo de su título.

a) *getPelisDeRating()*: Devuelve un listado en formato JSON con todas las películas que tienen una valoración media dentro de un rango que se le pasa al método como parámetro.

b) *getCantidPorRating()*: Devuelve la cantidad de películas que tienen una nota media dentro de un rango que se le pasa al método como parámetro. Este método ha sido añadido por nosotros para obtener la información necesaria para la segunda gráfica de la parte pública. La consulta que realizamos en la base de datos es la siguiente:

```
SELECT count(tconst) FROM ratingpelis WHERE (( ratio >= [nota mínima]) and (ratio <= [nota máxima]));
```

donde [nota mínima] y [nota máxima] son los valores indicados por el rango recibido como parámetro.

7) *DBActionsUsers*: Esta clase contiene métodos que interactúan con la tabla “Usuarios” de la base de datos.

a) *getUserAccess()*: Dado un nombre de usuario y una contraseña por parámetro, mira si el usuario existe y devuelve su nivel de privilegios. Si el usuario no existe, devuelve -1. Por simplicidad, modificamos este método de forma que en lugar de devolver un string en formato JSON cuyo único parámetro es el nivel, devuelva el nivel como un entero.

b) *createUser()*: Hemos añadido este método con el objetivo de poder crear usuarios desde nuestro sitio web. Dado un nombre de usuario y una contraseña por parámetro, introduce una entrada de un nuevo usuario en la base de datos. Dado que no hemos implementado ninguna funcionalidad en la que se puedan utilizar los diferentes niveles de privilegios, todos los usuarios se crean con nivel 1.

c) *checkUser()*: Antes de crear un usuario, es necesario comprobar que no exista ningún otro con el mismo nombre en la base de datos. Por ese motivo hemos creado esta función booleana encargada de realizar esa comprobación. Dado un nombre de usuario comprueba si existe un usuario con dicho nombre. En caso de que lo encuentre devolverá TRUE, en caso contrario devolverá FALSE.

8) *DBConnection*: Esta clase contiene métodos para establecer (y cerrar) una conexión con la base de datos de forma sencilla. También contiene un atributo “con” en el cual se almacenará la conexión hasta el momento en que se cierre la misma.

9) *DBProperties*: Esta clase contiene los parámetros que se utilizarán para establecer la conexión con la base de datos. Esta clase es utilizada solamente por métodos de la clase *DBConnection*.

V. DIFICULTADES

Pese a que no hemos tenido ninguna dificultad que nos haya supuesto un problema muy grande, si hemos tenido unos cuantos problemas que nos han hecho tener que dedicarle más tiempo a la práctica.

De los dos integrantes, a David no le funcionaba la máquina virtual en su ordenador de sobremesa ya que al rato de abrir cualquier aplicación, esta se cerraba indicando que había ocurrido un error con el sistema operativo. Afortunadamente no hubo ningún problema al utilizar la máquina virtual en su ordenador portátil por lo que esto solo fue un inconveniente menor.

A Sergio, de vez en cuando y de forma aleatoria, le dejaba de funcionar el cuarto superior izquierdo de la pantalla. Seguía visualizando todo bien, pero no permitía hacer clicks. Esto tampoco suponía mucho problema ya que bastaba con reiniciar la máquina virtual y ya volvía a funcionar todo bien.

Otro problema que nos encontramos es que el servidor de Apache por defecto utiliza el puerto 80 y al iniciarlo nos daba error porque nos decía que ya había un proceso utilizando ese puerto. Miramos qué proceso era e indicaba que era el proceso de Apache. Intentamos matarlo y volver a iniciar el servidor, pero el proceso volvía a iniciarse de manera automática y por tanto nos volvía a dar el mismo error. Finalmente lo que hicimos fue cambiar el puerto que utiliza el servidor de Apache (elegimos el 5050) y ya funcionó bien.

VI. BIBLIOGRAFÍA

- [1] Oracle VM VirtualBox, <https://www.oracle.com/es/virtualization/technologies/vm/downloads/virtualbox-downloads.html>.
- [2] Adobe XD, <https://www.adobe.com/es/products/xd.html>
- [3] Bootstrap, <https://getbootstrap.com/docs/4.4/getting-started/introduction/>
- [4] Highcharts, <https://www.highcharts.com/blog/products/highcharts/>
- [5] Highcharts maps, <https://www.highcharts.com/products/maps/>
- [6] JQuery, <https://jquery.com/download/>
- [7] Qué es un servlet, <https://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html>
- [8] Atributo name del servlet, <https://help.perforce.com/hydraexpress/4.7.0/html/rwfservletug/4-8.html#:~:text=The%20servlet%20name%20element%20declares.the%20URL%20of%20the%20servlet.>
- [9] ISO 3166-1 alfa-2 - Wikipedia, https://es.wikipedia.org/wiki/ISO_3166-1
- [10] Qué es un servicio web https://es.wikipedia.org/wiki/Servicio_web#:~:text=Un%20servicio%20web%20
- [11] Qué es un servidor de aplicaciones, https://es.wikipedia.org/wiki/Servidor_de_aplicaciones
- [12] Diferencias entre Servidor web y Web Service, <https://sistemaniatico.wordpress.com/2010/05/04/diferencia-entre-servidor-web-y-web-service/>
- [13] Java EE, https://es.wikipedia.org/wiki/Java_EE
- [14] Qué es un servicio web, <https://www.cleo.com/blog/knowledge-base-web-services>