



PRÁCTICA 2

Desarrollo e integración de software



20 DE ENERO DE 2021

REALIZADO POR:

Jesús Tejada, David Arranz, David González y Marcos Rodríguez

Indice

1. Descripción de la práctica
 - a. Parte 1: Desarrollo de la aplicación.
 - b. Parte 2: Despliegue de la aplicación

2. Descripción del código
 - a. CrudWithVaadinApplication
 - b. MainView
 - c. Actores
 - d. Películas
 - e. Bases de datos
 - i. ActoresBBDD
 - ii. Películas BBDD

3. Funcionamiento de la interfaz de la aplicación

4. Conclusiones generales de la práctica

1. Descripción de la práctica

El objetivo principal de esta práctica es demostrar el contenido adquirido a lo largo de la asignatura. Para ello crearemos una aplicación en java, la que uniremos a una base de datos para poder comprobar su utilización. Posteriormente configuraremos un proceso CI/CD y desplegaremos la misma.

Para la realización de la aplicación utilizaremos git hub para subir los contenidos, y para el desarrollo de la misma en java utilizaremos Vaadin con el Docker al igual que lo configuraremos con Maven . También realizaremos la gestión de dependencias, el control de versiones y las pruebas unitarias. Necesitaremos utilizar TDD, Paas y Jenkins.

Parte 1: Desarrollo de la aplicación

- Utilización del framework **Vaadin 8.6.2**
- Todas las dependencias deberán ser gestionadas por **Maven**.
- Deberán escribirse **pruebas unitarias** para comprobar el correcto funcionamiento de la aplicación con un alto porcentaje de **coverage**.

Parte 2: Despliegue de la aplicación

- Configuramos el proyecto para poder ejecutarlo
- Montamos una imagen de Jenkins en Docker
- Creamos el Job en Jenkins
 - Pasamos las pruebas con MVN test y guardamos el log de ejecución.

2. Descripción del código

CrudWithVaadinApplication

```
@Bean
public CommandLineRunner loadData(PelículasBBDD repository, ActoresBBDD aut) throws FileNotFoundException {
    // abrimos el archivo .json
    // esto es como la practica 1
    JsonParser parser = new JsonParser();
    Object object = parser.parse(new FileReader("Películas.json"));
    JsonObject gsonObj = (JsonObject) object;
    //accedemos a la videoteca
    gsonObj = gsonObj.getAsJsonObject("Videoteca");
    gsonObj = gsonObj.getAsJsonObject("Películas");
    JSONArray demarcation = gsonObj.get("Película").getAsJSONArray();
```

En un primer lugar deberemos abrir el archivo .json en el que se guardaran todas nuestras modificaciones de las películas llamado “películas.json”. Una vez creado accederemos a la videoteca al igual que crearemos otro objeto para las películas.

```
return (args) -> {
    // extraemos todos los datos del archivo Películas.Json y los guardamos en la base de Datos
    for (JsonElement demarc : demarcation) {
        String titulo = ((JsonObject) demarc).get("Titulo").getString();
        String sinopsis = ((JsonObject) demarc).get("Sinopsis").getString();
        String imbd = ((JsonObject) demarc).get("IMBD").getString();
        String genero;
        try{
            genero = ((JsonObject) demarc).get("Genero").getString();
        }catch (Exception e) { // por si no tiene genero
            genero="None";
        }int numeroActores=0;
        String[] nombreA = new String[10]; // suponemos que el maximo numero de actores va a ser 10, esto se puede modificar
        String[] enlaceP = new String[10];
```

A continuación extraeremos todos los archivos que hemos añadido en nuestro “Películas.json” los que añadiremos directamente dentro de la base de datos, estos serán Título, Sinopsis, IMBD, Género, Reparto, y el Actor con sus características, Nombre y Enlace. También hemos supuesto que la cantidad de autores máximos es de 10, pero este dato puede ser fácilmente modificado.

```
// ponemos un try catch en el caso de que haya 1 actor o varios actores
try { // si hay mas de un actor
    JSONArray autores = autors.get("Actor").getAsJSONArray();
    for (JsonElement demarc1 : autores) {
        String nmb = ((JsonObject) demarc1).get("Nombre").getString();
        String EnlaceWiki = ((JsonObject) demarc1).get("EnlaceWikipedia").getString();
        nombreA[numeroActores] = nmb;
        enlaceP[numeroActores] = EnlaceWiki;
        numeroActores += 1;
    }
} catch (Exception e1) { // si solo hay un actor, dara una excepcion y se metera aqui
    autors = autors.getAsJsonObject("Actor");
    numeroActores = 1;
    String nmb = autors.get("Nombre").getString();
    String EnlaceWiki = autors.get("EnlaceWikipedia").getString();
    nombreA[0] = nmb;
    enlaceP[0] = EnlaceWiki;
}
```

Este try catch para comprobar si son 1 o varios actores en la película, en el caso de que sean varios se creara un array para el cual pedirá para cada uno de ellos tanto su nombre como el enlace a su Wikipedia. En el caso de que solo haya un actor se inicializará la excepción y se preguntará lo mismo, pero no será un array, sino que solo lo realizará una única vez.

```
}
//guardamos la pelicula
Películas nuevo = new Películas(titulo, sinopsis, genero, imbd, numeroActores);
repository.save(nuevo);
Long idpeli=nuevo.getId();
//guardamos los autores, para asociar los autores con las películas, lo que hacemos es
// pasar a la tabla de los actores el id de la película, que esta asociada a dicho actor
for(int i=0;i<numeroActores;i++){
    Actores nuevoAutor= new Actores(nombreA[i],enlaceP[i],idpeli);
    aut.save(nuevoAutor);
}
```

Por último, guardamos la película con todas sus características al igual que los autores para poder asociar estos mismos a las películas. Para poder unirlo en la base de datos pasamos en la tabla de actores el ID de su película.

MainView

```
public MainView(ActoresBBDD aut,PeliculasBBDD repo) {
    //inicializamos las variables
    this.repo = repo;
    this.grid = new Grid<>(Películas.class);
    this.filter = new TextField();
    this.addNewBtn = new Button( text: "Nueva Película", VaadinIcon.PLUS.create());
    this.Exportar = new Button( text: "Exportar", VaadinIcon.ARROW_CIRCLE_RIGHT.create());

    // build layout
    HorizontalLayout actions = new HorizontalLayout(filter, addNewBtn,Exportar);
    add(actions, grid);
}
```

Primero inicializamos las variables, y creamos nuestro layout con nuestros tres elementos para poder buscar una película, añadir una nueva y subirla.

```
grid.setHeight("300px");
grid.removeColumnByKey( columnKey: "id");
grid.setColumns("titulo", "sinopsis", "genero", "imbd", "numeroDeActores");
filter.setPlaceholder("Buscar por Título");
filter.setValueChangeMode(ValueChangeMode.EAGER);
filter.addValueChangeListener(e -> listCustomers(e.getValue()));
grid.asSingleSelect().addValueChangeListener(e -> {
    modal(aut,e.getValue(),repo);
});|
addNewBtn.addClickListener(e -> modalnuevapelicula(aut,repo));
```

Primero ponemos un height de 300, indicamos la cantidad y que columnas va a tener nuestro Grid y le ponemos un placeholder al filter. Cuando el usuario busque una película se ira actualizando al igual que cuando se quiera añadir una nueva película se abrirá el modal correspondiente con cada una de las características de esta.

```
// si queremos guardar la lista de peliculas actuales
Exportar.addClickListener(e ->
{
    try {
        //guardamos en el json
        guardamosenjson(aut,repo);
        // notificacion, para avisar al usuario que se ha exportado correctamente
        Notification.show("Datos exportados correctamente a Peliculas.json");
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
}
);

// inicalizamos la lista
listCustomers( filterText: null);
```

Ahora simplemente guardaremos la lista que se ha generado de las películas actuales, esto se almacenara en nuestro archivo “Pelciulas.json” y por último recibiremos una notificación para informar al usuario de que la exportación de los datos ha salido con éxito.

```
// el modal, aqui se abrir en detalle la pelicula clicada
void modal(ActoresBBDD aut,Peliculas c,PeliculasBBDD repo) {
    try{
        // declaramos el modal
        Dialog dialog = new Dialog();
        dialog.setCloseOnEsc(false);
        dialog.setCloseOnOutsideClick(false);
        // añadimos al modal los elemntos de la pelicula correspondiente
        dialog.add(new HorizontalLayout(new Html( outerHtml: "<b>Titulo: </b>"), new Text(c.getTitulo())));
        dialog.add(new HorizontalLayout(new Html( outerHtml: "<b>Sinopsis: </b>"), new Text(c.getSinopsis())));
        dialog.add(new HorizontalLayout(new Html( outerHtml: "<b>Genero: </b>"), new Text(c.getGenero())));
        dialog.add(new HorizontalLayout(new Html( outerHtml: "<b>IMBD: </b>"), new Text(c.getImbd())));
        // añadimos los autores
        for (Actores autorActual : aut.findByIdPelicula(c.getId())) {
            dialog.add(new HorizontalLayout(new Html( outerHtml: "<b>Actor: </b>"), new Text(autorActual.getNombre())));
            dialog.add(new HorizontalLayout(new Html( outerHtml: "<b>Enlace: </b>"), new Text(autorActual.getEnlace())));
        }
        // ponemos los botones de editar, donde se abrbira otro modal para asi poder editar la pelicula y otro
        // para salir
        Button confirmButton = new Button( text: "Editar", event -> { dialog.close(); modaleditar(aut,c,repo); });
        Button cancelButton = new Button( text: "Cancelar", event -> { dialog.close(); });
        HorizontalLayout actions2 = new HorizontalLayout(confirmButton, cancelButton);
        dialog.add(actions2);
        //abrimos el modal
        dialog.open();
    }catch (Exception e) {
```

Aquí simplemente como se puede observar, creamos el modal que albergará las distintas características de cada película y añadimos los autores al igual que la referencia a sus datos para poder editarlos. Por ultimo ponemos los botones de Editar y Cancelar.

```
// este modal es para poder editar
void modaleditar(ActoresBBDD aut,Peliculas c,PeliculasBBDD repo) {
    //declaramos el modal
    Dialog dialog = new Dialog();
    dialog.setCloseOnEsc(false);
    dialog.setCloseOnOutsideClick(false);
    //declaramos los textedit y les ponemos los datos correspondientes
    TextField titulo = new TextField( label: "Titulo");
    titulo.setValue(c.getTitulo());
    dialog.add(new HorizontalLayout(titulo));
    TextField Sinopsis = new TextField( label: "Sinopsis");
    Sinopsis.setValue(c.getSinopsis());
    dialog.add(new HorizontalLayout(Sinopsis));
    TextField Genero = new TextField( label: "Genero");
    Genero.setValue(c.getGenero());
    dialog.add(new HorizontalLayout(Genero));
    TextField Imbd = new TextField( label: "Imbd");
    Imbd.setValue(c.getImbd());
    dialog.add(new HorizontalLayout(Imbd));
    int numerodeactores = c.getNumeroDeActores();
    Actores todoslosatuores[] = new Actores[numerodeactores];
    TextField nombreaktor[] = new TextField[numerodeactores];
    TextField enlaceactor[] = new TextField[numerodeactores];
    int i =0;
    //ponemos los actores
    for (Actores autorActual : aut.findByIdPelicula(c.getId())) {
        todoslosatuores[i]=autorActual;
        nombreaktor[i] = new TextField( label: "Nombre actor");
    }
}
```

Aquí simplemente creamos el modal que se abrirá cuando queramos editar alguna de las películas de la tabla.

```
//si le da aceptar se editara los datos de la base de datos
Button confirmButton = new Button( text: "Aceptar", event -> {
    c.setTitulo(titulo.getValue());
    c.setSinopsis(Sinopsis.getValue());
    c.setGenero(Genero.getValue());
    c.setImbd(Imbd.getValue());
    // por si ha modificado los actores
    for(int x=0;x<numerodeactores;x++){
        todoslosatuores[x].setNombre(nombreaktor[x].getValue());
        todoslosatuores[x].setEnlace(enlaceactor[x].getValue());
        aut.save(todoslosatuores[x]);
    }
    //guardamos los cambios en la base de datos
    repo.save(c);
    // actualizamos
    listCustomers( filterText: "");
    dialog.close();
});
Button cancelButton = new Button( text: "Cancelar", event -> { dialog.close(); });
// si le da a eliminar eliminaremos esa pelicula de la base de datos
Button EliminarButton = new Button( text: "Eliminar", VaadinIcon.TRASH.create(), event -> { repo.delete(c);listCustomers( filterText: "");dialog.close(); });
HorizontalLayout actions2 = new HorizontalLayout(confirmButton, cancelButton, EliminarButton);
dialog.add(actions2);
dialog.open();
```

Por último, si acepta los nuevos datos se modificará en la tabla y guardaremos los cambios en la base de datos. Si le da a “eliminar” se eliminara automáticamente.


```

void modalnuevapelicula(ActoresBBDD aut,PeliculasBBDD repo) {
    // declaramos el modal
    Dialog dialog = new Dialog();
    //ponemos los textview
    TextField titulo = new TextField( label: "Titulo");
    dialog.add(new HorizontalLayout(titulo));
    TextField Sinopsis = new TextField( label: "Sinopsis");
    dialog.add(new HorizontalLayout(Sinopsis));
    TextField Genero = new TextField( label: "Genero");
    dialog.add(new HorizontalLayout(Genero));
    TextField Imbd = new TextField( label: "Imbd");
    dialog.add(new HorizontalLayout(Imbd));
    // le preguntamos al usuario cuantos actores tiene la pelicula, dependiendo del numero de actores,
    //se pondran los textview

    //declaramos el NumberField
    NumberField NumeroActores = new NumberField( label: "Numero de Actores");
    dialog.add(new HorizontalLayout(NumeroActores));

    Button confirmButton = new Button( text: "Aceptar", event -> {

        Double nacto=NumeroActores.getValue();
        int nmactores=0;
        if (nacto!=null) { // si ha metido un numero valido
            nmactores = nacto.intValue();
        }
        //creamos y guardamos la nueva pelicula
        Peliculas nuevo = new Peliculas(titulo.getValue(), Sinopsis.getValue(), Genero.getValue(), Imbd.getValue(), nmactores);
        repo.save(nuevo);
        //actualizamos
        listCustomers( filterText: "");
        dialog.close();
        // si hay algun actor abriremos un modal, para que el usuario los escriba
        if (nmactores>0) {

```

A continuación crearemos la función que será la encargada de añadir una nueva película a la tabla, primero preguntamos las características principales de cada película y posteriormente se guardara en la tabla y actualizara la base de datos.

```

void modalagregamosautores(ActoresBBDD aut,int numerodeactores,Long idpeli) {
    Dialog dialog = new Dialog();
    dialog.setCloseOnEsc(false);
    dialog.setCloseOnOutsideClick(false);
    //creamos los arrays
    TextField nombreaautor[]= new TextField[numerodeactores];
    TextField enlaceautor[]= new TextField[numerodeactores];
    // ponemos los correspondientes textfiel para que el usuario introduzca los datos
    for(int i =0;i<numerodeactores;i++){
        nombreaautor[i] = new TextField( label: "Nombre actor");
        dialog.add(new HorizontalLayout(nombreaautor[i]));
        enlaceautor[i] = new TextField( label: "Enlace actor");
        dialog.add(new HorizontalLayout(enlaceautor[i]));
    }
    Button confirmButton = new Button( text: "Aceptar", event -> {
        for(int i =0;i<numerodeactores;i++) {
            Actores nuevo = new Actores(nombreaautor[i].getValue(),enlaceautor[i].getValue(),idpeli);
            aut.save(nuevo);
            dialog.close();
        }
    });
    Button cancelButton = new Button( text: "Cancelar", event -> { dialog.close(); });
    HorizontalLayout actions2 = new HorizontalLayout(confirmButton, cancelButton);
    dialog.add(actions2);
    dialog.open();

```

Este fragmento sirve para diferenciar dependiendo de el numero de actores que se hayan elegido para la película. Dependiendo de si es uno o varios, se nos desplegara otro modal correspondiente para introducir tanto su nombre como en enlace de su Wikipedia

```
void guardamosenjson(ActoresBBDD aut,PeliculasBBDD repo) throws IOException {
    String json2 = "{\"Success\":true,\"Message\":\"Invalid access token.\"}";
    String json = "{\"Videoteca\":{\"Nombre\":\"Marcos\",\"Ubicacion\":\"Madrid\",\"Fecha\":2020,\"Películas\":{\"Pelicula\":[
//extraemos todas las películas
List<Películas> películas= repo.findAll();
//vamos recorriendo todas las películas
for(int i = 0; i < películas.size(); i++)
{
    String titulo= películas.get(i).getTitulo();
    String sinopsis= películas.get(i).getSinopsis();
    String imdb= películas.get(i).getImbd();
    String genero= películas.get(i).getGenero();
    Long idpelicula= películas.get(i).getId();
    int numActores=películas.get(i).getNumeroDeActores();
    json+="{\"Titulo\":\""+titulo+ "\",\"Sinopsis\":\""+sinopsis+ "\",\"Genero\":\""+genero+ "\",\"IMBD\":\""+imbd+ "\",\"";
    if (numActores==0){ // si no hay actores
        // para el minar la ultima: ,
        json = json.substring(0, json.length()-1);
    }else{// si hay actores
        List<Actores> actorespeli=aut.findByIdPelicula(idpelicula);
        json+="{\"Reparto\":{\"Actor\":[\"";
        // bucle para recorrer todos los actores
        for(int x = 0; x < actorespeli.size(); x++)
        {
            String NombreAutor=actorespeli.get(x).getNombre();
            String EnlaceAutor=actorespeli.get(x).getEnlace();
            json+="{\"Nombre\":\""+NombreAutor+ "\",\"EnlaceWikipedia\":\""+EnlaceAutor+ "\",\"";
        }
        // para el minar la ultima: ,
        json = json.substring(0, json.length()-1);
        json+="}";
    }
}
}
```

```

// para el minar la ultima: ,
json = json.substring(0, json.length()-1);
|
json+="]]}";
//convertimos el string en un objeto de json
Gson gson = new Gson();
JsonElement jelem = gson.fromJson(json, JsonElement.class);
JsonObject jobj = jelem.getAsJsonObject();
// lo guardamos todo en un archivo
try (Writer writer = new FileWriter( fileName: "Películas.json")) {
    Gson gson1 = new GsonBuilder().create();
    gson1.toJson(jobj, writer);
}
}
```

Por último, guardamos todos los datos actualizados dentro de nuestra base de datos generando un nuevo archivo Películas.json.

Películas

```
@Id
@GeneratedValue
private Long id;
private String titulo;
@Column(length=1000)
private String sinopsis;
private String genero;
private String imbd;
private int numeroDeActores;
protected Peliculas() {
}
//creamos una nueva pelicula
public Peliculas(String Titulo, String Sinopsis,String Genero,String IMBD,Integer numerodeactores) {
    this.titulo = Titulo;
    this.sinopsis = Sinopsis;
    this.genero = Genero;
    this.imbd=IMBD;
    this.numeroDeActores=numerodeactores;
}
```

Introducimos las variables correspondientes y creamos unas para poder introducir una nueva película.

```
public Long getId() { return id; }

public String getTitulo() { return titulo; }

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

public String getSinopsis() { return sinopsis; }

public void setSinopsis(String Sinopsis) { this.sinopsis = Sinopsis; }

public String getGenero() { return genero; }

public void setGenero(String Genero) { this.genero = Genero; }

public String getImbd() { return imbd; }

public void setImbd(String imbd) { this.imbd = imbd; }

public int getNumeroDeActores() { return numeroDeActores; }

public void setNumeroDeActores(int numerodeactores) { this.numeroDeActores = numerodeactores; }
```

A continuación generamos todos los gets y sets para poder almacenar y trabajar con todos los distintos valores dentro de Películas.

Actores

```
public class Actores { // la clase actor tiene los elementos Idautor, no
    @Id
    @GeneratedValue
    private Long idAutor;

    private String nombre;

    private String enlace;
    private Long idPelícula;

    protected Actores() {}

    // para guardar un nuevo actor
    public Actores(String nombre, String enlace, Long idpelícula) {
        this.nombre = nombre;
        this.enlace = enlace;
        this.idPelícula = idpelícula;
    }

    //los gets y sets
    public Long getIdPelícula() { return this.idPelícula; }
    public String getNombre() { return this.nombre; }
    public String getEnlace() { return this.enlace; }

    public void setNombre(String nombre) { this.nombre = nombre; }
    public void setEnlace(String enlace) { this.enlace = enlace; }
}
```

Esta es la clase utilizada para los autores en la que introducimos los elementos del IDautor, su nombre, el enlace y el IDpelícula. También necesitaremos establecer los gets y sets correspondientes para las variables nombre y enlace.

Bases de datos

ActoresBBDD

```
package com.DIS.Practica2;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface PeliculasBBDD extends JpaRepository<Peliculas, Long> {
    List<Peliculas> findByTituloStartsWithIgnoreCase(String lastName);
}
```

Esta es la base de datos relacional de nuestra clase Actores, en la que guardaremos todos los datos relacionados con la misma.

PeliculasBBDD

```
package com.DIS.Practica2;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ActoresBBDD extends JpaRepository<Actores, Long> {
    List<Actores> findByIdPelicula(Long i);
}
```

Esta es la base de datos relacional de nuestra clase Película, en la que guardaremos todos los datos relacionados de la misma.

3. Funcionamiento de la interfaz de la aplicación

Titulo ↕	Sinopsis ↕	Genero ↕	Imbd ↕	Numero De Actores ↕
Superman	Clark Kent usa sus superpoderes, ...	Ciencia ficcion	www.imbd.com/superman	2
Avatar	En un exuberante planeta llamad...	None	www.imbd.com/avatar	2
V de vendeta	Historia de un futuro ficticio don...	Novela Grafica	www.imbd.com/vdevendetta	1
In time	Un hombre es acusado de asesin...	None	www.imbd.com/intime	1
a	a	a	a	0

Como podemos observar nada mas entrar en nuestra aplicación nos encontramos una tabla en la cual podemos distinguir las distintas características en relación con las películas. Podemos distinguir de forma clara el título de la obra, sus sinopsis, el género del que trata y el número de actores que participan en ella. También podemos observar que en cada una de las características o columnas que nos encontramos en la aplicación puede ser ordenada de tres maneras distintas; por orden alfabético, tanto de arriba abajo como viceversa y por último según el orden establecido al introducir los datos en el código.

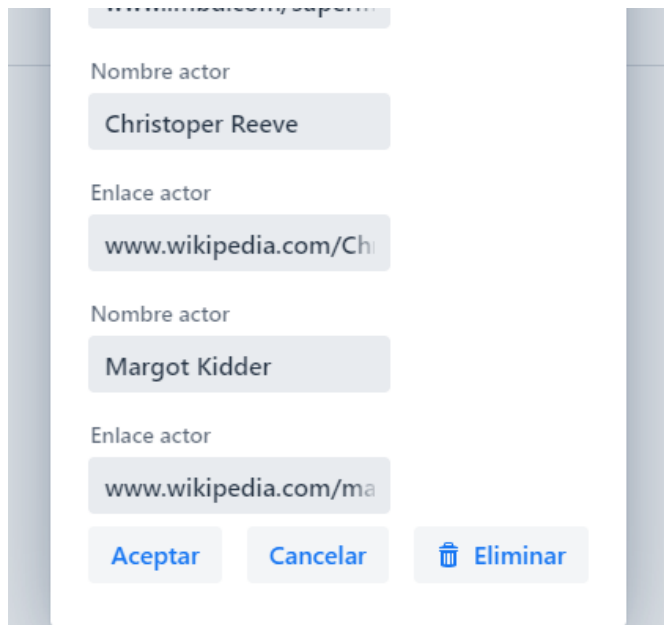
En un exuberante planeta llamad...	None	www.imbd.com/avatar	2
------------------------------------	------	---------------------	---

Titulo:Superman
Sinopsis:Clark Kent usa sus superpoderes, para arruinar el complot de Lex Luthor, quien intenta destruir la Costa Oeste.
Genero:Ciencia ficcion
IMBD:www.imbd.com/superman
Actor:Christoper Reeve
Enlace:www.wikipedia.com/ChristianReeve
Actor:Margot Kidder
Enlace:www.wikipedia.com/margotkidder

Editar Cancelar

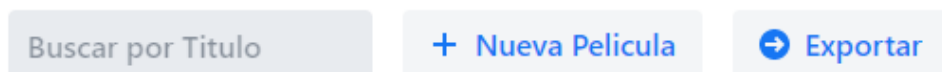
Cada una de las películas introducidas dentro de la aplicación podemos seleccionarlas (se pondrá de otro color la película seleccionada) de tal

manera que se nos abrirá un modal en el que veremos la información mucho más fácilmente y aparte tendremos la opción de editarla.



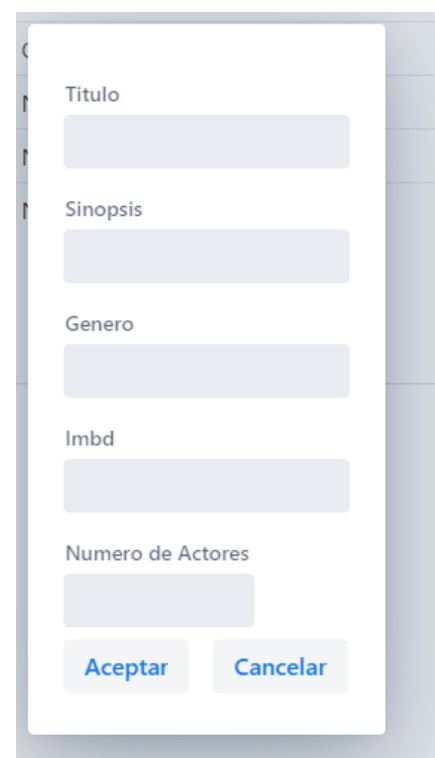
A modal window for editing actor information. It contains two sets of input fields. The first set is for 'Nombre actor' with the value 'Christoper Reeve' and 'Enlace actor' with the value 'www.wikipedia.com/Ch'. The second set is for 'Nombre actor' with the value 'Margot Kidder' and 'Enlace actor' with the value 'www.wikipedia.com/ma'. At the bottom, there are three buttons: 'Aceptar', 'Cancelar', and 'Eliminar' (with a trash icon).

Si seleccionamos la opción de “editar” se nos desplegará otro modal en el que seremos capaces de modificar cada una de las distintas características de la película en cuestión e incluso eliminarla.

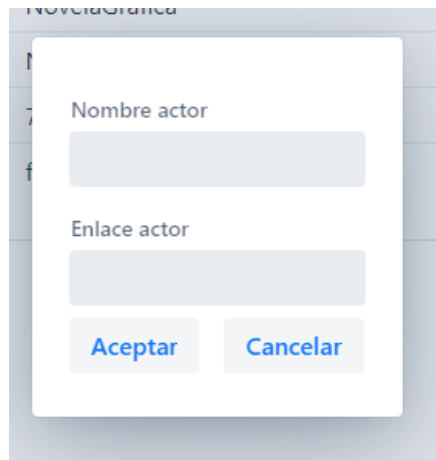


A row of three buttons: 'Buscar por Titulo', '+ Nueva Pelicula', and 'Exportar' (with a right arrow icon).

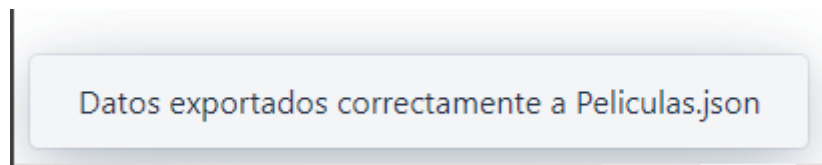
En la parte superior de la tabla nos encontramos tres elementos que son los que forman nuestro “HorizontalLayout”. Esta parte de la aplicación es la mas significativa ya que con ella podremos realizar la mayor parte de cosas dentro de la misma. Como podemos ver en primer lugar nos encontramos con un buscador, lo utilizaremos por si queremos buscar una película por su título, de esta manera se nos irán actualizando automáticamente los datos en la tabla según vayamos escribiendo. Por otro lado podemos introducir una nueva película en nuestra tabla, para ello presionamos el botón “Nueva Película” se nos abrirá modal para que rellenemos los distintos campos o características de la misma.



A modal window for adding a new movie. It contains five input fields labeled 'Titulo', 'Sinopsis', 'Genero', 'Imbd', and 'Numero de Actores'. At the bottom, there are two buttons: 'Aceptar' and 'Cancelar'.

A modal form with a white background and rounded corners, centered on a gray background. It contains two text input fields. The first field is labeled 'Nombre actor' and the second is labeled 'Enlace actor'. Below the fields are two buttons: 'Aceptar' (Accept) and 'Cancelar' (Cancel), both in blue text. The modal is slightly offset from the top and left edges of the background.

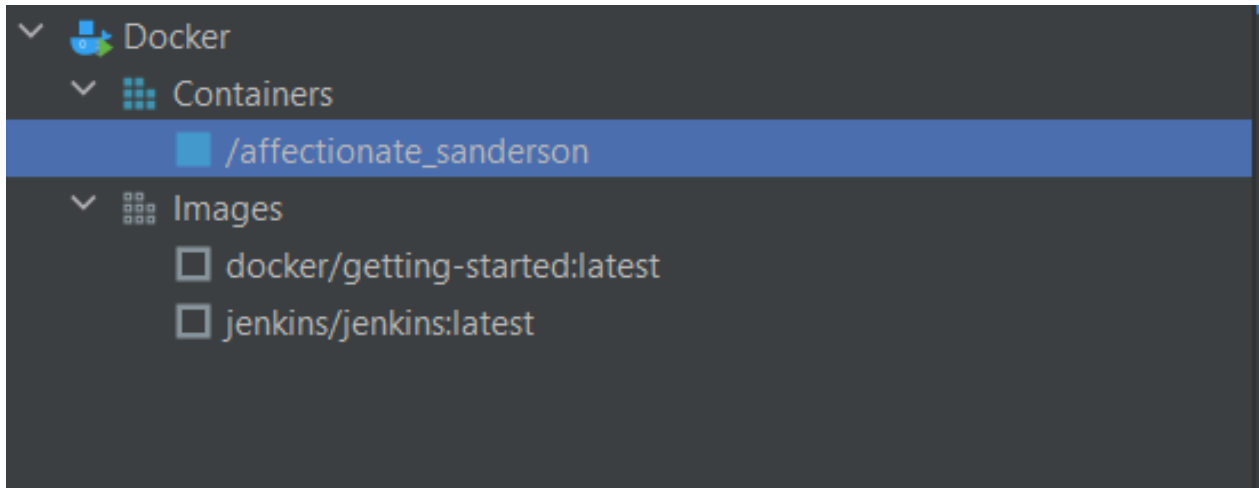
Finalmente, se nos abrirá un nuevo modal en el que tendremos que especificar dependiendo de la cantidad de actores que hayamos introducido, tanto el nombre de cada uno de ellos como su enlace.



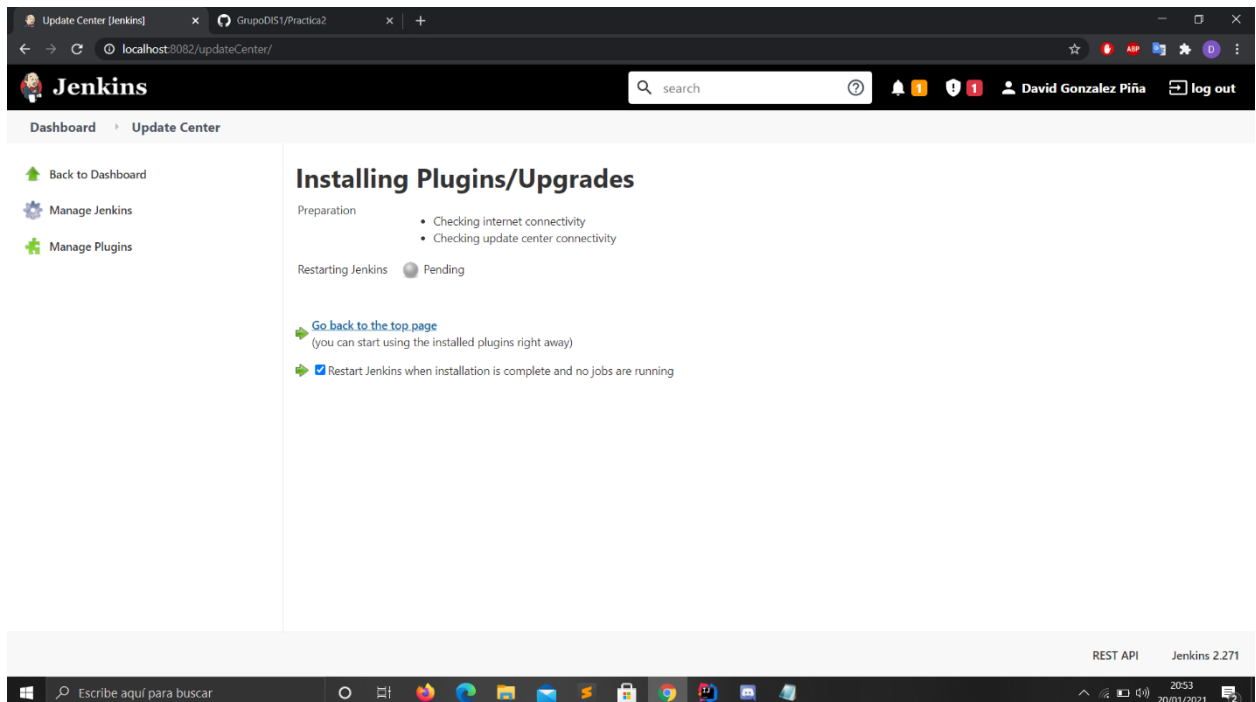
Por último, tenemos el botón de “exportar” el cual actualizará nuestro Json dependiendo de los cambios que hayamos realizado dentro de nuestra tabla, posteriormente se guardará en la base de datos y nos llegará un mensaje en la parte inferior izquierda confirmándolo.

Jenkins

Para realizar el Jenkins, ya habremos inicializado nuestra imagen de Docker y ya habremos creado el contenedor con dicha imagen.



Después accederemos al local host, y ya una vez dentro de Jenkins tendremos que instalar los plugins necesarios para trabajar con Git Hub y así poder realizar la práctica.



Para poder realizar el job pedido en la práctica tendremos que enlazar el Jenkins con la cuenta de Github donde se encuentra el repositorio de la práctica. Una vez enlazada la cuenta y el repositorio, crearemos los pipelines para su correcto y debido uso.

