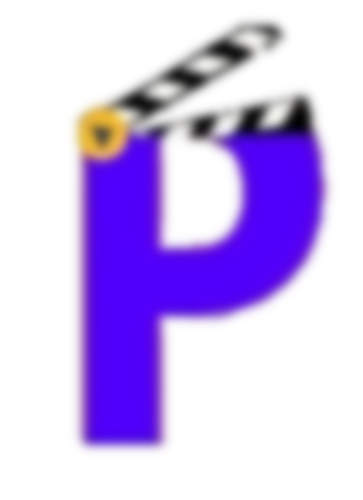
***DOCUMENTACIÓN***

PELICULEO !

Aplicación multiplataforma de gestión de tickets para espectadores y gestión semi-automatizada de estrenos para administradores.

******



Miguel Ángel Fuentes Valenzuela – 2ºDAW | IES Las Fuentezuelas

FichaTécnica **– Peliculeo!**

|  |  |
| --- | --- |
| DATOS GENERALES | |
| **Lenguajes de Programación** | JAVA (Spring), Maven, Gradle, SQL (motor: MariaDB) |
| **Entornos de Desarrollo (IDE) utilizados** | IntelliJ IDEA, Apache NetBeans, Notepad++ |
| **Modo de Despliegue** | Servidor local SpringBoot; Apache + mySQL (XAMPP stack) para servidor MySQL |
| **Motor de BBDD** | MariaDB + PHPMyAdmin |
| **Nombre de la Aplicación** | PELICULEO! |

|  |  |
| --- | --- |
| REFERENTE TÉCNICO y FUNCIONAL | |
| **Nombre** | Miguel Ángel Fuentes Valenzuela |
| **E-mail** | [mafvpersonal@gmail.com](mailto:mafvpersonal@gmail.com) – fuentesvalenzuela789@gmail.com |
| **Teléfono de contacto** | 603 58 96 82 |

Esta aplicación tiene acceso a **consultar la información que el Usuario introduce** en la BBDD.

1. Estudio del problema y análisis del sistema

El proyecto consiste en crear una Aplicación Multiplataforma que permita gestionar tanto películas de una Base de Datos relacional, como ‘tickets’ para los estrenos de dichas películas en un salón de cine.

Los clientes que quieran acceder a ver una película, se deberán registrar en el sistema, a través de la aplicación de móvil “MyPeliculeo”

Una vez se han registrado, tendrán acceso a las películas disponibles en cartelera y si ya se han identificado, a los tickets que ya hayan adquirido con opción de descartarlos o mantenerlos.

**ANÁLISIS DE LA SOLUCIÓN**

Con el fin de lograr cumplir estos objetivos, se han desarrollado tres “módulos”o aplicativos separados, cada uno con un objetivo específico:

1. Aplicación de servidor con SPRING (SERVIDOR)

2. App Escritorio con Java Swing (CLIENTE 1 - ADMINISTRACIÓN)

3. App para Android con Retrofit (CLIENTE 2 – USUARIO FINAL)

1. Módulos del ciclo que intervienen

**Acceso a Datos** 🡪 Diseño de la base de datos y de los DAOs (clases para conectar los diferentes módulos de la app a la BBDD). Codificación de los distintos componentes del script DDL para MySQL.

**Desarrollo de Interfaces** 🡪 Todo lo relacionado con GUIs, y la codificación interna de estas. Tanto en Android (api de Toast/Android/Glide…) como en Java Swing, para la app de Escritorio (api de awt/swing)

**Programación Multimedia y Disp. Móviles** 🡪 Módulo de aplicación para cliente (recyclerview con CRUD y autenticación) en Android.

**Programación de Servicios y Procesos (PSP)** 🡪 API REST usando SPRINGBOOT.

Uso de la función “sleep” en el hilo principal de las aplicaciones, para mostrar información de manera temporal al usuario (tanto en la aplicación de admón., como en la de Android).

1. Apartado Técnico: Funcionalidad de la solución

Según cada módulo he usado diversas formas de agregarle funcionalidades útiles a la aplicación, siendo las más destacables:

**MySQL** Scripts de creación para tablas y esquemas

<https://drive.google.com/file/d/1GrpG6fsu0Gc1Om_HBwFySKKfMNiG89R9/view?usp=drive_link>

**MySQL**

Procedure para eliminar películas cuya fecha de estreno ya ha pasado (gestión automática)

DELIMITER //

CREATE PROCEDURE DeleteOldMovies()

BEGIN

DELETE FROM peliculas WHERE STR\_TO\_DATE(fechaEstreno, '%Y-%m-%d %H:%i:%s') < CURDATE();

END //

DELIMITER ;

Trigger que se encarga de darle la fecha actual a las nuevas películas insertadas

CREATE TRIGGER before\_insert\_peliculas

BEFORE INSERT ON peliculas

FOR EACH ROW

BEGIN

IF NEW.fechaEstreno IS NULL THEN

SET NEW.fechaEstreno = DATE\_FORMAT(NOW(), '%Y-%m-%d %H:%i:%s');

END IF;

END //

Reflejar cambios de la PK de ‘Peliculas’

ALTER TABLE `tickets`

ADD CONSTRAINT `FK\_cliente\_tickets` FOREIGN KEY (`NIF\_cliente`) REFERENCES `clientes` (`NIF`) ON DELETE CASCADE ON UPDATE CASCADE,

ADD CONSTRAINT `FK\_codPelicula\_tickets` FOREIGN KEY (`codPelicula`) REFERENCES `peliculas` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;

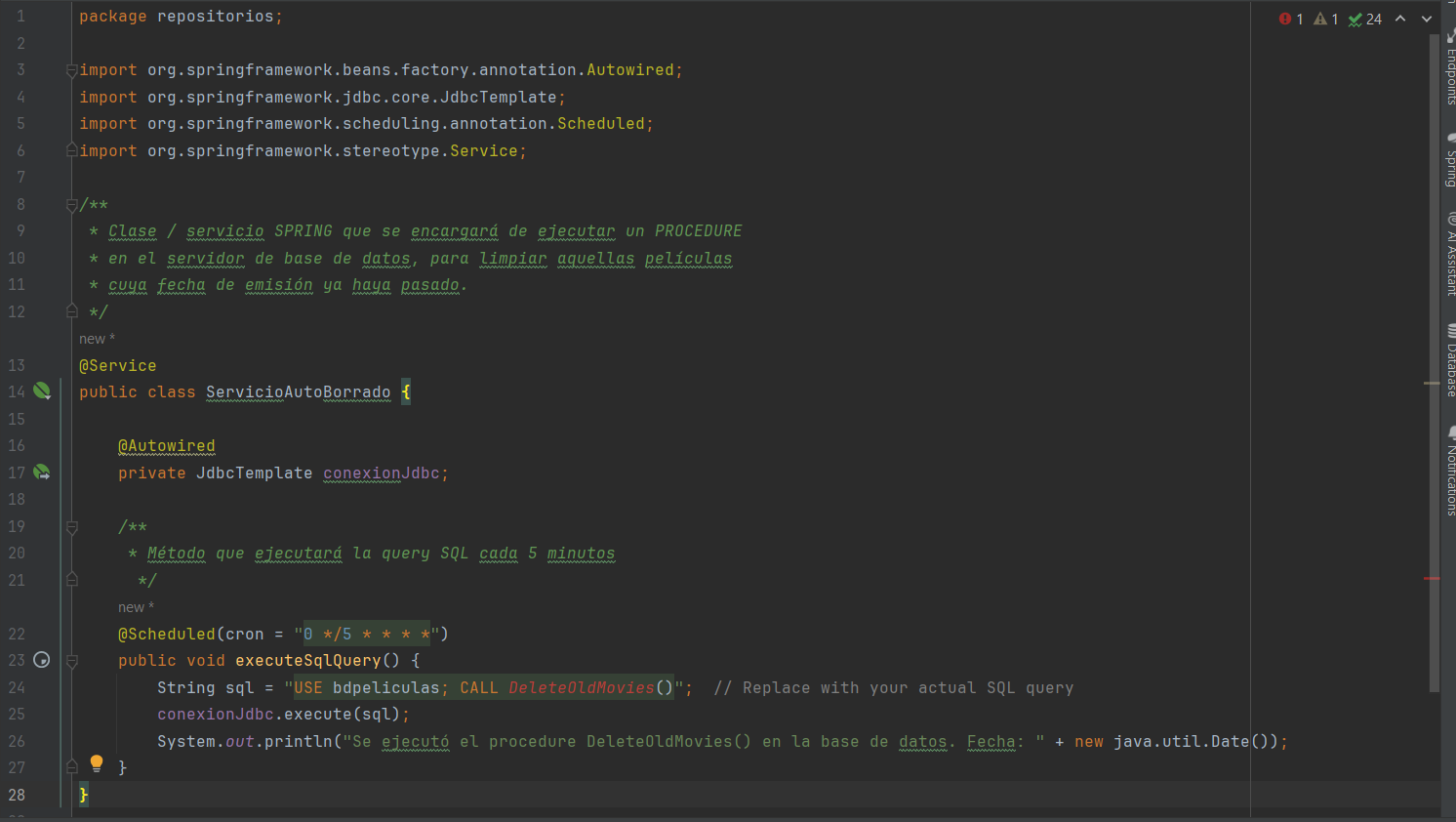
COMMIT;

**JAVA - SPRING BOOT: Aplicación “Peliculeo! Server”**

Se encarga de poner en marcha una API RESTful desde la que se podrá consumir las distintas entidades (Pelicula, Usuario y Ticket), así como de manejar la autenticación y ‘hasheo’ de contraseñas directamente desde el servidor.

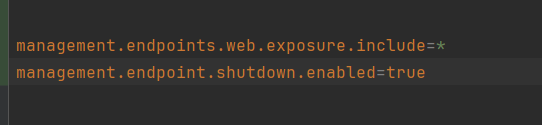
Utiliza Hibernate para comunicarse con la base de datos, e incluye un método cronometrado (esto es, se ejecuta reiteradamente cada cierto tiempo), que cada 5 minutos envía una petición a un PROCEDURE en la base de datos para eliminar todas las Películas cuya fecha de emisión ya haya pasado (fecha actual > fecha\_emision).

Para lograr esto, se ha empleado una clase con la anotación **@Scheduled(cron = …)** de SpringBoot, que a partir de un servicio a parte, es capaz de poner en marcha una tarea específica con la frecuencia que necesitemos:

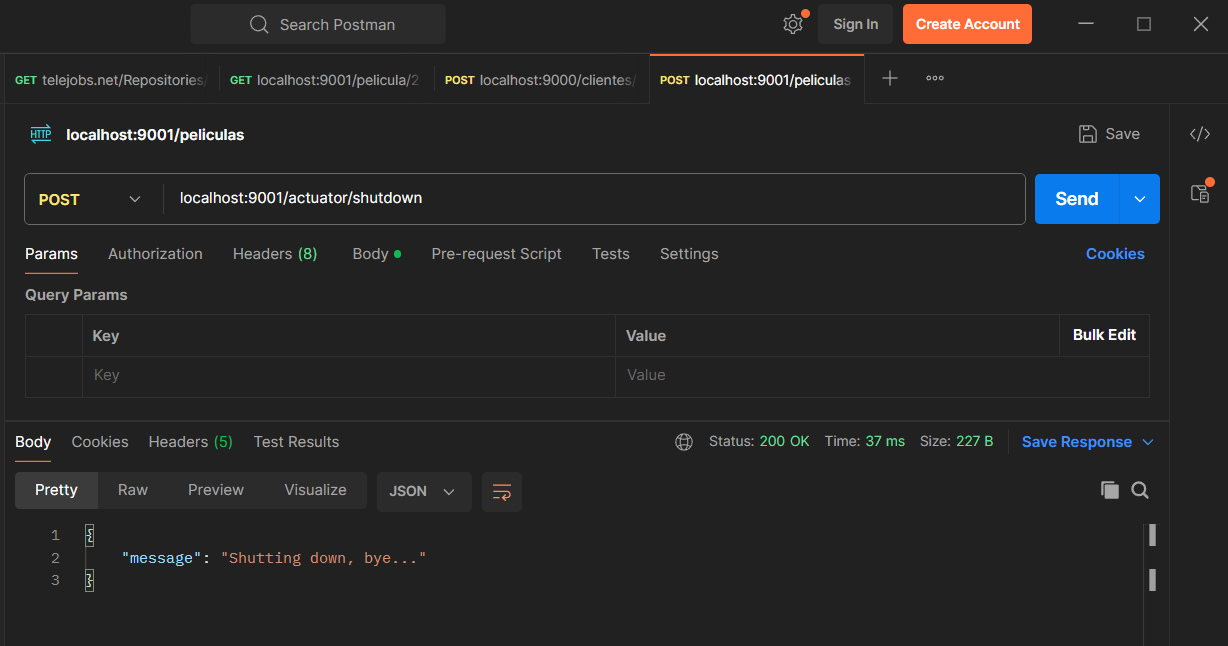


Otra funcionalidad a destacar: la posibilidad de cerrar/apagar el servicio de Spring a través de una request HTTP.

Para ello, se han introducido las siguientes líneas en el fichero de configuración *Application.properties* de Spring:



Con esto, al realizar una llamada http POST al endpoint ***/actuator/shutdown***, ocurre lo siguiente…



…y tras realizar la llamada se habrá cerrado la aplicación.

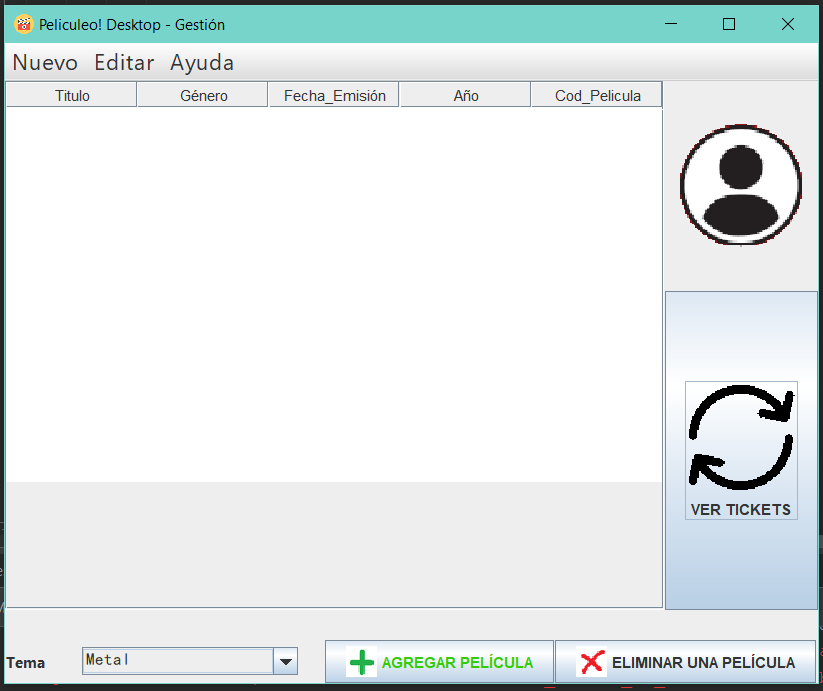
**JAVA – “Peliculeo! Desktop”**

Se encarga de proporcionar una interfaz gráfica o GUI para los usuarios de escritorio. A través de esta, se pueden gestionar los estrenos, tickets y Usuarios de la aplicación.

Para poder usarla, se necesitará de una instancia del servidor y Docker de MySQL iniciados (esto se puede lograr manualmente, o bien usando el script facilitado en el repositorio de “Peliculeo!”)\*\*

\*\*Usar el script start.sh // start.bat, dependiendo de nuestro Sistema Operativo

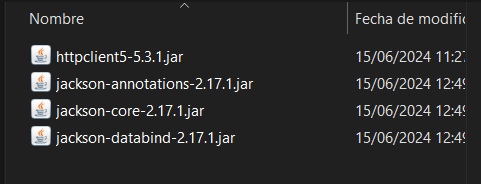
Como principales funcionalidades cabe destacar los listados para películas y tickets:



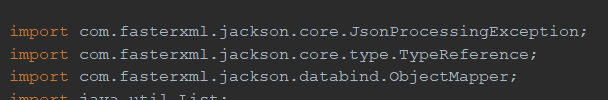
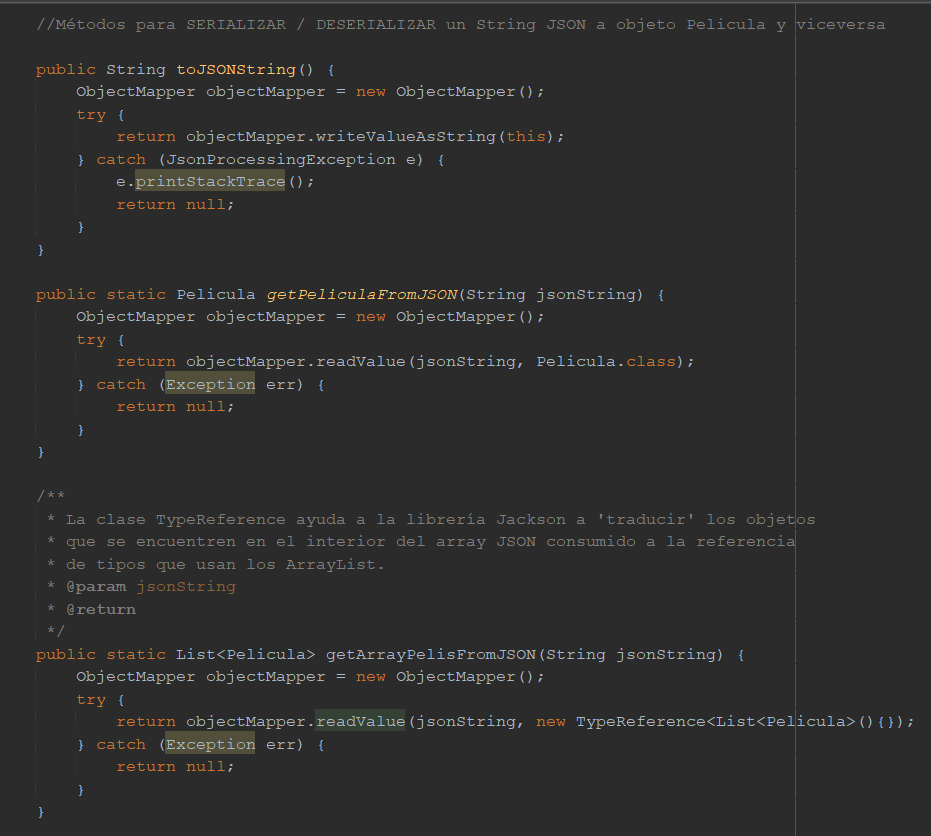
Además de las diferentes pestañas para gestión de Tickets sacados, Películas en emisión y Usuarios.

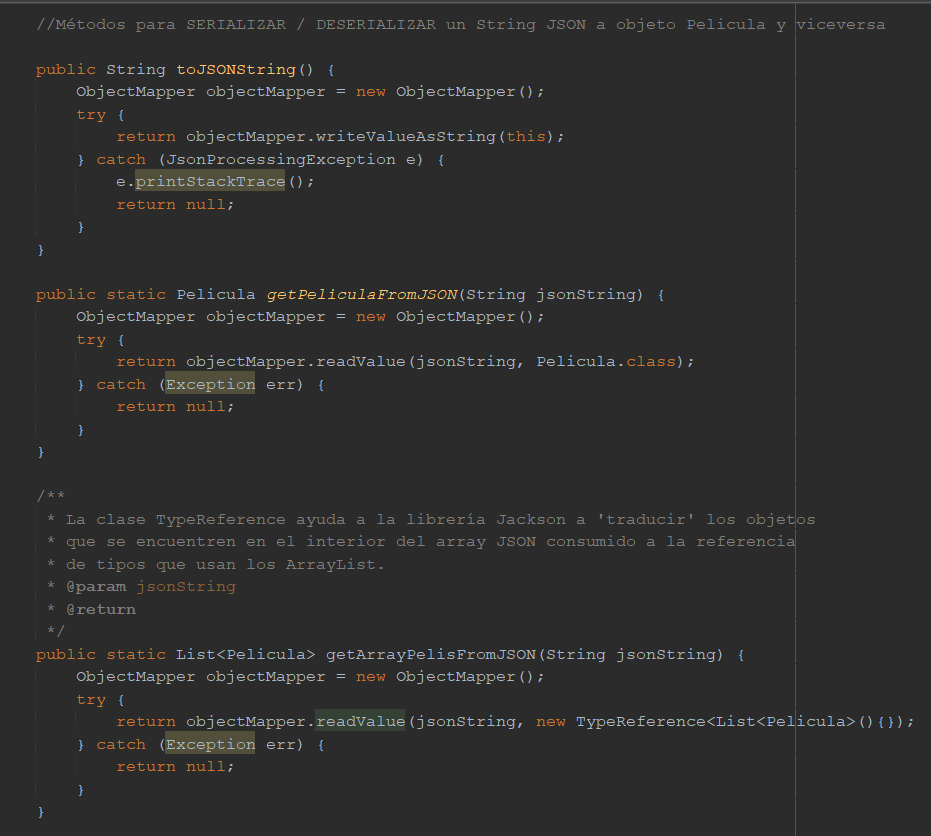
Para recuperar los objetos POJO de Java desde la API con Spring, en esta aplicación he optado por usar la librería **Jackson** *(faster xml)* que permite transformar resultados JSON de una llamada HTTP a un objeto de la clase de Java que necesitemos.

Los ficheros JAR usados son los siguientes:



Su implementación en las clases correspondientes:





1. Seguridad de la Aplicación

Para hacer la aplicación más segura, se usa un método de autenticación a nivel de Servidor (API Springboot) que usa **Spring Security** como base para interceptar las operaciones de login y registro.

Esta librería (Spring Security) incluye diferentes medidas que se aplican en momento de ejecución para evitar ataques de inyección SQL/código Java…

A nivel de código SQL, se implementan también métodos de comprobación antes de realizar cualquier operación DML (trigger before INSERT or UPDATE or DELETE), para comprobar que no se esté intentando introducir código malicioso.

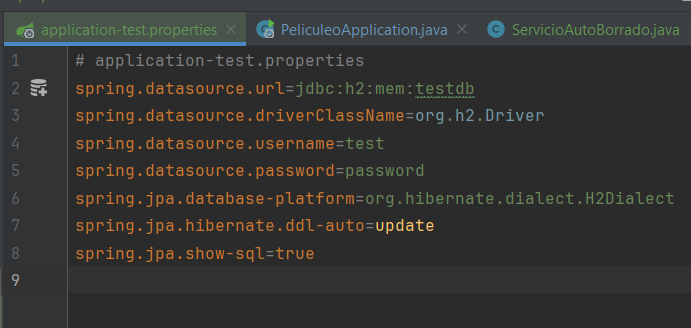
**TESTS SpringBoot**

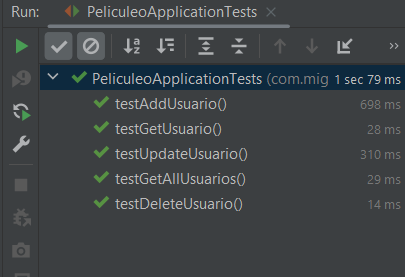
Se han estipulado y pasado los siguientes tests unitarios en el módulo SpringBoot de Peliculeo! “Peliculeo\_server”, usando **MockMvc** para simular operaciones CRUD, y una base de datos en memoria **H2**:

*/\*\*  
 \* TESTS UNITARIOS peliculeo!  
 \*  
 \** ***@TestMethodOrder*** *-- Anotación usada para ejecutar los tests en un orden especificado por  
 \* anotaciones.  
 \*/*@SpringBootTest  
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)  
@AutoConfigureMockMvc  
public class PeliculeoApplicationTests extends TestsUtils {  
  
 @Autowired  
 private MockMvc mockMvc;  
  
 @Autowired  
 private WebApplicationContext webApplicationContext;  
  
 //La anotación @BeforeEach se encarga de ejecutar este bloque de código antes de cada  
 // método de testing que incluya @Test detrás  
 @BeforeEach  
 @Order(1)  
 public void configuracionInicial() {  
 mockMvc = MockMvcBuilders.*webAppContextSetup*(webApplicationContext).build();  
 }  
  
 @Test  
 @Order(2)  
 void testAddUsuario() throws Exception {  
 Usuario usuario = new Usuario();  
 usuario.setNIF("12345678A");  
 usuario.setNombre("Pippi");  
 usuario.setApellidos("Calzaslarga");  
 usuario.setFechAlta("2024-06-15");  
 usuario.setProvincia("Cordoba");  
 usuario.setPassword("ejemplo1");  
  
 mockMvc.perform(*post*("/api/usuarios/add")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(usuario)))  
 .andExpect(*status*().isCreated())  
 .andExpect(*content*().string("Usuario introducido correctamente!"));  
 }  
  
 @Test  
 @Order(3)  
 void testGetUsuario() throws Exception {  
 mockMvc.perform(*get*("/api/usuarios/12345678A"))  
 .andExpect(*status*().isOk());  
 }  
  
 @Test  
 @Order(4)  
 void testUpdateUsuario() throws Exception {  
 Usuario usuario = new Usuario();  
 usuario.setNombre("Pipo");  
 usuario.setApellidos("Calzolargo");  
 usuario.setFechAlta("2024-06-14");  
 usuario.setProvincia("Jaen");  
 usuario.setPassword("contrasenia1234");  
  
 mockMvc.perform(*put*("/api/usuarios/12345678A")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(usuario)))  
 .andExpect(*status*().isOk());  
 }  
  
 @Test  
 @Order(5)  
 void testGetAllUsuarios() throws Exception {  
 mockMvc.perform(*get*("/api/usuarios/"))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 @Order(6)  
 void testDeleteUsuario() throws Exception {  
 mockMvc.perform(*delete*("/api/usuarios/12345678A"))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string("Usuario eliminado."));  
 }  
}

@Test  
 @Order(2)  
 void testAddUsuario() throws Exception {  
 Usuario usuario = new Usuario();  
 usuario.setNIF("12345678A");  
 usuario.setNombre("Pippi");  
 usuario.setApellidos("Calzaslarga");  
 usuario.setFechAlta("2024-06-15");  
 usuario.setProvincia("Cordoba");  
 usuario.setPassword("ejemplo1");  
  
 mockMvc.perform(*post*("/api/usuarios/add")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(usuario)))  
 .andExpect(*status*().isCreated())  
 .andExpect(*content*().string("Usuario introducido correctamente!"));  
 }  
  
 @Test  
 @Order(3)  
 void testGetUsuario() throws Exception {  
 mockMvc.perform(*get*("/api/usuarios/12345678A"))  
 .andExpect(*status*().isOk());  
 }  
  
 @Test  
 @Order(4)  
 void testUpdateUsuario() throws Exception {  
 Usuario usuario = new Usuario();  
 usuario.setNombre("Pipo");  
 usuario.setApellidos("Calzolargo");  
 usuario.setFechAlta("2024-06-14");  
 usuario.setProvincia("Jaen");  
 usuario.setPassword("contrasenia1234");  
  
 mockMvc.perform(*put*("/api/usuarios/12345678A")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(usuario)))  
 .andExpect(*status*().isOk());  
 }  
  
 @Test  
 @Order(5)  
 void testGetAllUsuarios() throws Exception {  
 mockMvc.perform(*get*("/api/usuarios/"))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 @Order(6)  
 void testDeleteUsuario() throws Exception {  
 mockMvc.perform(*delete*("/api/usuarios/12345678A"))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().string("Usuario eliminado."));  
 }  
}

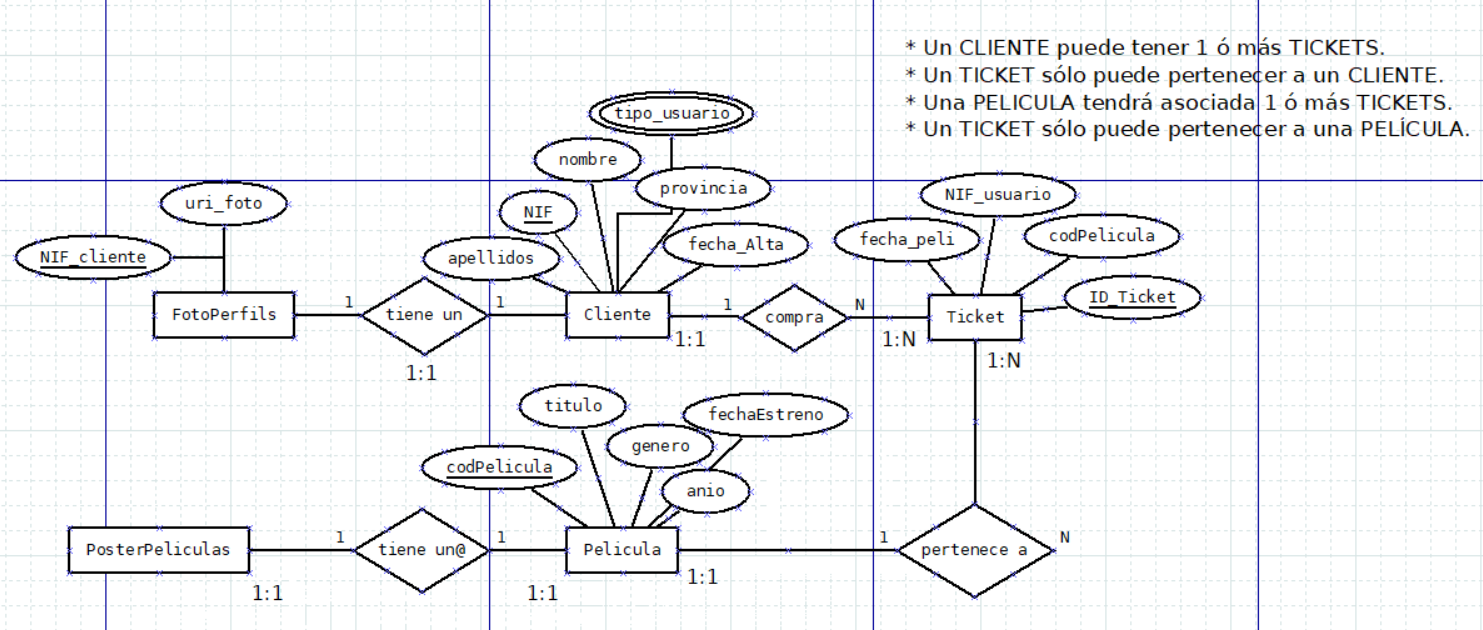
Ficheros de configuración para testing y demostración de los tests pasados:





1. Diagramas UML y Entidad-Relación

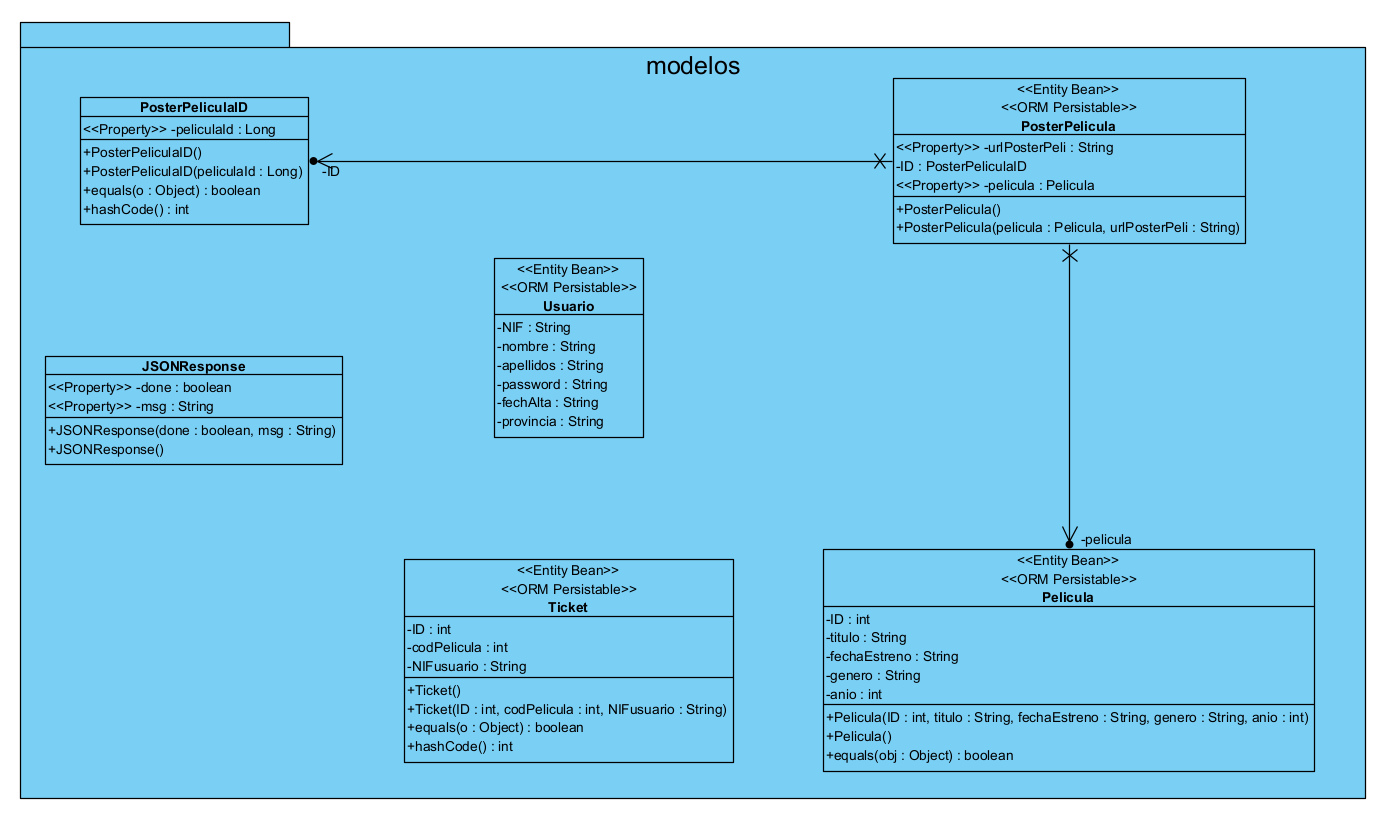
**MODELO E--R – Peliculeo!**

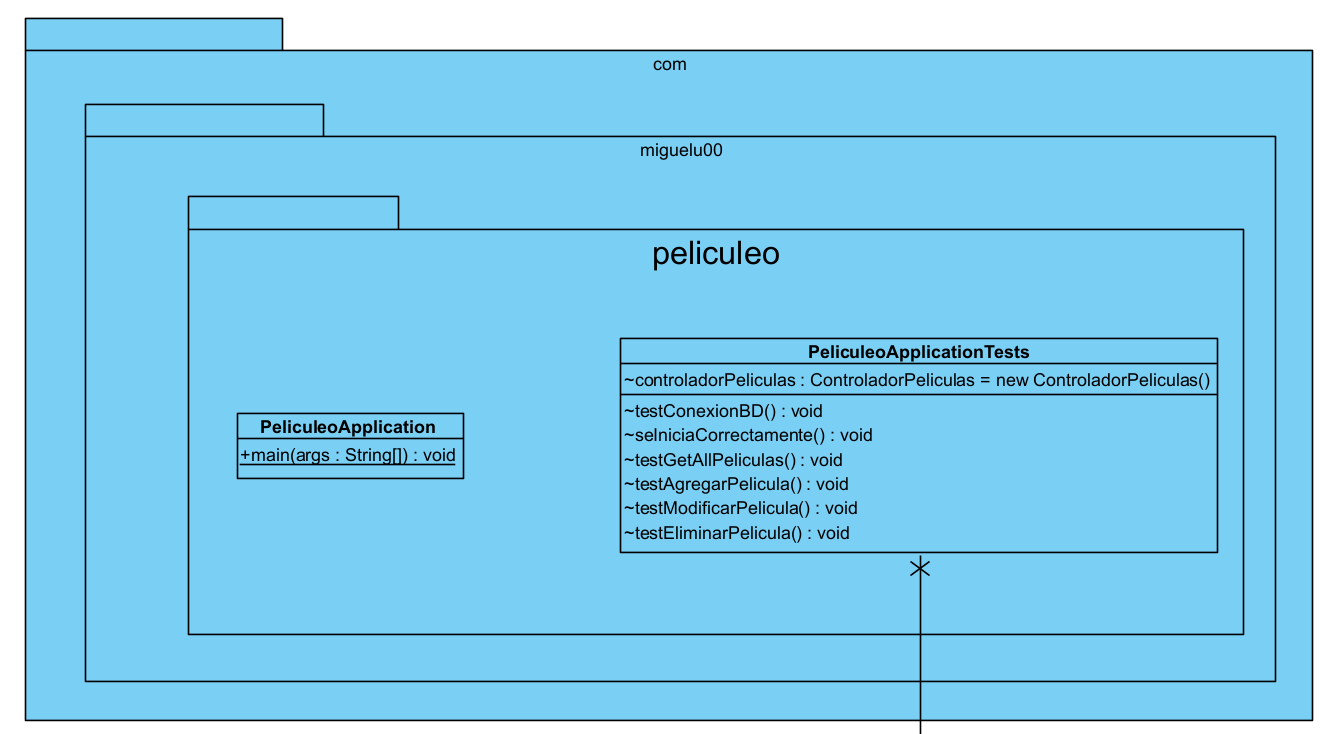
****

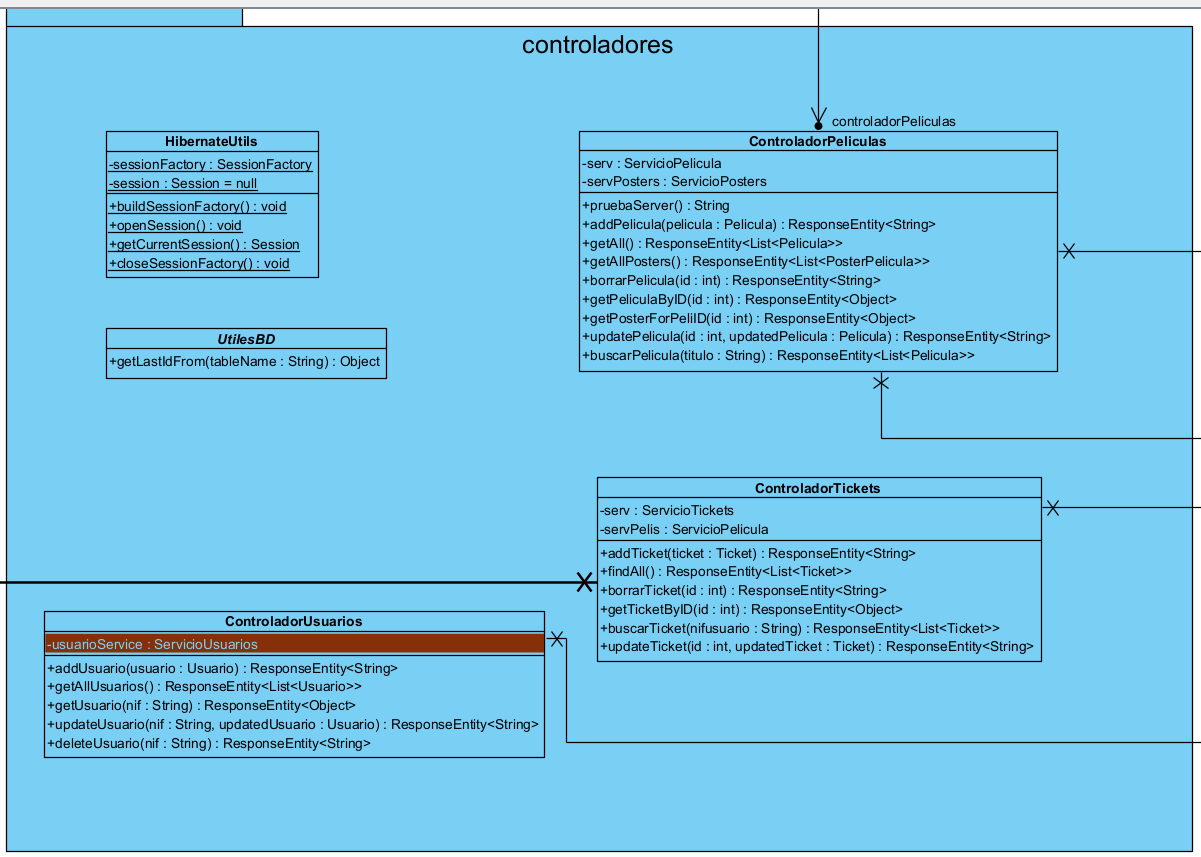
**DIAGRAMAS DE CLASE / PAQUETES**

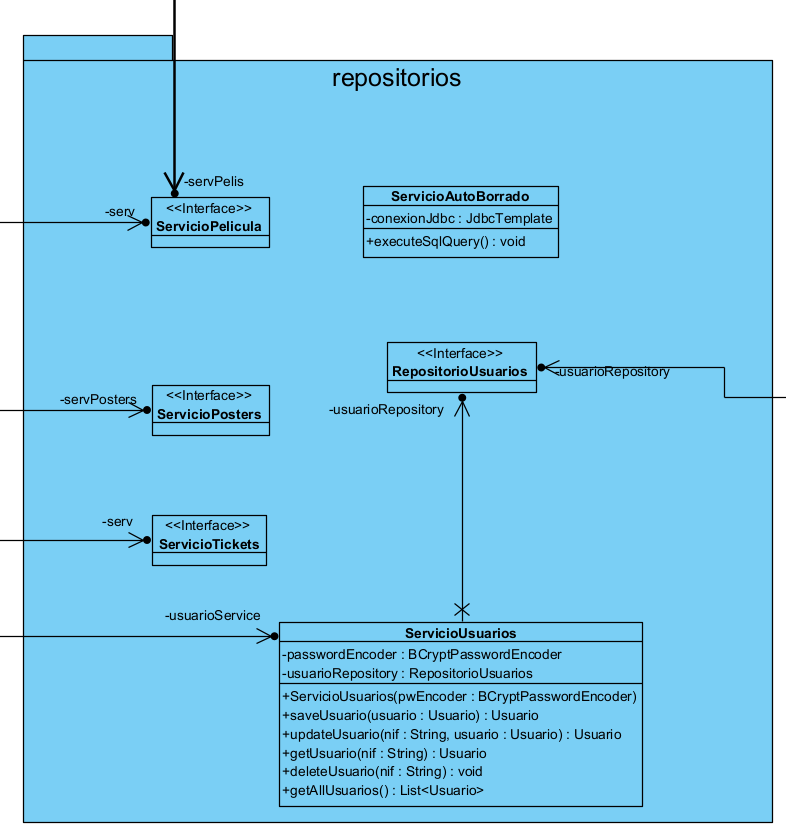
(Separados por packages):

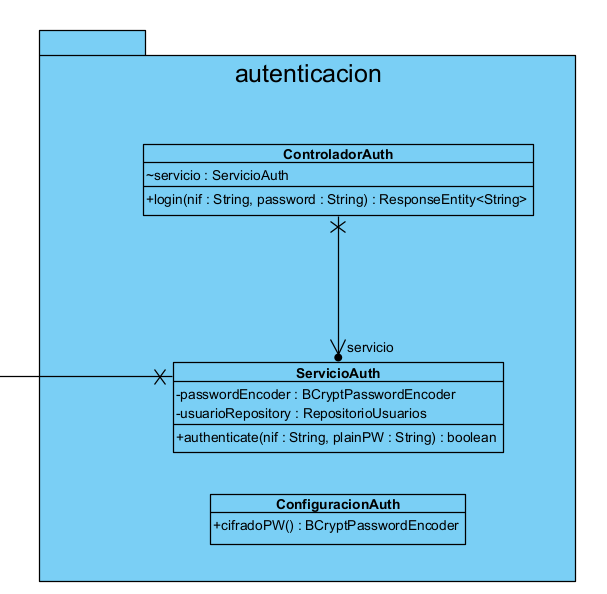
**Módulo servidor – Spring Boot**

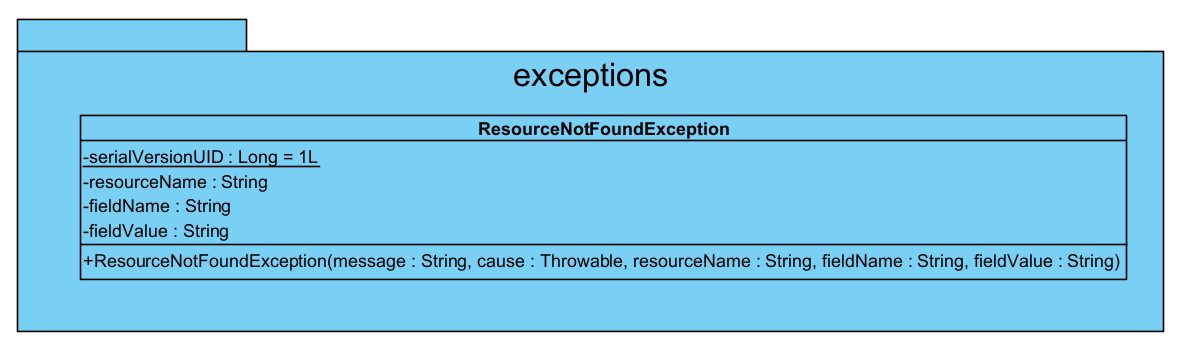
****

****

****

****

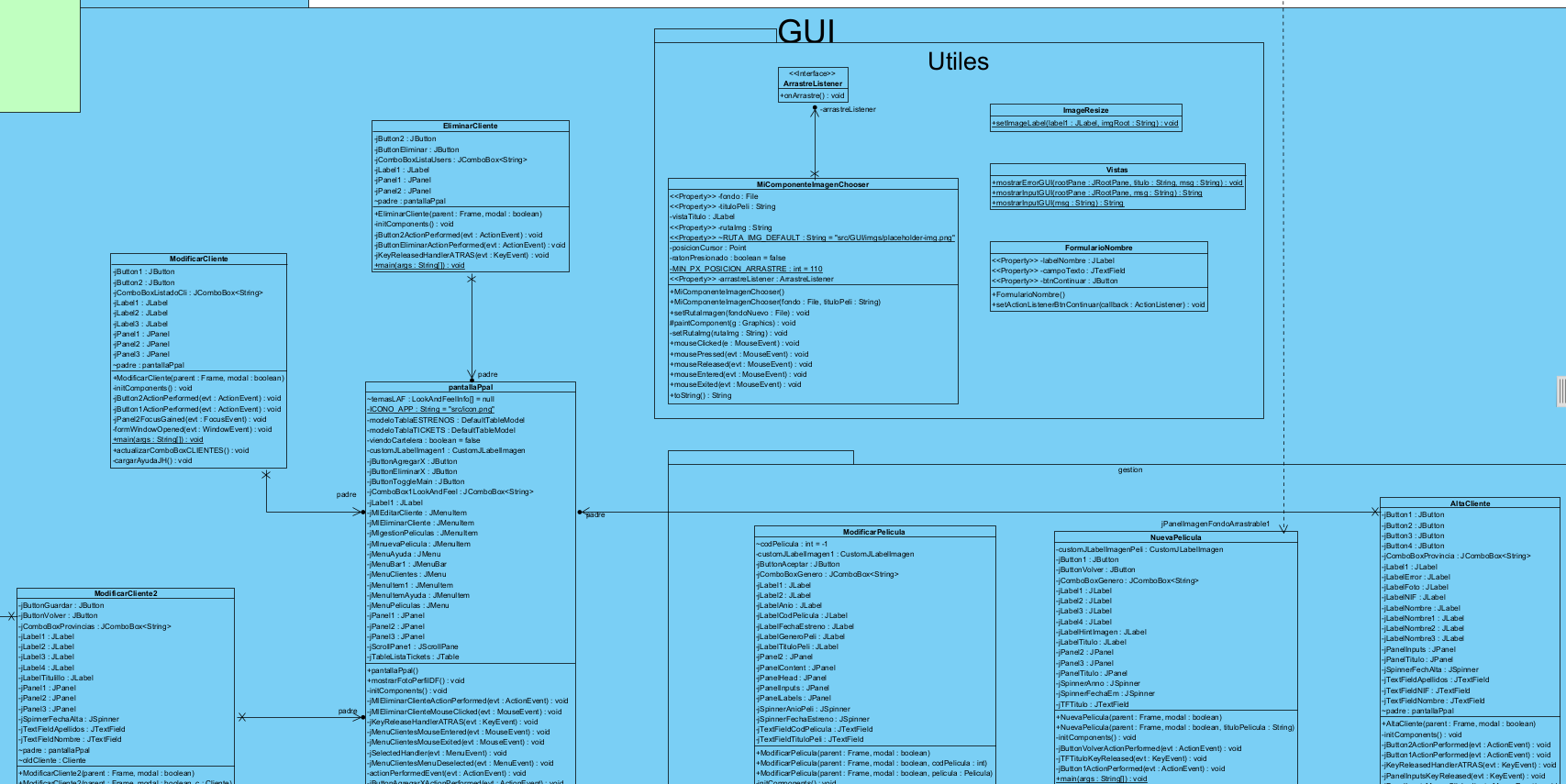
****

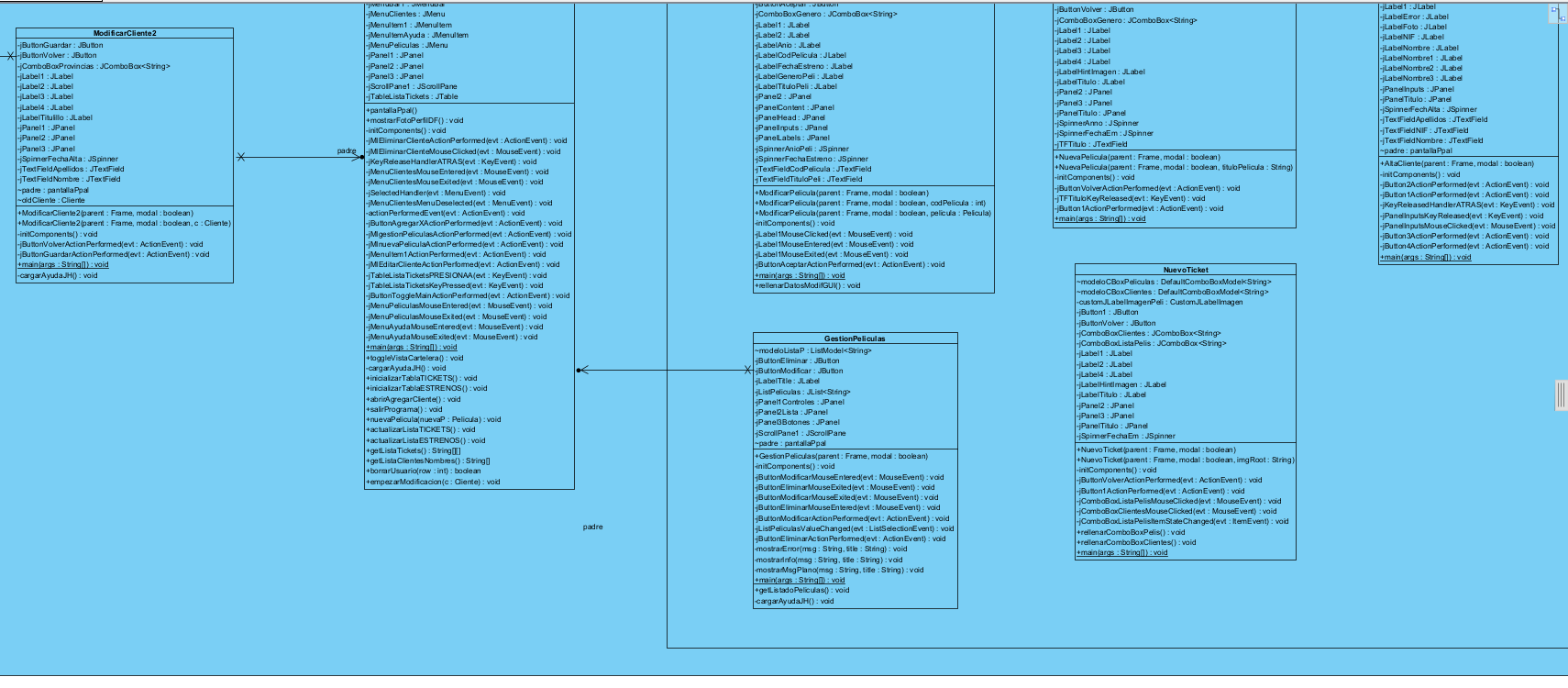
****

**(esta última fue auto-generada por IntelliJ)**

**Módulo Cliente 1 – “Peliculeo! Desktop”**

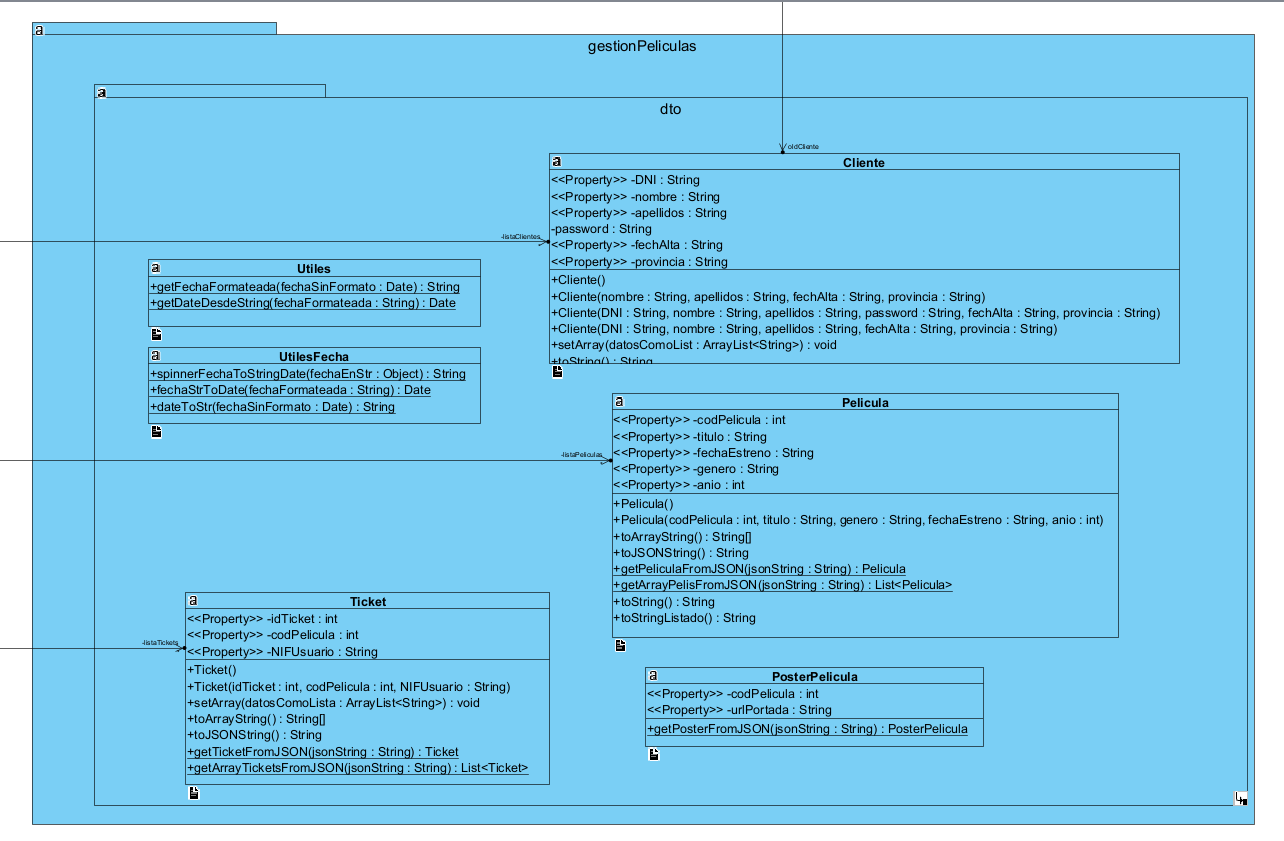
**PAQUETE GUI + subpaquetes (Utiles y gestión)**

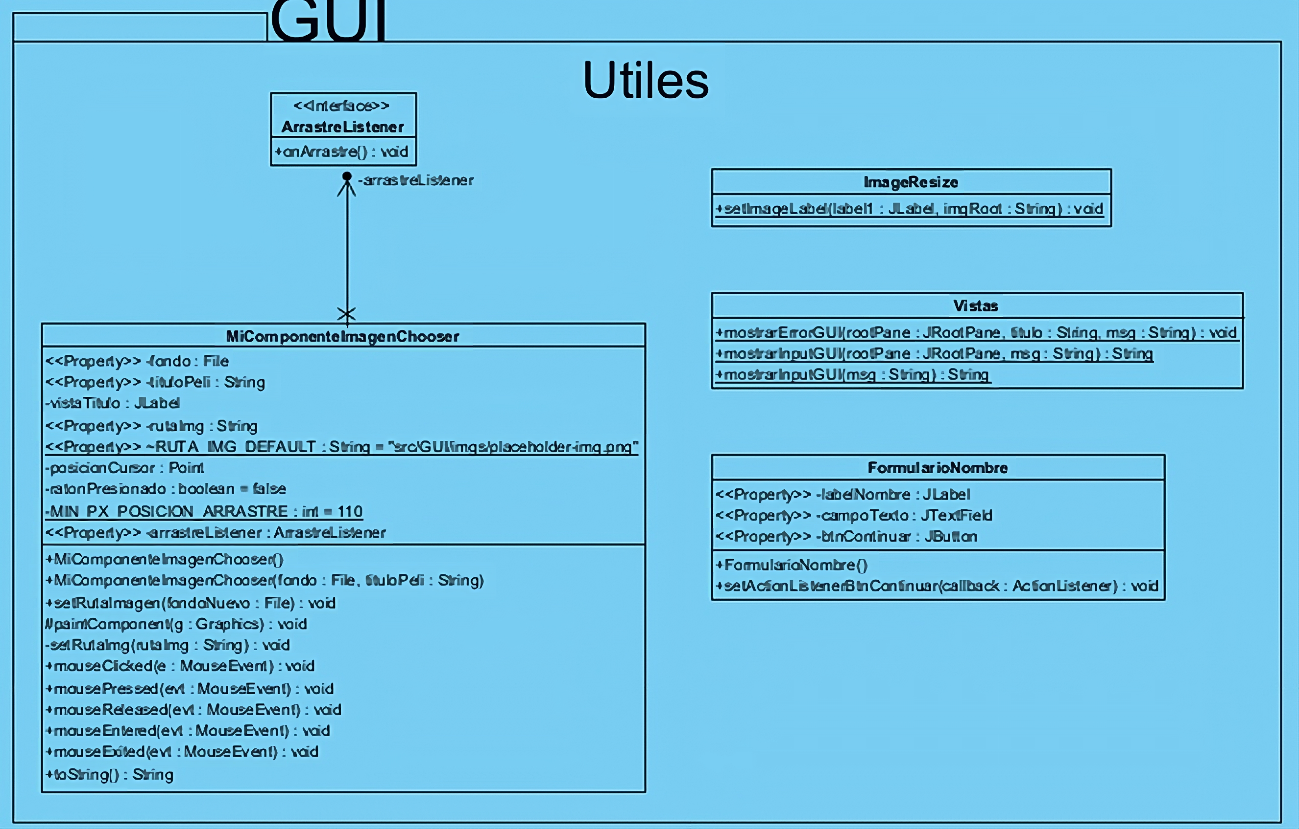
****

****

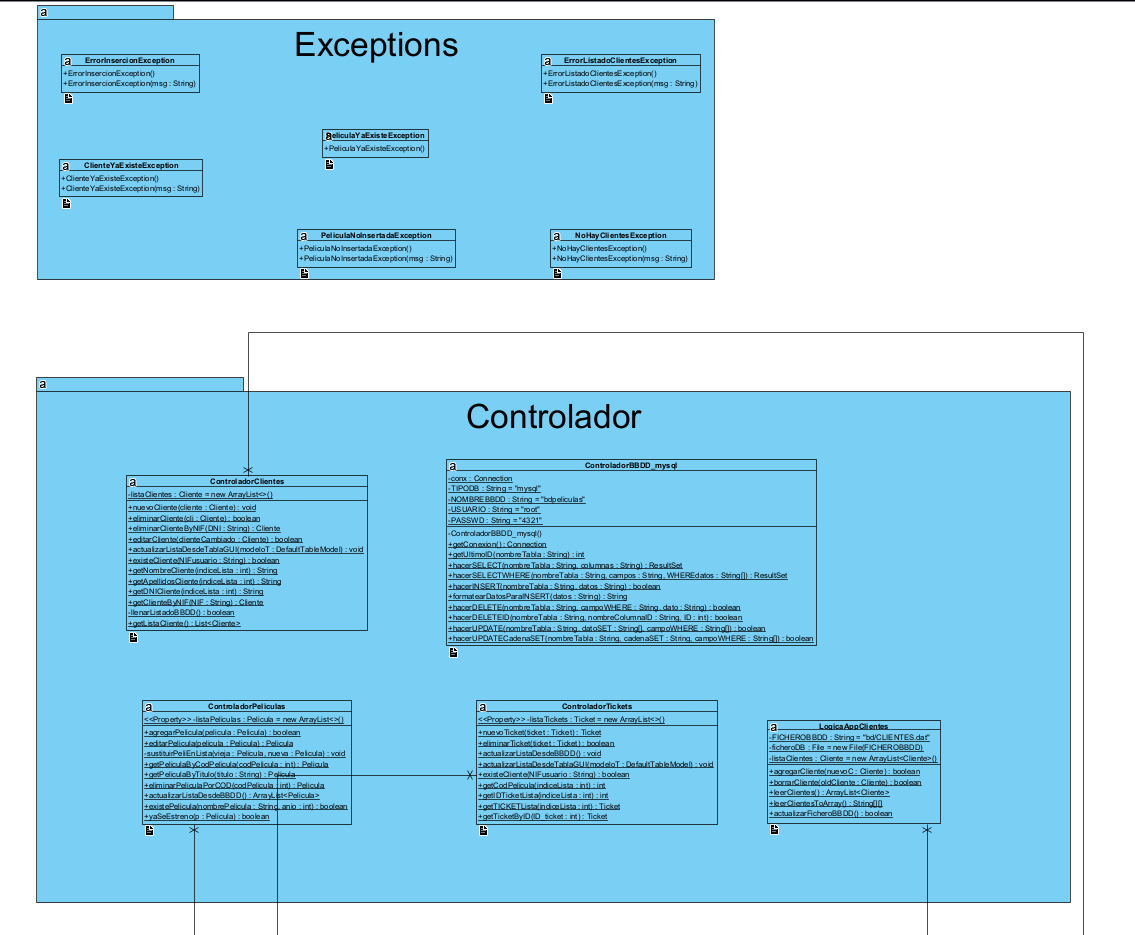
**\*muchas propiedades y variables son extrañas por el tratamiento que les hace el asistente de formularios de NetBeans**

Paquetes **DTO y Utiles (más cerca)**

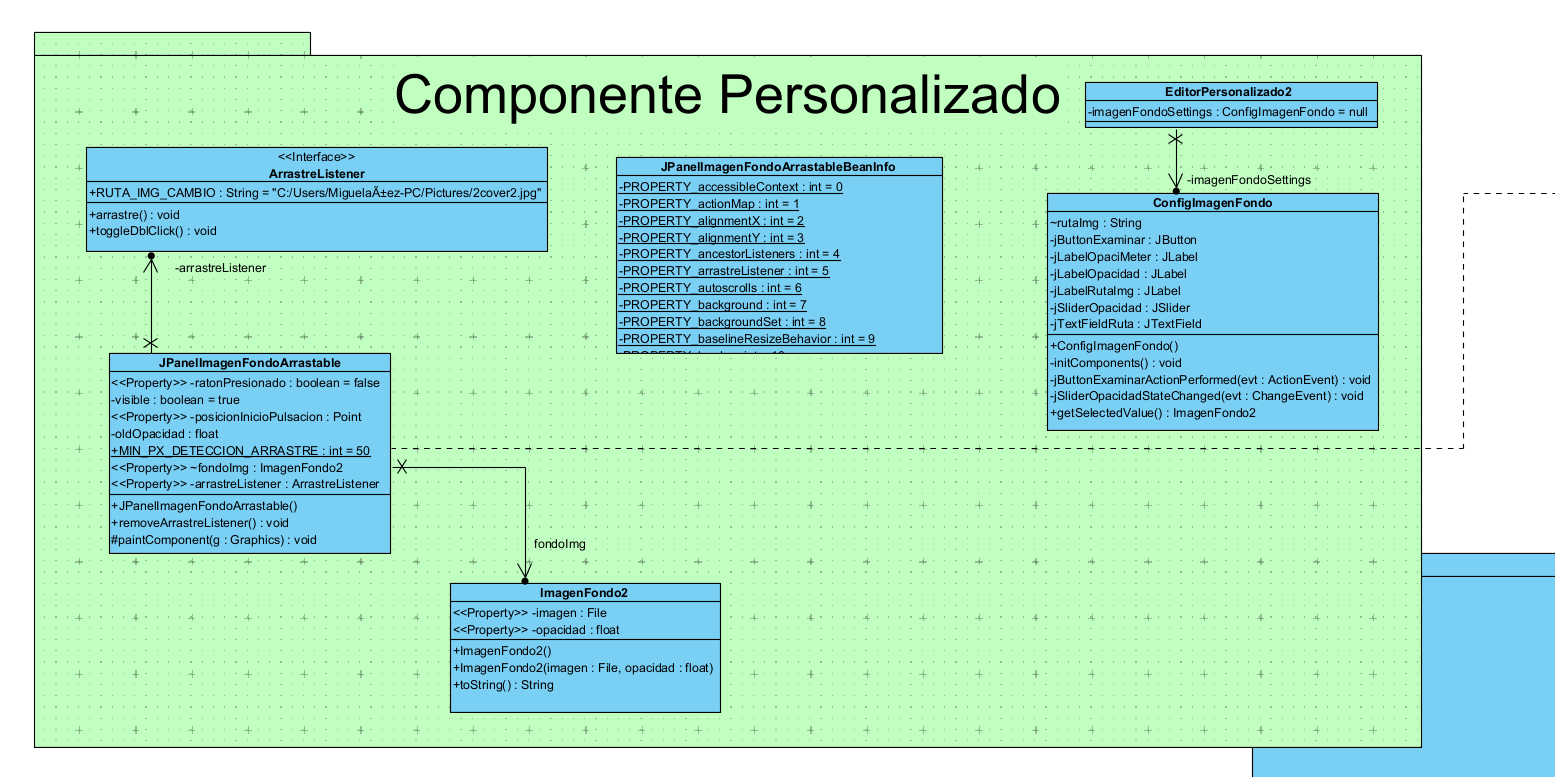
****

****

Paquetes **Exceptions y Controlador**

****

Paquete **Componente Personalizado**

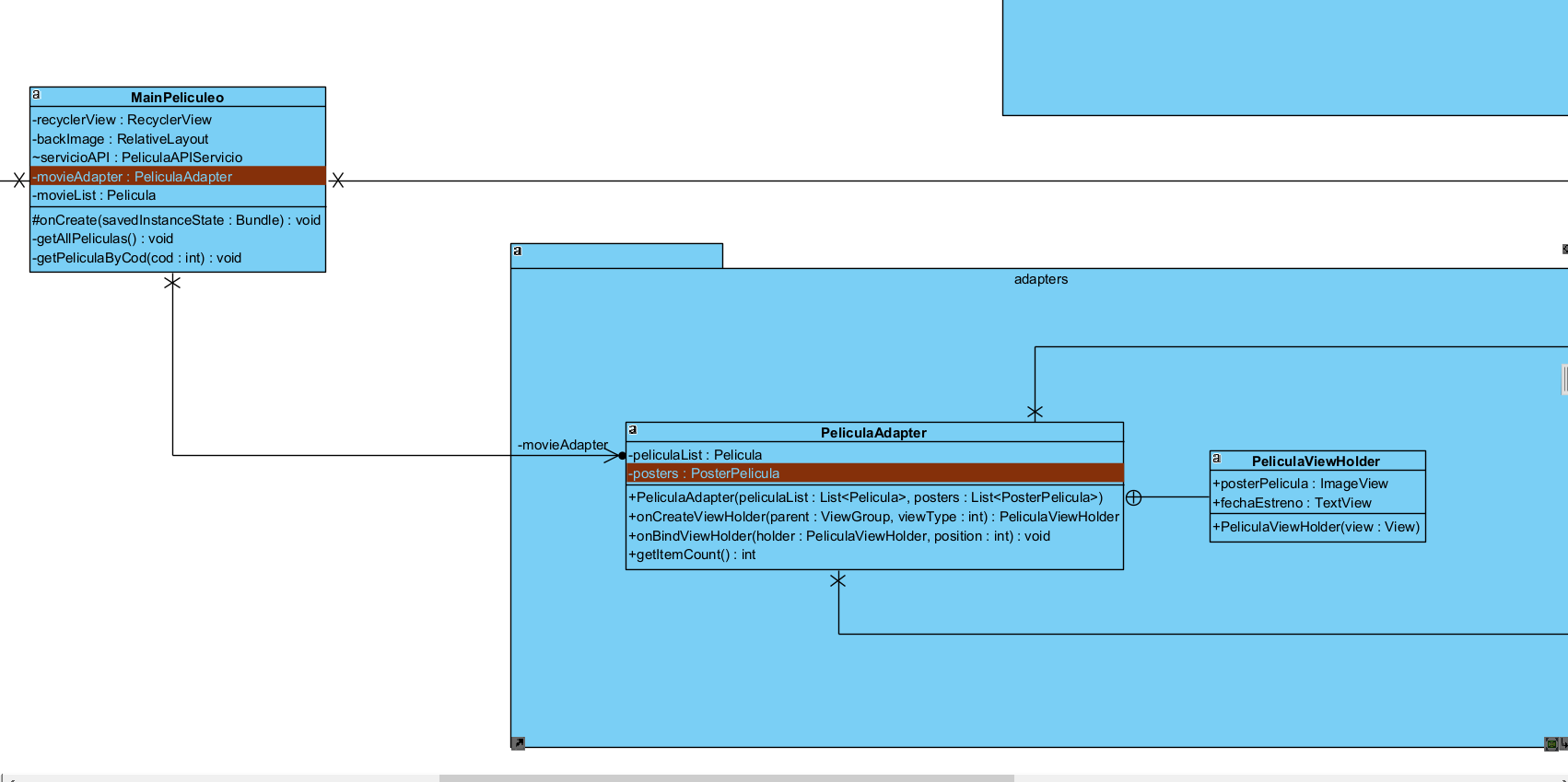
****

**Módulo Cliente 2 – “Peliculeo! Mobile”**

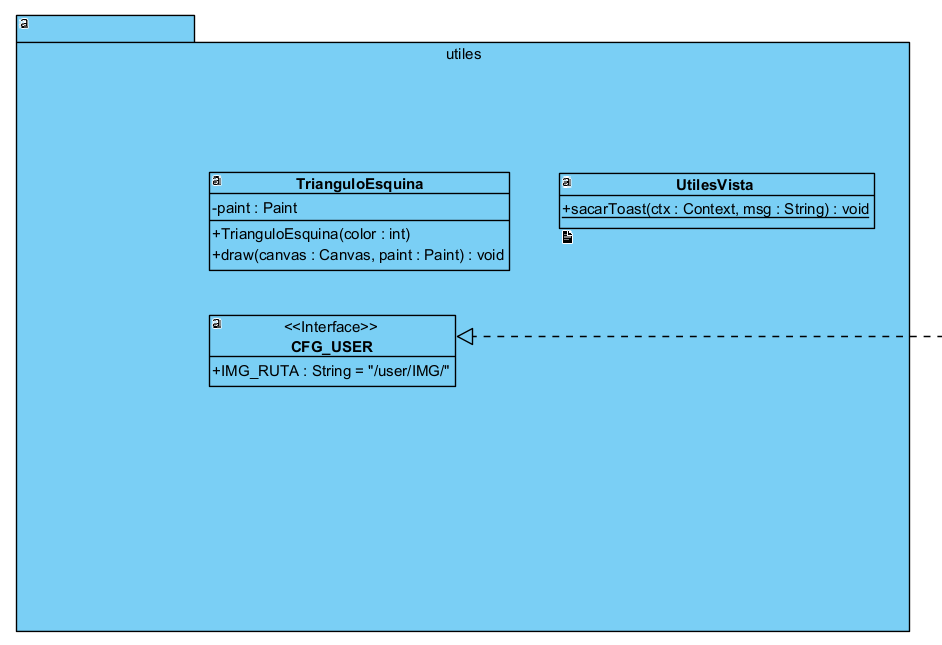
MainActivity, Cliente y uso Retrofit (APIs de Servicio)



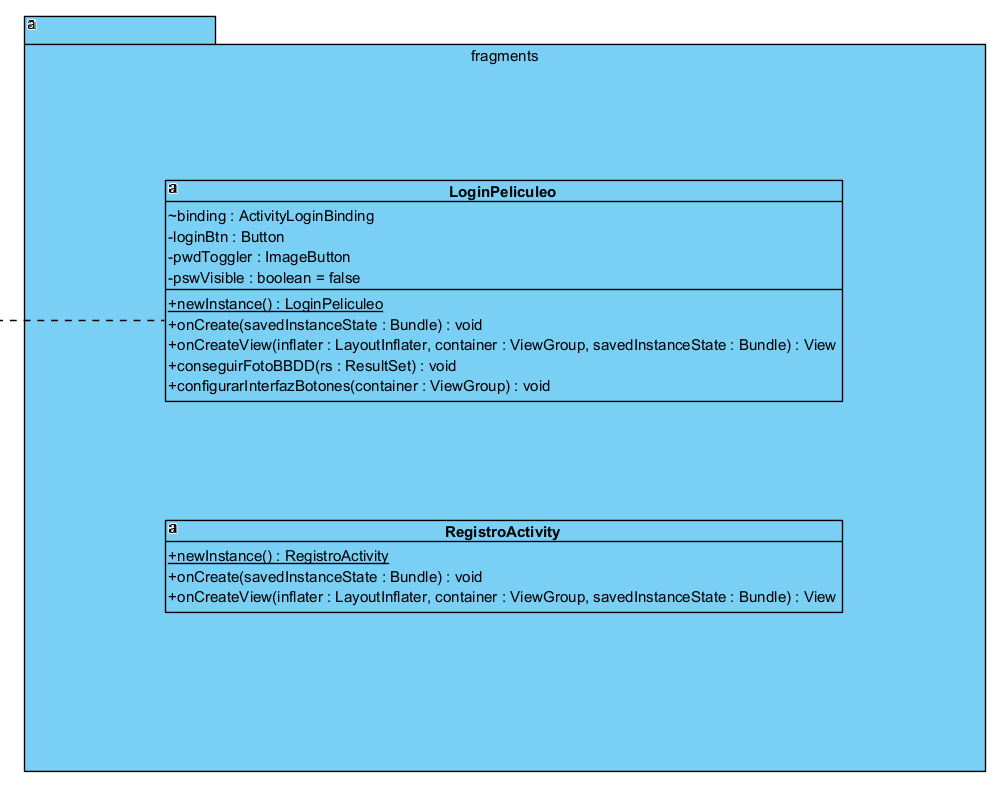
**MainPeliculeo** (main cartelera) y **Adapters** (RecyclerView)



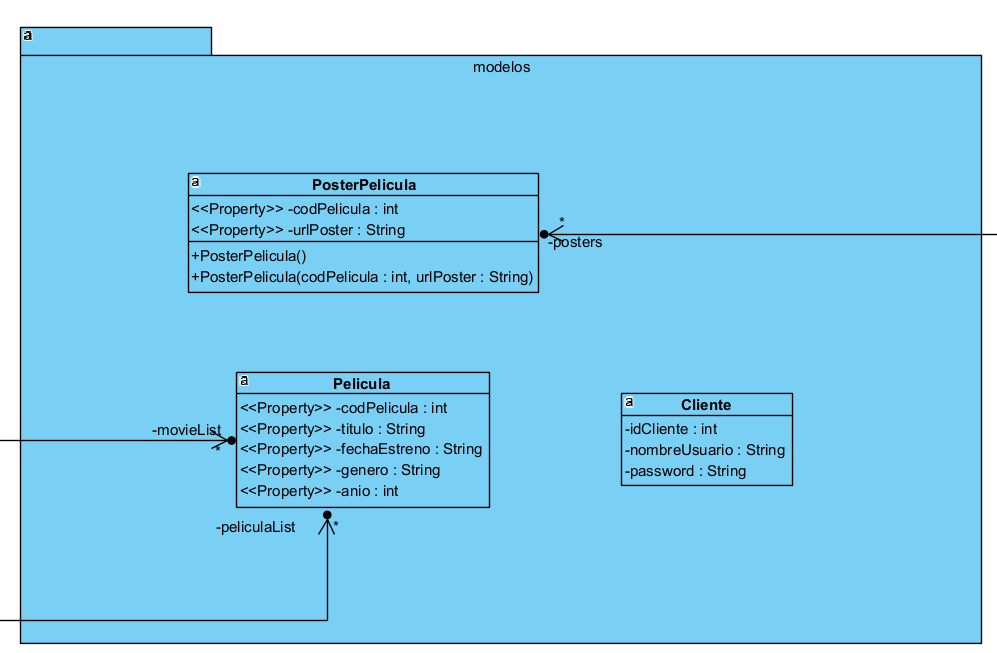
**Utiles**

****

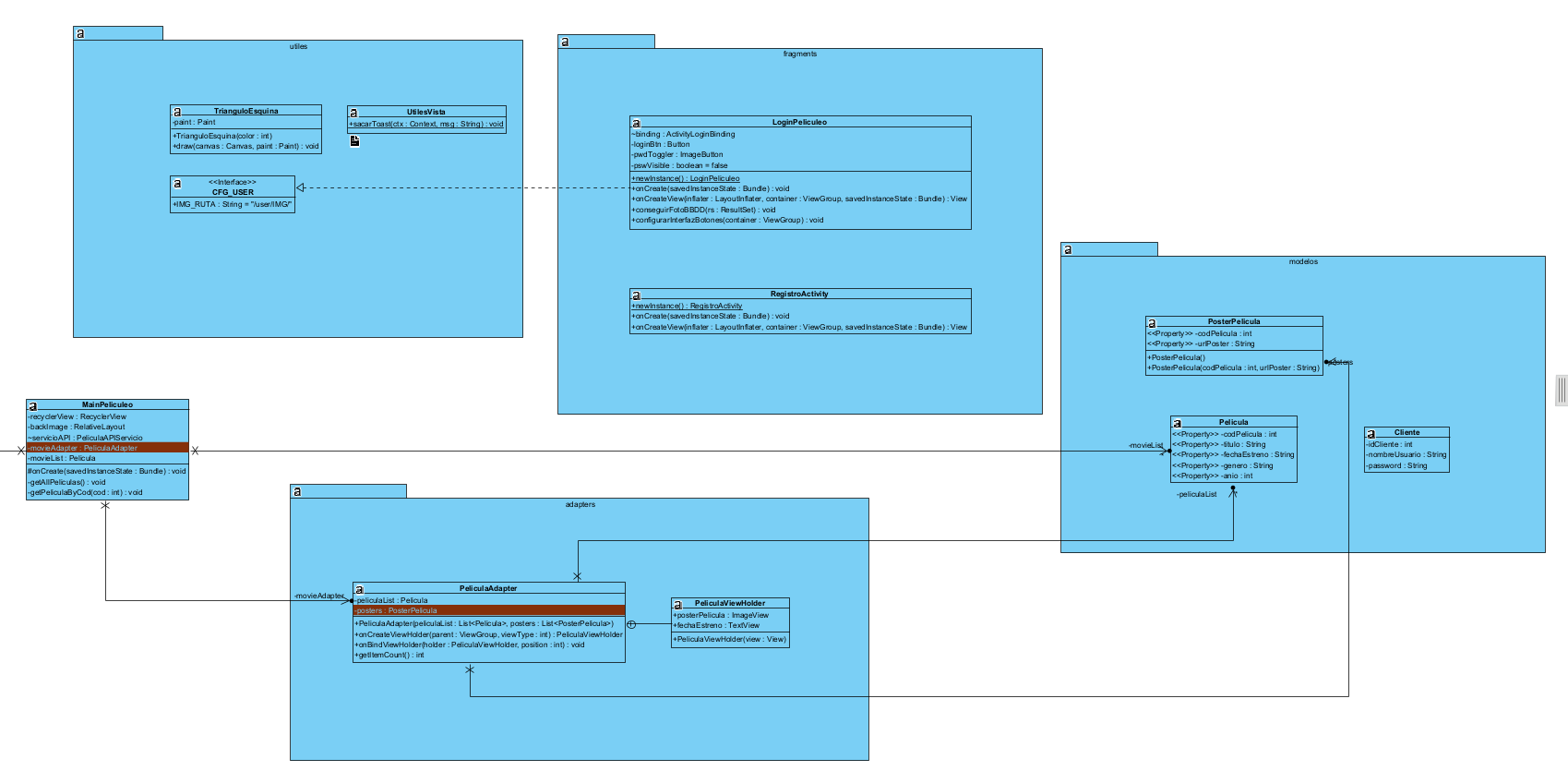
**FRAGMENTS** (Login y Registro)

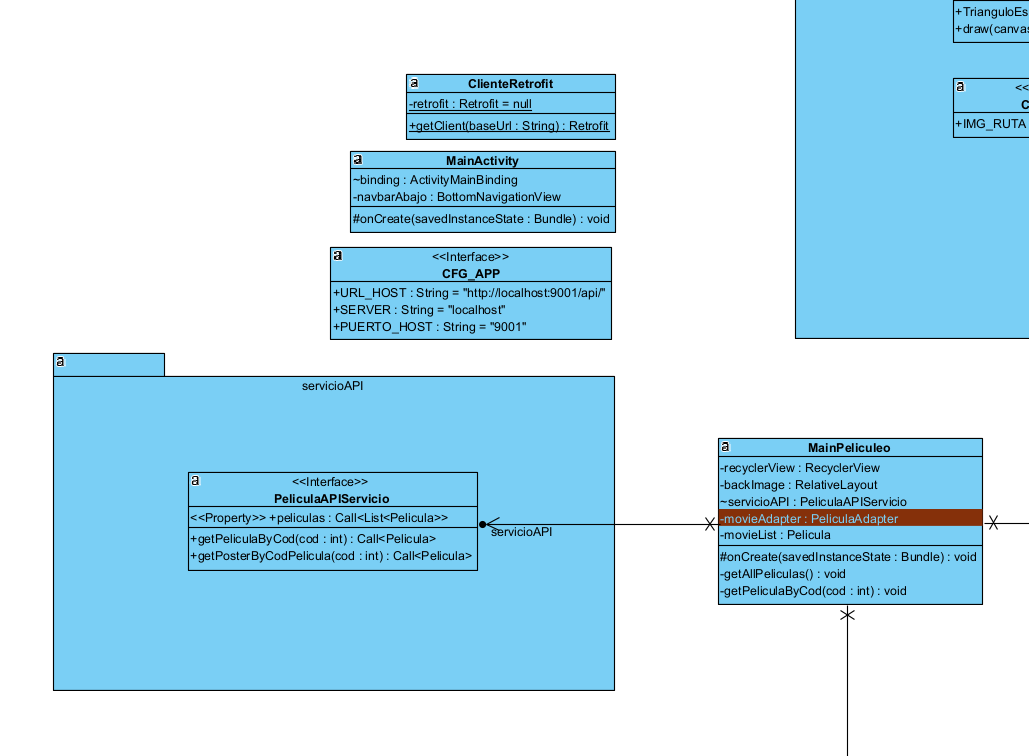
****

**Modelos**

****

**VISTA GENERAL**

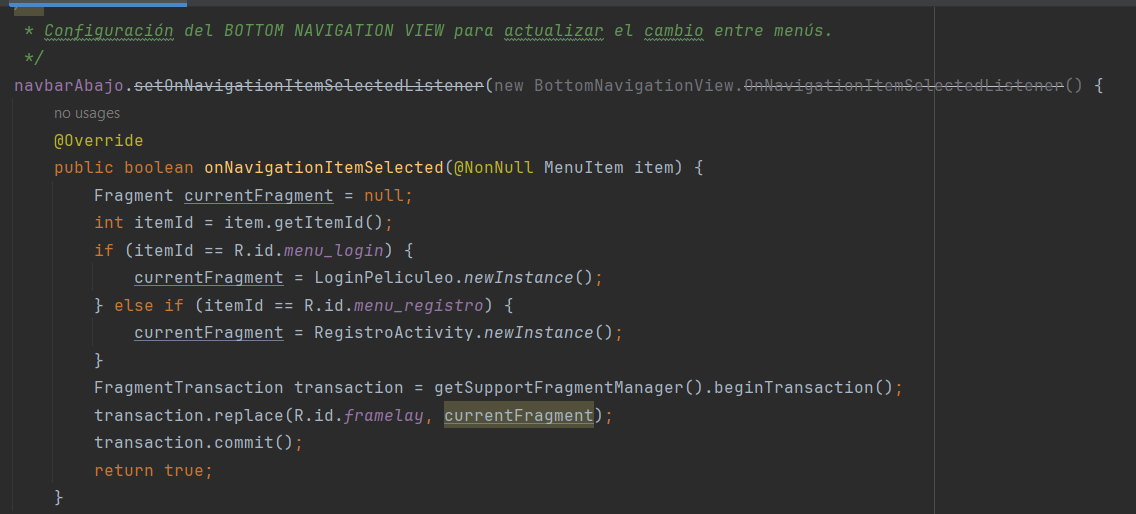
****

****

1. Vulnerabilidades conocidas

1. En la aplicación de Android, se reconoce el uso de Listeners deprecados (en Java, la dependencia **com.google.android.material:material:1.20.0@aar** ).

Más concretamente ***setOnNavigationItemSelected…*** usado para manejar qué fragmento (página) de Android mostrar cuando presionemos un botón del menú inferior u otro.



1. Bibliografía

Con el fin de perfilar el proyecto, se han recurrido a los recursos de clase que l@s profesor@s de Las Fuentezuelas han proporcionado a lo largo del curso, además de las siguientes fuentes de información:

[**Stackoverflow**](http://www.stackoverflow.com)–

Consultas sobre todo lo relacionado con revisión de errores.

Dudas menos frecuentes sobre la codificación de los módulos.

Uso de [dockerfile](https://stackoverflow.com/questions/24958140/what-is-the-difference-between-the-copy-and-add-commands-in-a-dockerfile?rq=2) para automatizar la containerización de la puesta en marcha del servidor

[**Superuser.com**](https://superuser.com/questions/1753996/how-to-fix-fresh-wsl-install-wslregisterdistribution-failed-with-error-0x80370) –

Aprender sobre el funcionamiento de Docker y refrescando mi memoria sobre el uso de permisos y aplicativos en Linux

[**Oracle DOCS (Java)**](https://docs.oracle.com/en/java/javase/11/docs/api/java.net.http/java/net/http/HttpClient.html) –

Documentación sobre uso de las librerías de Java para realizar peticiones HTTP a un servidor, así como uso de la API general de java.net

[**CodevsColor.com**](https://www.codevscolor.com/material-design-tutorial-6-navigationview)–

Android studio. Creación de un front-end intuitivo y fácil de dominar

[**Baeldung**](https://www.baeldung.com/) –

Distintas formas de aprovechar la API REST de SpringBoot. Spring security y estados HTTP.

[**Docker docs**](https://docs.docker.com/reference/cli/docker/image/build/) **… entre otros**

1. Conclusiones

Me ha encantado llevar a cabo este proyecto y he encontrado valiosísimo el hecho de poder sentarme a extender sobre todo lo aprendido en clases, descubrir toda la utilidad y potencial que las herramientas que ya conocía podían llegar a tener ha sido un proceso que me ha llenado muchísimo.

A pesar de la cantidad de trabajo con la que me he acabado encontrando en la formación de la FCT, he apretado y seguido adelante con este proyecto. La verdad es que no podría estar más orgulloso de lo que he conseguido.

Viniendo de un ciclo superior de Web, me ha sorprendido gratamente la enorme cantidad de herramientas que he adquirido este año cursado para llevar a cabo proyectos independientes, así como el hecho de que existan tantas librerías y formas de realizar una aplicación compatible con miles de dispositivos usando tan solo Java y herramientas como Android Studio.

Gracias a todo el profesorado del IES Las Fuentezuelas por ser tan abiertos y trabajadores, y por haberme dado esta gran oportunidad de sacar el proyecto adelante.

En especial, un fuerte abrazo a Álvaro, nuestro profesor de Desarr. de Interfaces y tanto a Jesús Pérez como a Soledad, si no fuera por ellos, me hubiera faltado toda la fundación básica para llevar a cabo estos proyectos

También darle un cordial abrazo a Manuel José, “Cazalla” por haberme acompañado durante 2 años consecutivos en mi camino por los grados superiores de Desarrollo de Aplicaciones y enseñarme todo lo que sé de scripting en Linux y redes. ¡Nunca olvidaré mi paso por el IES Alfonso XI y por Jaén!