**Proyecto de desarrollo de aplicaciones multiplataforma**

**GESTION ENTRADAS DE CINE**

CICLO FORMATIVO DE GRADO SUPERIOR

**Desarrollo de Aplicaciones Multiplataforma (IFCS02)**

**Curso 2021-22**

Autor/a/es:

**JEREMY ALEXANDER RAMOS SEGURA**

Tutor/a:

**JOSE LUIS GONZALEZ SANCHEZ**

Departamento de Informática y Comunicaciones

**I.E.S. Luis Vives**

Contenido

[2 INTRODUCCIÓN 3](#_Toc136897716)

[OBJETIVO 3](#_Toc136897717)

[ALCANCE 4](#_Toc136897718)

[Descripción del ptoyecto 4](#_Toc136897719)

[3 IMPLEMENTACIÓN 5](#_Toc136897720)

[REQUISITOS FUNCIONALES Y NO FUNCIONALES Y DE INFORMACION 5](#_Toc136897721)

[ANALISIS DEL MERCADO 6](#_Toc136897722)

[ANALISIS TECNOLÓGICO 8](#_Toc136897723)

[Ktor + Mongo 10](#_Toc136897724)

[Spring + Security 10](#_Toc136897725)

[IMPLANTACIÓN 15](#_Toc136897726)

[Aspectos Varios 15](#_Toc136897727)

[Código de la aplicación: 16](#_Toc136897728)

[Config 16](#_Toc136897729)

[JWT 17](#_Toc136897730)

[WebSockets 18](#_Toc136897731)

[Controllers 19](#_Toc136897732)

[DTO´S 21](#_Toc136897733)

[ResultsError 21](#_Toc136897734)

[Exceptions 22](#_Toc136897735)

[Mappers 22](#_Toc136897736)

[Modelos 23](#_Toc136897737)

[Repositories 24](#_Toc136897738)

[Interfaz Flows Con Results 25](#_Toc136897739)

[Validators 26](#_Toc136897740)

[Properties Back 26](#_Toc136897741)

[4 RESULTADOS Y DISCUSIÓN 28](#_Toc136897742)

[5 TRABAJO FUTURO (Opcional) 28](#_Toc136897743)

[6 CONCLUSIONES 28](#_Toc136897744)

[7 BLIOGRAFÍA 28](#_Toc136897745)

[ANEXOS 28](#_Toc136897746)

[I. ANEXO 1 28](#_Toc136897747)

[II. ANEXO 2… 28](#_Toc136897748)

# INTRODUCCIÓN

Como introducción al proyecto quiero recalcar que ya he tenido en cuenta la cantidad y gama de aplicaciones similares las cuales desempeñan el mismo papel. Pero como proyecto me pareció interesante ya que en estas aplicaciones hay algunas funcionalidades que para desempeñarlas como proyecto me resultan interesantes tales como la gestión de las reservas, la pasarela de pago etc. Por eso como opción al proyecto me resulto la más factible y teniendo en cuenta que el tema a tratar va de la mano con las películas será más ameno.



## OBJETIVO

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteCon este proyecto pretendo tanto aprender nuevas tecnologías orientadas al desarrollo multiplataforma. A través de encontrarme con Frameworks y otras características las cuales no he visto. También desarrollarme en el mundo multiplataforma para tener más cerca el objetivo de un desempeño full-stack.

Reformar y adquirir nuevos conocimientos para que de forma autóctona me refuercen la base para entrar en el mundo laboral.

## ALCANCE

-Establecer una comunicación simultanea en entre clientes y un servidor de forma íntegra.

-Realizar de forma correcta la gestión de validación mediante QR´s.

-Envío en Tiempo Real de Actualizaciones de estados de las Butacas reservadas. (Mediante el mecanismo de webSockets)

-Exportar la aplicación al apartado web.

Servidor

Se encargará de gestionar las peticiones de uno o varios clientes a la vez y almacenará dichos datos dentro de una base de datos no relacional.

Cliente

Web: Los cuales tendrán como funcionalidad el login de si mismos, la apreciación de un catálogo activo con las películas donde puedan realizar reservas a estas.

## Descripción del ptoyecto

Como he indicado en mi anteproyecto tengo como objetivo proporcionar un sistema de gestión de entradas de cine. El programa contará con un **cliente** que se le mostrará al usuario el cual dispondrá de ciertas opciones **que puede ver estando logueado o no dentro del sistema. “*Tal vez implementar algún tipo de descuento por estar logueado*”** Tales como: Ver en cartelera, Próximamente, junto con su respectiva portada, titulo, referencia de edades y tandas. **Sera una vista en cuadricula para maximizar el atractivo de la aplicación**. Una vez seleccionada la película deseada aparecerá la opción de elegir hora de visita y a su dicha selección aparecerá el apartado de butacas donde el usuario seleccionará el numero de butacas que desea siempre y cuando no estén seleccionadas, una vez hecho aparecerá una pantalla de resumen para que el usuario vea sus opciones y se pase al método de pago. Una vez hecho el pago al usuario se le entregará las entradas con un **código QR** adjunto el cual se usará para validar la entrada. Una vez finalizada la selección de entradas, película y butacas la aplicación te indicará si quieres seguir en la compra por si le interesa comprar más entradas y en caso de estar con la sesión iniciada desloguear también nos llevaría al apartado de películas habiendo estado logueados o no dentro de la aplicación.

En el aparado del servidor todo los datos internos de la aplicación: (*Datos de películas, datos de usuarios entrantes en la app, registros de entradas y redistribución de las mismas*) se guardarán dentro del mismo en su respectiva base de datos. Por consiguiente, al cliente contar con un sistema de logueo el servidor gestionará esa entrada con un framework que expondremos más adelante. También la gestión de las entradas se tiene que gestionar en tiempo real para evitar disturbios y sobrecarga de selección de entradas. Cuyo comportamiento y/o funcionalidades serán consumidos por nuestro cliente.

# IMPLEMENTACIÓN

## REQUISITOS FUNCIONALES Y NO FUNCIONALES Y DE INFORMACION

Requisitos Funcionales

Un requisito funcional es una declaración de cómo debe comportarse un sistema. Define lo que el sistema debe hacer para satisfacer las necesidades o expectativas del usuario. Los requisitos funcionales se pueden considerar como características que el usuario detecta. Son diferentes de los requisitos no funcionales, que definen cómo debe funcionar internamente el sistema (p. ej., rendimiento, seguridad, etc.).

Los requisitos funcionales se componen de dos partes: función y comportamiento. La función es lo que hace el sistema. El comportamiento es cómo lo hace el sistema.

R-F1) Crear, modificar, eliminar y mostrar películas a elección.

R-F2) Crear, modificar, eliminar y listar salas.

R-F3) Crear, modificar, eliminar y listar usuarios.

R-F4) Gestionar el sistema de generación de reservas.

R-F5) Sistema de identificación de usuario.

R-F6) Sistema de Registro de un Usuario.

R-F7) Método de pago asociado al formato especificado: Tarjeta, Paypal etc.

R-F8) Identificación del tipo de usuario para la gestión de ofertas.

R-F9) Permitir búsquedas mediante filtros.

R-F10) Validar la información introducida en el apartado de inicio de sesión.

R-F11) El usuario tiene la capacidad de restablecer su contraseña en cualquier momento.

R-F12) Permitir al usuario ver el resumen total de su pedido y/o modificar el mismo.

R-F13) No permitir que el varios usuarios de forma simultanea tengan la misma butaca.

R-F14) Generar una entrada impresa la cual mediante QR´s se pueda validar.

Requisitos No Funcionales

Un requisito no funcional representa las restricciones o condiciones que impone el

cliente al programa que necesita, por ejemplo, el tiempo de entrega del programa, el

lenguaje o la cantidad de usuarios.

Los requisitos no funcionales de la aplicación son:

RNF1) La aplicación cliente web y móvil ha de estar expuesta en Angular.

RNF2) Cliente y servidor deben ser dos subsistemas distintos interconectados.

RNF3) La parte del servidor tiene que albergar la posibilidad de gestionar varios clientes simultaneamente.

RNF4) El lenguaje de programación en él servidor debe ser Kotlin.

RNF6) El intercambio de información entre cliente y servidor será mediante WebSockets.

RNF7) El servidor contará con una base de datos relacional para la gestión de reservas.

RNF8) El servidor contará con una base de datos relacional para la gestión de Usuarios.

RNF9) La aplicación contará con un inicio de sesión de usuarios.

RNF10) La aplicación contará con un sistema de roles los cuales desempeñarán una acción.

RNF11) Las entradas se validarán mediante un sistema QR.

RNF12) La aplicación contará con un inicio de sesión de usuarios Mediante tokens.

RNF13) No es necesario que el usuario esté logueado para ver las butacas.

RNF14) El usuario tiene que estar logueado para comprarlas.

RNF15) La Base de datos de la aplicación estará regulada mediante SQL.

Requisitos De Información

RI1) Usuario. Es aquel que ejecuta la acción de interactuar con nuestra aplicación. Cada usuario tendrá un rol en este caso admin o usuario normal. El admin realizará acciones de actualización del listado de películas y gestión de los usuarios dentro de la aplicación, así como la gestión de entadas salas, etc. Y un usuario puede hacer reservas y consultar catálogo. Tiene como parámetros: Nombre usuario, email, contraseña, tarjeta , monto, rol, estado.

RI2) Películas. Son las distintas variantes que se mostrarán en un catálogo. Tienen como parámetros: Nombre, Portada, Directores, Actores, Duración, Categorías, Descripción y estado.

RI3) Salas. Son los diferentes espacios los cuales albergarán los usuarios para el disfrute de la película. Tiene como parámetros: Nombre, Tipo, Numero de Butacas, Tipo de butacas. Fila, Butaca,

RI5) Reserva. El resumen total con los parámetros de gestión. Tiene como parámetros: Nombre, Nombre Usuario, Identificador, Fecha, estado, Película, Sala, Numero Butacas, Total.

RI6) Entradas. Modelo el cual presentará la entrada una vez realizada la reserva. Tiene como parámetros: Nombre Película, Sala, Fecha, Hora, Fila, Butaca, Tipo Sala, Precio, Código QR.

RI7) Butacas. Son los componentes de las salas que tendremos en el cine. Tienen como parámetros: Fila, Numero, Tipo.

## ANALISIS DEL MERCADO

Como ya sabemos tenemos como base que partimos de que ya existen aplicaciones similares una de estas es la aplicación del **cinesa**.



Características de la app

Dicha aplicación cuenta con un sistema similar a lo expuesto en este proyecto ya que cuenta con un apartado del login el cual se complementa de usuario y contraseña con la posibilidad de registrarse y/o cambiar contraseña. También cuentan con la implementación de la gestión de las butacas por lo visto en tiempo real lo que conlleva un punto muy fuerte de la aplicación.

También cuenta con un buscador de cines lo cual lo hace independiente ya que podremos localizar cualquier cine de la franquicia y hacer lo pertinente sobre la aplicación.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Imagen que contiene teclado

Descripción generada automáticamente

## ANALISIS TECNOLÓGICO

Aquí se expondrán los distintos puntos de vista desde el nivel tecnológico de los cuales he elegido las tecnologías pertinentes que voy a utilizar en mi aplicación.

Base de Datos

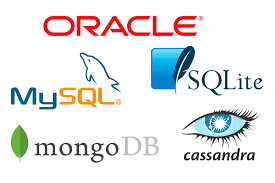
**Bases de Datos relacionales**

Las bases de datos relacionales se caracterizan por ser una colección ordenada de registros que se organizan en un conjunto de tablas. Estas tablas se relacionan entre sí.

Para acceder a estos datos, usaremos lo que se conoce como Lenguaje de Consultas Estructuradas, (SQL, Structured Query Language). Con SQL podemos obtener y alterar datos de una forma organizada siempre y cuando tengamos en cuenta cuál es la estructura de la base de datos con la que estamos trabajando.

Las bases de datos relacionales se organizan a través de identificadores. De este modo, cada tabla tiene un identificador único que es el que va a establecer su relación con el resto de tablas. A su vez, estos identificadores hacen que sea más fácil organizar cada una de las tablas por separado.

Los principales sistemas gestores de bases de datos relacionales son: MySQL, MariaDB, SQLite, PostgreSQL, SQL Server y Oracle.



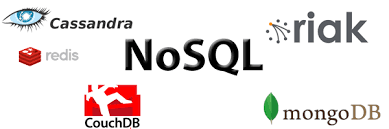
**Bases de datos no relacionales**

Las bases de datos no relacionales están diseñadas para modelos de datos específicos y que no necesitan ser relacionados con otros modelos. Cada tabla funciona de forma independiente y son mucho más sencillas que los modelos relacionales.

Las bases de datos no relacionales pueden tener identificador único, es decir, para identificar cada uno de los registros de la base de datos, pero este identificador no se usará (generalmente) para relacionar unos registros con otros. Como veremos, la información se organiza normalmente mediante documentos y es muy útil cuando no tenemos un esquema exacto de lo que se va a almacenar.

Con respecto a los formatos que se utilizan en las bases de datos no relacionales, podríamos decir que el formato más popular es el del documento. En muchos casos, lo que se utiliza es un objeto con una clave y un valor para que el acceso a la información sea pueda realizar de una forma sencilla.

Los principales sistemas gestores de bases de datos no relacionales son: MongoDB, Redis y Cassandra.



Elección

En este caso me decanté por el sistema de base de datos relacional ya que en función de lo que tengo planteado para mi aplicación tanto gestión de reservas, como de los distintos modelos representativos de la información, será mas provechosa que una base de datos Relacional. También en la gestión de usuarios será provechoso.

Logotipo, nombre de la empresa

Descripción generada automáticamente

Back-End

En este caso tengo Dos opciones: **Ktor y SpringBoot**

Mi elección en este caso ha sido Spring + Spring Security, ya que por consiguiente a lo expuesto anteriormente en el apartado de introducción las tecnologías expuestas en este programa me gustarían sentar una base para que en el entorno profesional prospere como programador y pienso que spring será una gran oportunidad.

## Ktor + MongoLogotipo Descripción generada automáticamente con confianza media

Imagen que contiene Logotipo

Descripción generada automáticamente

Podríamos utilizar ktor gracias a que nos proporciona un sistema de programación asíncrono gracias a las corrutinas lo que nos permite una gestión eficiente de los recursos. Nos ofrece la posibilidad de una arquitectura basada en **plugins** y de módulos y/o librerías de forma eficiente y rápida.  
Como opción a la base de datos tenemos MongoDB en este apartado ya que nos ofrece ventajas frente a una base de datos Relacional que nos permitirán manejar de forma íntegra gran cantidad de datos junto con su diseño NoSQ.

## Spring + Security

Logotipo, nombre de la empresa

Descripción generada automáticamentePodríamos utilizar Spring mas especifico SpringBoot para el manejo Api´s. Lo decidimos de esta manera gracias a que podremos crear Api’s con facilidad ya que cuenta con un conjunto de características RESTful las cuales nos ayudarán con la gestión de solicitudes http como ktor y como apunte importante es su gran escalabilidad y su buena integración frente a otros Frameworks como en este caso que hemos integrado la comunicación entre Ktor y Spring. La seguridad está basada en Spring 3 ya que es una versión reciente con lo cual estará actualizada y sobre todo la seguridad con spring es muy completa contando con clases específicas para la gestión de usuarios tales como: UserDetails y UserDetailService. Gracias a esta la gestión de usuarios se ve monitorizada por spring y en todo caso podremos configurar nosotros la gestión de estos en caso beneficiario.

Front-End

Estudiando las alternativas multiplataforma, y que tenga algún interés en aprender de este apartado he dado con la conclusión que tengo dos opciones Angular + Capacitor o Flutter.

### Angular + Capacitor

Angular es un framework Javascript potente, muy adecuado para el desarrollo de aplicaciones frontend modernas, de complejidad media o elevada. El tipo de aplicación Javascript que se desarrolla con Angular es del estilo SPA (Single Page Application) o también las denominadas PWA (Progressive Web App).

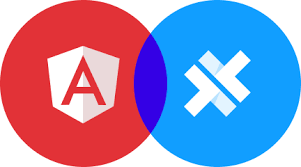
El framework Angular ofrece una base para el desarrollo de aplicaciones robustas, escalables y optimizadas, que promueve además las mejores prácticas y un estilo de codificación homogéneo y de gran modularidad.

El desarrollo en Angular se hace por medio de TypeScript (aunque también se podría desarrollar con Javascript, todas las guías y recomendaciones se basan en usar TypeScript), un superset del lenguaje Javascript que ofrece muchas herramientas adicionales al lenguaje, como el tipado estático o los decoradores.

**Capacitor**

Ionic es un framework para el desarrollo de aplicaciones híbridas. Con Ionic podemos construir aplicaciones para móviles Android, iOS, así como PWA y aplicaciones de escritorio, con una única base de código, aprovechando tus conocimientos de HTML, CSS y Javascript.

Gracias a Capacitor es muy sencillo acceder al SDK nativo de cada plataforma, con una interfaz de desarrollo unificada y optimizada para su uso con el framework Ionic. Capacitor es potente, sencillo y extensible vía plugins creados por el propio equipo de Ionic y la enorme comunidad de este framework.



### Flutter

Flutter es el kit de herramientas de UI de Google para realizar hermosas aplicaciones, compiladas nativamente, para móvil, web y escritorio desde una única base de código.

Algunas características son:

-**Desarrollo ágil**, con features como Hot reload, que te permite hacer cambios y ver las actualizaciones de manera instantánea.

-**Apps nativas con una misma base de código.** A diferencia de otros frameworks donde la gran parte del código se comparte, en Flutter el 100% del código funciona para ambas plataformas, y no tienes que escribir código personalizado para ninguna, esto mientras compila a aplicaciones nativas.



Elección

A mi criterio esta elección es muy difícil ya que si comparamos los pros y contras de cada framework flutter cuenta con unas ventajas sobre angular. Pero me puse a pensar y viendo que en el entorno profesional el cual me desenvolveré por ahora la preferencia de las aplicaciones es en angular y simplemente por el motivo de la opción a aprender tecnologías referentes a lo esencial actualmente me decanto por angular. También la idea del capacitor me llamo mucho la atención con lo cual sí, me decanto por angular. También de forma simultanea en las prácticas estaba desempeñando la creación de interfaces mediante angular con lo cual compaginar estas tecnologías será ameno.

Icono

Descripción generada automáticamente

**WebSockets**

**Como funciona WebSocket:**

La comunicación a través de plataformas web está basada en HTTP, la cual permite requerir información desde un cliente, usualmente un navegador, a un servidor, por ejemplo en Node.js. El problema surge cuando queremos hacer el proceso inverso. WebSocket resuelve el problema permitiendo que la información se transmita a través de mensajes en los dos sentidos en una misma conexión.

Esto nos trae muchas ventajas, como la reducción de carga a través de la red y principalmente una arquitectura que nos permite diseñar nuestros proyectos de una manera más sencilla, minimizando los problemas que nos podemos encontrar. Ahora bien, suele ser muy conveniente utilizar abstracciones que trabajan encima de WebSocket al desarrollar nuestras aplicaciones, ya que ellas nos proveen de utilidades ya pensadas para resolver los problemas más comunes a ocurrir.

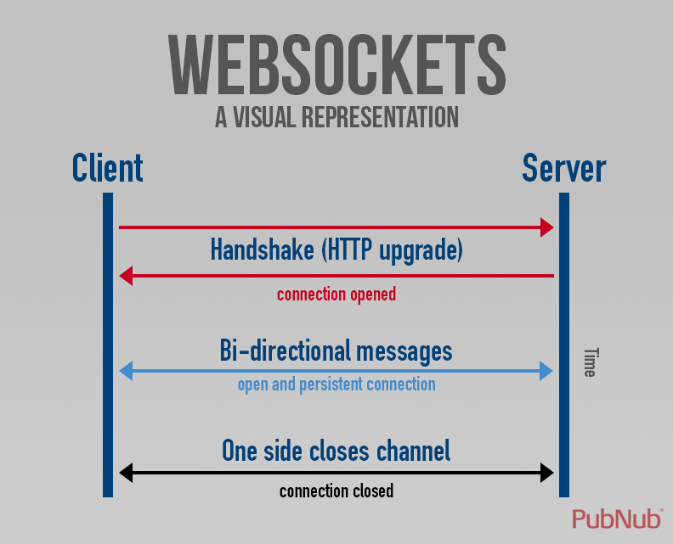


Diagrama De Clases

Diagrama

Descripción generada automáticamente

Explicación del Diagrama

En el diagrama apreciamos que nuestro sistema se compone de usuarios los cuales pueden visualizar de forma abierta las salas (1-1) donde dentro de cada sala dentro de esta se pueden reproducir en un día varias películas las cuales cuentan con su titulo el cual solamente se añadirá al realizar la reserva.

Dentro de la sala también nos encontramos con butacas, las cuales cuentan con su identificador y su numero butaca junto con la fila en la que se encuentra.

De otra manera un usuario puede realizar una reserva la cual está asociada a las butacas donde seleccionarán una butaca y dependiendo de las directrices de la reserva se le brindará la entrada o no.

También el en caso de las entradas tenemos que cada entrada tendrá su identificador y tendrá el título de la película, junto a que conlleva en sí su respectiva hora de creación, la reserva, el precio y un numero el cual será el de la sala y que en físico una vez impresa en su parte posterior contará con un QR para validar la misma.

### Diseño

Este apartado trata sobre la creación de los diagramas y bocetos varios los cuales se estarán propiciados en esta documentación.

Logotipo, nombre de la empresa

Descripción generada automáticamente

Despliegue de aspectos variados  
  
**Docker**

Docker es una plataforma de contenedores que permite empaquetar y distribuir entornos aislados y portátiles. Los contenedores son unidades de software livianas y autónomas que encapsulan todo lo que hace falta para ejecutar una aplicación, incluidas bibliotecas, herramientas y archivos de configuración, proporciona una forma eficiente y portátil de desplegar bases de datos, ofreciendo ventajas como la portabilidad, el aislamiento, la escalabilidad y la gestión de versiones.

Logotipo, Icono

Descripción generada automáticamente

## IMPLANTACIÓN

## Aspectos Varios

QR

Un dibujo de una persona

Descripción generada automáticamente con confianza bajaUn aspecto a tener en cuenta que he dicho que tendré que implementar será el tema de los QR´s ya que le dará realismo a la aplicación y a su contenido de salida.

Para ello he investigado y he encontrado una solución la cual consta de introducir una librería externa sobre mi proyecto llamada Zxing.

Para ello he introducido las siguientes dependencias sobre mi proyecto

* implementation 'com.google.zxing:core:3.4.1'
* implementation 'com.google.zxing:javase:3.4.1'

Las cuales ayudarán al mapeado y pintado del QR`s sobre el programa.

Plataforma de Pago

Icono

Descripción generada automáticamenteEn este tema he tenido que improvisar ya que para hacer que la aplicación contenga metodología de pago totalmente real tendría que pagar una pasarela de plataforma de pago para asociarla con mi proyecto y así poder representar el método de pago a través de esta. Lo que he decido es: Y si simulo la pasarela?. Que fue idea de mi tutor y me decidí por ello con lo cual por esa razón nuestro usuario en sus datos cuenta con una tarjeta y un monto el cual aumentará o disminuirá dependiendo de las reservas que se hagan o sobre las butacas. Lo haré todo mediante base de datos ya que es simulado y da la apariencia de realismo.

## Código de la aplicación:

La aplicación se compone de una arquitectura clave para su entendimiento y que sea lo más legible posible hacia las personas que la mantenga. La estructura es esta:

Texto

Descripción generada automáticamente

Donde cada paquete hará referencia a las clases que tenemos ya expuestas dentro del mismo. La aplicación trabaja con la gestión de apis por eso vemos controladores, validadores, results etc.

Aquí la explicación de cada paquete:

### Config

Texto

Descripción generada automáticamente

En este apartado expongo las configuraciones previas de mi programa.

Como primera opción tenemos el apartado de **APIConfig** el cual se caracteriza por tener la url de la api en caso de cambio.

**SecurityConfig**

En este apartado se exponen los accesos a la api junto con los Authentication Manager el cual configurará el acceso a nuestra api también utilizaremos el JWT para la creación y el manejo de tokens para acceder a las peticiones pertinentes.

Texto, Escala de tiempo

Descripción generada automáticamente

### JWT

Un JWT consta de tres partes separadas por puntos: el encabezado (header), la carga útil (payload) y la firma (signature). Cada una de estas partes se codifica en Base64URL y se concatenan con puntos para formar el JWT completo.

Imagen que contiene Icono

Descripción generada automáticamenteCuando se envía un JWT a un servidor, el servidor puede verificar la integridad del token extrayendo el encabezado y la carga útil, generando la firma utilizando la clave secreta correspondiente y comparándola con la firma incluida en el token. Si las firmas coinciden, se puede confiar en que el token no ha sido modificado y se puede acceder a la información contenida en él.

### WebSockets

Como por siguiente me pongo a relatar el tema de los webSockets en este caso tenemos una clase **ServerWebSocketConfig** la cual aplicamos el websSocket sobre los siguientes endpoints: **Butacas, Reservas, Salas.**

Dentro de sí tenemos los handlers los cuales nos harán de intermendiario entre los clientes y el websocket.

Con esto vamos a controlar las sesiones conectadas para poder establecer la conexión en tiempo real con la aplicación.

Por consiguiente, explico la creación de los handler:

En este caso tenemos la creación de las sesiones por cada conexión las cuales almacenaremos en una lista para que cada sesión se encuentre dentro del propio servidor dentro de la lista.



Con cada conexión nos aparecerá ciertos mensajes y se añade la sesión a la lista.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

También tenemos un método para enviar el mensaje establecido a cada sesión.

Texto

Descripción generada automáticamente

### Controllers

Aquí tenemos la parte crucial de nuestro back que en este caso es las llamadas a la api desde los controladores pertinentes.

Como ejemplo comenzamos a explicar el controlador de **usuarios.**

Tenemos varios apartados los cuales son importantes como primero tenemos un login y un register los cuales van referidos a los usuarios y Admins de la aplicación.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteComo versión especial tenemos un método para ver nuestro perfil en caso de solamente verlo y/o actualizar mi propio usuario. Las llamadas pertinentes tienen como etiqueta el preAuthorized para que solamente los usuarios con Token puedan acceder a estas llamadas para control de seguridad.

Texto

Descripción generada automáticamente

Como segundo ejemplo tenemos ButacasController que este controlador se encargará de controlar la creación y asignación de butacas a una sala.

Texto

Descripción generada automáticamente

En este caso vemos que de forma correcta devolvemos la respuesta pertinente y en otro caso devolvemos un handleError. En este caso estamos haciendo uso del apartado de los Results para controlar lo pertinente de entrada y salida sobre los endpoints junto a ese método tenemos un conjunto de atributos los cuales son los errors sobre cada objeto. Donde relatamos que según el error devolveremos un *NOT\_FOUND* o un *BAD\_REQUEST* en caso de fallo.

Texto

Descripción generada automáticamente

### DTO´S

Siguiendo con las butacas ya que básicamente los controladores son iguales simplemente controlamos la entrada y salida de cada uno. Siguiendo con el ejemplo anterior sobre cada método nos entra un Dto en el caso de butacas existen unos validadores previos para comprobar la entrada de la butaca. En este caso tenemos ciertas etiquetas validadoras que podemos establecer contra los atributos.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Donde cada entrada tendrá sus respectivos validadores previos a la entrada de los datos a la API.

### ResultsError

En este caso tenemos los results error, vemos que básicamente es una sealed class

con un mensaje que entra como parámetro el cual mapea las clases notFound y BadRequest.

Texto

Descripción generada automáticamente

### Exceptions

Un caso para analizar es el tema de las excepciones las cuales nos permite devolver errores pertinentes en este caso como ejemplo tenemos el UsuarioException el cual mapeamos dos status: NOT\_FOUND y BAD\_REQUEST.

Texto

Descripción generada automáticamente

### Mappers

Texto

Descripción generada automáticamenteAquí expongo el tema de los mapeos de los Dtos entrantes y salientes dentro de la API en este caso presento el mapper de entradas el cual de forma interesante le entra una reserva la cual busca primeramente si existe y esa reserva la busca y dependiendo de esa reserva crea la entrada con esa reserva creada.

También existen modelos de mapeo a modelos de forma directa en este caso tenemos el de reservaDto.toModel el cual mapea los dtos a modelos propios de la base de datos.

Texto

Descripción generada automáticamente

### Modelos

En este caso tenemos el apartado de modelos los cuales son las entidades que mapearemos para el transcurso de entrada y salida de la base de datos en este caso como ejemplo tenemos butacas los cuales cuentan con ciertas etiquetas tales como:

**@Table:** Argumento el cual indica el nombre de la tabla expuesta.

**@Id:** Crea los id´s de forma automática dentro de la base de datos.

**@Column:** Las columnas pertinentes de la base de datos.

Texto, Chat o mensaje de texto

Descripción generada automáticamente

Estos datos se encuentran dentro del archivo sql el cual muestra que han sido mapeada a la base de datos.

Texto

Descripción generada automáticamente

### Repositories

Aquí expondremos un apartado importante el cual es los repositorios los cuales hacen las operaciones pertinentes sobre la base de datos CRUD y métodos varios que no están explícitamente impuestos dentro de los métodos normales de la clase.

En consecuencia, tenemos como implementación un CoroutinieCrudRepository

al usar coroutines, podemos realizar operaciones de acceso a datos de forma asíncrona, lo que puede mejorar el rendimiento y la capacidad de respuesta de tu aplicación. Esto es especialmente útil cuando tienes operaciones de E/S costosas, como acceder a una base de datos o llamar a una API externa.

Tambien ya que estoy utilizando un enfoque reactivo en aplicación, como R2DBC en este caso, las coroutines son una buena opción para manejar operaciones asincrónicas de forma reactiva. Podemos combinar la programación reactiva y las coroutines para crear un flujo de trabajo eficiente y reactivo.

Logotipo

Descripción generada automáticamente con confianza baja

En este caso tenemos como ejemplo el repositorio de butacas donde tenemos una relación 1-N la cual se maneja mediante el método de findSalaId donde ese método según las butacas creadas en esta sala nos devuelve un flujo con las butacas pertinentes añadidas a esa sala.

Texto

Descripción generada automáticamente

### Interfaz Flows Con Results

En este caso tenemos como interfaz un conjunto de métodos los cuales vamos a modelar accediendo a un servicio el cual implementará todos estos métodos donde en algunos tendremos como objeto resultante un Result y su pertinente clase de error en este caso ButacasError.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamenteAlgunos métodos de nuestro servicio son los siguientes:

Texto

Descripción generada automáticamente

### Validators

Aquí tenemos como clase de validación sobre los dto´s de en este caso las butacas,

tenemos un método de extensión sobre los dto´s entrantes los cuales en caso de errores devolverán un result de forma errónea y en caso de pasar por cada argumento tendremos el objeto devuelto de forma positiva.

Texto

Descripción generada automáticamente

### Properties Back

En este caso tenemos un archivo llamado **application.properties** el cual nos permite configurar apartados externos los cuales aplican sobre la aplicación los cuales pueden ser: La conexión con la BBDD, la creación y tiempo de expiración de los tokens jwt, ssl etc..

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

# RESULTADOS Y DISCUSIÓN

…

# TRABAJO FUTURO (Opcional)

…

# CONCLUSIONES

…

# BLIOGRAFÍA

…

# ANEXOS

…

## ANEXO 1

## ANEXO 2…