# PROYECTO DE PORTAFOLIO

COMUNIDAD

VROL

Autor:   
Cristian Carrillo

# Introducción

Como proyecto de portafolio aborde parte de las necesidades de la Comunidad VRol de la cual participo, así que hagamos una leve introducción para repasar los puntos que dan razón a este proyecto:

#### ¿Qué es la Comunidad VRol?

La comunidad de VRol, es como indica una Comunidad de distintas personas, siendo la mayoría de la Quinta Region (ende la V para indicar Quinta) y de otros lugares de Chile, siendo su nombre oficial Quinta Rol. Esta parte como la iniciativa de ser un punto de encuentro para los amantes de los [Juegos de Rol](https://es.wikipedia.org/wiki/Juego_de_rol). Esto ya que en la Quinta región no se encontraban tan organizados como los que juegan en la región Metropolitana, y que sobre todo en tiempos de COVID donde las juntas presenciales no se pudiesen realizar con normalidad (o realizarse siquiera) hiciera que varios quedaran aislados. El grupo en si parte en Facebook por miembros integrantes de un Grupo mas grande en Chile que es Roleros.cl, también organizando un grupo de chat de WhatsApp para tratar de reunir a todos los roleros de la región para así poder organizar partidas online, y su ocasional presencial, y es aquí en este punto donde comienza a surgir esta necesidad.

#### Textos de chat perdidos

Una de las primeras cosas que se empezaron a notar, es que pasaba a menudo que la información pertinente a la organización de las partidas se perdía en el gran flujo de mensajes del chat (estamos hablando que hay mínimo un flujo de 200 mensajes al día) por lo cual siempre se tenia que estar preguntando cada tanto a los administradores por esta información pertinente. Esto creo la primera herramienta de control que fue crear una Calendario Google que tiene una información básica de las partidas que se están llevando a cabo y parte de la planificación de eventos de la comunidad.

#### Información relevante poco accesible.

Depender del calendario ayudo, pero no eliminaba todo, sin mencionar que el calendario era visible solo para administradores, quienes procedían a sacar una leve captura cada vez que se preguntaba (hasta el día de hoy aun se hace), lo que llevo a que se usara el medio usual para realizar las partidas on-line para también dejar una agendacion de los eventos, Discord. Pero esta plataforma aun funciona como chat, por lo cual si bien las secciones dejadas para el posteo de eventos estaban bloqueadas para posteo, esto aun no deja una claridad inmediata sobre algunos de los datos de la partida, aunque ha sido una mejora sin duda.

Entonces, como se puede apreciar, en favor de crear algo que sirva para unificar las fuentes de información de las mesas y eventos es que decidí tomar esto como caso de desarrollo para mi Portafolio.

# DESARROLLO

Vamos entonces a abordar el desafío de este proyecto, el cual ha sufrido de bastantes transformaciones desde su primer génesis como simplemente un portal web el cual aun es visible a través del sitio web que sigue publicado [VRol](https://vrol.netlify.app/) a través de [Netlify](https://www.netlify.com/) (también pueden ver una que realice en base a modelo pero en Angular [VrolSite](https://vrolsite.netlify.app/)).

## Análisis del modelo

Después de decidir utilizar un modelo en base a [JPA](https://es.wikipedia.org/wiki/Java_Persistence_API), lo primero fue derivar una estructura básica de lo que seria el modelo de la base de datos la cual para las necesidades mas inmediatas cuenta solo con las siguientes 4 Tablas: la tabla de Mesas, Jugadores Inscritos, Contacto y Usuarios. Por lo tanto el primer paso fue crear estas Clases, y colocar las anotaciones necesarias para definirlas como Entidades, dejo una imagen del modelo final (a la fecha).

Diagrama UML

Estructura de la Base de Datos

Al ocupar JPA, nuestro principal objetivo es simplemente crear el Schema y dejarlo definido en el archivo de las propiedades de la base de datos para poder así indicar al sistema que estamos trabajando con él y de ahí empezar a trabajar en las clases.

### Dependencias

Un punto importante antes de siquiera hablar de crear Entidades, vamos a por las dependencias usadas para este proyecto Springboot, de las cuales aquí dejo marcada la primera más importante:

Texto

Descripción generada automáticamente Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Por supuesto la Dependencia de JPA, también voy a destacar otras que son de relevancia en post de este proyecto, que son las Dependencias de [Thymeleaf](https://es.wikipedia.org/wiki/Thymeleaf) pues marcan una diferencia marcada con respecto a la de mis compañeros de curso: Thymeleaf no soporta JSP, solo trabaja con HTML y otros, ¿ahora porque Thymeleaf? La verdad lo vi en mas de un tutorial y me pareció interesante y que trabajaba de una forma más interesante, aunque similar al uso de las JSTL (tiene comandos que incluso operan en integración con Spring Security) por lo que decidí darle una prueba y así permitirme aprender a ocupar algo nuevo. De ahí las otras dependencias son algo mas clásicas, como las necesarias de Spring para trabajar, el Conector J para trabajar con MySQLL, Junit y otras.

### Entidades

Texto

Descripción generada automáticamenteEl concepto de Entidad es clave para su interacción con JPA a diferencia del modelo por conexión JDBC, ya que en este se necesita insertar la anotación exclusiva que indica que esta Clase se indica como entidad y de ahí poder crear su espejo en la base de datos.

@Entity: Anotación Principal para indicar que esta es una Clase tipo Entidad.

@Table: Anotación que Indica que esta clase es una Tabla, y que nombre llevara.

@Id: Anotación que indica que este atributo es del tipo Identidad.

@GeneratedValue: generalmente usada junto con Id, para indicar que se autogenera.

@NotEmpty: esto indica que este valor no puede estar nulo.

@Email: anotación que da un parámetro general para indicar que el valor indicado aquí debe de llevar mínimo el @.

@Pattern: sirve para determinar que el valor ingresado debe cumplir con un patrón, en este caso, que solo serán números y son 9 dígitos solamente.

@ManyToOne y @JoinColumn: estas anotaciones son las que hacen posible determinar que esta tabla esta enlazada a otra (en este caso Mesa), y que su relación es de una a muchas.

Aquí dejo un ejemplo con el caso de la Clase Jugador, la cual crea entonces su Tabla jugadores\_inscritos, e inserto los atributos que tendrá esta clase los cuales son idInscrito, nombreJugador, email, teléfono y Mesa. Esta clase tiene como objetivo ser el medio para que cualquier persona que desea se pueda inscribir a una Mesa de Juego, por ende, su relación de Uno es a Muchos, pues en este caso 1 Mesa puede tener X jugadores.

### Sobre los Usuarios

Como clarificación, en este sistema, hay tres tipos de usuarios: usuario general o visita, Narrador, y Administrador. El primero es el usuario genérico que visite la página, no necesita permiso especial para visitar casi todos los menús y vistas. El segundo, Narrador, puede ver los jugadores inscritos en las mesas y crear mesas. Mientras que solo el Administrador puede crear otro usuario Administrador o Narrador, crear mesas, listar usuarios y en general total acceso a la aplicación.

## Análisis de la Rubrica

A continuación, me basare en la rubrica para cubrir aspectos del proyecto y demostrar su aplicación a lo largo del proyecto:

### Consulta a Base de Datos

#### 1.- Selecciona las columnas requeridas para presentar la información solicitada

Como mencionado anteriormente, al estar usando JPA, el uso y selección de muchas de las posibles “queries” esta ya hecha en si, JPA a través de JpaRepository, ya nos entrega las Queries mas básicas que componen el Select que son las funciones que parten de un find, Insert Into y Update, que se absorben en el save, y Delete por delete.

Imagen que contiene Texto

Descripción generada automáticamente

Aquí por ejemplo se está llamando a una función que pide llamar a todos los jugadores inscritos, y pasarlos como objeto al modelo de vista para obtener lo siguiente

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

#### 2.- Utiliza JOIN para relacionar la información de distintas tablas

Como se indico al mostrar la clase Jugador, la Clase Jugador, esta estaba vinculada a la Clase Mesa, lo que nos indica están enlazadas en una relación de 1 es a muchas. Aquí dejo señalado de forma explicita cuales datos son de la tabla Jugador y cual de Mesa

**Jugador**

**Mesa**

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

#### 3.- Utiliza WHERE para filtrar la información requerida,

#### 4.- Utiliza cláusulas de ordenamiento para presentar la información, Y

#### 5.- Utiliza cláusulas de agrupación de información para obtener datos agregados

Texto

Descripción generada automáticamente

Aquí, en la clase repositorio podemos ver estos métodos que salvo el que lleva la anotación @Query, las otras solo por “escribirse” tal cual conllevan a decir que son queries en si lo que se llama “queries nativas” donde simplemente por decir “find” es sinónimo de decir Select, findAll es “Select \*”, By viene siendo el “Where”, OrderBy es evidente, y acompañando al OrderBy es el campo por el cual queremos que ordene, si escribo Asc ordenara por ascendente, y si acompaño con Desc e forma descendente. Por ultimo tenemos el caso del método findByMesa en donde tenemos la anotación @Query e indicamos la Query para expresar lo siguiente “cuenta y agrupa todos los jugadores donde la mesa sea X. Dejare las vistas de los casos 3, 4 y 5

##### Utiliza WHERE para filtrar la información requerida

Aquí es “Buscame a todos los jugadores que están en la mesa 1”.

O “SELECT \* FROM jugadores\_inscritos WHERE idMesa = 1.”

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

##### Utiliza cláusulas de ordenamiento para presentar la información

“Buscame a todos los jugadores y ordenalos de forma descendiente según su id”

“SELECT \* FROM jugadores\_inscritos ORDER BY idInscrito Desc”

Texto

Descripción generada automáticamente

##### Utiliza cláusulas de agrupación de información para obtener datos agregados

“Cuéntame todos los jugadores que están en la mesa 1”

“SELECT nombre\_mesa, count(id\_inscrito) from jugadores\_inscritos j JOIN mesas m WHERE m.id\_mesa = 1 group by id\_mesa”

Pantalla de video juego

Descripción generada automáticamente con confianza media

### Algoritmo de cálculo y unidades de prueba

#### 6.- Utilización general del lenguaje, sintaxis, selección de tipos de datos, sentencias lógicas, expresiones, operaciones, comparaciones

Ejemplo de selección de datos

Texto

Descripción generada automáticamente

Ejemplo de comparaciones, sentencias expresiones

Texto

Descripción generada automáticamente

#### 7.- Utilización de sentencias repetitivas

Ejemplo

Texto

Descripción generada automáticamente

#### 8.- Utilización de clases, encapsulamiento y responsabilidad única

Ejemplo

Texto

Descripción generada automáticamente

#### 9.- Se utilizan correctamente interfaces o relaciones de herencia para hacer polimorfismo donde fuese necesario

En el caso de este proyecto, el uso de interfaces, herencia es marcado sobretodo entre las clases de repositorio y servicio

Ejemplo Clase Jugador Repository la cual extiende (hereda) de JpaRepository

Texto

Descripción generada automáticamente

Ejemplo Interface (lo cual antes seria un DAO) IJugadorService

Texto

Descripción generada automáticamente

Ejemplo de Clase JugadorServiceImpl la cual implemente Interface IJugadorService

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

#### 10.- Convenciones y estilos de programación

Varias desde el Uso de camelCase y otras como uso de getters, Setters, etc.

Como ejemplo en la siguiente clase podemos apreciar las siguientes:

* El nombre de la clase "Jugador" usa una convención de nomenclatura en CamelCase, donde la primera letra de cada palabra se escribe en mayúscula, excepto la primera letra de la primera palabra.
* Las anotaciones utilizadas, como @Entity, @Table, @Id, @GeneratedValue, @ManyToOne, @JoinColumn, son convenciones que pertenecen a la especificación JPA (Java Persistence API), y se utilizan para mapear la clase a una tabla de base de datos.
* Los nombres de los atributos y métodos siguen la convención de nomenclatura en camelCase, donde la primera letra de la primera palabra se escribe en minúscula, y la primera letra de cada palabra siguiente se escribe en mayúscula.
* Las constantes como "serialVersionUID" se nombran utilizando una convención de nomenclatura en mayúsculas y minúsculas (camelCase) y en mayúsculas, respectivamente.
* Los nombres de los métodos, como "getIdInscrito", "setNombreJugador", "getEmail", etc., siguen la convención de nomenclatura de los métodos getter y setter.
* La clase tiene un método toString() que sigue la convención de nomenclatura de este método y escribe la representación en cadena del objeto.
* Texto

  Descripción generada automáticamenteTexto

  Descripción generada automáticamenteEn general, la clase sigue el estilo de codificación en Java, que incluye indentación, uso de llaves, comentarios, etc.



#### 11.- Utilización de unidades de prueba

Dejo muestra de Caso de Prueba de Clase ContactoServiceImpl

Texto

Descripción generada automáticamente

### Página web html y css

#### 12.- Utilización de tags html, estilos y responsividad

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

#### 13.- Utilización de Bootstrap

Tal cual como demostrado en [punto 12](#_12.-_Utilización_de) en el cual se muestra la plantilla de HTML, se usa archivos de estilos de Bootstrap, además de archivos de css propio, mismo caso ocurre con el archivo de JS de Bootstrap

### Spring MVC

#### 14.- Utilización de controladores

Este proyecto contempla la creación (al momento) de 10 Controladores activos los cuales son: CalendarController (controla la vista de la sección de Calendario en la pagina), ContactoController (controla la vista de formulario y el envio de mensaje), HomeController (controla el despliegue de home o vista principal), JuegosController (controla el despliegue de la vista de juegos y a futuro también controlara los juegos por individual), JugadorController(controla la inscripción de un nuevo jugador, listarlos, editarlos y borrarlos), JuntasController(controla el despliegue de la vista de álbum con fotos de las juntas), ListarJController(controlador REST, despliega información de los jugadores que se han inscrito en mesas, se espera que también la información sea descargada como un pdf próximamente), LoginController(controla el login, desde vista, proceso y logout), MesasController (controla la vista de las mesas, y controlara la creación, edición y borrado de mesas), y UsuarioController (controla la creación, listado, edición y eliminación de usuarios). En el [punto 8](#_8.-_Utilización_de) deje un print de la clase controladora ContactoController.

#### 15.- Utilización de vistas JSP y Taglib

Tal y como señalo en el ítem de [Dependencias](#_Dependencias) casi al final, decidí utilizar Thymeleaf, la cual no soporta el uso de JSP. Las TagLib de JSP son etiquetas personalizadas que se pueden utilizar para simplificar y mejorar la legibilidad de los archivos JSP. Thymeleaf, como motor de plantillas, no utiliza taglib en el sentido tradicional de JSP, pero proporciona una serie de atributos y directivas personalizadas para simplificar la escritura de plantillas HTML.

Aquí hay algunos ejemplos de etiquetas Thymeleaf que imitan el uso de TagLib en JSP:

**th:include** - Utilizado para incluir fragmentos de plantilla en otra plantilla.

<th:block th:include="fragments/footer :: footer"></th:block>

**th:replace** - Utilizado para reemplazar un elemento HTML con otro elemento o fragmento de plantilla.

**<th:block** th:replace="fragments/header :: header"></th:block>

th:with - Utilizado para definir una variable local en una plantilla.

<th:block th:with="titulo='Bienvenido'">

<h2 th:text="${titulo}"></h2>

</th:block>

**th:attr** - Utilizado para establecer atributos HTML en un elemento con el valor de una variable o expresión.

<a th:attr="href=@{/producto/{id}(id=${producto.id})}, title=${producto.nombre}">

<h2 th:text="${producto.nombre}"></h2>

</a>

**th:text** - Utilizado para establecer el contenido de un elemento HTML con el valor de una variable o expresión.

**th:if y th:unless** - Utilizados para establecer estructuras condicionales en una plantilla.

<th:block th:if="${usuario != null}">

<h2>Bienvenido, <span th:text="${usuario.nombre}"></span>!</h2>

</th:block>

**th:each** - Utilizado para iterar sobre una lista o colección de elementos.

<th:block th:each="producto : ${productos}">

<h2 th:text="${producto.nombre}"></h2>

<p th:text="${producto.descripcion}"></p>

<p th:text="${producto.precio}"></p>

</th:block>

#### 16.- Creación de Servicio Spring y

#### 17.- Creación de DAO acceso a Datos

Este al ser un proyecto SpringBoot con JPA no tiene la presencia de las clases DAO tan explicitas como lo es en un proyecto Spring MVC normal, en ves de los DAO tenemos las clases Repositorio y los Service que son Interfaces (estos últimos los cuales se implementan en una clase que las Implementa). En el [punto 9](#_9.-_Se_utilizan) deje prints de JugadorRepository, IJugadorService y JugadorServiceImpl

#### 18.- Creación del Proyecto y Configuración

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

#### 19.- Funcionamiento General del Aplicativo

Captura de pantalla de computadora

Descripción generada automáticamente

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación, Calendario

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Captura de pantalla de computadora

Descripción generada automáticamente con confianza media

Una captura de pantalla de una computadora

Descripción generada automáticamente

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente con confianza media

### API REST

#### 20.- Creación servicio Rest

De momento tengo solo la Clase Controladora ListarJController, la cual permite a alguien acceder a el listado de los jugadores inscritos a mesas en formato JSON al acceder a través de la dirección /ListarJ y ver la información de los jugadores en forma individual a través de cada numeral (ej: /ListarJ/1)

Texto

Descripción generada automáticamente

## Conclusión

Puedo decir que ha sido una Aventura, sin más, entre buscar información, documentarse, ir, intentar, probar, seguir probando, ver las innumerables veces que el código me ha estallado. Pero el resultado final me deja satisfecho. Hay mucho mas por hacer para seguir perfeccionando este proyecto muchas mas funcionalidades que le quiero agregar y mucho más. ¿Y quién sabe?, aprendiendo nuevas tecnologías, permitirme darle una vuelta mas tarde y convertirla en una app para celular.

A destacar debo dejar que hacer la lógica de la construcción de la inscripción de Jugadores y la de mesas es una que me hizo pasar bastante tiempo.