MELI SHOWS API

BIENVENIDO AL SHOW BUSINESS! Estás incursionando en el mercado del espectáculo brindando **una API que permita a diferentes portales la venta de tickets para shows artísticos** (actuación, música, etc.) concentrando la oferta de diferentes teatros, sus shows y **pudiendo realizar la reserva de la o las localidades para una función del show**.

Nuestra API tendrá dos usuarios directos:

1. Quién alimenta de información la plataforma agregando los teatros con sus correspondientes salas y butacas.
2. El portal (que nos utilizara para hacer una reserva en nombre de un espectador).

El primero, aparte de dar de alta los teatros, sus salas y butacas, también carga los shows y funciones disponibles (especificando en qué sala de cada teatro, será cada función); además de los precios de las butacas para cada sección (grupo de butacas) de la sala en cuestión.

Nota: Una obra puede decidir tener diferentes precios para diferentes secciones o puede directamente decidir cobrar lo mismo para cualquier ubicación. También es posible que para una misma sala, dos obras diferentes tengan distintas secciones o como mencionamos, una sola.

El otro usuario será el portal, quien mediante su aplicación nos utilizará para hacer la reserva en nombre del espectador (su cliente final). Para esto la API deberá poder disponer de la siguiente funcionalidad (que es la te pedimos que implementes)**:**

- Poder **listar los shows y funciones disponibles** con todas sus características.
- **Para una función de un show** deberá ser posible listar **las butacas disponibles con el precio** de las mismas.
- Y por último deberá poderse **realizar una reserva para la función de un show**, indicando DNI, nombre del espectador y las butacas con el precio registrando todo a modo de Ticket de reserva.

Nota: Toma en cuenta la posibilidad de que 2 clientes vean disponible la misma localidad, uno haga la reserva y después el segundo cliente intente también reservar. La API debería no dejar al segundo realizar la reserva si alguna de las localidades que solicita ya fue reservada.

Todo lo referente a la carga de datos puede ser ignorado y cargado del origen que elijas (archivo, resource, hardcoded, etc.). **No es necesario que la API exponga esta funcionalidad (** podes ignorar los endpoints para hacer la carga de la información de shows, teatro, salas, etc.)**,** mas si que al iniciar la información la obtenga de algún lado. En resumen tiene que tener datos para poder utilizarla pero no es necesario que esten expuestos los endpoints para hacerlo via la API de modo de simplificarle el problema.

Te pedimos:

1) Implementar la API con la funcionalidad solicitada respetando los principios RESTful.
2) Agregale a la API la posibilidad de realizar búsquedas de shows con criterios complejos como: todos los shows que tengan funciones programadas entre la fecha X y la fecha Y, que su precio esté dentro del rango de precios Z y K y pudiendo solicitar que los resultados esten ordenados de forma ascendente o descendente según algún atributo del show.
3) Implementar una forma de asegurar que la información solicitada más frecuentemente no se busque en el repositorio de datos, sino que esté disponible de una forma más eficiente.
4) Hostear la API en un cloud computing libre (Google App Engine, Amazon AWS, etc).

EXTRA OPP - BUGS BUNNY y EL CAZADOR (opcional)

Enfocado 100% en lograr un modelo orientado a objetos lo más elegante posible, este ejercicio solicita programar una simulación de la siguiente situación que cumpla con los requerimientos indicados (puede ejecutarse directo en consola, no hagas una API ni una UI). **Las reglas del juego son la siguientes:**

- Tu modelo debe tener el comportamiento solicitado.
- El evaluador de tu modelo podrá instanciar tu modelo en el main y configurarlo para que funcione.
- Debes tratar de que el evaluador no pueda violar el comportamiento deseado (hacer trampa) sea por como setea la simulacion o porque implementa una nueva clase que permite violar el comportamiento.
- El evaluador no tocará tu código ni modificará tu modelo pero si el problema lo especifica debería poder implementar nuevas clases en base a tu modelo para que sean utilizadas en la simulación.

Situación:
El cazador Elmer está buscando a Bugs Bunny en el bosque para cazarlo y lleva una escopeta, por ejemplo una escopeta Remington modelo 750 con balas calibre 45. Elmer le dispara a un venado. Para un venado, un disparo de calibre 45 es mortal. Si el cazador estuviese utilizando una pistola Colt con balas calibre 22 en vez de la escopeta con balas calibre 45, el venado podría sobrevivir a 1 disparo, pero no a 2.
Elmer sigue buscando a Bugs Bunny, pero si se encontrase con un oso (el Oso Yogui), necesitaría dispararle por lo menos 3 veces con la escopeta, mientras que se calcula que si le disparase con la pistola calibre 22, debería asestarle por lo menos 20 disparos para cazarlo. Por suerte Elmer no encontró a Bugs, ya que el impacto de cualquier arma en él sería mortal.
Por la descripción se puede apreciar que no todos los calibres tienen el mismo efecto en las diferentes presas de Elmer.

Tu modelo debería poder implementar nuevos tipos de animales y nuevos tipos de armas de fuego que Elmer (un cazador) podría utilizar pero no debería ser posible ni implementar armas que maten cualquier cosa ni animales que sean invencibles.

Nota: El evaluador deberá instanciar un cazador, darle un arma de un tipo específico para que le dispare a la presa que se le indique capturar.

```
public static void main(String args[]) {

    Hunter elmer = new Hunter();
    Gun gun = new RemingtonRifle( model: 750,  calibre: 45);
    Animal yogi = new Bear( age: 10, weight: 1500,  height: 300);

    elmer.shoot(yogi, gun);
    System.out.println(yogi.isAlive()? "Run Yogi!":"Oh no, poor Yogi");

    //Any of us (reviewers of your challenge) should be able to implement
    // new guns & animals using your base classes.
    // Remember not to let us cheat (being able to set isAlive to false or similar)
    // i.e:
    Gun crossbow = new Crossbow(); //Ballesta
    Animal dino = new Dinosaur();
    elmer.shoot(dino, crossbow);
    System.out.println(dino.isAlive()? "Run Elmer!!!":"Man, what a powerful crossbow!");
```

Entregables del Challenge:

- Código fuente en repositorio de GitHub.
- Documentación que indique cómo ejecutar el programa.
- Documentación del proyecto que considere importante.
- URL en donde esté hosteado el servicio.
- Contemplar buenas prácticas (tip: imaginar que estas poniendo una aplicación productiva).