**Examen Práctico de Spring Boot - Uso de manejo de hilos, alta disponibilidad y concurrencia**

**Instrucciones:**

En este examen, se le pedirá que desarrolle una aplicación Spring Boot que maneje ***concurrencia y alta disponibilidad mediante el uso de hilos***. Siga las siguientes instrucciones para completar el examen.

**Proyecto de Spring Boot:**

* Cree un nuevo proyecto de Spring Boot utilizando Spring Initializer o su método preferido.
* Asegúrese de incluir las dependencias necesarias para crear una aplicación web y para el manejo de hilos.

**Requisitos de la Aplicación:**

Su aplicación debe cumplir con los siguientes requisitos:

1. Se requiere que el servicio posea alta disponibilidad, por lo que se solicita que se explique cómo sería la arquitectura. El objetivo es que la arquitectura sea on premise. No se acepta arquitecturas que maneje todo en la nube.
2. Debe tener un ***controlador REST*** que acepte solicitudes HTTP POST en la ruta **/api/tarea-concurrente**. A este controlador le llegara una lista en formato JSON con el siguiente formato (ejemplo: example.json):

{

"envios": [

{

"shippingId": 1,

"time-start-in-seg": 120,

"next-state": true

},

{

"shippingId": 2,

"time-start-in-seg": 180,

"next-state": false

},

{

"shippingId": 3,

"time-start-in-seg": 240,

"next-state": true

}

]

}

Donde:

* **shippingId:** id del envío.
* **time-start-in-seg:** momento en el tiempo donde será lanzada la tarea. El momento 0 (cero) se considera cuando se llama al endpoint.
* **next-state:** si es “true”, pasa al siguiente estado el envío; si es “false”, cancela el envío.
  + En caso de true, el siguiente estado respeta lo siguiente:
    - Inicial -> Entregado al correo -> En camino -> Entregado.

1. Cuando se envía una solicitud a **/api/tarea-concurrente** la aplicación deberá ejecutar cada elemento de la lista según “time-start-in-seg” y aplicar lo que indique “next-state”. ***Cuando corresponda cambiar el estado, se debe usar los siguientes endpoints del proyecto “mareo-envios”(aclaración personal: yo le llamo “mercadoEnvios”, no “mareo-envios”)***. Aclaracion: ***Se debe enviar los comandos al proyecto “mareo-envios”,*** no se acepta que en un solo proyecto se resuelva la lógica de “mareos-envios”, **el objetivo es evaluar el manejo entre los 2 proyectos**.

/shippings/{shippingId} <- PATCH

\* Body:

\* {transition: sendToMail } -> Debe demorar mínimo 1 segundo

\* { transition: inTravel } -> Debe demorar mínimo 3 segundos

\* { transition: delivered } -> Debe demorar mínimo 5 segundos

\* { transition: cancelled }-> Debe demorar mínimo 3 segundos.

1. La aplicación debe coordinar la ejecución de las tareas de manera que no se puedan ejecutar 2 o más tareas a la vez para el mismo shippingId. En caso de que no se puede ejecutar una tarea, porque ya existe otra ejecutándose, se debe registrar en una tabla.
2. La aplicación debe registrar en la tabla cuando una tarea se terminó exitosamente o cuando no pudo continuar por algún motivo.
3. Además, la aplicación debe contener un endpoint **/api/status**, del tipo **HTTP GET**, que informe el estado de los envíos, y que se pueda consultar, sin que interfiera con la ejecución de las tareas.
4. En la verificación que se va a realizar, se probara dejar en ejecución lo enviado a **/api/tarea-concurrente**, y luego se solicitara el **status** de la misma, y ***ambas solicitudes deben poder convivir sin problemas***.

Se adjunta casos de ejemplo de input.

**~~Puntos Extra (Opcional):~~**

~~Si desea puntos adicionales, puede implementar cualquiera de las siguientes características adicionales:~~

* ~~Agregar un límite de tiempo para la ejecución de las tareas y responder con un mensaje de error si se supera ese límite.~~
* ~~Implementar Swagger para la ejecución de los endpoints.~~
* Implementar test-unitarios que considere necesarios.

**Recomendaciones:**

* Utilice el enfoque de anotaciones de Spring para configurar su aplicación y crear controladores.
* Utilice la clase ***Thread*** o utilice la API de **ExecutorService** de Java para crear y gestionar hilos.
* Utilice ***mecanismos de sincronización***, como **CountDownLatch**, para coordinar la ejecución de los hilos.
* ~~El objetivo del ejercicio es evaluar como desarrolla los diferentes requerimientos, en caso de no llegar con los tiempos, también es válido, enviar en seudocódigo de como resolvería los requerimientos faltantes.~~
* ~~No olvide desarrollar también lo solicitado en el proyecto “mareo-envios” adjunto.~~

**Mareo Envíos**

Marcos junto a sus amigos, cansados de las malas gestiones de las empresas de correo, se plantearon armar una empresa de envíos de mercadería a todo el país gestionadas digitalmente. Debido a la urgencia de salida al mercado, te contactaron para que le entregues WebService para que ellos puedan evolucionar con facilidad.

**Requerimientos técnicos**

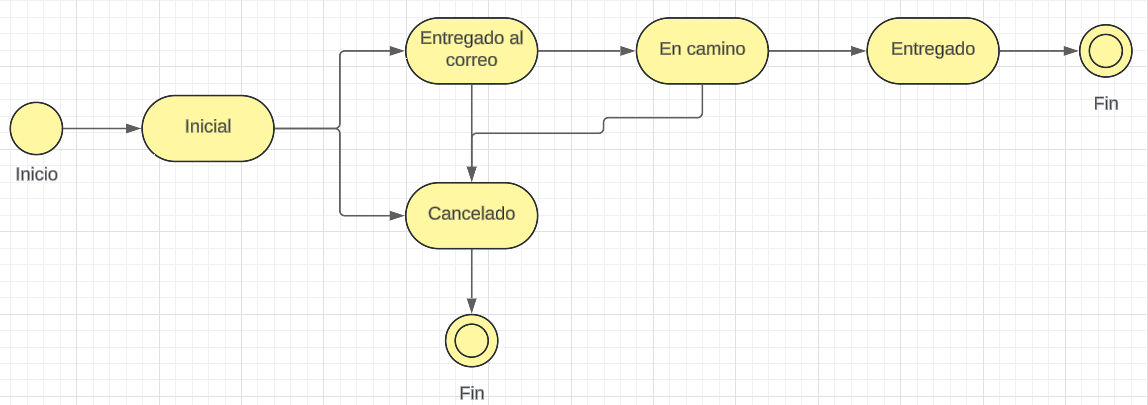
* **Arquitectura WS**: RESTful Json
* **Lenguaje**: Java 17
* **Formato proyecto**: maven
* **Framework**: Spring Boot
* **Base de datos**: MySQL o PostgreSQL

**Opcionales**

* ~~Incluir~~ ***~~Dockerfile~~*** ~~de 2 etapas para buildear con Docker~~
* ~~Preparar todo el stack con~~ *~~docker-compose~~* ~~(IE: app + db)~~
* Escribir *tests unitarios y de integración*
* ~~Documentar la API con~~ **~~Swagger~~**

**Requerimientos funcionales**

* Poder obtener información del comprador.(Customer)
* Poder obtener información del envío y productos comprados.(Shipping)
* Poder cambiar el estado del envío siguiendo las siguientes reglas de negocio:
  + El estado inicial es “Inicial” y solo se puede transaccionar a “Entregado al correo” y o “Cancelado”
  + El estado “Entregado al correo” solo se puede transaccionar a “En camino” y o “Cancelado”
  + El estado “En camino” solo se puede transaccionar a “Entregado” o “Cancelado”
  + Los estados “Entregado” y “Cancelado” son estados finales y no se pueden transaccionar.



* Obtener un listado con la descripción y cantidad de los 3 productos más solicitados para su envío.

**Firmas para utilizar**

* /customers/{customerId} <- GET
* /shippings/{shippingId} <- GET
* Firmas de transacciones:
  + /shippings/{shippingId} <- PATCH
  + Body:
    - { transition: sendToMail }
    - { transition: inTravel }
    - { transition: delivered }
    - { transition: cancelled }
* /reports/top-sent <- GET

**Aclaraciones**

* El usuario no pidió la creación de pedidos ya que nos comentó que lo tiene resuelto.
* Dejó la creación de su esquema de datos y algunos datos de ejemplo para que sea más sencillo realizar las pruebas
* Nos avisaron que el modelo de datos quedará fijo como nos lo pasaron, por lo cual no se pueden alterar las columnas ni crear tablas nuevas. Excepto para la tabla Task\_shipping donde podrá agregar más columnas si lo considera necesario.
* Es necesario implementar manejo de errores en las respuestas de las distintas firmas.
* Se incluye pom.xml, scripts de base de datos y codigo mínimo para inicializar el proyecto.

**Entrega:**

Envíe su proyecto completo, incluyendo el código fuente y cualquier archivo necesario para ejecutar la aplicación. ~~Asegúrese de que su aplicación se ejecute correctamente y cumpla con los requisitos mencionados anteriormente.~~

**~~Plazo de Entrega:~~**

~~La fecha límite de entrega es hasta el 14/07/2024 a las 23:59hs.~~

~~¡Buena suerte!~~