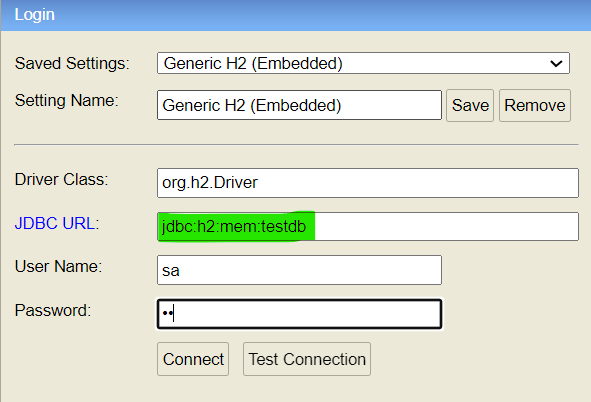
***README***

requisito previo antes de levantar “mercadoEnvios” tener una base creada en mySQL con el nombre “mercado\_envios”.Las tablas y datos de prueba, son cargados automáticamente al levantarse el proyecto.

Se adjunta el zip “mercenvios” el cual deberá ser importado. El IDE utilizado por mí fue Eclipse.

Levantar ambos proyectos: “sistema” y “mercadoEnvios”

Sistema se ejecuta en el puerto 8080 y usa h2. Podes entrar a la consola para ver los datos que va guardando: <http://localhost:8080/h2-console/>



Usuario:sa

Password:sa

mercadoEnvios se ejecuta en el puerto 8081 y usa mysql para la persistencia. El username y password son ambos: root

**endpoints**

mercadoEnvios:

1. Poder obtener información del comprador:   
   GET [http://localhost:8081/customers/{customerId}](http://localhost:8081/customers/%7bcustomerId%7d)
2. Poder obtener información del envío y productos comprados:

GET [http://localhost:8081/shippings/{shippingId}](http://localhost:8081/shippings/%7bshippingId%7d)

1. Poder cambiar el estado del envío:

PATCH [http://localhost:8081/shippings/{shippingId}](http://localhost:8081/shippings/%7bshippingId%7d)

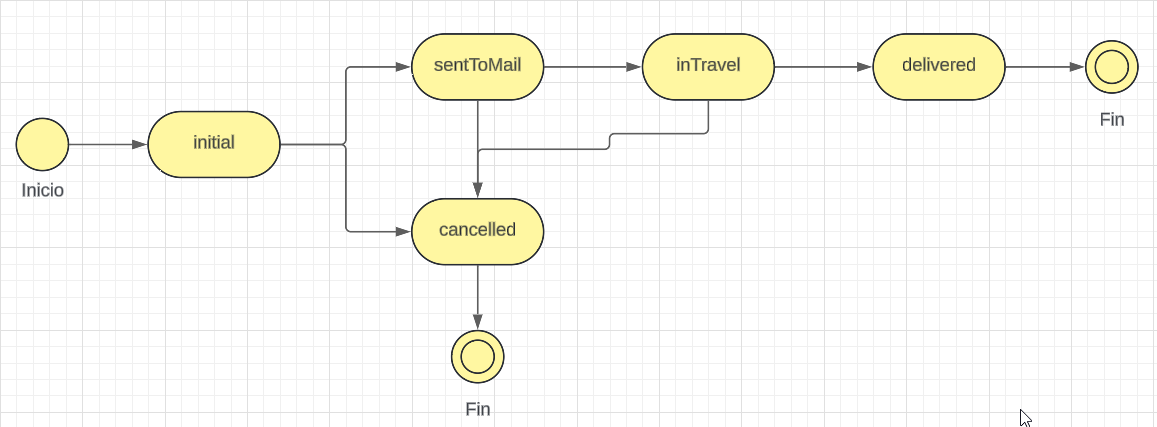
Body:

{ transition: sendToMail }

{ transition: inTravel }

{ transition: delivered }

{ transition: cancelled }



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

GET <http://localhost:8081/reports/top-sent>

Sistema:

1. ejecuta cada elemento de la lista enviada en el body, según “time-start-in-seg” y aplicar lo que indique “next-state”.

POST <http://localhost:8080/api/tarea-concurrente> (teniendo levantado los 2 proyectos, al mandar este POST, comienza la comunicación con el proyecto “mercadoEnvios” y su posterior guardado de los estados de los envios ejecutados de manera concurrente.)

body:  
{

"envios": [

{

"shippingId": 1,

"time-start-in-seg": 1,

"next-state": true

},

{

"shippingId": 2,

"time-start-in-seg": 1,

"next-state": true

},

{

"shippingId": 3,

"time-start-in-seg": 2,

"next-state": true

},

{

"shippingId": 4,

"time-start-in-seg": 2,

"next-state": true

},

{

"shippingId": 5,

"time-start-in-seg": 3,

"next-state": true

},

{

"shippingId": 1,

"time-start-in-seg": 2,

"next-state": true

},

{

"shippingId": 1,

"time-start-in-seg": 2,

"next-state": true

},

{

"shippingId": 3,

"time-start-in-seg": 5,

"next-state": true

},

{

"shippingId": 4,

"time-start-in-seg": 4,

"next-state": false

},

{

"shippingId": 5,

"time-start-in-seg": 7,

"next-state": true

},

{

"shippingId": 2,

"time-start-in-seg": 4,

"next-state": true

},

{

"shippingId": 2,

"time-start-in-seg": 6,

"next-state": false

},

{

"shippingId": 4,

"time-start-in-seg": 8,

"next-state": false

},

{

"shippingId": 1,

"time-start-in-seg": 6,

"next-state": true

},

{

"shippingId": 2,

"time-start-in-seg": 7,

"next-state": false

},

{

"shippingId": 3,

"time-start-in-seg": 8,

"next-state": true

},

{

"shippingId": 5,

"time-start-in-seg": 12,

"next-state": true

}

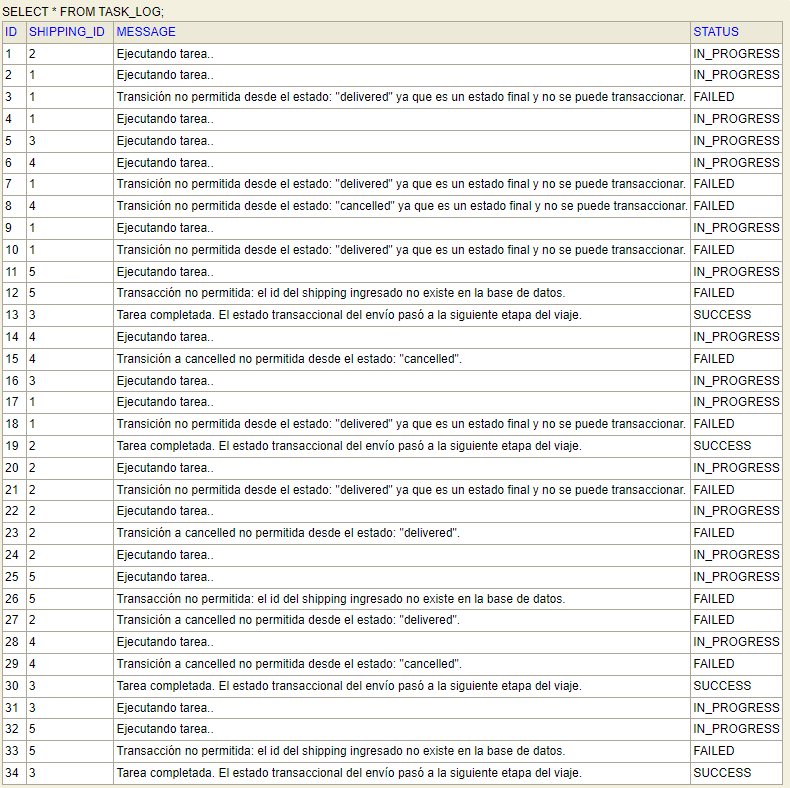
]

}

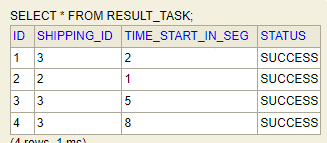
1. Informa el estado de los envíos, y que se pueda consultar, sin que interfiera con la ejecución de la tarea del POST.

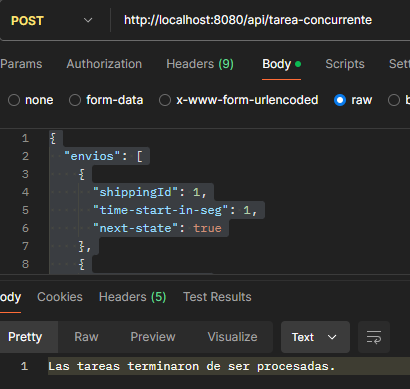
En la verificación se puede probar dejando en ejecución lo enviado a /api/tarea-concurrente, y luego solicitar el status de la misma. Ambas solicitudes conviven sin problemas.

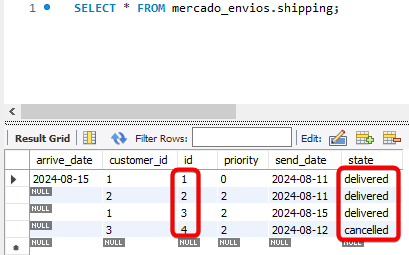
GET <http://localhost:8080/api/status>

El api/status decidí que va a traer el status (IN\_PROGRESS, FAILED, SUCCESS) a modo de log en tiempo real. He ahí su nombre de la tabla “TASK\_LOG”.  


Debo mencionar que también existe otra tabla llamada “RESULT\_TASK” en la cual decidí sólo guardar los SUCCESS.



Podemos apreciar que al final de la ejecución del POST, se verá un mensaje  


Luego de haberse procesa todas las Task o tareas. Tendrá como resultado final:  


Debajo su estado inicial (antes de mandar el POST a http://localhost:8080/api/tarea-concurrente)  
