Shipping Service Design V1.0

Designer: Alexander

Decisiones de diseño

1. **Replicación datos entre microservicios**

   Debido a que la arquitectura de el sistema Tongue esta basada en una arquitectura de microservicios orientada a eventos, y con el objetivo de reducir el acoplamiento entre microservicios. Se a decidido utilizar el patrón de replicación de datos, el cual consiste en la replicación parcial o proyección de datos de un microservicio a otro, logrando así una reducción en la cantidad de solicitudes HTTP entre ellos. Para soportar este proceso, se utilizará el bróker de mensajería RabbitMQ con el protocolo AMQP.

2. **Comunicación asíncrona**

   Se a decidió utilizar una comunicación asíncrona principalmente porque al haber considerado el enfoque síncrono, el tiempo de respuesta entre solicitudes (Especialmente la de solicitud de conductores) es muy tardado, por lo cual el riesgo de que se pierda la conexión HTTP original es más alto. Además, debido a que existen operaciones cuya realización es obligatoria, como la confirmación de que el conductor a finalizado la entrega. En el caso de alguna falla en las instancias de los microservicios, se debería persistir ese evento en algún lugar, por lo cual, en lugar de que el microservicio haga una llamada síncrona al servicio que solicitó la entrega, este publicará el evento en algún canal alojado en RabbitMQ hasta su consumo y resolución. El bajo acoplamiento es un plus.

3. **Solicitud y finalización de entregas**

   Shipping Service es un servicio que utiliza mensajería asíncrona con RabbitMQ como bróker, por lo cual todas las solicitudes de entrega de artefactos deberán ser publicadas en una cola predefinida, la cual será consultada continuamente por Shipping Service (Consumidor), el cual deberá implementar un Listener activo que cree un hilo por cada solicitud.

   El flujo de pasos que se debería realizar es el siguiente:

   1. El Listener recibe una solicitud de entrega.
   2. Se extrae la posición del punto de encuentro.
   3. Se filtra a los conductores más cercanos al punto de encuentro.
   4. Se notifica a los conductores.
   5. Un conductor acepta la solicitud.
   6. Se publica un evento en RabbitMQ, posiblemente en la cola *“accepted_ships”*.
   7. Un Listener de Shopping Service recibe el evento y lo publica en una cola con clave posiblemente igual a *“order_fulfilment_ready”.*
   8. El microservicio Order Service escucha en esa cola y empieza a preparar la orden.
   9. El conductor llega al establecimiento y notifica a Shipping Service que ha recogido la orden.
   10. Se envía la posición del consumidor al conductor.
   11. El conductor notifica a Shipping Service que el envío a terminado.
   12. Shipping Service publica un evento en una cola con clave posiblemente igual a *“finished_ships”.*

13. Shopping Service escucha el evento y realiza las actividades necesarias.

4. Autenticación

Se utilizará JWT para autenticar a los usuarios del servicio, para esto se plantea el siguiente flujo general.

1. El conductor o el consumidor deben autenticarse en los servicios correspondientes (Customer Management Service y Driver Management Service) con el mecanismo apropiado que proporcione cada servicio.
2. El microservicio correspondiente genera un JWT que se enviará al usuario autenticado.
3. El usuario llama al punto de autenticación correspondiente en Shipping Service, posiblemente */drivers/autenticate* enviando el JWT del servicio al que se autenticó inicialmente.
4. Shipping Service valida el token sea internamente o comunicándose con el servicio que generó el JWT.
5. Si el token es válido, entonces se debe obtener el *id* y el *username* del usuario comunicándose con los repositorios internos de datos replicados. Si el dato no existe, se debe comunicar de manera síncrona con el servicio que contiene el usuario original, si existe se recupera el usuario, se actualizan los repositorios de datos replicados y se crea una sesión con el *id* y el *username*, y se retorna esa información al usuario. Si existe algún error se envían las respuestas correspondientes.

5. Notificación de disponibilidad por parte del conductor

Para que un conductor sea objetivo de solicitudes de entrega, este debe subscribirse a un punto predeterminado, el cual posiblemente puede ser */shipping/couriers*. El microservicio Shipping enviará solicitudes solo a aquellos conductores que estén subscritos a ese tema.

6. Notificación a los conductores

Una vez que se haya obtenido una lista ordenada de n candidatos para la entrega, se debe notificar a los conductores siguiendo el orden en la lista, apegándose al siguiente flujo.

1. Se genera un token de acceso con tiempo de vida t.
2. Se Invalida cualquier otro token.
3. Se crea un objeto de solicitud al que se le adjuntará el token.
4. Se envía la solicitud al conductor.
5. Se inicia un contador con un tiempo $t - k$, con $0<k<t$.
6. Se verifica el estado de la solicitud. Si se a aceptado entonces se finaliza el ciclo y se notifica al usuario en los canales correspondientes o por el contrario se repite el flujo.

Para controlar que un conductor no acepte pedidos fuera de horario, se utiliza tokens de acceso.

7. Envío de la posición del conductor al usuario.

Para que un usuario reciba la posición del conductor, deben cumplirse las siguientes condiciones.

1. El conductor debe tener una sesión activa con el servicio.
2. El conductor debe publicar continuamente su ubicación en el tópico */drivers/geolocation*.
3. El usuario debe tener una subscripción activa en el punto */<user_id>/queue/shipping/geolocation*.

Para que el sistema entregue las posiciones, cada vez que el conductor envíe su ubicación se realizará lo siguiente.

1. Se obtiene el id del conductor.
2. Se obtiene el id del enlazador entre el conductor y la entrega.
3. Se obtiene el id del consumidor consultando la solicitud de entrega.
4. Se envía la posición del conductor en el tópico */<userId>/queue/shipping/geolocation*.

8. Uso de tokens temporales para estimación de costos de envío.

Debido a que el costo de una entrega depende de ciertos factores como pueden ser: el clima, la hora, el tráfico, las fecha, las posiciones de origen y destino, etc. Su costo puede llegar a ser muy distinto según el instante en que se calcule, por lo cual, para evitar que se le facture a un cliente un costo de envío distinto al que tenía inicialmente, se ha decidido utilizar tokens que pueden expirar, de modo que durante un tiempo en específico el costo que se le envió al usuario sea el mismo independientemente del dinamismo en el valor de las variables.

Para esto, se a decidido añadir el endpoint */shipping/summary*, el cual debe recibir las posiciones de origen y destino del envío, donde a partir de ellos se debería calcular el costo de entrega, la distancia, el tiempo de entrega y un token con vida temporal que deberá ser enviado dentro de la Solicitud de Envío del cliente al servidor, quien validará el token respondiendo de la manera correcta.