Second Checkpoint

BCI Training Plan

Objetivo

Evaluar los conocimientos adquiridos durante la segunda parte del entrenamiento concerniente a:

- Java 8 / Java 11
    - Uso de lambdas
    - Uso de Streams, Collections, Filters, Map, Reduce.
- Microservicios / Spring-boot / Spring Data / Spring Security.
- Gradle, Git (gitflow)
- Arquitectura EDA.

La prueba tiene un total de 100 puntos, se considera aprobada la prueba si se obtiene un puntaje igual o superior a 70 puntos.

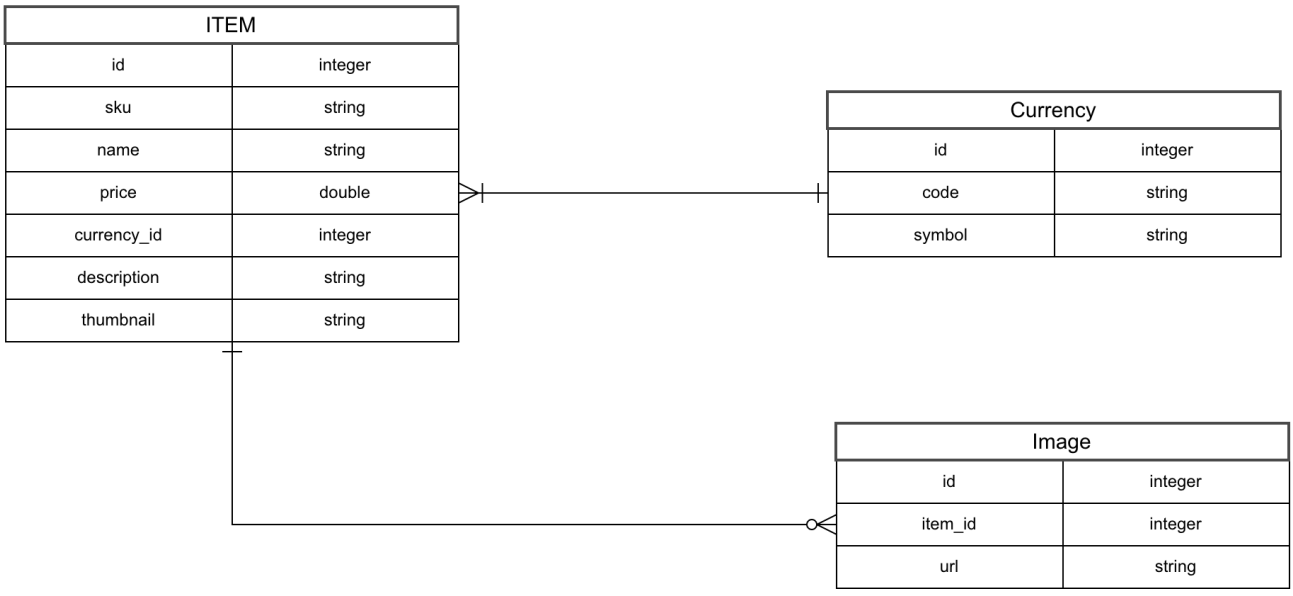
Deberá entregar la prueba técnica resuelta en un repositorio GIT, en donde cada package del código, será cada ejercicio propuesto en la presente evaluación.

Correo: jcardona@nisum.com, cc: aurso@nisum.com
Repositorio de ejemplo: https://github.com/aursog/SecondEvaluationBCI

Problema a resolver

Parte 1

En una conocida empresa de e-commerce, se poseen varios microservicios. Uno de estos microservicios - el cual es responsable de mantener los ítems (inventario) del sitio - tiene asociado una base de datos con la siguiente estructura:



La base de datos corresponde a un motor MariaDB / MySQL y el microservicios de Ítems, consulta esta base de datos y es el único responsable de mantener la información acá contenida. El servicio de ítems disponibiliza dos recursos clave.

1. /item/add
2. /item/{id}

/item/add

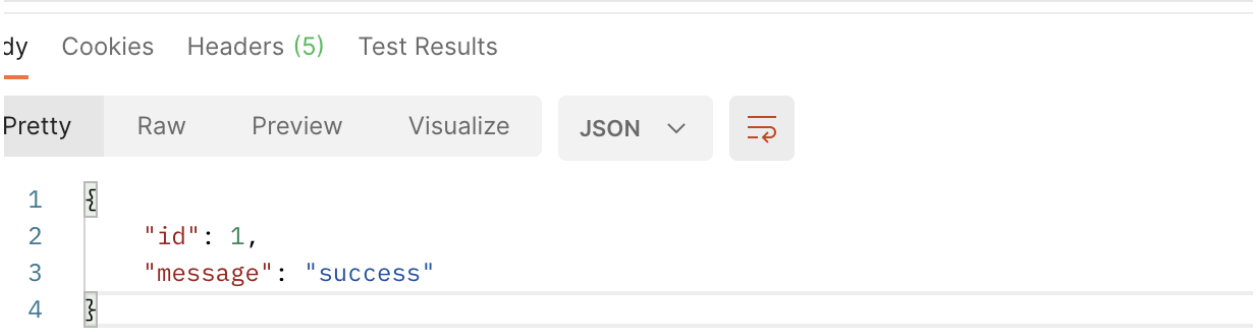
Este recurso recibe en el body un json con los datos del producto a ser agregado en la base de datos y responde un json con el id del producto creado y un mensaje de éxito.

Los datos obligatorios son:

- SKU
- Name
- Price
- Currency
- Thumbnail

En caso de que uno de los datos obligatorios no se encuentre presente, deberá retornar un error de BAD_REQUEST y en el mensaje indicar qué campo es el faltante.

```
{
    "sku": "44545654787",
    "name": "Pelota de playa",
    "price": 20000,
    "currency": "CLP",
    "description": "Pelota de playa redonda",
    "thumbnail": "https://cloud.images.google.com/thumbnail_pelotaplaya.png",
    "images": [
        "https://cloud.images.google.com/pelotaplaya1.png",
        "https://cloud.images.google.com/pelotaplaya2.png",
        "https://cloud.images.google.com/pelotaplaya3.png"
    ]
}
```



Una vez creado el ítem, deberá notificar a una cola Kafka en el topico: **items-update** el mensaje indicando que se ha creado un nuevo ítem en la base de datos. El mensaje deberá poseer la siguiente estructura:

```
{
    "id": "1",
    "message": "new"
}
```

/item/{id}

Este recurso admite dos caracteristicas:

- Si es para obtener datos, retornará un JSON con los datos del producto y todas sus relaciones.

```
{
    "id": 12,
    "sku": "44545654787",
    "name": "Pelota de playa",
    "price": 20000,
    "currency": {
        "id": 1,
        "code": "CLP",
        "symbol": "$"
    },
    "description": "Pelota de playa redonda",
    "thumbnail": "https://cloud.images.google.com/thumbnail_pelotaplaya.png",
    "images": [{
        "id": 1,
        "url": "https://cloud.images.google.com/pelotaplaya1.png"
    },{
        "id": 2,
        "url": "https://cloud.images.google.com/pelotaplaya2.png"
    },{
        "id": 3,
        "url": "https://cloud.images.google.com/pelotaplaya3.png"
    }]
}
```

  - En caso de que el ítem no exista, deberá retornar un error NOT_FOUND, con un mensaje indicando que el ítem no existe.
  - En caso de que el id ingresado sea inválido, deberá retornar un error BAD_REQUEST, con un mensaje indicando que la petición fue incorrecta.

- Si necesitamos actualizar un registro de un ítem, enviaremos en el body del request el JSON con los datos a ser actualizados.

```
{
    "description": "Pelota de playa cuadrada",
    "thumbnail": "https://cloud.images.google.com/thumbnail_pelotaplayacuadrada.png"
}
```

  - En caso de un error en los datos del body, deberá retornar un error tipo BAD_REQUEST y en el mensaje indicar el error.
  - En caso de éxito, retornará un OK y en el mensaje un success.
  - Una vez modificado el ítem, deberá notificar a una cola Kafka en el tópico: **items-update**, el mensaje indicará el id del ítem modificado y en el mensaje indicar “update”, como se muestra en la imagen.

/authenticate

El consumo de los diferentes endpoints deben validar el respectivo token(JWT) de autenticación, para esto debe existir un método de autenticación de usuario y contraseña para hacer el login y devolver el respectivo token para consumir los respectivos servicios .

Lo que se espera de esta parte

Se espera que usted determine:

- Los verbos HTTP adecuados para cada uno de los recursos disponibles de la aplicación.
- La construcción del microservicio utilizando spring-boot 2.x.x y Java 11.
- Implementar autenticación JWT.
- La construcción de un docker-compose que permita levantar la base de datos MariaDB / MySQL y la cola Kafka como servicios.

- Que su aplicación posea al menos un 80% de cobertura de test, utilizando para ello unit test en Spock (se sugiere Jacoco para medir coverage, otras herramientas son igualmente aceptadas).
- Un Dockerfile con los componentes necesarios para poder levantar su aplicación.

Parte 2

Como parte de la misma arquitectura de la empresa de e-commerce, posee otro microservicio que se alimenta de la API de ítems para enviar esa información a un Frontend. Este microservicio corresponde a un BFF (Backend-for-frontend) en donde se consolida cierta información proveniente de distintas fuentes.

Para no consultar de forma reiterada al microservicio de ítems, tiene una base de datos NoSQL (Redis) en donde almacena los ítems que ya han sido consultados.

Este microservicio se encuentra suscrito al tópico: **items-update**, en donde obtiene la información de cualquier tipo de modificación / creación que suceda en el microservicio de ítems, y así ir a consultar la información específica del ítem modificado / creado; y así generar las modificaciones necesarias en su base de datos NoSQL.

La arquitectura general es la siguiente:

Este microservicio posee un único recurso disponibilizado, el cual corresponde a:
/vip/item/{id}, el cual en caso de no tener registrado el dato en su base de datos NoSQL deberá ir a consultar al microservicio de ítems y almacenar dicha información en su base de datos y luego desplegar un JSON con la siguiente información:

```json
{
    "id": 12,
    "sku": "44545654787",
    "name": "Pelota de playa",
    "price": 20000,
    "currency": "CLP",
    "symbol": "$",
    "description": "Pelota de playa cuadrada",
    "thumbnail": "https://cloud.images.google.com/thumbnail_pelotaplaya.png",
    "images": [{
        "id": 1,
        "url": "https://cloud.images.google.com/pelotaplaya1.png"
    },{
        "id": 2,
        "url": "https://cloud.images.google.com/pelotaplaya2.png"
    },{
        "id": 3,
        "url": "https://cloud.images.google.com/pelotaplaya3.png"
    }]
}
```


La cual se corresponde con la data contenida en su base de datos NoSQL.

Lo que se espera de esta parte

- Un docker-compose que permita levantar la base de datos Redis (NoSQL), otros motores son igual de aceptados.
- Construcción del microservicio que siga las reglas de negocio planteadas.
- A lo menos un 80% de cobertura de test, a través de unit test desarrollados en Spock (se sugiere Jacoco para medir coverage, sin embargo otras herramientas son igual de aceptadas).
- Ejecutar revisión de código con Sonarqube.
- Pruebas de aceptación con Cucumber.
- Pruebas de carga con Jmeter.
- Implementar **Swagger** para la firma de los métodos.
- Un Dockerfile para poder correr su aplicación.

Detalles de la evaluación.

- La parte 1 consta de un puntaje de 60 puntos
- La parte 2 consta de un puntaje de 40 puntos.
- Tendrán 1 semana para desarrollar la solución (si lo pueden tener antes mejor).
- Deberán subir dos repositorios distintos (para cada parte) en GitLab / GitHub / BitBucket, incluyendo los archivos docker-compose.

- Se debe implementar Git Flow en cada uno de los repositorios y desarrollar en el respectivo feature para luego hacer el respectivo pull request.
- Deberán subir un script de creación de las bases de datos y poblado inicial con algunos datos para poder probar(Ideal uso de Liquibase).
- Se admiten dudas sobre qué solución es mejor, si así lo determinan necesario, se podrán armar reuniones ad-hoc con las personas que ustedes estimen conveniente dentro del proyecto.