

Solución Challenge: Cupón de compra

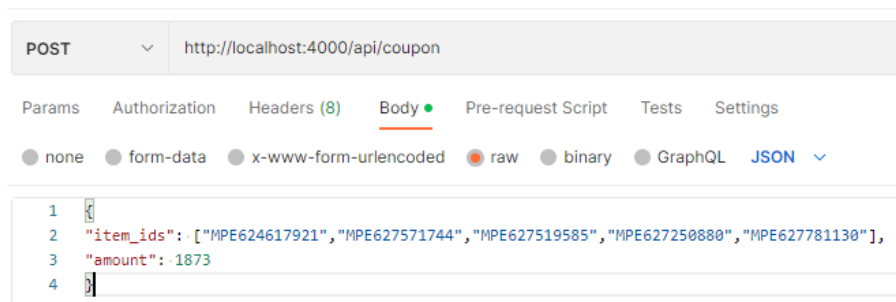
Configuración:

Lenguaje	JAVA 17
IDE	IntelliJ
Proyecto	Gradle
Framework	Spring Boot 3.0.4

Reto 1.

Crear una API REST, con el servicio “/coupon/” en donde se pueda enviar la lista de item_ids y el monto del cupón. El endpoint debe devolver los ítems que tendría que comprar el usuario.

Se crea una app (REST) con el servicio/coupon, el cual es controlado por “CuponController”, el cual posee un método principal llamado “listadoPriceProducts”, el cual es el encargado de recibir el body de la petición que se realiza de manera externa, el cual debe declararse de la siguiente manera.



```
{  
  "item_ids": ["MPE624617921", "MPE627571744", "MPE627519585", "MPE627250880", "MPE627781130"],  
  "amount": 1873  
}
```

En la línea marcada con "item_ids", debe ir una lista de ítems tal cual se encuentran descritos y disponibles desde la api de MercadoLibre.

"amount", debe ir un dato de tipo numérico, este dato será el valor del cupón que el cliente tiene, con el cual se podrá realizar la comprobación, de cuáles ítems serán los elegidos para que se maximice la utilización de este

La respuesta de esta petición será de la siguiente manera:

```
1 {
2   "item_ids":["MPE624617921","MPE627571744","MPE627519585","MPE627250880","MPE627781130"],
3   "amount":1869
4 }
```

Body Cookies Headers (5) Test Results

Pretty

Raw

Preview

Visualize

Text



```
1 {"total":1800,"item_ids":["MPE624617921","MPE627571744","MPE627250880"]}
```

```
{
  "total":1800,
  "item_ids":["MPE624617921","MPE627571744","MPE627250880"]
}
```

En esta respuesta se puede identificar dos elementos **"total"**, el cual es el encargado de mostrar cuál sería el monto máximo a usar del cupón si se intenta utilizar con los ítems enviados.

"item_ids", el cual sería el encargado de mostrar cuales serian los ítems que por su precio en conjunto se acercará más al valor del cupón enviado en la petición.

Dentro de la app y en el método antes mencionado, se realiza una suma inversa (sumInversaUp), la cual es la encargada de identificar mediante un llamado recursivo (sumInversaRecursive) de este.

```
public class SumInversa {
    static void sumInversaRecursive(ArrayList<Double> prices, double coupon, ArrayList<Double> partial, Map<Double,ArrayList<Double>> arrItemsSelected) {
        double s = 0.0;
        for (double x: partial) {
            s = s + x;
        }
        if (s > 0.0 && s <= coupon){
            arrItemsSelected.put(s,partial);
        }
        if (s > coupon){
            return;
        }

        for(int i=0;i<prices.size();i++) {
            ArrayList<Double> remaining = new ArrayList<>();
            double n = prices.get(i);
            for (int j=i+1; j<prices.size();j++) {
                remaining.add(prices.get(j));
            }
            ArrayList<Double> arrPartialRec = new ArrayList<>(partial);
            arrPartialRec.add(n);
            sumInversaRecursive(remaining,coupon,arrPartialRec,arrItemsSelected);
        }
    }

    public static List<Map.Entry<Double,ArrayList<Double>>> sumInversaUp(List<Double> prices, double target) {

        Map<Double,ArrayList<Double>> mapItemsSelected = new HashMap<>();
        sumInversaRecursive((ArrayList<Double>) prices,target,new ArrayList<>(),mapItemsSelected);

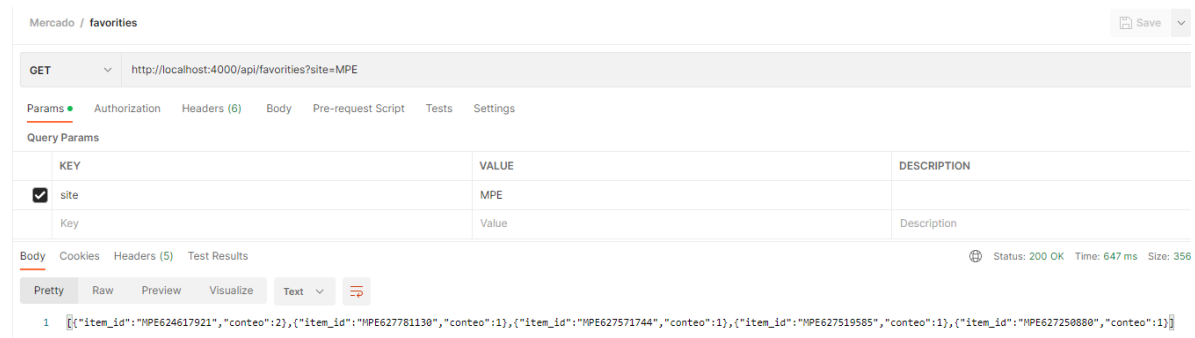
        List<Map.Entry<Double,ArrayList<Double>>> itemsCouponSorted;

        itemsCouponSorted = new ArrayList<>(mapItemsSelected.entrySet());
        itemsCouponSorted.sort(Map.Entry.comparingByKey(Comparator.reverseOrder()));
        return itemsCouponSorted;
    }
}
```

Reto 2.

Definir e implementar un endpoint el cual devuelva el top 5 de ítems más “favoriteados” a partir del endpoint anterior.

Este se enfrentó, utilizando una tabla de una BD, en la cual se van ingresando u actualizando según sea la situación puntual, de los ítems enviados en la petición usada en el **Reto 1**, para la utilización de este servicio utilizo el endpoint “/favorites”, y el uso de un parámetro que es el encargado de identificar el top 5 pero por “site”.



Mercado / favorites

GET http://localhost:4000/api/favorites?site=MPE

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
site	MPE	
Key	Value	Description

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 647 ms Size: 356

Pretty Raw Preview Visualize Text

```
[{"item_id": "MPE624617921", "conteo": 2}, {"item_id": "MPE627781130", "conteo": 1}, {"item_id": "MPE627571744", "conteo": 1}, {"item_id": "MPE627519585", "conteo": 1}, {"item_id": "MPE627250880", "conteo": 1}]
```

En este caso nos conectamos a una BD SqlServer, y por cada petición que se realice del Reto 1, se realiza un Query tipo MERGE, que es el encargado de ingresar o de actualizar los ítems favoritos por site.

```
MERGE INTO favoritos
    USING (SELECT ? AS item_id, ? AS site_id, 1 AS conteo) AS reg
    ON favoritos.item_id = reg.item_id
WHEN MATCHED THEN
    UPDATE SET site_id = reg.site_id, conteo = favoritos.conteo + reg.conteo
WHEN NOT MATCHED THEN
    INSERT(item_id , site_id ,conteo) VALUES (reg.item_id , reg.site_id, reg.conteo);
```

El cual recibe dos parámetros que serían el item_id y el site_id.

Reto 3.

Hostear las APIs en un cloud computing libre.

Este se enfrente de la siguiente manera:

Se utilizan las herramientas de Google Cloud.

Se crea un espacio de trabajo, el cual llámanos **favoritosmeli**.

Información del proyecto

Nombre del proyecto

favoritosmeli

Número del proyecto

768380969063

ID de proyecto

favoritosmeli

AGREGA PERSONAS A ESTE PROYECTO

→ Ir a la configuración del proyecto

Posteriormente se crea una instancia de SQL, para crear una BD tipo SqlServer

Filtro Ingresar el nombre o el valor de la propiedad					
<input type="checkbox"/>	ID de instancia	Tipo	Dirección IP pública	Dirección IP privada	Nombre de la conexión con la instancia
<input checked="" type="checkbox"/>	favoritosmeli	SQL Server 2019 Standard	34.171.210.241		favoritosmeli:us-central1:favoritosmeli

Todas las instancias > favoritosmeli

✓ favoritosmeli

SQL Server 2019 Standard

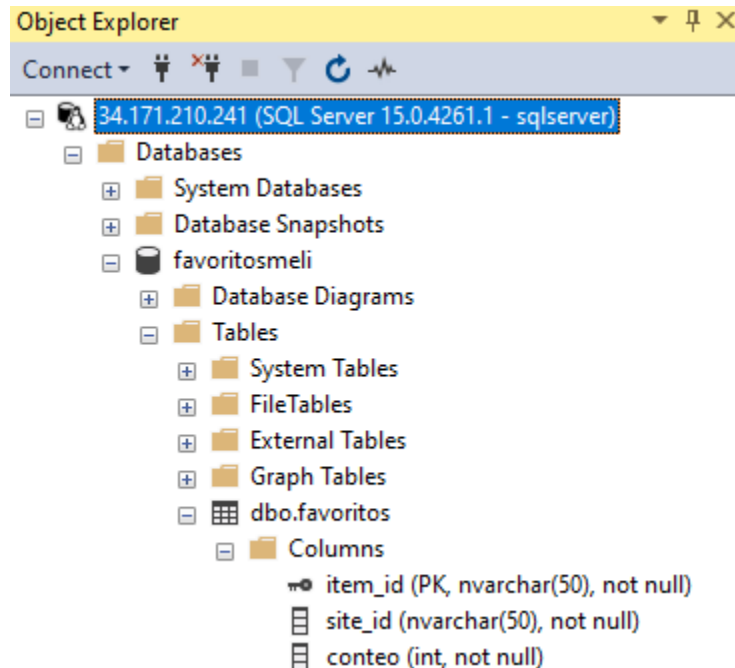
Intercalación SQL_Latin1_General_CP1_CI_AS
predeterminada

+ CREAR BASE DE DATOS

Nombre	Intercalación	
favoritosmeli	SQL_Latin1_General_CP1_CI_AS	⋮
master	SQL_Latin1_General_CP1_CI_AS	⋮
model	SQL_Latin1_General_CP1_CI_AS	⋮
msdb	SQL_Latin1_General_CP1_CI_AS	⋮
tempdb	SQL_Latin1_General_CP1_CI_AS	⋮

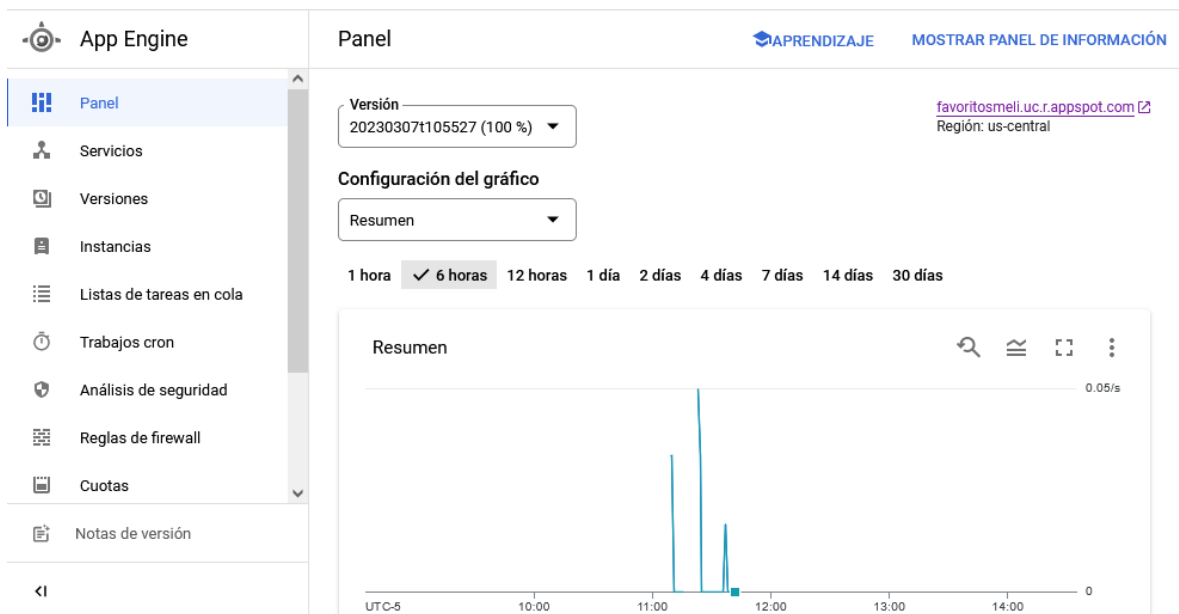
En la cual se crea una tabla en la que se guardará la información de todas las solicitudes que se realicen al servicio cuopon del Reto 1, esta tabla (favoritos) está

compuesta por 3 campos (item_Id(PK), site_id, conteo) que nos ayudarán a llevar un conteo de favoritos que se envían en el BODY de la solicitud del Reto 1.



Esta tabla se llena con un Query tipo MERGE, como se explica en el Reto 2.

En el espacio de trabajo creado, creamos una instancia en el App Engine.

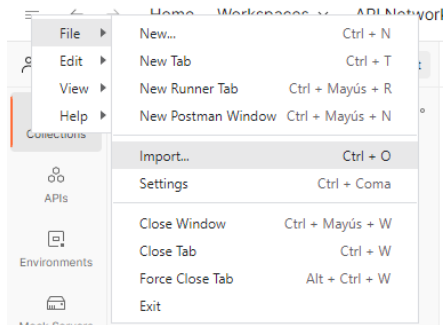


Para de esta manera poder subir nuestra App.

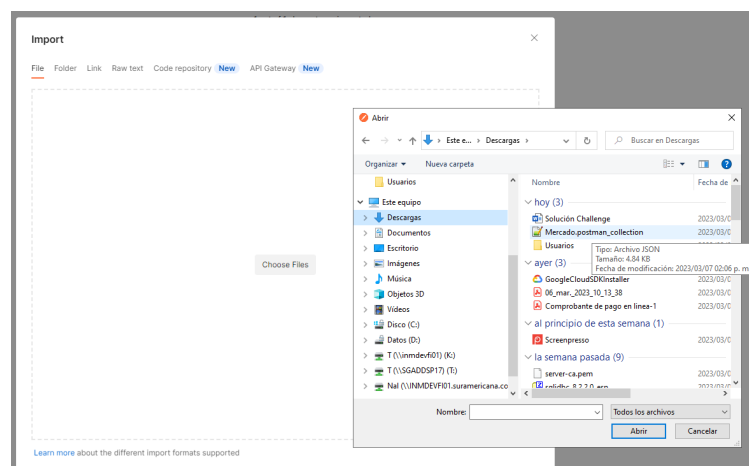
Luego de subir la aplicación, el Google Cloud nos entrega una dirección donde queda publicada esta (favoritosmeli.uc.r.appspot.com) con la cual podremos realizar nuestras peticiones directamente en la nube.

Importar Recurso en POSTMAN

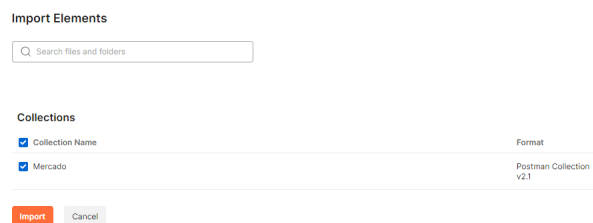
1. Bajar de GitHub la rama main del repositorio [Coupon](#).
2. Allí encontraremos el archivo Mercado.postman_collection.json.
3. Desde la herramienta POSTMAN.
 - a. realizar un import



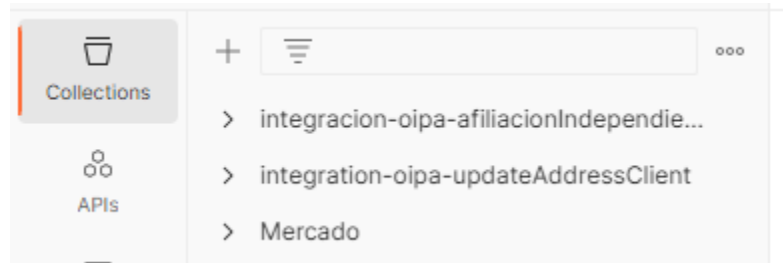
- b. Luego el sistema nos solicitará el archivo a importar que será el que anteriormente bajamos de GitHub.



- c. Posteriormente al ser seleccionado el archivo JSON, la herramienta nos mostrará un resumen.



- d. Por último damos Click en el botón Import, y podremos observar que dentro de las colecciones, se ha creado una con el nombre de “Mercado”.



4. Ya aquí podremos hacer el llamado de cada una de las URL disponibles para el Proyecto.

