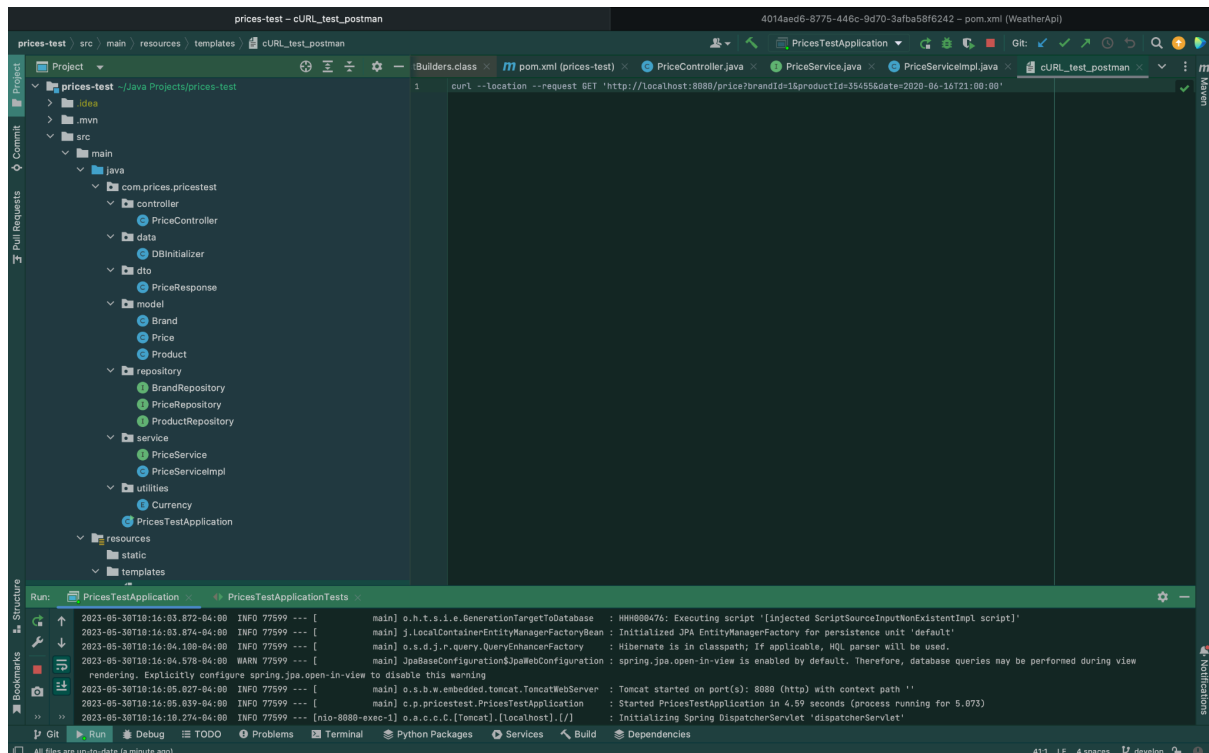


Lista de Precios Coding Test

Saludos, este documento tiene como finalidad explicar como se desarrollo la solución para el coding Test de Lista de Precios. Para esto fue creado un proyecto utilizando Spring boot, Maven, H2 (base de datos en memoria).



Estructura de Clases

Entity o Model

En este paquete se guardan las entidades involucradas en la solución.

- Brand: Entidad de la marca, obedece a la relación entre la tabla de brands y prices.
- Product: Entidad del producto, obedece a la relación entre la tabla products y prices.
- Price: Entidad que obedece a la tabla prices.

Repository

En este paquete se guardan los repositorios de las entidades creadas anteriormente, donde usando Spring Repository podemos utilizar las funciones básicas de persistencia para los repositorios de Brand, Product y Price. Adicionalmente para este último (Price) fue

necesario agregar una nueva funcionalidad con el query específico para filtrar la data de acuerdo como se requiere en la necesidad de negocio.

- BrandRepository
- ProductRepository
- PriceRepository

Dto

Para esta solución fue creado un DTO con los atributos que se requieren retornar al cliente que hace la consulta. El DTO PriceResponse contiene los siguientes atributos:

- productId
- brandId
- priceList
- startDate
- endDate
- finalPrice

Este objeto se crea para estar alineado con lo que el cliente espera recibir sin tener que exponer nuestras entidades.

Controller

Se creó un controlador donde se especifica un único endpoint para la solución, el cual tiene como nombre getPrice y recibe como parámetros (query param) el id de la marca (brandId) el id del producto (productId) y la fecha (date). Al final se retorna el response determinado y en caso de ser satisfactorio (ok) se retorna un objeto de tipo PriceReponse, el DTO mencionado en el paso anterior.

Service

Para esta solución y siguiendo el conjunto de buenas practicas se implemento un servicio donde se maneja lógica de llamado a la capa de persistencia, se valida el resultado y se retorna una respuesta mapeada al objeto de negocio dependiendo de la información obtenida de la base de datos.

Utilities

Fue creada la enumeración Currency solo con la finalidad de hacer el escenario más real, no agrega valor de negocio a la solución.

Data

En esta categoria fue creada una clase con un initializer para la base datos en memoria, con finalidad que una vez desplegada la solución se puedan ejecutar desde postman los request propuestos en el documento del Coding Test.

localhost:8080/h2-console/login.do?sessionId=9d3f5cdf5711c70dc621c4e2e692d254

tipos de datos Design Patterns English Class Cvent Karate | Test Auto... English Site Senior Java Devel... dolar americano a...

Auto commit Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:capitoliodb

- BRAND
- PRICE
- PRODUCT
- INFORMATION_SCHEMA
- Users
- H2 2.1.214 (2022-06-13)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PRODUCT;

SELECT * FROM PRICE;

SELECT * FROM BRAND;

SELECT * FROM PRODUCT;

PRODUCT_ID	NAME
35455	T-Shirt
35456	Sweater
35457	Blue Jean
35458	Jacket

(4 rows, 0 ms)

SELECT * FROM PRICE;

BRAND_ID	ID	PRICE	PRICE_LIST	PRIORITY	PRODUCT_ID	END_DATE	START_DATE	CURR
1	1	35.5	1	0	35455	2020-12-31 23:59:59	2020-06-14 00:00:00	EUR
1	2	25.45	2	1	35455	2020-06-14 18:30:00	2020-06-14 15:00:00	EUR
1	3	30.5	3	1	35455	2020-06-15 11:00:00	2020-06-15 00:00:00	EUR
1	4	38.95	4	1	35455	2020-12-31 23:59:59	2020-06-15 16:00:00	EUR

(4 rows, 1 ms)

SELECT * FROM BRAND;

BRAND_ID	NAME
1	ZARA
2	NIKE
3	GEF

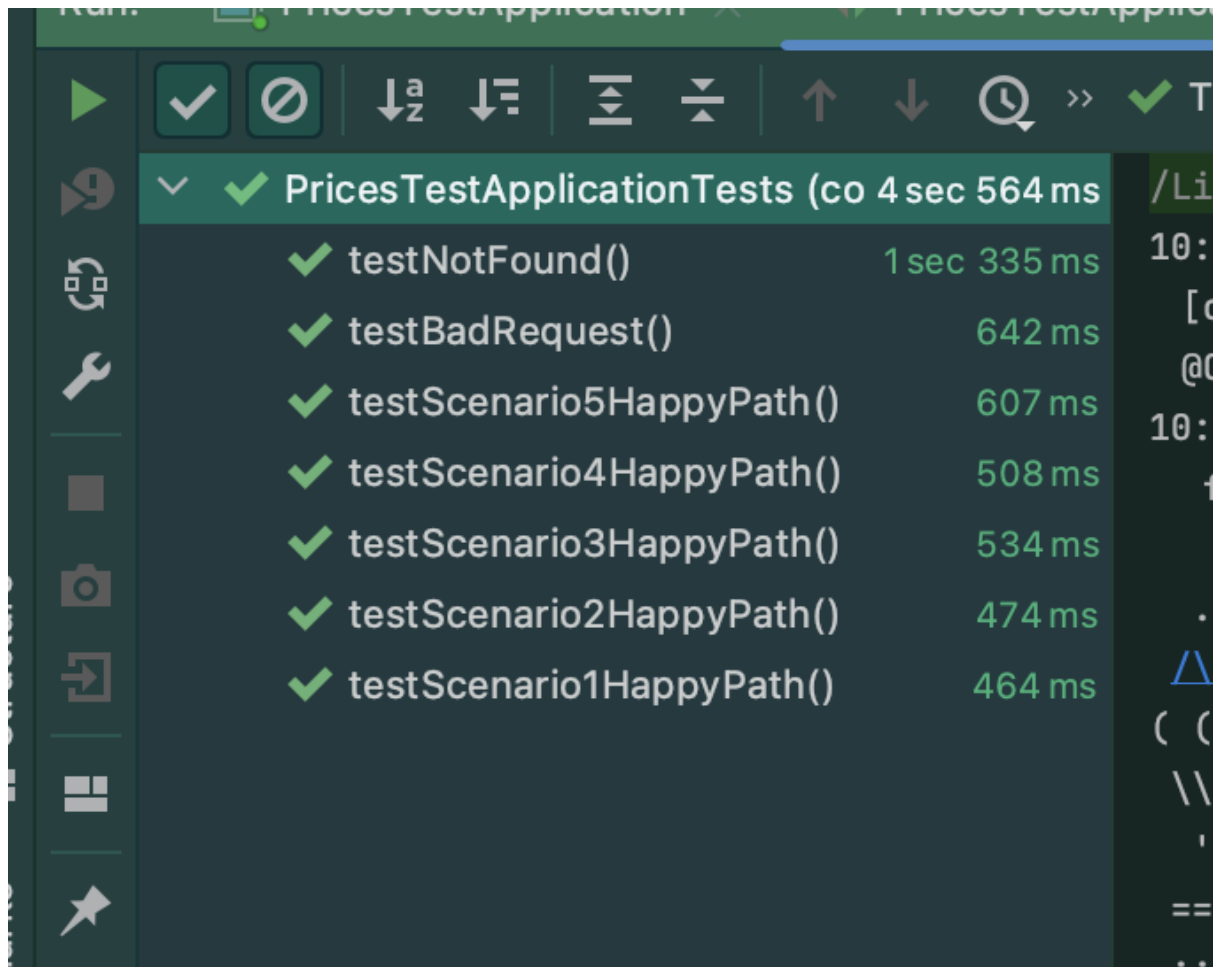
(3 rows, 0 ms)

Resources

En la carpeta templates fue creado un archivo plan con un cURL de ejemplo para realizar las pruebas funcionales desde postman.

Pruebas de Integracion

Fueron creadas las pruebas de integración en la clase PricesTestAplicationTests, las cuales cubren los escenarios planteados en el problema y se centran solo en los criterios de aceptación, no en ampliar cobertura.



Pruebas desde Postman

Para las pruebas en Postman podemos usar el siguiente request y validar los diferentes resultados variando los valores de los parametros

```
curl --location --request GET  
'http://localhost:8080/price?brandId=1&productId=35455&date=2020-06-16T21:00:00'
```

Home Workspaces ▾ API Network ▾ Explore [Invite](#) [Team](#) ▾

An update has been downloaded for Postman. Restart now to install the update. [Restart](#)

POST http POST Save PUT Update GET Get Env GET Get All DEL Delete POST Sen POST http GET http POST http GET http POST http GET http No Environment

[http://localhost:8080/price?brandId=1&productId=35455&date=2020-06-16T21:00:00](#)

GET [http://localhost:8080/price?brandId=1&productId=35455&date=2020-06-16T21:00:00](#) [Send](#)

[Params](#) [Authorization](#) [Headers \(7\)](#) [Body](#) [Pre-request Script](#) [Tests](#) [Settings](#) [Cookies](#)

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	brandId	1			
	productId	35455			
	date	2020-06-16T21:00:00			
	Key	Value	Description		

[Body](#) [Cookies](#) [Headers \(5\)](#) [Test Results](#) Status: 200 OK Time: 231 ms Size: 294 B [Save Response](#)

[Pretty](#) [Raw](#) [Preview](#) [Visualize](#) [JSON](#)

```
1 {
2   "productId": 35455,
3   "brandId": 1,
4   "pricelist": 4,
5   "startDate": "2020-06-15T16:00:00",
6   "endDate": "2020-12-31T23:59:59",
7   "finalPrice": 38.95
8 }
```

Online Find and Replace Console Cookies Capture requests Runner Trash

Gracias por la atención prestada y no dude en consultarme cuaquier inquietud.