# Geeks México

BLOG DE PROGRAMACIÓN EN ESPAÑOL SOBRE JAVA, FRAMEWORKS, BASES DE DATOS, CÓMPUTO EN LA NUBE, ETC. EN ESPAÑOL Y EN INGLÉS.





# Aprende a hacer tests funcionales

# utilizando rest-assured

HACE 1 MES 1 COMENTARIO

Rate This

Un problema común en la mayor parte de las aplicaciones es escribir pruebas funcionales, en este post explicaremos como hacer esas pruebas sobre servicios REST utilizando la herramienta **rest-assured.io**.

## Paso 1 Configuración

El primer paso será configurar nuestro proyecto, para esto veamos nuestro archivo pom.xml (https://github.com/raidentrance/rest-assured-example/blob/master/pom.xml).

Como se puede ver la dependencia de **rest-assured** tiene el scope de test, esto para que solo sea utilizada para realizar las pruebas del proyecto y no sea incluida al construir el proyecto.

## Paso 2 Analizando el endpoint a probar

Para que puedas ejecutar los tests en tu máquina los realizaremos sobre una api pública que todos podemos consultar, en este caso utilizaremos el api rest de bitso para realizar nuestras pruebas.

El endpoint que vamos a probar será el siguiente:

endpoint : https://api.bitso.com/v3/available_books/ (https://api.bitso.com/v3/available_books/)

Respuesta ejemplo :


```
{
    "success": true,
    "payload": [
        {
            "book": "btc_mxn",
            "minimum_price": "500.00",
            "maximum_price": "16000000.00"
            "minimum_amount": "0.00015",
            "maximum_amount": "500.0000000
            "minimum_value": "5",
            "maximum_value": "10000000.00"
        },
        {
            "book": "eth_mxn",
            "minimum_price": "10.00",
            "maximum_price": "200000.00",
            "minimum_amount": "0.001",
            "maximum_amount": "100000.0000
            "minimum_value": "5.00",
            "maximum_value": "10000000.00"
        },
        {
            "book": "xrp_btc",
            "minimum_price": "0.00000100",
            "maximum_price": "5000.0000000
            "minimum_amount": "0.00000100"
            "maximum_amount": "100000.0000
            "minimum_value": "0.00000100",
            "maximum_value": "100000.00000
        },
        {
            "book": "xrp_mxn",
            "minimum_price": "0.0000100",
            "maximum_price": "5000.0000000
            "minimum_amount": "0.5",
            "maximum_amount": "500000.0000
            "minimum_value": "5",
            "maximum_value": "10000000.000
        },
        {
            "book": "eth_btc",
            "minimum_price": "0.00000100",
            "maximum_price": "5000.0000000
            "minimum_amount": "0.00000100"
            "maximum_amount": "1000.000000
            "minimum_value": "0.00000100",
            "maximum_value": "1000.0000000
        },
        {
            "book": "bch_btc",
```

```
            "minimum_price": "0.0001",
            "maximum_price": "8000.00",
            "minimum_amount": "0.0001",
            "maximum_amount": "8000.00",
            "minimum_value": "0.0001",
            "maximum_value": "8000.00"
        },
        {
            "book": "ltc_btc",
            "minimum_price": "0.01",
            "maximum_price": "80000.00",
            "minimum_amount": "0.00015",
            "maximum_amount": "500.0000000
            "minimum_value": "0.001",
            "maximum_value": "100.00"
        },
        {
            "book": "ltc_mxn",
            "minimum_price": "10",
            "maximum_price": "50000.00",
            "minimum_amount": "0.001",
            "maximum_amount": "100000.0000
            "minimum_value": "5",
            "maximum_value": "10000000.00"
        }
    ]
}
```

La respuesta anterior representa los precios de las criptomonedas cuando este post fue escrito.

## Paso 3 Iniciar nuestras pruebas

Para crear nuestras pruebas utilizaremos el framework JUnit junto con rest-assured, veamos el primer test:

Prueba 1: valida que la bandera **success** en la respuesta es true:

En esta prueba se validará que la bandera success que se encuentra en la respuesta tiene el valor de verdadero, en caso contrario el test fallará.

```
import static io.restassured.RestAssured.g
import static org.hamcrest.Matchers.equalT

import org.junit.Test;

/**
 * @author raidentrance
 *
```

```
 */
public class AvailableBooksTest {
    @Test
    public void testGetAvailableBooks() {
        get("https://api.bitso.com/v3/avai
    }
}
```

En este ejemplo podemos ver los siguientes puntos:

- static imports: Como en la mayor parte de framework de pruebas rest-assured define algunos static imports que son necesarios para que nuestra aplicación funcione correctamente, en este caso estamos utilizando los siguientes:
  - **import static io.restassured.RestAssured.get**;
  - **import static org.hamcrest.Matchers.equalTo**;
- Nuestra prueba funcional: Como se puede ver, es posible hacer nuestra primera prueba funcional con una sola línea de código, en esta definimos los siguientes puntos:
  - Url: La url del endpoint que vamos a probar.
  - Método http: al utilizar el método get estamos definiendo que la petición http a realizarse será de tipo get.
  - Validación: Como se puede ver en el método body definimos que el valor de la propiedad success debe ser igual a true, en caso contrario el test fallará

Prueba 1: valida que al obtener el precio de una crypto moneda devuelve un status http 200

```
@Test
public void getTickerByBook() {
    given().param("book", "xrp_mxn").get("h
}
```

En este ejemplo se puede ver que le pasamos como parámetro el nombre de la crypto moneda al endpoint ticker y que validamos que el estatus http es 200.

Prueba 3 Valida que al obtener el precio de una crypto moneda el estatus http es 200 y que en el cuerpo de la respuesta el precio es diferente de null.

```
1  @Test
2  public void validateGetTickerPrice() {
3      given().param("book", "xrp_mxn").get("h
4          .body("payload.ask", notNullValue()
5  }
```

En el código anterior se puede ver que es posible utilizar más de una validación durante nuestra prueba.

## Conclusión

En este post podemos ver los beneficios de utilizar rest assured para las pruebas funcionales de nuestras aplicaciones, esto debido a que no tenemos que crear clientes http, des serializar las respuestas entre muchas otras tareas.

Puedes encontrar el código completo en la siguiente url https://github.com/raidentrance/rest-assured-example (https://github.com/raidentrance/rest-assured-example).

Si te gusta el contenido y quieres enterarte cuando realicemos un post nuevo síguenos en nuestras redes sociales https://twitter.com/geeks_mx (https://twitter.com/geeks_mx) y https://www.facebook.com/geeksJavaMexico/ (https://www.facebook.com/geeksJavaMexico/).

*Autor: Alejandro Agapito Bautista*

*Twitter: @raidentrance*

*Contacto:raidentrance@gmail.com*

# 1 comentario »

quede mas confundido! no se que ide usar o que libreria bajar

★(https://geeks-mexico.com/2018/03/16/aprende-a-hacer-tests-funcionales-utilizando-rest-assured/?like_comment=193&_wpnonce=3c07478d23)
Me gusta