Testing II

¿Cómo funciona Rest-Assured?

Entendamos cómo funciona Rest Assured:

- Crear una solicitud HTTP con todos los detalles
- Enviar la solicitud a través de la red
- Validar la respuesta recibida

HTTP Request

Una HTTP request consiste en lo siguiente:

1. **URL -** La URL de solicitud es la dirección única utilizada para realizar una solicitud. La URL está compuesta por la URL base, el recurso (resource), la consulta (query) o los parámetros de la ruta (path parameters).

2. **HTTP Method/Verb** - Las cuatro operaciones básicas de creación, lectura, actualización y eliminación (CRUD) se realizan mediante los métodos POST, GET, PUT y DELETE en la interfaz REST.

3. **Headers** - Representa los metadatos asociados a las peticiones o respuestas HTTP. Es la información adicional que se pasa entre el cliente y el servidor junto con la solicitud o la respuesta. Las cabeceras se utilizan para diversos fines, como la autenticación, el almacenamiento en caché, la información del cuerpo del mensaje, la gestión de las cookies, etc. Las cabeceras serán pares clave-valor o pueden ser una clave con varios valores.

4. **Payload/Body** - Contiene la información que el usuario quiere enviar al servidor. La carga útil se utiliza únicamente con las solicitudes que modifican el recurso existente o crean otros nuevos.

5. **HTTP Response** - La respuesta HTTP consiste en el código de estado, las cabeceras (headers) y el cuerpo de la respuesta (response body).

6. **Códigos de status (status code)** - Los códigos de estado HTTP nos ayudan a comprender rápidamente el estado de la respuesta.
   100-199: Informativo. Se ha recibido la solicitud y el proceso está en marcha.
   200-299: Exitoso. La solicitud fue exitosa.
   300-399: Redirección. La petición se está redirigiendo a otra URL.
   400-499: Error del cliente. Se ha producido un error en el lado del cliente.
   500-599: Error del servidor. Se ha producido un error del lado del servidor.

¿Cómo probar la API Rest?

Cuando se prueba un recurso REST, suele haber unas cuantas responsabilidades ortogonales en las que deben centrarse las pruebas:

- El status code HTTP
- Headers HTTP en la respuesta
- Payload (JSON, XML)

Cada prueba debe centrarse en una sola responsabilidad e incluir una sola afirmación.

Centrarse en una separación clara siempre tiene ventajas, pero cuando se hace este tipo de pruebas de caja negra es aún más importante, ya que la tendencia general es escribir escenarios de prueba complejos desde el principio.

Ejemplo de prueba para comprobar el código de estado:

```
@Test
public void givenUserDoesNotExists_whenUserInfoIsRetrieved_then404IsReceived()
  throws ClientProtocolException, IOException {

    // Given
    String name = RandomStringUtils.randomAlphabetic( 8 );
    HttpUriRequest request = new HttpGet( "https://api.github.com/users/" + name );

    // When
    HttpResponse httpResponse = HttpClientBuilder.create().build().execute( request );

    // Then
    assertThat(
      httpResponse.getStatusLine().getStatusCode(),
      equalTo(HttpStatus.SC_NOT_FOUND));
}
```

Esta es una prueba bastante simple. Comprueba que una ruta básica está funcionando, sin añadir demasiada complejidad al conjunto de pruebas. Si por la razón que sea falla, no es necesario examinar ninguna otra prueba para esa URL hasta que se solucione.

Esto es sólo una parte de lo que debería ser un conjunto de pruebas de integración completo. Las pruebas se centran en garantizar la corrección básica de la API REST, sin entrar en escenarios más complejos, por ejemplo, no se contemplan los siguientes escenarios: descubrimiento de la API, consumo de diferentes representaciones para un mismo recurso, etc.

La implementación de este ejemplo y el fragmento de código se pueden encontrar en Github - este es un proyecto basado en Maven, por lo que debería ser fácil de importar y ejecutar tal cual.