Práctica 1. Testing con Spring y Node.js

Convocatoria ordinaria

Parte 1: Testing en Java con Spring (7 pts)

Se desea implementar las pruebas necesarias para comprobar la correcta funcionalidad de una aplicación que gestiona una librería online. Se proporciona el código de dicha aplicación (publicado en el Aula Virtual: **Java Enunciado**).

Se quieren realizar las siguientes pruebas de la aplicación:

Tests unitarios con MockMVC (3 pts)

- Comprobar que se pueden recuperar todos los libros (como usuario sin logear)
- Añadir un nuevo libro (como usuario logueado)
- Borrar un libro (como administrador)

Tests E2E con RESTAssured (3 pts)

- Comprobar que se pueden recuperar todos los libros (como usuario sin logear)
- Añadir un nuevo libro (como usuario logeado) y comprobar que se ha creado
- Borrar un libro (como administrador) comprobar que se ha borrado

Tests con WebTestClient (1 pts) - Opcional

- Realizar los test unitarios y de API REST de los ejercicios anteriores utilizando WebTestClient

Consideraciones

- Los tests **deben ser independientes entre sí** (no depender de información que otros tests hayan creado o eliminado). Además de ser un anti-patrón, JUnit no garantiza el orden de ejecución por defecto.
- La aplicación utiliza HTTPS y Basic Auth.
- Se valorará la modularización de los tests en paquetes o clases diferentes, dado que son de diferente naturaleza.
- En los test unitarios, es **obligatorio el uso de Mocks**, dado que la persistencia se realiza utilizando una base de datos H2 (y queremos evitarlo en este tipo de test).

Material de ayuda

Dado que no se ha explicado en clase cómo manejar los test unitarios y de API cuando utilizamos autenticación, se proporciona a continuación código de ayuda para estos casos:

Autenticación en test unitarios

Es necesario añadir la anotación `@WithMockUser` para generar un mock de un usuario con un rol que necesitemos para la prueba. Los valores (a excepción de roles) no es relevante.

```
@Test
@WithMockUser(username = "user", password = "pass", roles = "USER")
public void myTest() throws Exception {
```

Es necesario importar la siguiente dependencia para importar la anotación:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Autenticación en RestAssured

Es necesario añadir en el `given()` la autenticación necesaria. En este caso, es necesario poner credenciales existentes en la base de datos.

```
given()
  .auth()
     .basic("user", "pass")
```

Además, al trabajar con aplicaciones seguras que utilizan HTTPS, es necesaria una configuración adicional en RestAssured. Se proporciona a continuación código de ayuda para estos casos:

HTTPS en RestAssured

Es necesario sobrescribir la URL base a la que “atacarán” nuestros test, forzando HTTPS. Además, por defecto, no acepta certificados autofirmados, por lo que es necesario relajar esta restricción.

```
  @BeforeEach
  public void setUp() {
      RestAssured.port = port;
      RestAssured.useRelaxedHTTPSValidation();
      RestAssured.baseURI = "https://localhost:"+port;
  }
```

Parte 2: Testing en Node.js con Express (3 pts)

Se desea implementar las pruebas necesarias para comprobar la correcta funcionalidad de una aplicación que gestiona películas. Se proporciona el código de dicha aplicación (publicado en el Aula Virtual: **Node enunciado**).

Se quieren realizar las siguientes pruebas de la aplicación:

Tests de API REST *mockeando* la conexión a BBDD (1.5 pts)
- Añadir una nueva película
- Recuperar todas las películas

Tests de API REST utilizando TestContainers (1.5 pts)
- Añadir una nueva película
- Recuperar todas las películas

Consideraciones

- Los tests **deben ser independientes entre sí** (no depender de información que otros tests hayan creado o eliminado).
- Se valorará utilizar módulos distintos por cada tipo de test, dado que son de diferente naturaleza.
- Los test se deben realizar con Jest y Supertest.
- La aplicación usa una base de datos DynamoDB, normalmente gestionada por el proveedor de servicios cloud Amazon Web Services. Es posible ejecutar la aplicación en local levantando una instancia de esta base de datos a través de este comando (solo es necesario tener instalado Docker):
  - `docker run --rm -p 8000:8000 -d amazon/dynamodb-local:1.13.6`
- En los **test unitarios**, es **obligatorio el uso de Mocks**, dado que la persistencia se realiza utilizando una base de datos.
- En los test de API REST. no queremos usar una base de datos existente (ya sea remota o local). Para ello, es **obligatorio el uso de TestContainers**.

Material de ayuda

Configuración del cliente de DynamoDB (AWS)

Para utilizar servicios de AWS, como DynamoDB, ya sea en local o remoto, es necesario realizar una configuración del cliente de AWS antes de lanzar la aplicación (*server.js*):

```
AWS.config.update({
      region: process.env.AWS_REGION || 'local',
      endpoint: process.env.AWS_DYNAMO_ENDPOINT || 'http://localhost:8000',
      accessKeyId: "xxxxxx", // No es necesario poner nada aquí
      secretAccessKey: "xxxxxx" // No es necesario poner nada aquí
});
```

Esta configuración, **si no se proporciona**, por defecto fallará.

Configuración del schema en DynamoDB

DynamoDB es una base de datos NoSQL similar a MongoDB. A diferencia de este, requiere declarar un Schema para poder crear una tabla. Esta creación se hace de forma posterior a la configuración del cliente mencionada en el punto anterior, pero antes de lanzar la aplicación (*server.js*). Para ello utiliza el siguiente método (asíncrono):

```
      createTableIfNotExist("films");
```

HINT: Será necesario utilizarlo ambas configuraciones en nuestro test, aunque solo si realizamos una conexión real.

Formato de entrega

La práctica se entregará teniendo en cuenta los siguientes aspectos:

- La práctica se entregará como un fichero .zip. Este fichero contendrá dos carpetas:
  - Java: La aplicación Java junto a los test desarrollados
  - Node: La aplicación Node.js junto a los test desarrollados
- El nombre del fichero .zip será el correo URJC del alumno (sin @alumnos.urjc.es).
- El proyecto se puede crear con cualquier editor o IDE.

Las prácticas se podrán realizar de forma individual o por parejas. En caso de que la práctica se haga por parejas:

- Sólo será entregada por uno de los alumnos

- El nombre del fichero .zip contendrá el correo de ambos alumnos separado por guión. Por ejemplo `p.perezf2020-z.gonzalez2020.zip`