JavaScript
Herramientas de testing

Manuel Tomas Ortega Sanchez

Version 1.0.0 2019-05-10

Contenidos

Capítulo 1. Jest

URL: https://jestjs.io/

Capítulo 2. Introducción

Jest es un test runner basado en Jasmine y creado por el equipo de Facebook. Se va a encargar de buscar los tests en nuestra aplicación, ejecutarlos y nos mostrará unos resultados en pantalla que nos dirán si se han pasado todos los tests correctamente o no. Jest se puede integrar con otras librerías de testing como **Enzyme**. También nos muestra la cobertura de los tests sin necesidad tener que configurarlo o de añadir paquetes extra. Y trae consigo el **snapshoting test** que se encarga de comparar la salida de un componente con una copia que se almacena la primera vez que se testea y de esta forma, los tests fallarán si se ha cambiado algo en los componentes que se están testeando, porque ya no coincidirán las dos salidas.

Nuestros archivos de tests tienen que estar en una carpeta `tests` o en archivos con las siguientes extensiones: `.test.js` o `.spec.js`. De esta forma Jest sabrá que archivos van a contener los tests de nuestra aplicación.

Capítulo 3. Paquetes

Para poder crear los test de una forma sencilla, vamos a necesitar instalar las siguientes dependencias:

- jest: el paquete para ejecutar los test.
- jest-cli: la interfaz de linea de comandos para usar jest.
- enzyme: nos va a ayudar a manipular los componentes en los tests y comprobar que se cumplen de una forma muy sencilla.
- enzyme-to-json: serializa los snapshots que genera jest para que sean más fáciles de entender.

```
$ npm install --save jest jest-cli enzyme enzyme-adapter-react-16 enzyme-to-json
```

Una vez instalados vamos a configurar nuestro proyecto para poder ejecutar los test correctamente.

Capítulo 4. Configuración

Lo primero que vamos a hacer es modificar el script de test que hay en el `package.json` para que ejecute los tests usando jest.

/package.json

```
{
  "name": "jest-standalone",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "jest"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "enzyme": "^3.10.0",
    "enzyme-to-json": "^3.3.5",
    "jest": "^24.8.0",
    "jest-cli": "^24.8.0"
  }
}
```

Para ejecutar nuestros test, únicamente debemos de ejecutar el script test de NPM

```
$ npm run test
```

Capítulo 5. Cypress

URL: https://www.cypress.io/

Se trata de otro framework de pruebas basado en javascript que se le conoce como "el todo en uno"

Incluye librería de aserciones, temas de mocks y pruebas e2e, como hace Selenium

Cypress no utiliza Selenium por que consta de una nueva arquitectura construída desde cero

Tras Cypress se encuentra un hilo de proceso mediante NodeJs que constantemente se está comunicando, sincronizando y ejecutando tareas == Jasmine

URL: https://jasmine.github.io/

Características:

- Se trata de un framework de desarrollo dirigido por comportamiento para código javaScript
- No depende de ninguna otra librería javaScript
- No requiere de un DOM para su ejecución
- Sintaxis limpia para poder escribir test fácilmente

Se puede decir que se trata de uno de los estándares de facto en el mundo del testing cuando hablamos de javaScript

5.1. Utilizando Jasmine de forma aislada

Descargamos de la siguiente URL el .zip de la versión de Jasmine que deseemos: https://github.com/jasmine/jasmine/releases

Esta forma de utilizar puede resultar apropiada sobre todo en proyectos legacy, donde no es posible emplear una arquitectura más elaborada

Para nuestro caso, vamos a descargar la versión 3.4.0

Descomprimimos el archivo y nos quedamos con los archivos que se exponen, en la estructura propuesta

Vamos a crear una estructura de proyecto con la siguiente arquitectura

- jasmine-standalone
  - spec
  - src
  - lib
    - jasmine-standalone-3.4.0
      - boot.js

        - jasmine_favicon.png
        - jasmine-html.js
        - jasmine.css
        - jasmine.js
        - MIT.LICENSE
    - SpecRunner.html

Carpeta **src**, tendremos nuestros archivos de código fuente

Carpeta **spec**, tendremos nuestros archivos de test

Carpeta **lib**, tendremos nuestras librerías javascript para nuetro proyecto

El archivo **SpecRunner.html** es una plantilla que nos provee la propia librería cuando la hemos descargado, mediante ella podremos ejecutar el conjunto de test que deseemos, haciendo referencia a tantos include como tengamos

Si abrimos con el navegador el archivo SpecRunner.html observaremos como se ejecuta el conjunto de test de los archivos de la carpeta spec

5.2. Utilizando Jasmine mediante NPM

Otra forma de utilizar jasmine, es mediante la creación de un projecto NPM

Esta forma tiene el beneficio de que disponemos del gestor de dependencias NPM, lo que nos permite indicar versión de las librerías que utilizaremos, así como poder utilizar diferentes scripts como el de test

Primero vamos a crear nuestra carpeta del proyecto, indicamos como nombre **jasmine-npm**

Dentro de la carpeta, inicializamos el proyecto,este comando nos preguntará sobre el nombre del proyecto, licencia, repositorio git, etc. Le damos a intro al conjunto de opciones y listo

```
$ npm init
```

A continuación, vamos a instalar la dependencia de jasmine que nos interesa, de forma que quede anotada en el archivo de manifiesto

Indicamos que el uso es para desarrollo, por lo que quedará anotada en el package.json, en la sección de dependencias para desarrollo

```
$ npm install jasmine --save-dev
```

Una vez descargada nuestra dependencia, se habrá creado una carpeta llamada node_modules

En el raíz del proyecto, nos situamos y ejecutamos el script de inicialización de jasmine, para que

nos cree una estructura básica de proyecto

```
$ node node_modules/jasmine/bin/jasmine init
```

Ahora, editamos nuestro archivo package.json e indicamos que en la sección de scripts, en la fase test, queremos ejecutar el binario jasmine

El archivo package.json quedará de la siguiente forma

```
{
  "name": "jasmine-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "jasmine"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "jasmine": "^3.4.0"
  }
}
```

La arquitectura de nuestro proyecto quedaría de la siguiente forma

- jasmine-npm
  - node_modules
  - spec
    - support
      - jasmine.json
    - BookBsSpec.js
  - src
    - CalculatorBs.js
  - package-lock.json
  - package.json

Para ejecutar nuestros test ejecutamos

```
$ npm run test
```

5.3. Utilizando Jasmine mediante Maven

En la URL del plugin podremos consultar la documentación sobre el uso: https://justin.searls.co/jasmine-maven-plugin/index.html

Primero creamos nuestro proyecto java maven a partir de un arquetipo específico, ejecutamos por consola lo siguiente

```
$ mvn archetype:generate -DgroupId=com.app.tomcat.jasmine -DartifactId=jasmine-java -DarchetypeArtifactId=maven-archetype
-webapp -DinteractiveMode=false
```

Adicionalmente, dentro de la carpeta webapp creamos la estructura **webapp** > **js**, dentro de la carpeta js indicamos el archivo CalculatorBs.js

También, a la altura de la carpeta main, creamos la estructura **src** > **test** > **javascript** y cramos nuestro archivo CalculatorBsSpec.js

La estructura del proyecto quedaría así

- jasmine-java
  - src
    - main
      - resources
      - webapp
        - js
          - CalculatorBs.js
        - WEB-INF
          - web.xml
        - index.jsp
    - test
      - javascript
        - CalculatorBsSpec.js
  - pom.xml