NodeJS: Test unitarios

Gabriel Rodríguez Flores

October 28, 2021

ADA ITS
INSTITUTO TECNOLÓGICO SUPERIOR

- Realización de test unitarios con Jest o Ava
- Análisis de cobertura de código con Istanbul y SonarQube

Contents

1 Teoría

Introducción al tema:

1. Necesitamos un módulo para ejecutar los test: Ava o Jest
2. Podemos usar librerías para mock: Sinon y otras específicas
3. Exportamos los resultados de los test con Istanbul (nyc)
4. Visualizamos y analizamos la cobertura y calidad de código con SonarQube (sonarqube-scanner)

> Importante: Todas las librerías exclusivas para los test han de ser instaladas en modo Desarrollo (-D).

1.1 AVA

```
npm i -D ava
```

1.1.1 Configuración

En el fichero **package**.json

```
{
  ...
  "ava": {
    "files": [
      "test/**/*.test.js"
    ]
  }
  ...
}
```

1.2 Jest

1.3 Mock: Simular comportamientos externos

Cuando una función no es pura (no siempre tiene el mismo resultado cuando se invoca con los parámetros), tenemos que probar las distintas variantes de respuestas que esta funcion nos puede dar. O simplemente cuando no queremos que se realiza la petición real porque no estamos en el entorno adecuado. Podemos simular la respuesta de un método sustituyendo su ejecución real por una definida manualmente.

1.3.1 Sinon

```
npm i -D sinon
```

1.3.1.1 Stub: Evitar o alterar respuesta a un método según su entrada

```
const sinon = require('sinon');

const callback = sinon.stub();
callback.withArgs(42).returns(1);
callback.withArgs(1).throws("name");
callback.returns(23);
```

1.3.1.2 Stub: Para funciones void

```
const sinon = require('sinon');

const logInfo = sinon.stub(console, 'log');
logInfo.calledWithExactly('mensaje en el console.log()');
```

1.4 Istambul: Cobertura de código

```
npm i -D nyc
```

```
nyc --reporter=html ava
nyc --reporter=lcov --reporter=text-lcov ava
```

Directorios generados (a omitir en el seguimiento del repositorio):

- .nyc_output
- coverage –> (por la instrucción del html)

> Se suele requerir la una cobertura de código del 80%.

1.5 SonarQube: Revisión de test y código

Directorios generados (a omitir en el seguimiento del repositorio):

- .scannerwork

1.5.1 Docker-compose

```yaml
version: "3"
services:
  sonarqube:
    image: "sonarqube"
    ports:
      - "9000:9000"
      - "9092:9092"
```

1.5.2 Integración con Nodejs

```
npm i -D sonarqube-scanner
```

La configuracion se ajusta en el fichero `test/sonar.js` por defecto

1.5.3 Procedimiento

1. Arrancar el servicio de SonarQube
2. Generar el reporte de test
3. Acceder a SonarQube

2 Ejemplos

2.1 Configuración completa del `package.json`

```json
{
  "name": "fizzbuzz",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "compose:test": "docker-compose -f docker-compose.test.yml up -d",
    "test": "cross-env NODE_ENV=test nyc ava",
    "test:html": "cross-env NODE_ENV=test nyc --reporter=html ava",
    "test:watch": "cross-env NODE_ENV=test ava --watch --verbose",
    "test:report": "cross-env NODE_ENV=test nyc --reporter=lcov --
       reporter=text-lcov ava || echo 1; node test/sonar.js",
    "test:purge": "rm -r .nyc_output .scannerwork coverage"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
```

```
    "ava": "^3.15.0",
    "cross-env": "^7.0.3",
    "nyc": "^15.1.0",
    "sonarqube-scanner": "^2.8.1"
  },
  "ava": {
    "files": [
      "test/**/*.test.js"
    ]
  }
}
```

2.2 Configuración de SonarMQ: `sonar.js`

```
const sonarqubeScanner = require('sonarqube-scanner');

sonarqubeScanner({
  serverUrl: 'http://localhost:9000',
  options: {
    'sonar.sources': '.',
    'sonar.tests': 'test',
    'sonar.inclusions': 'src/**',
    'sonar.coverage.exclusions': 'src/**/index.js',
    'sonar.javascript.lcov.reportPaths': 'coverage/lcov.info',
    'sonar.login': 'admin',
    'sonar.password': 'adminadmin',
  },
}, () => { });
```

2.3 Ejemplo práctico: FizzBuzz

- fizzbuzz.js

```
function fizzBuzz(n){
  let result = '';
  if(n%3 === 0) result += 'fizz';
  if(n%5 === 0) result += 'buzz';

  return result || n;
}

module.exports = fizzBuzz;
```

- fizzbuzz.test.js

```
const test = require('ava');
```

```
2
3  const fizzBuzz = require('../src/fizzbuzz');
4
5  test('Should return 1', t => {
6    const result = fizzBuzz(1);
7    t.is(result, 1);
8  });
9
10 test('Should return fizz', t => {
11   const result = fizzBuzz(3);
12   t.is(result, 'fizz');
13 });
14
15 test('Should return buzz', t => {
16   const result = fizzBuzz(5);
17   t.is(result, 'buzz');
18 });
19
20 test('Should return fizzbuzz', t => {
21   const result = fizzBuzz(15);
22   t.is(result, 'fizzbuzz');
23 });
```

3 Ejercicios

1. Realizar ejercicio *fizzbuzz* con sus test correspondientes con AVA
2. Realizar ejercicios de comparación de fechas con sus test usando AVA. Se realiza una función llamada `dateCompare` que:
   - Al recibir dos fechas, devolverá cual es anterior y cual es posterior en un objeto `{ startDate: 'ISODateString', endDate: 'ISODateString'}`
   - Si solo recibe una fecha, se comparará con el momento actual
3. Investigar y realizar los test de los ejercicios usando `Jest`.
4. Modificar *fizzbuzz* para recibir el numero y las condiciones en un objeto.

```
1 const n = 100;
2 const condition = {
3   2: 'poo',
4   3: 'fizz',
5   5: 'buzz',
6   15: 'foo'
7 };
```

4 Entregables

4.1 En clase

Ejercicio 1 y 2

- Entrega del proyecto *fizzbuzz* con:
  - Programa funcionando correctamente
  - Test realizados con ava, con cobertura de código 100%
  - Docker-compose para arrancar SonarQube
  - Fichero de configuración de SonarQube
  - Script del `package.json` para automatizar las labores de: arrancar SonarMQ, lanzar los test, lanzar los test en modo *watch* y generar los ficheros de test.

4.2 Tarea

- Ejercicio 3: Se pide entregar los test realizados de ejemplos con JEST en lugar de AVA con el estudio de cobertura
- Ejercicio 4: Modificación sobre el script y test realizados de *fizzbuzz*. En Jest o AVA.