TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- Herramientas para el testing unitario: Jest y Mocha + Chai.
- Jest v/s Mocha + Chai.
- Objetos simulados, definición y contexto de uso.

HERRAMIENTAS PARA EL TESTING UNITARIO: JEST

Es un `framework` de código abierto, lanzado a fines del año 2015 por el equipo de Facebook, y basado en Jasmine. Originalmente creado como herramienta para testeo de aplicaciones generadas con `React`, su finalidad es llevar a cabo pruebas de aplicaciones desarrolladas en `JavaScript`, flexibles, rápidas, y con un output sencillo y comprensible. Trabaja con la ejecución de pruebas en paralelo, iniciando cada prueba de manera separada e independiente de las otras.

HERRAMIENTAS PARA EL TESTING UNITARIO: MOCHA + CHAI

Liberado inicialmente en el año 2011, `Mocha` es un `framework` para testeo de código abierto, mientras que `Chai`, es una biblioteca de aserciones que apareció, también como código abierto, solo un par de años después. Juntos, son dos de las herramientas más comúnmente empleadas para realizar pruebas de código `JavaScript`.

`Mocha` proporciona a los desarrolladores, un marco de prueba básico para la realización de los tests. Para funcionar necesita, y permite, seleccionar las bibliotecas de aserción, `mocking` y `stubs` que se van a usar. Ejecuta los tests de manera secuencial, tanto en un entorno de navegador, como a través de `Node.js`, otorgando la posibilidad de crear pruebas síncronas y asíncronas de forma muy sencilla.

`Chai` es una de las bibliotecas de aserciones, o afirmaciones, más populares. Ofrece toda una colección de éstas ya preparadas, permitiéndonos, además, escoger entre diversas sintaxis a la hora de aplicarlas. Proporciona los estilos de programación `TDD`, para probar su código en cualquier framework JavaScript, junto a una gran cantidad de complementos y extensiones.

JEST V/S MOCHA + CHAI

`Mocha` es un proyecto antiguo y maduro, que cuenta con una gran comunidad de usuarios y amplia documentación, además de preguntas en StackOverflow, y artículos de soporte para la configuración. Su desempeño es mejor testeando el back end, y funciona muy bien para proyectos grandes que requieran personalización y flexibilidad.

`Jest` es un `framework` un tanto más reciente. Sin embargo, ha adquirido bastante popularidad, y sigue mejorando considerablemente con cada actualización. Si bien se usa principalmente para probar aplicaciones `React`, puede integrarse fácilmente en cualquiera. A diferencia de otros, posee una amplia `API`, que no requiere incluir bibliotecas adicionales para su uso, a menos que realmente

se necesiten, lo cual la hace simple y rápida de utilizar, sobre todo en proyectos pequeños. Esto facilita la adopción, por parte de los desarrolladores, de la metodología `TDD`. Funciona mejor para el testeo del front end gracias al `snapshot testing`.

OBJETOS SIMULADOS, DEFINICIÓN Y CONTEXTO DE USO

Antes de continuar, vamos a incluir un par de conceptos nuevos: las aserciones (`assertions`) y expectativas (`expectations`). Si bien no corresponden a objetos simulados, son expresiones que forman parte de algunos de ellos, y permiten evaluar un resultado o comportamiento dentro de una prueba de software.

- Aserción: es una expresión lógica, donde se asume que siempre es verdadera en determinado punto de la ejecución de un programa. Por ejemplo: 1! = 0.
- Expectativa: expresión que define la expectativa de que el resultado de un comportamiento iguale un determinado valor.

Un objeto simulado, o doble de prueba, es simplemente uno que utilizamos en sustitución de otro, cuando queremos realizar una prueba que necesita recibir información de otra fuente. Por ejemplo, casos como el uso de `APIs` de un tercero, justifican su existencia, ya que, de utilizar la `API` real, el test dejaría de ser unitario. Además, sería lento en la respuesta, perdería la independencia del entorno, y en muchos casos, sería costoso, ya que varias son de pago por consulta. A continuación, presentaremos en orden alfabético, las definiciones de los objetos simulados más conocidos, basándonos en las entregadas por Martin Fowler.

- Dummy: son objetos de relleno, que se usan para completar el requerimiento de un método en testeo, pero no intervienen directamente en la funcionalidad de éstos.
- Fake: son componentes de nuestra aplicación, que funcionan como un objeto normal y operativo, sin ser simulados, pero que sólo implementan lo mínimo necesario para poder pasar las pruebas.
- Mocks: son objetos preprogramados, cuyo foco se centra en registrar el comportamiento durante la prueba, para luego reportarlo. Tienen expectativas que especifican las llamadas que espera recibir y así, cuando otro objeto los llama, reemplazan por completo al base, y solo devuelven valores por defecto o valores definidos por nosotros, permitiendo testear el paso de mensajes entre objetos, qué parámetros ha recibido, o cuantas veces se ha realizado la llamada.

Un `mock` podría ser útil, cuando un objeto:

- Proporciona resultados no deterministas (por ejemplo, la hora o la temperatura actual).
- Tiene estados que son difíciles de crear o reproducir (por ejemplo, un error de red).
- Es lento (por ejemplo, una base de datos completa).
- Todavía no existe, o puede cambiar el comportamiento.
- Tendría que incluir información y métodos exclusivamente para fines de prueba (y no para su tarea real).

- Spy: son objetos que reemplazan solo un subconjunto, de todos los métodos del objeto original, y el resto de sus métodos son idénticos a los originales.
- Stub: es un doble de test, que proporciona respuestas predefinidas cuando uno de sus métodos es llamado durante una prueba. Por lo general, no responden en absoluto a nada fuera de lo programado para el test. Se usan para evitar que el sujeto bajo prueba, o `SUT` (subject under test), llame a métodos externos cuando necesita obtener una respuesta de ellos. Es importante destacar que este tipo de dobles solo verifica el estado de los objetos, y nunca su comportamiento o relación con otras entidades. El `stub` no dispone de verificación de expectativas, para eso hay que emplear las aserciones.