Pruebas Unitarias y end-to-end en un entorno Vue

Objetos simulados

{desafío} latam_

Implementar pruebas unitarias utilizando las herramientas provistas por Vue para verificar el correcto funcionamiento del aplicativo.

- Unidad 1:
  Vue Router
- Unidad 2:
  Vuex
- Unidad 3:
  Firebase
- Unidad 4:
  Pruebas Unitarias y end-to-end en un entorno Vue

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Implementa pruebas unitarias utilizando las herramientas provistas por Vue para verificar el funcionamiento de un componente.*

¿Qué son los matchers en Jest?

{desafío} latam_

¿Qué tipo de matchers vimos en la sesión anterior?

¿Para qué utilizamos el matcher .toBeGreaterThanOrEqual()?

{desafío} latam_

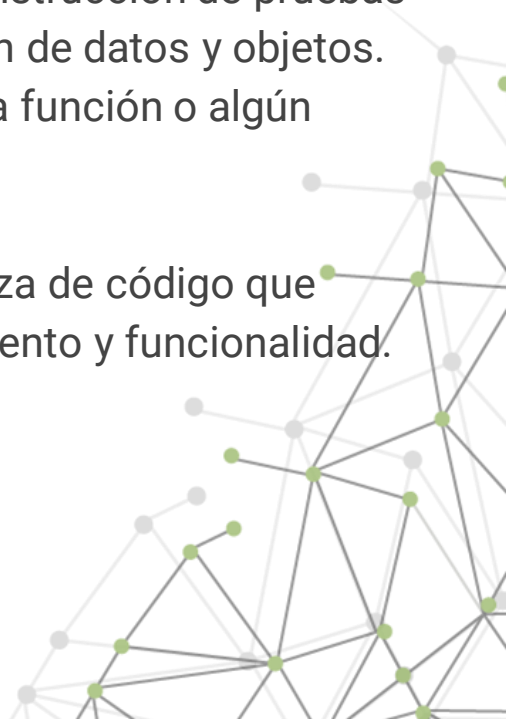
¿Para qué utilizamos el matcher .toHaverLength()?

{desafío} latam_

Contexto antes de iniciar

A continuación, veremos otras funcionalidades en el entorno de la construcción de pruebas unitarias en una aplicación Vue JS. Estaremos revisando la simulación de datos y objetos. Esta simulación permitirá validar el comportamiento hipotético de una función o algún elemento de nuestras aplicaciones.

Para ello, revisaremos el concepto de mocks, que consiste en una pieza de código que imita o simula otra porción de código para comprobar su comportamiento y funcionalidad.



/* Objetos simulados */

¿En qué consisten?

Los objetos simulados, en el ámbito del desarrollo basado en pruebas (TDD), consisten en piezas de código que imitan el comportamiento de otro código.

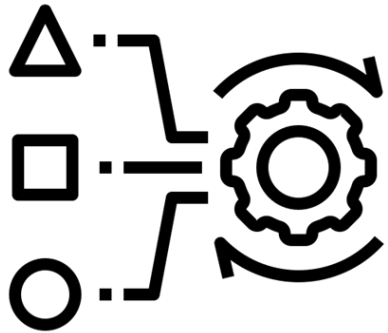
Imaginemos el siguiente caso:

- Tenemos un formulario de inicio de sesión con los campos email y contraseña.
- ¿Qué podríamos simular?
  - Que el campo email reciba datos de tipo email.
  - Que el campo contraseña reciba datos de tipo string y/o numéricos. Además, que sea de tipo password para que no se muestre la contraseña
  - Etc.

Veamos otro ejemplo, supongamos que tenemos una función que se conecta y extrae información de una API. ¿Qué simularíamos?

¿Por qué utilizarlos?

- Simular es importante en el ámbito del TDD.
- Esto nos habilita la opción de imitar el comportamiento de elementos y funciones, para verificar que se cumple con lo esperado.
- Cuando estamos escribiendo pruebas en el fondo, estamos simulando errores para su posterior corrección.



/* Mocks */

¿Qué son los mocks?

Los mocks son piezas de código que utilizamos para simular o verificar un comportamiento. Son utilizados durante la ejecución de las pruebas. Por ejemplo:

- Con mocks podemos confirmar si un método se llamó **n** cantidad de veces.
- Podemos confirmar los parámetros con que los métodos son ejecutados.
- Verificar, confirmar y simular son las palabras claves asociadas a los mocks.

Demostración:
"Ejemplo de implementación de mocks con Jest"

{desafío}
latam_

Ejemplo de implementación de mocks con Jest

A continuación, realizaremos un ejercicio en el cual aplicaremos la utilización de mocks para simular y confirmar un resultado en una función que haga el consumo de una API, para este ejemplo utilizaremos el archivo **“Material de apoyo - Mocks Jest”**, el cual está disponible en plataforma.

Una vez descargado debemos correr el comando **npm install** para la instalación de las dependencias.

Sigue los pasos...

- **Paso 1:** Abrimos la aplicación en el editor de código y nos dirigimos al archivo sumar.spec.js. En él, empezaremos a escribir la lógica de nuestra prueba con mocks.

```
import {sumar} from "@/utils/sumar";

jest.mock('@/utils/sumar', () => ({
  sumar: jest.fn(),
}));
```

Sigue los pasos...

- **Paso 2:** Definimos el cuerpo de nuestra prueba.

```
describe('sumar', () => {
  test('debería calcular la suma correctamente', () => {
    //Arrange
    sumar.mockReturnValue(6);
    //Act
    const resultado = sumar(2, 3);
    //Asserts
    expect(resultado).toBe(5);
  });
});
```

Resultado de ejecución

```
FAIL tests/unit/suma.spec.js
 sumar
   × debería calcular la suma correctamente (5 ms)

 ● sumar › debería calcular la suma correctamente

   expect(received).toBe(expected) // Object.is equality

   Expected: 5
   Received: 6

     14 |
     15 |     //Asserts
   > 16 |     expect(resultado).toBe(5);
        |                       ^
     17 |   });
     18 | });
```

Ejercicio:
"Implementa tus mocks"

Implementa tus mocks

Contexto

A continuación, deberán realizar un ejercicio individual en el que aplicarán los aprendizajes acerca de los mocks. Para lograrlo implementa lo siguiente:

- Crea una función en el directorio `/utils` que permita realizar una resta a partir de 2 números pasados por argumentos.
- Además, crea una función que permita ejecutar una operación de división entre dos números pasados como argumentos a una función.

Nota: Utiliza la misma aplicación de Vue JS utilizada en el ejercicio anterior.

Implementa tus mocks

Instrucciones

- En la carpeta de `tests/unit` crea los archivos correspondientes a cada función.
- Genera los mocks que controlarán el retorno de esta función.
- Genera el test donde se espere que el valor sea un número definido a libre elección.

Tips:

```
describe('resta', () => {
  test('debería calcular la resta correctamente', () => {
    //Arrange
    resta.mockReturnValue(2);
    //Act
    const resultado = resta(2, 3);
    //Asserts
    expect(resultado).toBe(-1);
  });
});
```

{desafío} latam_

Compartan los resultados de ejecución del ejercicio anterior.

{desafío} latam_

/* Stubs */

¿Qué son los stubs?

El concepto de stubs puede resultar muy parecido a Mocks, sin embargo, en el ámbito de las pruebas unitarias, los stubs nos permiten realizar pruebas simuladas sobre componentes o funciones que utilizan datos externos, por ejemplo una API.

- Si estamos probando un componente que recibe información de una API, podemos realizar un stub que simule estos datos.
- Con esto verificamos que el componente se comporta de manera correcta incluso si la API real llega no estar disponible.

Diferencias entre stubs y mocks

Los stubs los utilizamos para:

- Probar el comportamiento de un componente Vue en ausencia de una dependencia externa, por ejemplo una API

Los mocks los utilizamos para:

- Verificar el comportamiento de un componente Vue en respuesta a diferentes eventos y cambios de estados de la aplicación.
- Comprobar el comportamiento de un componente Vue en respuesta a errores.

En la guía de ejercicios veremos cómo comprobar la conexión a una API

Resumen de la sesión

- Los objetos simulados, en el ámbito del desarrollo basado en pruebas (TDD), consisten en piezas de código que imitan el comportamiento de otro código.
- Cuando estamos escribiendo pruebas en el fondo, estamos simulando errores para su posterior corrección.

¿Existe algún concepto que necesiten repasar de esta sesión?

Próxima sesión...

- *Guía de ejercicios.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales