Guía de ejercicios - Pruebas Unitarias en un entorno Vue

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

La siguiente guía de estudio tiene como objetivo practicar y ejercitar los contenidos que hemos visto hasta el momento.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Actividad guiada: Pruebas unitarias snapshot

A continuación, realizaremos un ejercicio en el cual a partir de un contador en Vue JS haremos pruebas unitarias para verificar algunos comportamientos en su ejecución.
Para ello, utilizaremos el **"Material de apoyo - Pruebas unitarias snapshot"** que estará disponible en la plataforma, una vez descargado ejecuta en terminal el comando `npm install` para que las dependencias sean instaladas correctamente. Una vez terminada la instalación, veremos la siguiente información en el navegador al levantar el servidor.

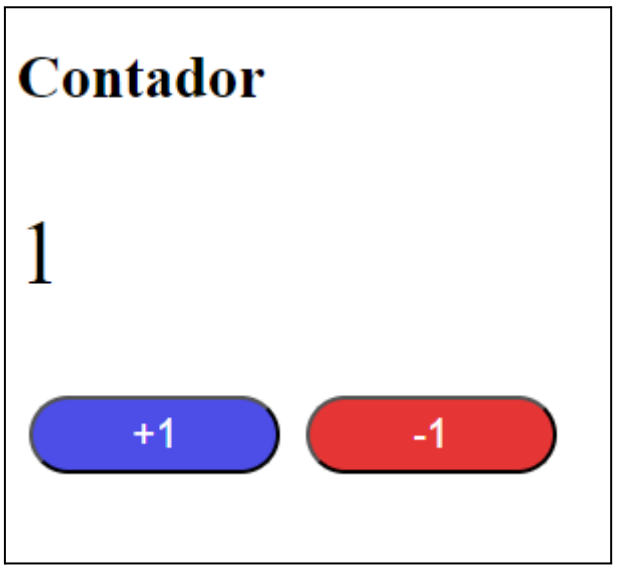


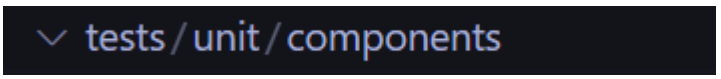
Imagen 1. Estado inicial de la aplicación
Fuente: Desafío Latam

Snapshots

Los snapshots son una herramienta que utilizaremos para capturar el estado de un componente. Son útiles para la verificación de la estructura HTML y CSS de los componentes.
Para incorporar snapshots, debemos hacer uso de `shallowMount` o `mount`. Estos, son métodos que tenemos disponibles en el paquete de Vue Test Utils y nos permitirán hacer el montaje de los componentes que deseamos probar.

Teniendo la información referente a los snapshots y su utilidad, procedamos con los pasos del ejercicio:

- **Paso 1:** En el directorio `/tests/unit` creamos una nueva carpeta llamada components, esta estructura es la recomendada dado que nos ayudará a identificar rápidamente cuál es el componente al que estamos haciendo pruebas.



- **Paso 2:** Creamos un archivo llamado contador.spec.js y definimos la estructura de la primera prueba que ejecutaremos, además debemos importar `shallowMount` y el componente `Contador.vue`.

```
import { shallowMount } from '@vue/test-utils'
import Contador from '@/components/Contador'


describe('Componente Contador.vue', () => {
  test('Validación de match con el snapshot', () => {
    //Selección del sujeto de pruebas

    //Aserción

  })
})
```

- **Paso 3:** Para seleccionar el objeto de pruebas utilizaremos un *wrapper,* un wrapper es como un envoltorio en el cual estará el objeto de pruebas, en este caso es el componente `Contador.vue`.

```
describe('Componente Contador.vue', () => {
  test('Validación de match con el snapshot', () => {
     //Selección el Sujeto de pruebas
     const wrapper = shallowMount(Contador)

     //Aserción
     expect(wrapper.html()).toMatchSnapshot()
  })
})
```

Wrapper, es la representación de un componente en Vue, lo podemos utilizar para interactuar con dicho componente y realizar las siguientes acciones a través de un conjunto de métodos:

1. **find():** Podemos buscar un elemento HTML dentro del componente.
2. **text():** Podemos obtener un texto en específico dentro de un elemento HTML.
3. **html():** Obtenemos la estructura HTML del componente.
4. **trigger():** Podemos simular un evento que esté definido dentro del componente.
5. **vm():** Obtenemos el objeto Vue en el componente.

En el código anterior estamos obteniendo la estructura HTML con `wrapper.html()`, seguidamente, esperamos que la estructura haga match con el snapshot que se genere.

- **Paso 4:** Ejecutemos las pruebas con el comando `npm run test:unit` y verifiquemos los cambios que se incorporan en la aplicación.

```
 % npm run test:unit

> pruebas-unitarias-snapshot@0.0.0 test:unit
> jest

 PASS  src/tests/unit/components/contador.spec.js
  Componente Contador.vue
    ✓ Validación de match con el snapshot (12 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   1 passed, 1 total
Time:        0.635 s, estimated 1 s
Ran all test suites.
```

Imagen 2. Corriendo pruebas y generación de snapshot
Fuente: Desafío Latam

Al correr esta prueba veremos que en la carpeta `/tests/unit/components` se incorpora un nuevo directorio llamado `/__snapshot__`.

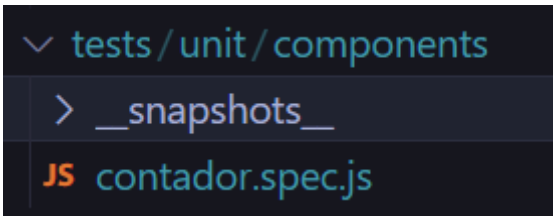


Imagen 3. Carpeta de snapshots
Fuente: Desafío Latam

Si ingresamos a esta carpeta veremos un archivo con extensión `.snap` el cual contiene una foto referencial de lo que el componente `Contador.vue` tiene en su estructura HTML. Esta "foto" que se genera, es un archivo minificado en el cual se eliminan los saltos de línea del componente original

Utilidad de esta prueba

La utilidad los snapshots radica en que podemos guardar una referencia de la estructura de un componente HTML, con esto tenemos una validación de la estructura que dicho componente debe tener.

En este sentido, si por alguna razón este componente recibe algún cambio, bien sea por error o con intención, la prueba mostrará una alerta indicando que no es compatible con el snapshot inicial.

- **Paso 5:** Hagamos una leve modificación incorporando una clase de CSS al título h2. Luego, veamos qué sucede al correr nuevamente la prueba.

```
<template>
  <h2 class="titulo-principal">{{ tituloPorDefecto }}</h2>
  <p class="counter">{{ contador }} </p>

  <div>
    <button class="btn primary" v-on:click="incrementar">+1</button>
    <button class="btn warning" v-on:click="disminuir">-1</button>
  </div>
</template>
```

Al correr nuevamente la prueba veremos el siguiente mensaje en terminal

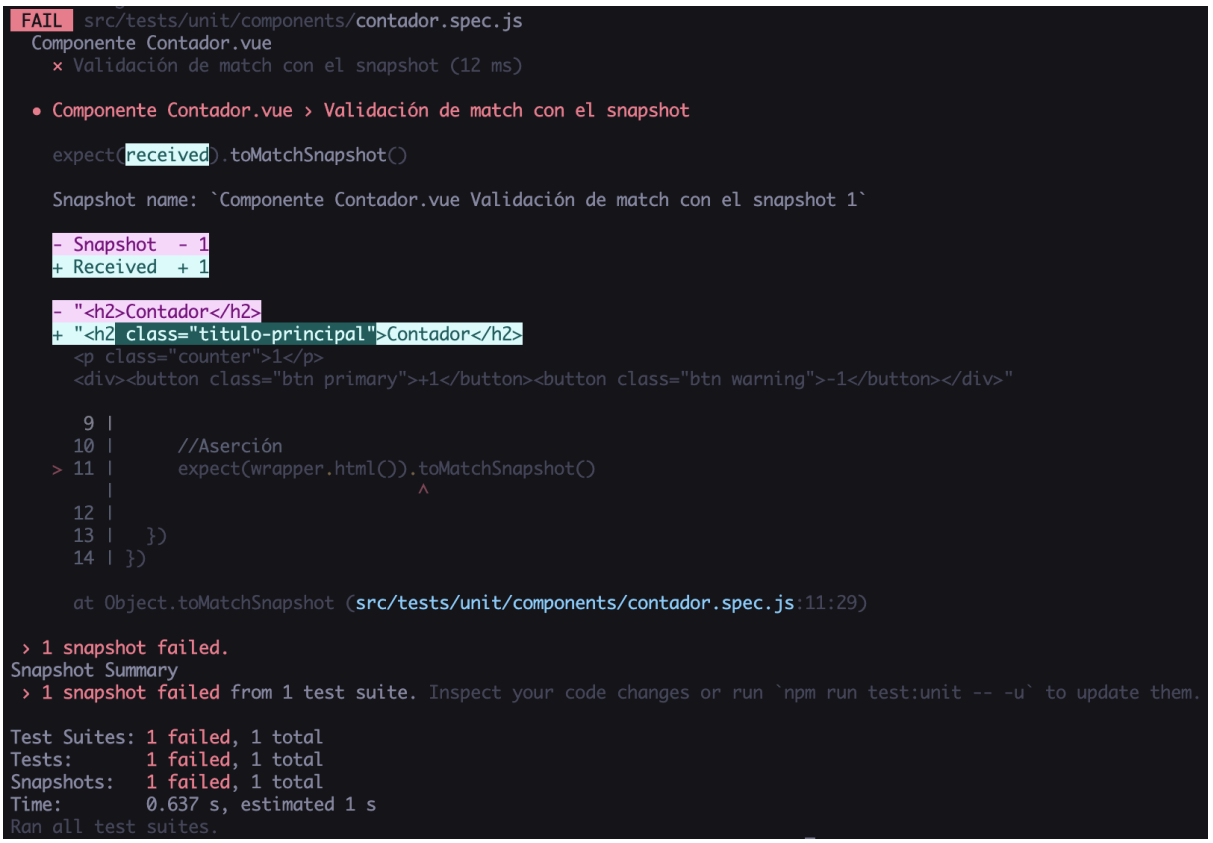


Imagen 4. Prueba fallida
Fuente: Desafío Latam

La prueba falló dado que la versión inicial del componente en su estructura HTML ya no es la misma, dado que incorporamos una clase nueva al título h2.

¡Importante! En el caso hipotético de que estos cambios que implementamos sean válidos, podemos utilizar el comando `npm run test:unit -- -u` para actualizar el snapshot y seguidamente nuestra prueba pasará correctamente.

Supongamos que ahora necesitamos validar la presencia de textos dentro de un elemento HTML en el componente. Para ello utilizaremos el método `.text()`.

- **Paso 6:** Generemos un botón en el componente `Contador.vue` y validaremos que el botón tenga el texto “Enviar”.

```
describe('Componente Contador.vue', () => {
    test('Validar el texto de un botón de enviar formulario', () => {
    const wrapper = shallowMount(Contador)
```

```
    const botonEnviar = wrapper.find('.boton-enviar')
    //Con este console.log comprobamos que estamos accediendo al texto del
elemento
    console.log(botonEnviar.text());

    expect(botonEnviar.text()).toBe("Enviar")
   })
})
```

Al ejecutar la prueba veremos el siguiente resultado en la consola.

```
> pruebas-unitarias-snapshot@0.0.0 test:unit
> jest

  console.log
    Enviar

      at Object.log (src/tests/unit/components/contador.spec.js:11:13)

 PASS  src/tests/unit/components/contador.spec.js
  Componente Contador.vue
    ✓ Validar el texto de un botón de enviar formulario (23 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.639 s, estimated 1 s
Ran all test suites.
```

Imagen 5. Comprobando presencia de textos en elementos HTML
Fuente: Desafío Latam

En el código utilizamos los siguientes aspectos:

1. Primero buscamos el elemento según su clase `const botonEnviar = wrapper.find('.boton-enviar')`
2. Luego hacemos la aserción donde esperamos que el texto de dicho botón sea "Enviar".

Este tipo de validaciones es muy útil si deseamos asegurar que la información a mostrar en los elementos HTML cumpla con lo que esperamos.

Otro ejemplo de validación de textos en un componente App donde exista una h1 con el texto "Hello World".

```
describe('App', () => {
```

```
  const wrapper = shallowMount(App);

  it('should have the correct title', () => {
    expect(wrapper.find('h1').text()).toBe('Hello World!');
  });
});
```

Simulación de eventos en un componente

Con Jest podemos simular eventos, para ello utilizaremos el método `.trigger()` donde verificaremos el comportamiento de los botones de incrementar y disminuir del componente Contador.

- **Paso 7:** Añadimos la lógica con el siguiente código:

```
describe('Componente Contador.vue', () => {
    test('Incrementar en 1 el valor del contador', async ()=> {
     const wrapper = shallowMount(Contador)
     const botonIncrementar = wrapper.find('button');

     botonIncrementar.trigger('click')

     const valor = wrapper.find('.incrementar').text()

     expect(valor).toBe('+1')
    })
 })
```

1. Guardamos en una variable la búsqueda del elemento tipo botón y le decimos que dispare el evento click `const botonIncrementar = wrapper.find('button').trigger('click')`
2. Seguidamente, capturamos el valor del botón con la clase incrementar y accedemos a su contenido con el método `.text()`.
3. Por último esperamos que al darle click al botón incrementar, su valor esperado es de “+1”.

Nota: Nuestro test en acciones definidas en el componente es implementado a través de un proceso asíncrono, recordemos que cuando trabajamos con propiedades reactivas, se ejecuta una actualización del DOM y luego se muestra la información. En este proceso existe un lapso de tiempo, es por ello que al momento de realizar test en métodos y funciones que modifiquen el DOM es necesario trabajarlo con async y await.

Resumen del ejercicio

Con este ejercicio aplicamos algunas técnicas de comprobación y pruebas en un componente Vue JS. Los métodos que provee Jest son muy útiles en casos específicos y nos ayudarán a ejecutar pruebas que vayan acorde a las necesidades funcionales de nuestros componentes.

Actividad guiada: Pruebas en Vue router

A continuación, realizaremos un ejercicio donde ejecutaremos pruebas en una aplicación Vue JS que utilice Vue router.

- **Paso 1:** Generamos una aplicación usando `create-vue` con el nombre `router-test`.
- **Paso 2:** En los pasos de instalación, seleccionamos la opción de Vue router
- **Paso 3**: Instalamos dependencias y hacemos la configuración de Jest.
- **Paso 3:** Una vez creada la aplicación definimos las siguientes vistas en el archivo `index.js`.

```
import { createRouter, createWebHistory } from 'vue-router'
import HomeView from '../views/HomeView.vue'

const router = createRouter({
 history: createWebHistory(import.meta.env.BASE_URL),
 routes: [
   {
     path: '/',
     name: 'home',
     component: HomeView
   },
   {
     path: '/posts',
     name: 'posts',
     component: () => import('../views/PostsView.vue')
   }
  ]
})

export default router
```

- **Paso 4:** Agregamos el código de la vista `PostsView.vue`.

```
<template>
  <div class="posts">
```

```
    <h1>Vista de Posts</h1>
    <ul>
      <li v-for="post in posts" :key="post.id">
        {{ post.name }}
      </li>
    </ul>
  </div>
</template>

<script>
  export default {
    name: 'PostsViewVue',
    data() {
      return {
        posts: [
          { id: 1, name: 'Post 1' },
          { id: 2, name: 'Post 2' },
          { id: 3, name: 'Post 3' },
          { id: 4, name: 'Post 4' }
        ]
      }
    }
  }
</script>
```

Esta vista está haciendo un render de un arreglo de Posts que utilizamos solo de ejemplo para mostrar información.

- **Paso 5:** En la vista `HomeView`, mostraremos la información del componente `PostsView`.

```
<template>
  <div class="home">
    <img alt="Vue logo" src="@/assets/logo.svg">
    <PostsView />
  </div>
</template>

<script>
import PostsView from './PostsView.vue';

// @ is an alias to /src

export default {
```

```
  name: 'HomeView',
  components: {
    PostsView
}
}
</script>
```

- **Paso 6:** Luego, nos dirigimos al directorio de `tests/unit` y creamos una nueva carpeta llamada `/views`. Seguidamente, creamos un archivo llamado `postView.spec.js`.

Configuración del router en el entorno de pruebas

Para ejecutar pruebas con Vue router debemos realizar las siguientes acciones:

1. Crear un enrutador local con todas aquellas rutas que deseamos testear.
2. Indicar la ruta a la cual deseamos navegar
3. Verificar que la navegación haya sido exitosa, esto lo realizaremos haciendo el check de la existencia del componente que se encarga de hacer el render.

- **Paso 7:** Importamos las dependencias necesarias para crear el sistema de enrutamiento y el `mount` para el montaje del componente o vista que vamos a testear.

```
import { mount } from '@vue/test-utils'
import { createRouter, createWebHistory } from 'vue-router'

import PostsView from '@/views/PostsView.vue'
```

- **Paso 8:** Luego, empezamos con la estructura de la prueba a realizar con el siguiente código.

```
describe('PostsView', () => {
    test('Probando la existencia del componente o vista PostsView ', async
() => {
        const router = createRouter({
            history: createWebHistory(),
            routes: [{
                path: '/posts',
                name: 'PostsViewVue',
                component: PostsView
            }],
        })
```

```
        router.push('/posts')
        await router.isReady()

        const wrapper = mount(PostsView, {
            global: {
                plugins: [router]
            }
        })
        expect(wrapper.findComponent(PostsView).exists()).toBe(true)
    })
})
```

Entendamos el código

1. Primero estamos creando el router y definiendo la ruta y componente que mostrará la información.

```
const router = createRouter({
            history: createWebHistory(),
            routes: [{
                path: '/posts',
                name: 'PostsViewVue',
                component: PostsView
            }],
        })
```

2. Luego, hacemos la redirección con el `.push` y dado que este tipo de peticiones toman un tiempo, declaramos que a través del await esperemos se haga la carga del DOM.

```
router.push('/posts')
        await router.isReady()
```

   Recuerda que para que el await funcione, se debe declarar que el test es asíncrono.

3. Seguidamente, seleccionamos el objeto de prueba, en este caso el `PostsView`. Cabe destacar que, debemos definir la configuración del enrutador local durante el montaje, esto se logra a través del siguiente código.

```
const wrapper = mount(PostsView, {
            global: {
```

```
            plugins: [router]
        }
    })
```

4. Por último, realizamos la aserción, donde primero hacemos la búsqueda del componente y luego validamos que la existencia sea true.

```
expect(wrapper.findComponent(PostsView).exists()).toBe(true)
```

Resumen del ejercicio

De esta manera, hemos realizado una prueba simple sobre vue router para verificar el comportamiento y la existencia de un componente que controla una ruta en específico. **Recordemos que para ejecutar este tipo de test, tuvimos que crear un ejemplo de router local en el archivo de pruebas**.

¡Manos a la obra! - Prueba unitaria parte I

Utilizando la aplicación de la " Pruebas unitarias snapshot", implementa una prueba que realice y valide la disminución del contador en 1.

¡Manos a la obra! - Prueba unitaria parte II

Utilizando la aplicación de la " Pruebas unitarias snapshot", implementa una prueba que genere un snapshot de un nuevo componente llamado Users.vue. En este componente existirá una lista ficticia de usuarios.