

# Principios de Testing y seguridad

## Actividad 1

En el desarrollo de esta actividad está dividida en 2 set de pruebas

Para el primer caso se realizaron pruebas funcionales para distintos casos en un código de prueba, utilizando comprobación de funciones y callbacks

Estas pruebas se realizaron en un entorno de node utilizando Tyescript y Jest

En la segunda parte se documentan los test realizado en React usando la librería

Prueba de funcionalidades React Test Library y Vitest

Primeramente, definiremos un función que entregue un strings random de un arreglo de strings.

app.ts

```
src > app.ts > default
1
2  const cities = [ 'Santiago' , 'Temuco', 'Concepcion ', 'Antofagasta' ,' Castro' , 'Coquimbo'];
3
4  const randomString = ( ) => {
5    const city = cities[Math.floor(Math.random() * cities.length)];
6    return city;
7  };
8
9  export default randomString;
```

App.test.ts

Comprobaremos la funcionalidad app.ts de randomString , y además comprobaremos que no entregue un determinado dato, en este caso una ciudad que no existe en el array de origen

```
src > app.test.ts > ...
1  import randomString from "./app";
2
3
4  describe("Probar funcionalidades", () => {
5    test("Probar la funcionalidad", () => {
6      expect(typeof randomString()).toBe("string");
7    });
8
9    test("comprobar que no existe una ciudad", () => {
10     expect(randomString()).not.toMatch(/Iquique/);
11   });
12 });
13 |
```

Ejecutamos las pruebas para este primer set con Jest

```
patricio@patricio-Hyper-v:~/Documents/Master/testing /src$ npx jest app.test.ts
PASS ./app.test.ts (7.644 s)
  Probar funcionalidades
    ✓ Probar la funcionalidad (4 ms)
    ✓ comprobar que no existe una ciudad (1 ms)

-----|-----|-----|-----|-----|-----
File    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files|     100 |     100 |     100 |     100 |
  app.ts|     100 |     100 |     100 |     100 |
-----|-----|-----|-----|-----|-----
Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        7.934 s, estimated 9 s
Ran all test suites matching /app.test.ts/i.
patricio@patricio-Hyper-v:~/Documents/Master/testing /src$
```

Para este set de pruebas realizaremos test sobre las funciones definidas en functions.ts

```
src > functions.ts > ...
1  // Ejemplo de archivo de funciones en TypeScript
2
3  export function sum(a: number, b: number): number {
4    if (typeof a !== 'number' || typeof b !== 'number') {
5      return NaN;
6    }
7    return a + b;
8  }
9
10 export function reverseString(str: string): string {
11   if (typeof str !== 'string') {
12     throw new Error('El argumento debe ser una cadena');
13   }
14   return str.split('').reverse().join('');
15 }
16
17 export function isUpperCase(str: string): boolean {
18   return str === str.toUpperCase();
19 }
20
21 export function isLowerCase(str: string): boolean {
22   return str === str.toLowerCase();
23 }
24
```

Las pruebas se organizan en bloques usando la función describe de Jest para cada función, y dentro de cada bloque, hay pruebas que comprueban si las funciones se comportan correctamente en diferentes situaciones.

Las funciones que se prueban son sum, reverseString, isUpperCase y isLowerCase.

En resumen, las pruebas verifican si:

- Sum: suma correctamente dos números y devuelve el resultado esperado, y devuelve NaN si se pasa un argumento que no es un número.

- reverseString: devuelve la cadena invertida correctamente, y lanza una excepción si se pasa un argumento que no es una cadena.
- isUpperCase: devuelve true si la cadena está en mayúsculas, y false si no lo está.
- isLowerCase: devuelve true si la cadena está en minúsculas, y false si no lo está.

```
src > fuctions.testits > describe('Funciones') callback
1 // Ejemplo de archivo de pruebas en Jest para funciones en TypeScript
2
3 import { sum, reverseString, isUpperCase, isLowerCase } from './functions';
4
5 describe('Funciones', () => {
6   describe('sum', () => {
7     it('debería sumar dos números y devolver el resultado', () => {
8       expect(sum(1, 2)).toBe(3);
9       expect(sum(-1, 1)).toBe(0);
10    });
11
12    it('debería devolver NaN si alguno de los argumentos no es un número', () => {
13      expect(sum(1, '2' as any)).toBeNaN();
14      expect(sum('foo' as any, 'bar' as any)).toBeNaN();
15    });
16  });
17
18  describe('reverseString', () => {
19    it('debería devolver la cadena al revés', () => {
20      expect(reverseString('hello')).toBe('olleh');
21      expect(reverseString('foo bar')).toBe('rab oof');
22    });
23
24    it('debería lanzar una excepción si se le pasa un argumento que no sea una cadena', () => {
25      expect(() => reverseString(123 as any)).toThrow();
26      expect(() => reverseString(['foo', 'bar'] as any)).toThrow();
27    });
28  });
29
30  describe('isUpperCase', () => {
31    it('debería devolver true si la cadena está en mayúsculas', () => {
32      expect(isUpperCase('HELLO')).toBe(true);
33      expect(isUpperCase('WORLD')).toBe(true);
34    });
35
36    it('debería devolver false si la cadena no está en mayúsculas', () => {
37      expect(isUpperCase('Hello')).toBe(false);
38      expect(isUpperCase('world')).toBe(false);
39    });
40  });
41
42  describe('isLowerCase', () => {
43    it('debería devolver true si la cadena está en minúsculas', () => {
44      expect(isLowerCase('hello')).toBe(true);
45      expect(isLowerCase('world')).toBe(true);
46    });
47
48    it('debería devolver false si la cadena no está en minúsculas', () => {
49      expect(isLowerCase('Hello')).toBe(false);
50      expect(isLowerCase('World')).toBe(false);
51    });
52  });
53
54 });
```

Ejecutamos jest sobre el archivo function.test.ts

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
patricio@patricio-Hyper-v:~/Documents/Master/testing /src$ npx jest functions.test.ts
PASS ./functions.test.ts (5.534 s)
  Funciones
    sum
      ✓ debería sumar dos números y devolver el resultado (7 ms)
      ✓ debería devolver NaN si alguno de los argumentos no es un número (1 ms)
    reverseString
      ✓ debería devolver la cadena al revés (1 ms)
      ✓ debería lanzar una excepción si se le pasa un argumento que no sea una cadena (45 ms)
    isUpperCase
      ✓ debería devolver true si la cadena está en mayúsculas (1 ms)
      ✓ debería devolver false si la cadena no está en mayúsculas (1 ms)
    isLowerCase
      ✓ debería devolver true si la cadena está en minúsculas (1 ms)
      ✓ debería devolver false si la cadena no está en minúsculas (1 ms)

-----|-----|-----|-----|-----|-----
File    | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |     100 |       100 |     100 |     100 |
functions.ts |     100 |       100 |     100 |     100 |
-----|-----|-----|-----|-----|-----
Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        5.971 s, estimated 7 s
Ran all test suites matching /functions.test.ts/i.
patricio@patricio-Hyper-v:~/Documents/Master/testing /src$
```

Finalmente en estas pruebas se comprueban diferentes aspectos de la funcionalidad de algunas funciones, como verificar si un valor es verdadero, falso o nulo, si un valor numérico es mayor que otro, si una cadena contiene ciertos caracteres, si una matriz contiene un elemento determinado, y si una función de devolución de llamada y una promesa funcionan correctamente.

Además, hay algunos métodos de ciclo de vida que se ejecutan antes o después de todas las pruebas y que imprimen mensajes en la consola.

```
src > global.test.ts > ...
1  const mensaje = "Hola Mundo";
2  const fruits = ["manzana", "pera", "uva"];
3
4  const reverseString = (str: string, callback: Function) => {
5    callback(str.split("").reverse().join(""));
6  };
7  const reverseString2 = (str: string) => {
8    return new Promise((resolve, reject) => {
9      if (!str) {
10        reject(Error("Error"));
11      }
12      resolve(str.split("").reverse().join(""));
13    });
14  };
15  test("debe contener un texto", () => {
16    expect(mensaje).toMatch(/Mundo/);
17  });
18  test("tiene una uva", () => {
19    expect(fruits).toContain("uva");
20  });
21  test("Mayor que", () => {
22    expect(10).toBeGreaterThan(9);
23  });
24  test("Verdadero", () => {
25    expect(true).toBeTruthy();
26  });
27  test("Falso", () => {
28    expect(false).toBeFalsy();
29  });
30  test("Nulo", () => {
31    expect(null).toBeNull();
32  });
33  test("Probando un Callback", () => {
34    reverseString("Hola", (str: String) => {
35      expect(str).toBe("aloH");
36    });
37  });
38  test("Probando una promesa", () => {
39    return reverseString2("Hola").then((string) => {
40      expect(string).toBe("aloH");
41    });
42  });
43  test("probar async/await", async () => {
44    const string = await reverseString2("Hola");
45    expect(string).toBe("aloH");
46  });
47  test
48  //afterEach(() => console.log("Despues de cada prueba"));
49  afterAll(() => console.log("Despues de todas las pruebas"));
50
51  beforeAll(() => console.log("Antes de todas las pruebas"));
52  //beforeEach(() => console.log("Antes de cada prueba"));
53  |
```

El resultado de las pruebas:

```
patricio@patricio-Hyper-v:~/Documents/Master/testing /src$ npx jest global.test.ts
console.log
  Antes de todas las pruebas

    at Object.<anonymous> (src/global.test.ts:50:25)

console.log
  Despues de todas las pruebas

    at Object.<anonymous> (src/global.test.ts:48:24)

PASS ./global.test.ts (6.309 s)
  ✓ debe contener un texto (9 ms)
  ✓ tiene una uva (1 ms)
  ✓ Mayor que (1 ms)
  ✓ Verdadero (10 ms)
  ✓ Falso (1 ms)
  ✓ Nulo (1 ms)
  ✓ Probando un Callback (1 ms)
  ✓ Probando una promesa (8 ms)
  ✓ probar async/await (1 ms)

-----|-----|-----|-----|-----|-----
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |      0  |      0   |      0   |      0   |
-----|-----|-----|-----|-----|
Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        6.595 s, estimated 8 s
Ran all test suites matching /global.test.ts/i.
patricio@patricio-Hyper-v:~/Documents/Master/testing /src$ +
```

## Pruebas en React

Para realizar las pruebas de testing en react utilizamos la librería de vitest junto con @testing-library/React

Para este caso crearemos un componente tipo accordion del cual probaremos su funcionalidad con una serie de test,

Es este caso Vitest nos ofrece un entorno de desarrollo para la ejecución de los test

Componente accordion.tsx

Este es un componente de React llamado Accordion que recibe dos propiedades:

- title: el título que se muestra en la interfaz de usuario.
- children: el contenido que se oculta o se muestra cuando se hace clic en el botón de "Abrir".

El componente utiliza el hook useState de React para mantener el estado de open, que indica si el contenido debe mostrarse o no.

La función Accordion devuelve un elemento div que contiene el título y un botón de "Abrir" que muestra u oculta el contenido cuando se hace clic. El botón utiliza una expresión ternaria para cambiar el texto que se muestra según si el contenido está abierto o cerrado. El contenido se muestra si la variable open es verdadera, de lo contrario no se muestra.

En resumen, este es un componente simple que se utiliza para crear un acordeón en la interfaz de usuario, permitiendo al usuario mostrar u ocultar el contenido.

```

src > components > Accordion.tsx > ...
1  import React from "react";
2  import { useState } from "react";
3
4  type AccordionProps = {
5    title: string;
6    children: React.ReactNode;
7  };
8
9  function Accordion({ title, children }: AccordionProps) {
10   const [open, setOpen] = useState(false);
11
12   return (
13     <div>
14       <div>
15         <h3>{title}</h3>
16         <button onClick={() => setOpen(!open)}>
17           {open ? "Close" : "Open"}
18         </button>
19       </div>
20       {open ? children : null}
21     </div>
22   );
23 }
24
25 export default Accordion;
26

```

#### Accordion.test.tsx

Este archivo de prueba se utiliza para probar la funcionalidad del componente Accordion en React, asegurándose de que el título y el contenido se muestren y oculten correctamente en respuesta a los clics del usuario.


La función describe establece un grupo de pruebas que se enfocan en el componente Accordion. Dentro de describe, la función beforeEach se llama para establecer el componente Accordion con un título y algunos elementos de contenido de ejemplo utilizando la función render de @testing-library/react. Esto se hace antes de cada prueba en este grupo.

Las pruebas en este archivo de prueba son:

- La prueba "should show the title all time" verifica si el título se muestra todo el tiempo en el componente Accordion.
- La prueba "should not show the content by default" verifica si el contenido está oculto por defecto en el componente Accordion.
- La prueba "should show the content when the button is clicked" verifica si el contenido se muestra cuando se hace clic en el botón en el componente Accordion.
- La prueba "should hide the content when the button is clicked twice" verifica si el contenido se oculta cuando se hace clic dos veces en el botón en el componente Accordion.

En cada prueba, se usa la biblioteca @testing-library/react para buscar elementos en la interfaz de usuario y simular eventos del usuario utilizando la función fireEvent. Las afirmaciones se realizan utilizando las funciones de comparación proporcionadas por Jest y @testing-library/react, como expect, toBe, toBeDefined, toBeTruthy, toBeNull, queryByText y getByText.



src > components >  Accordion.test.tsx > ...

```
1
2 import { render, screen ,fireEvent } from "@testing-library/react";
3 import Accordion from "../Accordion";
4
5
6 describe("Accordion", () => {
7   beforeEach(() => {
8     render(
9       <Accordion title="hello">
10         <h3>My </h3>
11         <p>some content</p>
12       </Accordion>
13     );
14   });
15 });
16
17
18 test("should show the title all time ", () => {
19   expect ( screen.getByText("hello")).toBeDefined();
20 })
21
22 test("should not show the content by default", () => {
23   expect(screen.queryByText(/content/i)).toBeNull();
24 })
25
26 test("should show the content when the button is clicked", () => {
27   const button = screen.getByText(/open/i);
28   fireEvent.click(button);
29   expect(screen.queryByText(/content/i)).toBeTruthy();
30   console.log(screen.queryByText(/content/i));
31 })
32
33 test("should hide the content when the button is clicked twice", () => {
34   const button = screen.getByText(/open/i);
35   fireEvent.click(button);
36   fireEvent.click(button);
37   expect(screen.queryByText(/content/i)).toBeNull();
38 })
39
40 });
41
```

Vitest, nos permite un entorno de pruebas automatizado que incluso nos proporciona una interface web para interactuar con el código de testing.

**RERUN** src/components/Accordion.test.tsx x1

✓ src/components/Accordion.test.tsx (4)

Test Files **1 passed** (1)

Tests **4 passed** (4)

Start at 03:25:00

Duration 791ms

**PASS** Waiting for file changes...  
press **h** to show help, press **q** to quit



## Render de Vitest en navegador

The screenshot displays the Vitest test runner interface in a web browser. The interface is organized into three main sections:

- Left Sidebar:** Shows the file explorer with the file `src/components/Accordion.test.tsx` selected. It indicates that the tests passed, showing **PASS (1)**.
- Middle Panel:** Displays a list of test results for the `Accordion` component. The tests are as follows:
  - ✓ `Accordion` 168ms
  - ✓ `should show the title all time` 84ms
  - ✓ `should not show the content by default` 15ms
  - ✓ `should show the content when the button is clicked` 35ms
  - ✓ `should hide the content when the button is clicked twice` 22ms
- Right Panel:** Shows the source code of the `Accordion.test.tsx` file. The code includes imports for `render`, `screen`, and `fireEvent` from `@testing-library/react`, and `Accordion` from `../Accordion`. It defines a `describe` block for the `Accordion` component, with `beforeEach` hooks for rendering and `test` cases for each of the five assertions listed in the middle panel.