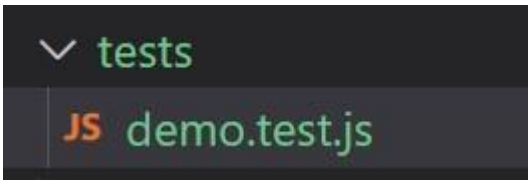
Pruebas unitarias o de integración

Descargar repositorio para los ejercicios

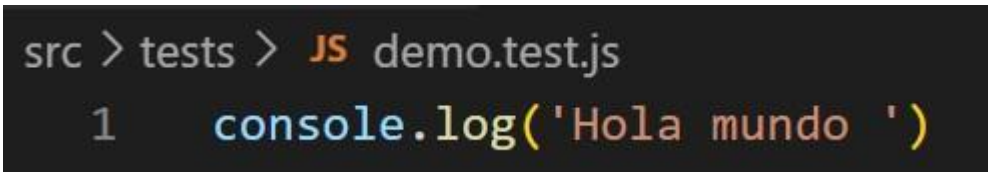
https://github.com/Garcia091/Ejercicios_Test

instalar las dependencias `npm install`

Crear la carpeta de trabajo

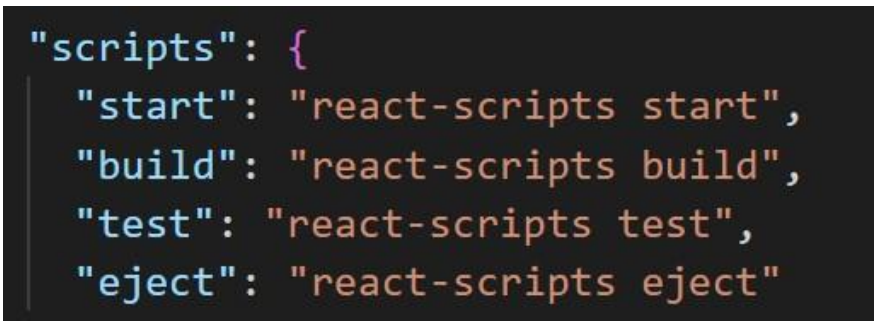


En el archivo demo.test.js escribir un Hola mundo



Ejecutar el archivo de pruebas con el comando **npm run test**

Cuando ejecutamos este comando estamos haciendo referencia a react-scripts



Este error que obtenemos es intensional, se ejecuta muestra el mensaje de Hola mundo, pero no encuentra ninguna prueba para ejecutar.

Pruebas unitarias o de integración

```
> counter-app@0.1.0 test C:\Users\silvi\Documents\Agile_Innova_2021\Contenido_Frontend\Testing\Sesion_Uno\rea
 FAIL  src/tests/demo.test.js
  ● Test suite failed to run

    Your test suite must contain at least one test.

      at node_modules/@jest/core/build/TestScheduler.js:242:24
      at asyncGeneratorStep (node_modules/@jest/core/build/TestScheduler.js:131:24)
      at _next (node_modules/@jest/core/build/TestScheduler.js:151:9)
      at node_modules/@jest/core/build/TestScheduler.js:156:7
      at node_modules/@jest/core/build/TestScheduler.js:148:12
      at onResult (node_modules/@jest/core/build/TestScheduler.js:271:25)

  console.log src/tests/demo.test.js:1
    Hola mundo
```

Si realizamos alguna modificación del archivo podemos ejecutar los siguientes comandos

```
Watch Usage
 › Press a to run all tests.
 › Press f to run only failed tests.
 › Press q to quit watch mode.
 › Press p to filter by a filename regex pattern.
```

A: Ejecutar todas las pruebas.
F: ejecutar solo las pruebas que fallaron.
Q: salir
P: Filtrar por el nombre de una expresión regular.

Mi primera Prueba:

Estructuras de una prueba

```
test('should ', () => {

})
```

Con solo esta estructura ya reconocer como la ejecución exitosa de una prueba

```
 PASS  src/tests/demo.test.js
  √ should  (1ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        4.178s
Ran all test suites related to changed files.
```

Pruebas unitarias o de integración

Ejemplo:

```
src > tests > JS demo.test.js > ...
1
2  test('Hola clase, el Ejemplo deber ser verdadero ', () => {
3
4      const isActive = true;
5
6      if(isActive){
7          throw new Error('No esta activo')
8      }
9  })
```

```
● Hola clase, el Ejemplo deber ser verdadero

  No esta activo

    5 |
    6 |     if(isActive){
  > 7 |         throw new Error('No esta activo')
      |               ^
    8 |     }
    9 | })
   10 |

      at Object.<anonymous> (src/tests/demo.test.js:7:15)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
Snapshots:   0 total
Time:        4.894s
Ran all test suites related to changed files.
```

Test Sweet

Ejercicio #1:

Comparar mensaje1 con mensaje2

```
src > tests > JS demo.test.js > ...
1
2   describe('Pruebas demo.test.js', () => {
3
4       test('los string deben ser iguales', () => {
5
6           //Arrange
7           const mensajeUno = 'Hola Mundo'
8           const mensajeDos = 'Hola Clase'
9
10          //Assert
11          expect(mensajeUno).toBe(mensajeDos); // ===
12      })
13
14  })
```

```
FAIL src/tests/demo.test.js
  Pruebas demo.test.js
    × los string deben ser iguales (6ms)

  ● Pruebas demo.test.js › los string deben ser iguales

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

    Expected: "Hola Clase"
    Received: "Hola Mundo"

      9 |
     10 |         //Assert
   > 11 |         expect(mensajeUno).toBe(mensajeDos); // ===
          |                            ^
     12 |     })
     13 |
     14 | })

      at Object.<anonymous> (src/tests/demo.test.js:11:28)

Test Suites: 1 failed, 1 total
Tests:       1 failed, 1 total
```

Ejercicio #2:

Importar la librería jest-dom para digitar los comandos de forma más rápida.

```
import '@testing-library/jest-dom'
```

Realizar la prueba del archivo 02-template-string de la carpeta base.

src > base > JS 02-template-string.js > ...

```
1
2  export function getSaludo(nombre) {
3      return 'Hola ' + nombre;
4  }
```

Se desea validar que string de hola mundo con la variable que se envía por parámetros.
Solucion:
Crear en la carpeta de pruebas la respectiva carpeta base y el archivo 02template-string.test.js
Crear la estructura de la prueba y su validación.

src > tests > base > JS 02-template-string.test.js > ...

```
1  import '@testing-library/jest-dom'
2  import {getSaludo} from '../../base/02-template-string'
3
4  describe('Template-string', () => {
5
6      const nombre = 'Carlos'
7      const saludo = getSaludo(nombre)
8
9      test('Verificar nombre ', () => {
10         expect(saludo).toBe('Hola ' + nombre)
11     })
12 })
```

Ejercicio #3:

Crear la validación del archivo funciones de la carpeta base. Nota: Para este ejercicio es necesario usar la propiedad toEqual, en la función getUsuarioActivo si el usuario no envía el nombre debe aparecer el nombre de Carlos.

src > base > JS 05-funciones.js > ...

```
1  const getUser = () => ({
2          uid: 'ABC123',
3          username: 'El_Papi1502'
4  });
5
6  // Tarea
7  const getUsuarioActivo = ( nombre ) =>({
8      uid: 'ABC567',
9      username: nombre
10 })
```

Solución:

1. Exportar las funciones para usarlas en el test.
2. Crear los archivos necesarios para la prueba.

En la siguiente solución la prueba es negativa de forma intencional, ya que se desea observar la funcionalidad de la propiedad toEqual.

```
import '@testing-library/jest-dom'
import { getUser, getUsuarioActivo } from '../../base/05-funciones'

describe('Funciones', () => {

    test('Valores iguales ', () => {
        expect(getUser()).toEqual({
            uid: 'ABC123',
            username: 'Silvia'
        })
    })
})
```

```
expect(received).toEqual(expected) // deep equality

- Expected
+ Received

  Object {
    "uid": "ABC123",
-   "username": "Silvia",
+   "username": "El_Papi1502",
  }

   5 |
   6 |     test('Valores iguales ', () => {
>  7 |         expect(getUser()).toEqual({
     |                           ^
   8 |             uid: 'ABC123',
   9 |             username: 'Silvia'
  10 |         })
```

Terminar el ejercicio…

Ejercicio #4: Validar que los elementos del vector del ejercicio 7 de la carpeta base sean letras y números. Adicional validar el contenido, para que sea igual al esperado.

```
const retornaArreglo = () =>{
    return ['ABC', 123];
}
```

Ejercicio #5: Realizar el filtro de los héroes por el id. Adicional consumir un objeto de héroes que permita traer la misma información y comparar entre los dos objetos si la información es igual.

Validar que el id del héroe exista.

src > tests > base > JS 08-imp-exp.test.js > ...

```
1   import {getHeroeById,getHeroesByOwner } from '../../base/08-imp-exp'
2   import '@testing-library/jest-dom'
3   import {heroes} from '../../data/heros'
4
5   describe('Pruebas con función heroes', () => {
6       test('Retornar id ', () => {
7
8           const id=2;
9           const personaje = getHeroeById(id);
10
11          //Hacer el find de los horoes
12          const heroeData = heroes.find((h) =>h.id === id);
13          expect(heroeData).toEqual(personaje)
14      })
15
16      test('Retornar undefined si heroe no existe ', () => {
17
18          const id=10;
19          const personaje = getHeroeById(id);
20          expect(personaje).toBe(undefined)
21      })
22  })
```

Ejercicio #6:

En la prueba del ejercicio anterior filtrar los héroes por el atributo owner para Marvel y DC. Adicional crear un nuevo array y comparar si el contenido es igual.

Pruebas con funciones asíncronas

Las pruebas asíncronas deben ser usadas cuando esperamos que el recurso a consumir requiera de tiempo para obtenerlo. Para esto utilizamos el **done** para indicar que la ejecución debe terminar.

src > base > JS 09-promesas.js > ...

```
1  import { getHeroeById } from './08-imp-exp'
2
3  export const getHeroeByIdAsync = ( id ) => {
4
5      return new Promise( (resolve, reject) => {
6          setTimeout( () =>  {
7              // Tarea
8              // importen el
9              const p1 = getHeroeById( id );
10             if ( p1 ) {
11                 resolve( p1 );
12             } else {
13                 reject( 'No se pudo encontrar el héroe' );
14             }
15         }, 2000 )
16     });
17 }
```

Solución:

src > tests > base > JS 09-promesas.test.js > ...

```
1   import {getHeroeByIdAsync} from '../../base/09-promesas'
2   import {heroes} from '../../data/heros'
3   import '@testing-library/jest-dom'
4
5   describe('Pruebas con promesas', () => {
6
7       test('Prueba héroe async ', (done) => {
8           const id=2;
9           getHeroeByIdAsync(id)
10          .then(h=>{
11              expect(h).toBe(heroes[0])
12              done()
13          })
14      })
15  })
```

Vamos a validar cuando un héroe no existe, se debe ejecutar “No se pudo encontrar el héroe”

```
test('Validar héroe no existente ', (done) => {
    const id =10;
    getHeroeByIdAsync(id)
    .catch(error=>{
        expect(error).toBe('No se pudo encontrar el héroe')
        done();
    })
})
```

Ejercicio recibiendo una url de una api con async - await

src > tests > base > JS 11-async-await.test.js > ...

```
1   import {getImagen} from '../../base/11-async-await'
2
3   describe('Validar url de API', () => {
4       test('retornar url', async() => {
5           const url = await getImagen()
6           expect(typeof url).toBe('string')
7   })
8   })
```

Validar si la url trae la https://

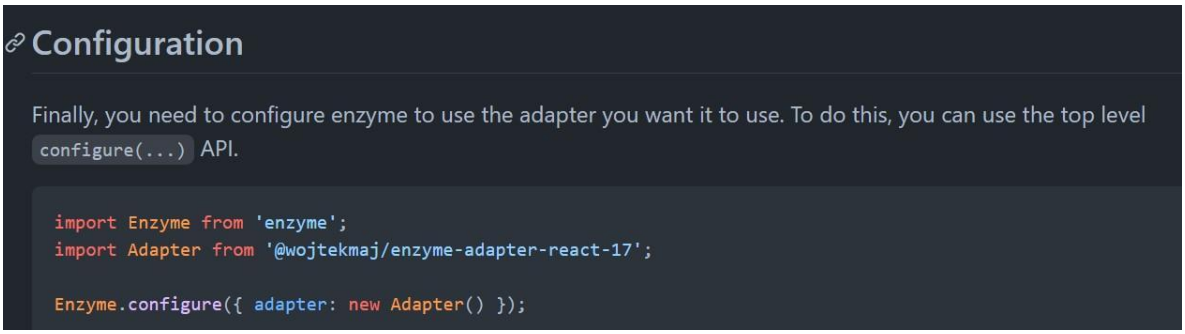
src > tests > base > JS 11-async-await.test.js > ...

```
1   import {getImagen} from '../../base/11-async-await'
2
3   describe('Validar url de API', () => {
4       test('retornar url', async() => {
5           const url = await getImagen()
6           console.log(url)
7           expect(url.includes('https://')).toBe(true)
8   })
9   })
```

Pruebas con componentes en React

1. Installación: https://enzymejs.github.io/enzyme/

Si la versión de su React es superior a la 17 debe realizar la respectiva instalación y configuración con el siguiente enlace:



https://github.com/wojtekmaj/enzyme-adapter-react-17

De lo contrario seguir instalación oficial de Enzyme.

```
npm i --save-dev enzyme enzyme-adapter-react-16
```

```
import Enzyme from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

Enzyme.configure({ adapter: new Adapter() });
```

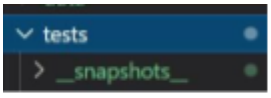
2. Crear un archivo setupTests.js en la raíz de la carpeta src y agregar la configuración del adaptador
3. Vamos a realizar la validación del componente de mi PrimeraApp.js 4. Crear el archivo para las pruebas y crear estructura de trabajo
5. **Shallow** nos permite controlar eventos de clic y hacer referencias parecidas al querySelector

```
src > tests > JS PrimeraApp.test.js > ...
1   import React from 'react'
2   import {shallow} from 'enzyme'
3   import PrimeraApp from '../PrimeraApp'
4
5   describe('pruebas <PrimeraApp/>', () => {
6       test('Validar render de <PrimeraApp/>', () => {
7           const saludoC = 'Hola clase'
8           const compo = shallow(<PrimeraApp saludo={saludoC} />)
9       })
10  })
```

6. Instalar **npm install --save-dev enzyme-to-json**
   https://www.npmjs.com/package/enzyme-to-json
7. Realizar las configuraciones pertinentes de la dependencia de enzyme-tojson

```
src > JS setupTests.js > ...
1   import Enzyme from 'enzyme';
2   import Adapter from 'enzyme-adapter-react-16';
3   import {createSerializer} from 'enzyme-to-json';
4
5   Enzyme.configure({ adapter: new Adapter() });
6   expect.addSnapshotSerializer(createSerializer({mode: 'deep'}));
```

8. El objetivo de esta prueba es validar la información que se va a renderizar en el navegador
9. Al agregar la función **toMatchSnapshot()** se crea una nueva carpeta




```
src > tests > JS PrimeraApp.test.js > ...
1   import React from 'react'
2   import {shallow} from 'enzyme'
3   import PrimeraApp from '../PrimeraApp'
4   import '@testing-library/jest-dom'
5
6   describe('pruebas <PrimeraApp/>', () => {
7       test('Validar render de <PrimeraApp/>', () => {
8           const saludoC = 'Hola clase'
9           const compo = shallow(<PrimeraApp saludo={saludoC} />)
10
11          expect(compo).toMatchSnapshot();
12
13      })
14  })
```

10. La nueva carpeta guarda la información del componente que queremos validar, su información del render se guarda como una fotografía y de esta manera podemos realizarle pruebas a su contenido. **Nota:** La carpeta de **__snapshots__ solo es de lectura.**
11. Con la tecla u podemos actualizar nuestros snapshots, dado el caso que el componente haya sufrido cambios.
12. Vamos a validar si el contenido del párrafo es igual al que le enviamos

src > tests > JS PrimeraApp.test.js > ...

```
1   import React from 'react'
2   import {shallow} from 'enzyme'
3   import PrimeraApp from '../PrimeraApp'
4   import '@testing-library/jest-dom'
5
6   describe('pruebas <PrimeraApp/>', () => {
7       test('Validar render de <PrimeraApp/>', () => {
8
9           const saludoC = 'Hola clase'
10          const subTitulo ='Soy un subtitulo'
11
12          const compo = shallow(
13              <PrimeraApp
14              saludo={saludoC}
15              subtitulo={subTitulo}
16               />)
17
18          const textParrafo = compo.find('p').text().trim();
19
20          expect(textParrafo).toBe(subTitulo)
21          expect(compo).toMatchSnapshot();
22      })
23  })
```

Pruebas unitarias o de integración

Ejercicio independiente:

1. Crear las siguientes pruebas para el <CounterApp/>
    - Debe de mostrar <CounterApp/> directamente(hacer un match con un snapshot) y sus valores por defecto
    - El contador debe mostrar como valor inicial 100. Usar la propiedad .find y validar en el html que se muestre el valor del contador(100)