**React**

**Pruebas en el archivo 02-template-string.js.**

Documentación: <https://es.reactjs.org/>

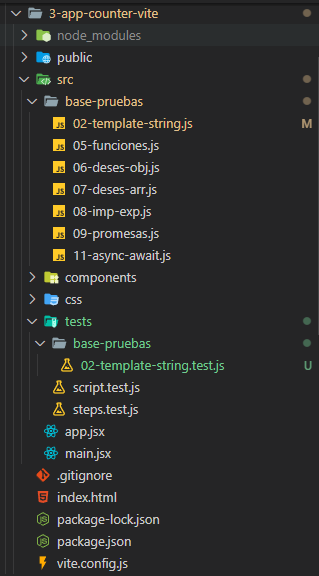
<https://es.reactjs.org/docs/getting-started.html>

<https://jestjs.io/>

<https://jestjs.io/docs/snapshot-testing>

<https://jestjs.io/docs/expect>

Al momento de realizar las pruebas a las funciones programadas, es importante mencionar que nosotros no vamos a poder realizar ninguna prueba a funciones que no están exportadas. Al momento de crear el archivo de prueba se acostumbra a tener la misma estructura de archivos en la carpeta de **“tests”** que se creó dentro de **“src”** imaginemos que la carpeta de **“tests”** es otra carpeta igual a **“src”** como un tipo espejo. Ejemplo:



Y antes de empezar a programar nuestro archivo de pruebas, vamos a instalar otra dependencia de desarrollo para poder correr bien las pruebas, tendremos que ejecutar el siguiente comando en la terminal dentro de nuestra carpeta de la aplicación: **npm install --save-dev babel-jest @babel/core @babel/preset-env**

Después de esto vamos a crear y configurar el archivo de **babel.config.cjs** agregándolo a nuestro proyecto en la carpeta raíz con el siguiente código dentro del archivo:

module.exports = {  
 presets: [['@babel/preset-env', {targets: {node: 'current'}}]],  
};

**Nota de actualización para la extensión de archivos.**

En la siguiente clase se hace la configuración para las pruebas de nuestros distintos archivos, en caso de ver el error **"You appear to be using a native ECMAScript module configuration file, which is only supported when running Babel asynchronously." Cambiar extensión de los archivos jest.config.js y babel.config.js a .cjs**

**Para ver más detalles al respecto pueden ir a:** [**https://nodejs.org/docs/latest/api/modules.html#enabling**](https://nodejs.org/docs/latest/api/modules.html#enabling)

**Vamos a realiza nuestra prueba.**

Al importar los archivos de código a los que le realizaremos las pruebas de las funciones en nuestro archivo de testing, cabe mencionar que esos archivos importados se van a ejecutar con todo su código en el archivo de prueba. Ejemplo:

**Código de la aplicación – Archivo 02-template-string.js**

const nombre = "Benjamín";

export function getSaludo (nombre) {

    return 'Hola ' + nombre;

}

console.log( `Este es un texto: ${ getSaludo( nombre ) }  ` );

**Código de prueba – Archivo 02-template-string.test.js**

import { getSaludo } from "../../base-pruebas/02-template-string"

describe('Pruebas en 02-template-string', () => {

    test('getSaludo debe retornar un "Hola Benjamín"', () => {

        const name = "Pedro";

        const message = getSaludo( name );

        expect( message ).toBe( `Hola ${ name }` )

    })

})