TEST UNITARIO: JEST

ANGELINA MAVERINO, AGUSTIN MARSICO, PEDRO BERTOLOTTI

PRUEBAS (TESTING)

El testing es el proceso utilizado para verificar y validar cada etapa de desarrollo sobre la que avanzamos en nuestro código fuente.

- DETECCIÓN TEMPRANA DE ERRORES
- MEJORA LA CALIDAD DEL CÓDIGO
- AUMENTO DE LA CONFIABILIDAD
- REDUCCIÓN DE RIESGOS
- FACILITA LA COLABORACIÓN
- MANTENIMIENTO MÁS SENCILLO
- DOCUMENTACIÓN VIVA
- MAYOR CONFIANZA EN LOS CAMBIOS
- CUMPLIMIENTO DE REQUISITOS

PRUEBAS UNITARIAS (UNIT TESTS)

Las pruebas unitarias funcionan descomponiendo funciones y/o procedimientos del programa con el fin de probar individualmente estas pequeñas unidades de código.

Para aislar funciones de código o unidades de código, los desarrolladores realizan **stubbing** que resumidamente es simular el comportamiento de un código existente o ser un sustituto temporal de un código aún por desarrollar.

Por otro lado, los usuarios deben definir las **expectativas** dentro de los casos de prueba y a medida que se ejecutan las pruebas unitarias, los valores de salida se pueden recopilar e inspeccionar para verificar que sean correctos.

F.I.R.S.T

Las pruebas unitarias deben seguir cinco reglas , cada una es parte de la sigla.

- FAST
- INDEPENDENT
- REPEATABLE
- SELF-VALIDATING
- TIMELY

TEST UNITARIO: JEST

Jest es un testing framework de JavaScript de código abierto diseñado para simplificar y agilizar el proceso de escritura y ejecución de pruebas.

- RAPIDEZ
- FACILIDAD DE USO
- POTENCIA
- FLEXIBILIDAD

CONCEPTOS CLAVES PARA LAS PRUEBAS CON JEST

Los siguientes conceptos nos ayudarán a entender la terminología y palabras reservadas que se utiliza Jest para su funcionamiento.

- **Describe**: Función que nos ayudará a crear bloques que agrupan varias pruebas relacionadas.

- **Test:** Función en donde se probará la casuística.

- **Expect:** Función que permite validar coincidencias durante la prueba. La función expect se utiliza cada vez que se necesite validar un valor.

JEST MATCHERS

Los matchers son las funciones utilizadas por el framework de test para comprobar si el valor esperado por la prueba automática coincide realmente con el obtenido

Ejemplos:

```
test('2 + 2 es igual a 4', () => {
 expect(2 + 2).toBe(4);
});
test('array contiene un 2', () => {
const array = [1, 2, 3];
expect(array).toContain(2);
});
```

LISTADO DE ALGUNOS DE LOS MATCHERS MÁS UTILIZADOS EN JEST

- **toBe(value):** Compara si el valor es idéntico (===) al valor esperado.
- **toEqual(value):** Compara si el valor es igual al valor esperado en términos de propiedades y valores de objetos o arrays.
- **toBeCloseTo(value,numDigits):**compara números de punto flotante con una precisión determinada.
- **toBeNull():** Comprueba si el valor es null.
- **toBeDefined():** Comprueba si el valor está definido.
- **toBeUndefined():** Comprueba si el valor es undefined.
- **toBeTruthy():** Comprueba si el valor es verdadero.
- **toBeFalsy():** Comprueba si el valor es falso.

- **toBeGreaterThan(value):** Comprueba si el valor es mayor que el valor esperado.
- **toBeLessThan(value):** Comprueba si el valor es menor que el valor esperado.
- **toContain(item):** Comprueba si un array o string contiene un elemento específico.
- **toHaveLength(length):** Comprueba si un array o string tiene la longitud esperada.
- **toMatch(pattern):** Comprueba si un valor coincide con un patrón de expresión regular.
- **toThrow(error?):** Comprueba si una función arroja una excepción.
- **toBeInstanceOf(constructor):** Comprueba si un valor es una instancia de una clase específica.

LISTADO DE ALGUNOS DE LOS MATCHERS MÁS UTILIZADOS EN JEST

- **toHaveBeenCalled():** Comprueba si una función (spy) se ha llamado al menos una vez.

- **toHaveBeenCalledTimes(count):** Comprueba si una función (spy) se ha llamado un número específico de veces.

- **toHaveBeenCalledWith(arg1, arg2, ...):** Comprueba si una función (spy) se ha llamado con argumentos específicos.

LISTADO DE ALGUNOS DE LOS MATCHERS MÁS UTILIZADOS EN JEST

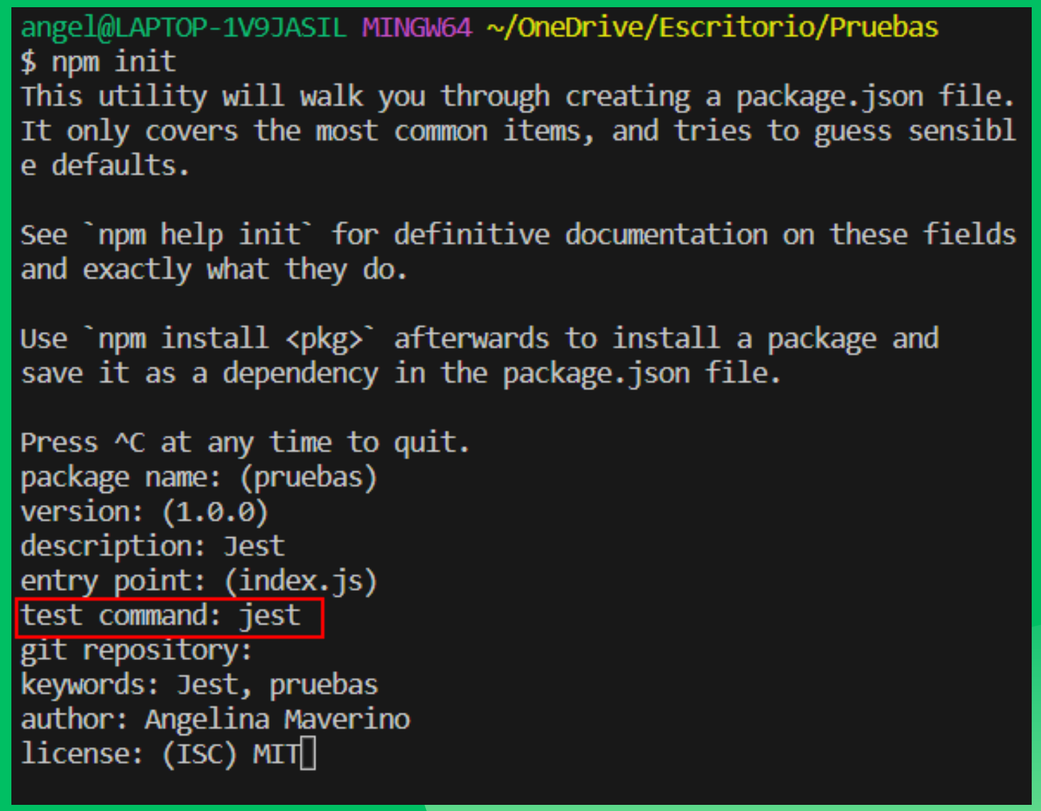
INSTALACIÓN DE JEST

Antes de comenzar, para realizar este ejemplo, **es necesario tener instalado Nodejs** en nuestro entorno de trabajo. Una vez instalado generamos una carpeta de nombre prueba para realizar el test de ejemplo.

Ejecutaremos el comando

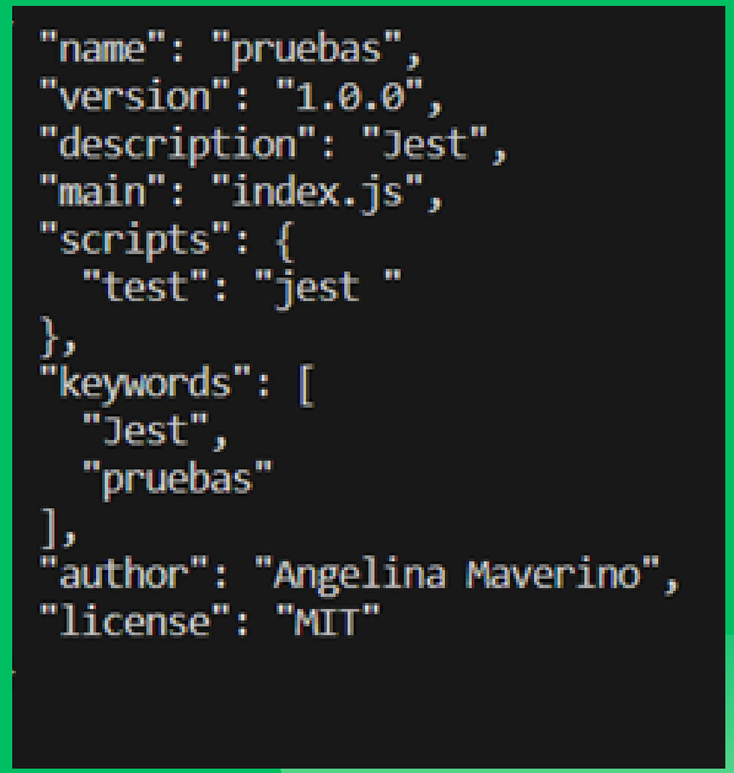
`npm init`

en nuestra terminal que generará el primer paquete json de nuestro proyecto, nos pedirá ingresar algunos datos, **donde en test command ingresaremos jest**.



INSTALACIÓN DE JEST

Damos enter e ingresamos yes, esto generara el archivo package.jason con toda la información registrada en nuestra terminal



```
"name": "pruebas",
"version": "1.0.0",
"description": "Jest",
"main": "index.js",
"scripts": {
  "test": "jest "
},
"keywords": [
  "Jest",
  "pruebas"
],
"author": "Angelina Maverino",
"license": "MIT"
```

INSTALACIÓN DE JEST

El siguiente paso instalaremos jest usando el gestor de paquetes de su preferencia (npm, Yarn o pnpm), en este caso utilizaremos nmp, ingresando en terminal el siguiente comando:

`npm install -save-dev jest`

Esperaremos que Jest se instale en nuestro sistema operativo.

```
angel@LAPTOP-1V9JASIL MINGW64 ~/OneDrive/Escritorio/Pruebas
$ npm install --save-dev jest

added 286 packages, and audited 287 packages in 42s

31 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

PRUEBA DE EJEMPLO

LINKS DE INTERÉS

- **PAGINA PRINCIPAL DE JEST**

  HTTPS://JESTJS.IO/ES-ES/

- **REPOSITORIO DE GITHUB DE JEST CON EJEMPLOS**

  HTTPS://GITHUB.COM/JESTJS/JEST

¡GRACIAS POR SU ATENCIÓN!

¿PREGUNTAS?