**Objetivo**

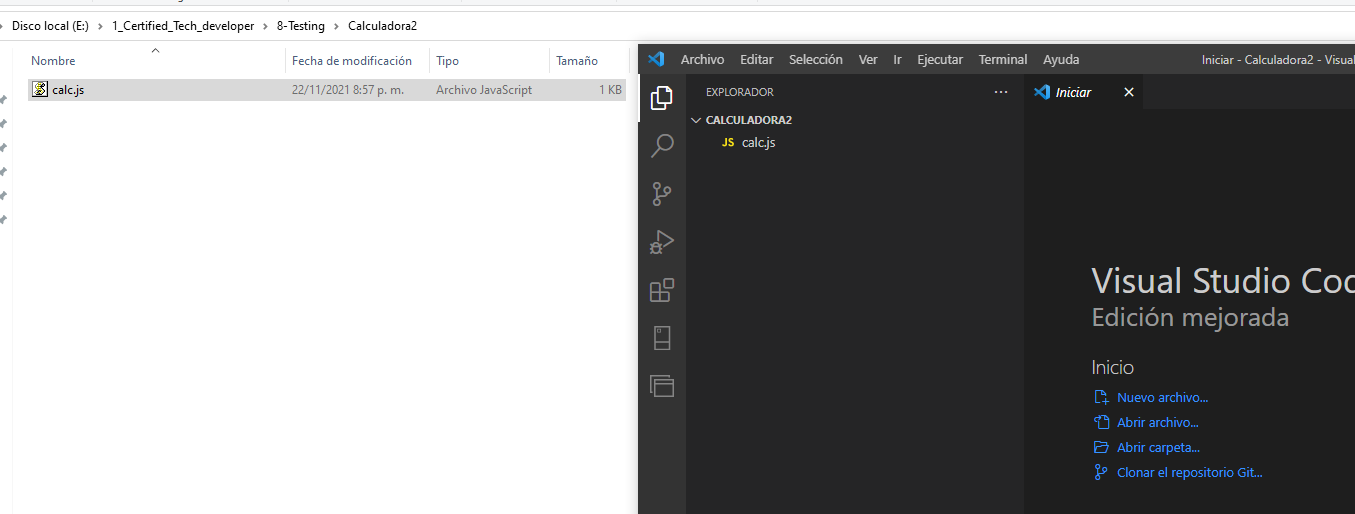
Realizar las pruebas unitarias de una calculadora hecha en javascript

**Paso 1**

Bajar el script al que realizaremos las pruebas[**Calc.js**](https://drive.google.com/drive/folders/16TJtup03ZQuPtoMvfX-ifqEroUpB4TBW)

**Paso 2**

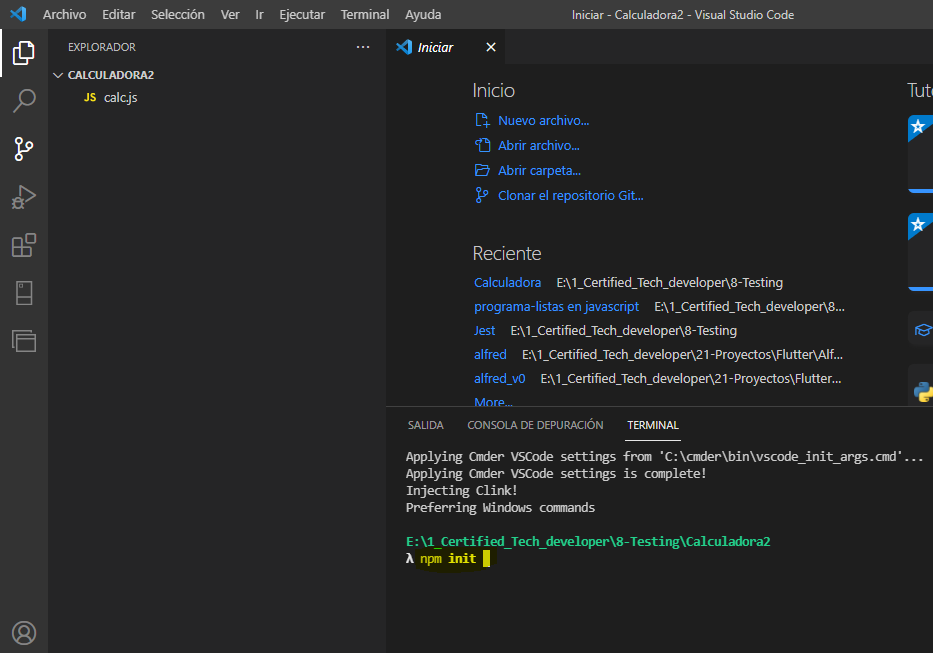
Crea una carpeta con el nombre que desees para la prueba, lleva allí el script de la calculadora, una vez hecho esto, abre visual studio code y abre la carpeta ( puedes usar **ctrl+O** y navegar hasta encontrar la carpeta que has creado)

****

**Paso 3**

En el visual studio y estando en la carpeta (desde ahora será este el proyecto en que trabajaremos) abriremos la terminal (**ctrl+Mayus+ñ** ) o en la barra superior menú---> Terminal. y digitamos **npm init -y** paracrear el package.json

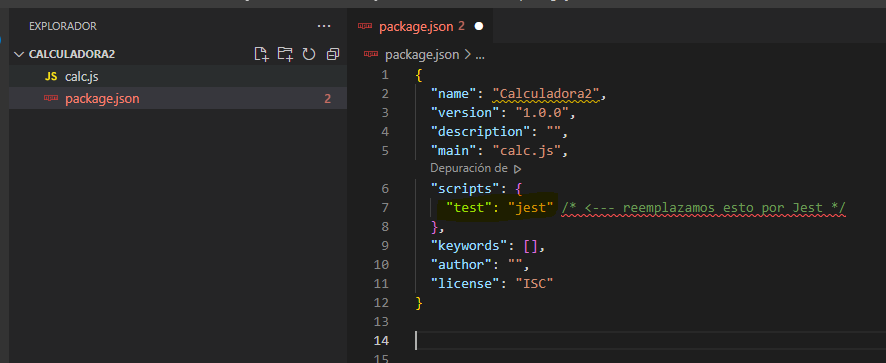
que tendrá la configuración de nuestro proyecto.



ahora ingresamos al .json y editamos la linea:

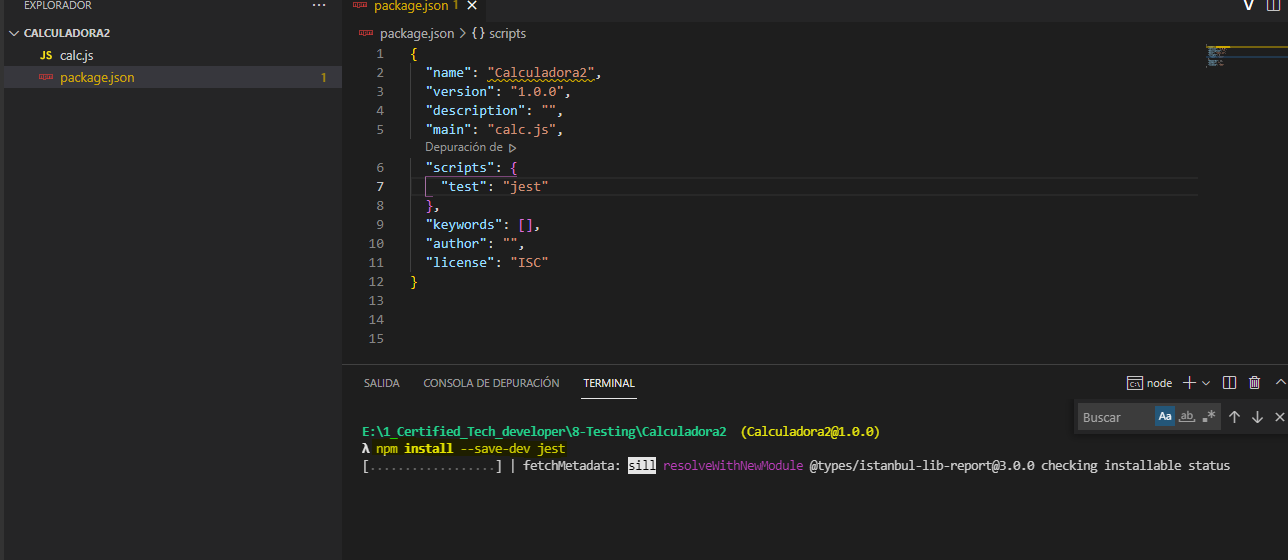
"test": "echo \"Error: no test specified\" && exit 1"

La cual debe quedar así:

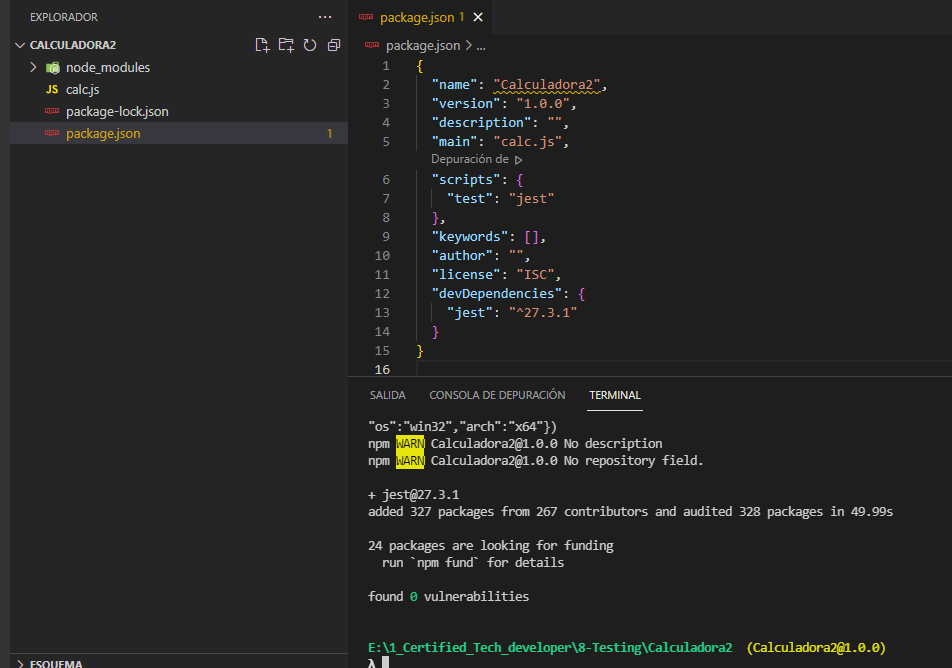


**Paso 4**

Abrimos nuevamente la terminal y digitamos **npm install --save-dev jest** para incluir Jest en el proyecto **(cuidado, si hay error en el .json no te permitirá continuar)**

****

**...**Una vez termine, habrá creado las siguientes carpetas y archivos:

****

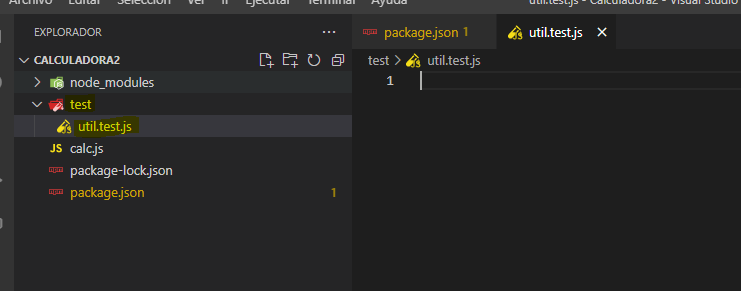
**Nota: Estos pasos se deben realizar en todos los proyectos en los que queramos hacer las pruebas unitarias.**

**Paso 5**

Ya casi terminamos de configurar…

Crearemos una carpeta de test y un archivo javascript de tipo test:

**util.test.js <--- Esta parte es importante, si no lleva .test.js Jest no lo reconocerá.**

****

**Paso 6**

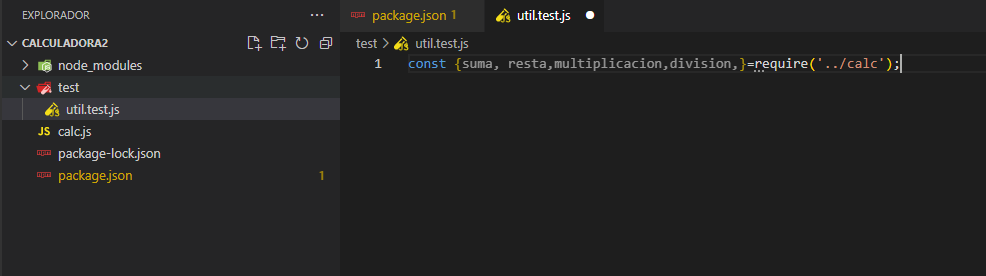
Ya casi …

En el archivo de test que hemos creado importamos el script al cual le realizaremos las pruebas, para esto digitamos:

const {aquí vas a poner los métodos o funciones a probar ,}=require('../acá el nombre del script');

Te debe quedar algo como esto, recuerda que en el **require** debes indicar el nivel de la carpeta donde está el script, en este caso está por fuera de la carpeta test, por lo tanto usamos dos puntos y slash ../Calc

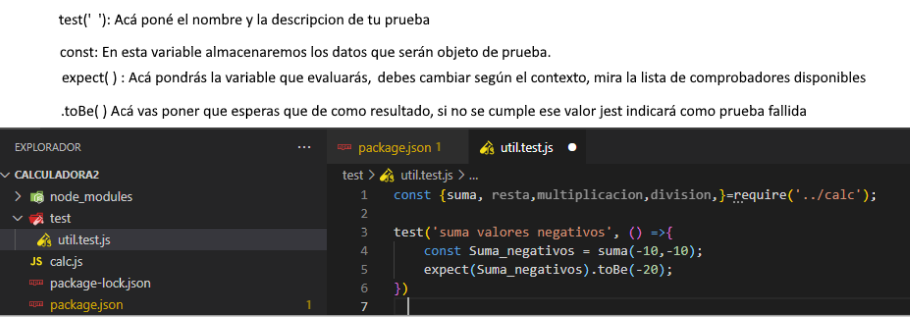
(cómo nos enseñó Vero **T\_T**)



**Paso 7**

¡Ya puedes empezar a jugar!

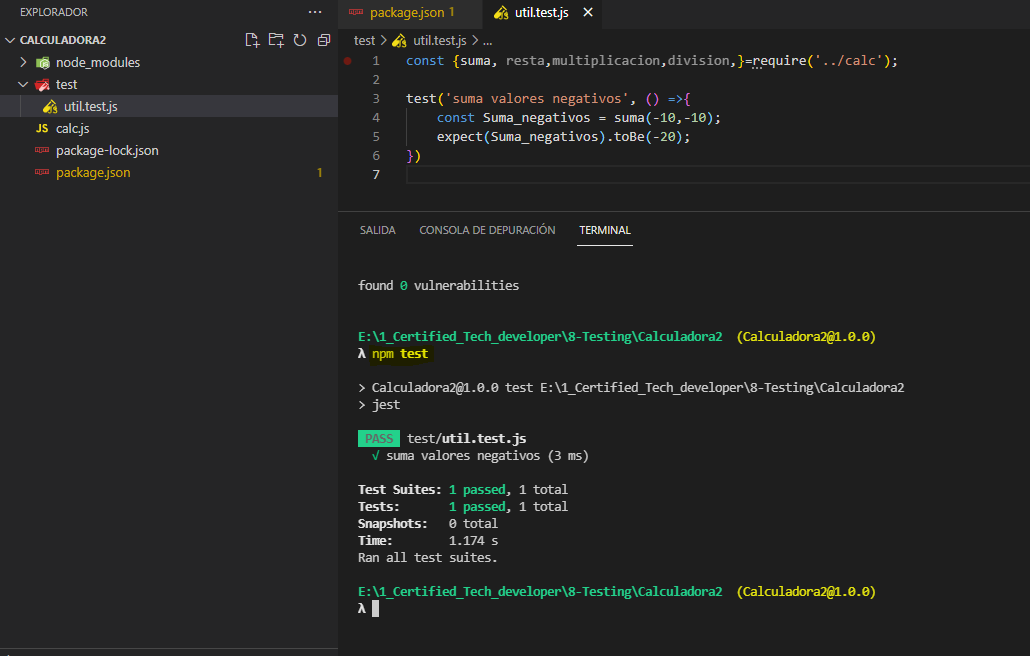
La estructura básica para la prueba será algo así:



Acá tenemos la lista de métodos [comparadores “expect](https://jestjs.io/docs/expect)” que utilizaremos según el contexto.

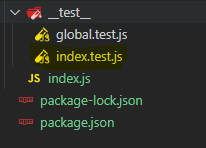
finalmente para ejecutar la prueba, nos vamos nuevamente a la terminal **ctrl+Mayus+ñ** y tipeamos **npm test** (**npm** es para decirle al node.js que nos ayude a ejecutar nuestro script)

Cuando se ejecute, tendrás algo como esto, donde se te muestra cuáles pruebas pasaron y cuáles no:



**TIPS**

**npm install -g jest :** Nos permite instalar esta funcionalidad que permite ejecutar solo una parte de nuestra suite de pruebas, ejemplo que estén en archivos diferentes. ejemplo: tipeamos en la terminal: **jest scr/\_\_test\_\_/index.test.js <- -**Solo ejecuta las pruebas de este archivo.



**npm install --save @types/jest** → para que el intellisense nos ayude a completar el código al momento de tipear.

**npm test -- --coverage :** crea la carpeta coverage

**test.only** → para correr un test específico y saltar los otros.

**test.skip** → para saltar un test específico.

**typeof→** Nos permite agregar al expect para validar que una variable tenga un tipo de dato específico ejemplo: expect (**typeof** (mivariable)).toBe (‘number’)

**describe**→ Permite agrupar los test.

En el package.json también podemos activar algunas flags de funcionalidades:

**"test:cobertura":** "jest --coverage", → nos muestra la cobertura y lo corremos en la consola asi npm run test:coverage

**"test:vigilar":** "jest --watchAll" → para que esté escuchando constantemente los cambios y pueda hacer las pruebas

**npm run test:watch** : Nos permite ser más rápidos al ejecutar los tests, estará siempre escuchando, por lo tanto solo debemos presionar alguna de las siguientes opciones:

› Press a to run all tests.

› Press f to run only failed tests.

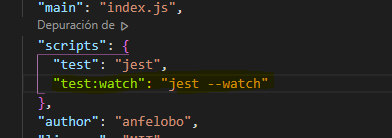
› Press p to filter by a filename regex pattern.

› Press t to filter by a test name regex pattern.

› Press q to quit watch mode.

› Press Enter to trigger a test run.

**Nota:** Para que funcione debemos agregar una línea adicional en el package.json

****

[**Utilizando Comparadores**](https://jestjs.io/docs/expect#resolves)

**.toBe→** Compara un valor que debe ser igual a lo que necesitamos ejemplo: expect (mivariable).**toBe**(‘mivalor’)

**.toMatch**→ Busca por expresión regular un valor, es como un like va entre slash, ejemplo: expect (mivariable).toMatch((**/mivalor/**))

**toHaveLength**: Comprueba la longitud

¿Qué ocurre si queremos evaluar el test y comprobar si el valor es igual, superior o inferior?, pues jest tiene unos métodos para esa tarea:

**toBeLessThan:** El valor es menor que.

**toBeLessThanOrEqual:** El valor es menor o igual que.

**toBeGreaterThanOrEqual:** El valor es mayor o igual que.

**toBeGreaterThan:** El valor es mayor que.

Ahora vamos a crear un test para comprobar los valores boolean, undefined o null,

**toBeTruthy:** El valor es verdadero.

**toBeFalsy:** El valor es falso.

**toBeUndefined:** El valor es ‘undefined’

**toBeNull:** El valor es ‘null’

Si vamos a realizar testing sobre arrays jest tiene algunos métodos.

**.toContain:** Contiene el elemento dentro del array

**.toHaveLength:** El array tiene la longitud