**Implementación de pruebas   
NEO-CONT WEBAPP**

**Consideraciones iniciales:**

Dado que la aplicación posee una capa SSR, las pruebas que deben llevarse a cabo deben tomar en cuenta dos elementos, la parte backend perteneciente a ASP y la parte frontend referente a JS.

Como es bien sabido, el proyecto NEO-CONT (NC) basa su implementación JS frontend en JS Vainilla y JQuery. La integración de una librería de testing por ende deberá incluir la vinculación de un manejador de paquetería como NPM.

**Acercamiento inicial:**

***Dependencias iniciales***

Para empezar, debe inicializarse un nuevo proyecto. Esto será posible gracias al comando **npm init -y**

Luego se instalarán las dependencias node-fetch en su versión 2 y la librería de pruebas unitarias Jest. Los comandos necesarios se detallan a continuación.

**npm install node-fetch@2 jest -D**

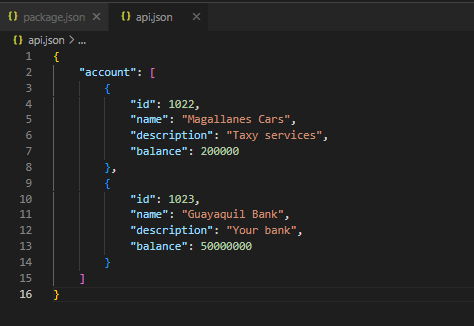
La opción **-D** del comando indica que las dependencias instaladas serán solo para desarrollo y no forman parte del build final para que la aplicación funcione en producción. Esto debido a que son dependencias para pruebas, no deberían incluirse en el compilado de producción.

***FAKE API***

Adicionalmente, instalaremos la dependencia **json-server** para generar un fake API mediante un archivo JSON. Para ello instalaremos la dependencia de manera global, no solo como parte del proyecto:

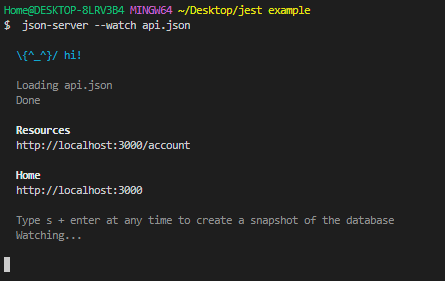
**npm install -g json-server**

Para generar el fake API es necesario construir un archivo JSON con la estructura de dicha API. A continuación, se muestra un ejemplo de este archivo:



Para inicializar el fake API se utiliza el siguiente comando en la ruta en la cual se encuentra el archivo generado:

**json-server --watch api.json**

****

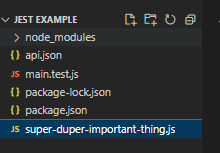
Se nos proporcionará una ruta para consumir el api generado, para el ejemplo adjunto esta ruta es <http://localhost:3000/account>.

Con esta API podremos hacer peticiones GET - POST - PUT – DELETE – PATCH, obtener elementos por ID, filtrarlos, paginarlos, etc. Para más detalle de las características del fake API generada con JSON-SERVER acceda a su [documentación](https://github.com/typicode/json-server#paginate).

***Creación de la prueba unitaria***

Una vez generada la API de prueba, procederemos a crear el archivo de pruebas unitarias para nuestro código. Recuerde que existen tres tipos de código fuente para el propósito previamente mencionado primero, el código fuente que se desea probar (funciones, métodos y clases) para el caso de NEO-CONT WEBAPP son los controladores del código ASP, segundo el código para el caso de prueba (la función o método que hace la llamada al código ASP con los parámetros y datos necesarios para la prueba) y por último el código de la prueba unitaria. Para este ejemplo se consideran los dos últimos.

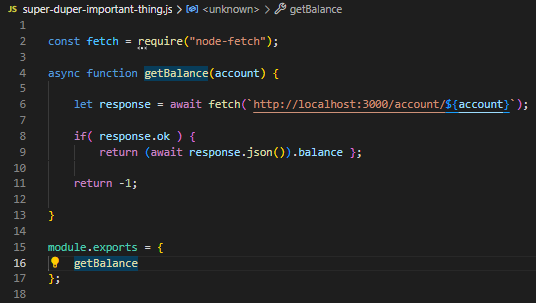
La estructura de directorios es la siguiente:



El código para el caso de prueba se encuentra en el archivo **super-duper-important-thing.js** mientras que el código para las pruebas unitarias estará en el archivo, **main.test.js**. La nomenclatura del segundo archivo no es opcional, a menos que se hagan las configuraciones necesarias. Esta nomenclatura lleva la palabra test como prefijo de la extensión del archivo.

***Archivo super-duper-important-thing.js***

Dentro de este archivo se encuentra el código utilizado para el caso de prueba. El código del ejemplo busca generar un caso de prueba para obtener el balance actual de una cuenta determinada.

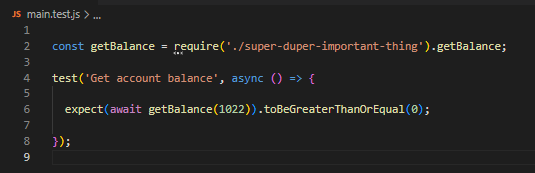


El código devuelve -1 si no encuentra una cuenta o la respuesta del servidor tiene un código de estado distinto a 200. Por otro lado, si se obtiene la cuenta se retornará el balance total de la misma.

La línea 15 permite externalizar la función **getBalance** para ser utilizada en otro módulo o archivo. En caso de tener más funciones para casos de prueba, se incluiría en la lista de objetos, de la línea 16. Esto es así debido a la destructuración de nombres de [ECMA Script](https://lenguajejs.com/javascript/modulos/export/#exportaci%C3%B3n-post-declaraci%C3%B3n).

**Archivo main.test.js**

En este archivo se encuentra el código para el test unitario, note que solo se realiza la importación del código de caso de prueba, las funciones para ejecutar el test unitario, tanto como las que realizan la verificación de resultados se pre-cargarán al ejecutar el comando para lanzar los tests.

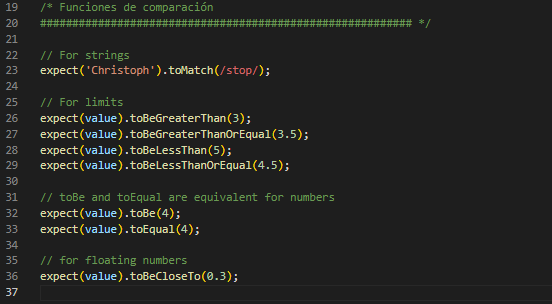


A simple vista es un archivo sencillo y lo es. La línea 2 importa la función **getBalance** del archivo que contiene el código para los casos de prueba. Luego a partir de la línea 4 se hace uso de las funciones para pruebas otorgadas por la librería de pruebas JEST.

La estructura es simple:

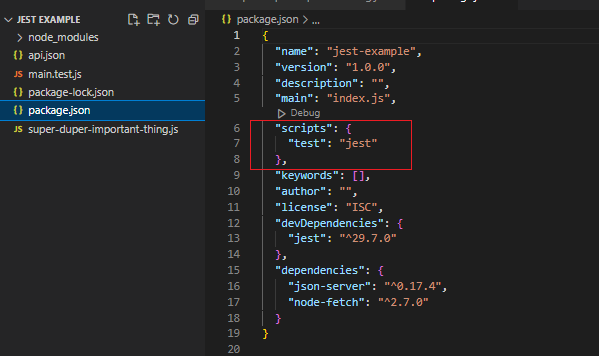
1. Se invoca la función **test** y se pasa como parámetros el nombre de la prueba y la función que ejecutará las verificaciones.
2. Dentro del segundo argumento (la función) se invoca a la función importada que contiene el código de prueba para el caso de uso, como argumento de la función **expect**. Esto debido a que expect recoge el resultado de la ejecución de la función del caso de prueba, **getBalance** para nuestro ejemplo.
3. Finalmente, se invoca la función de comparación para el resultado, en este ejemplo es toBeGreaterThanOrEqual la cual compara que el valor recogido por **expect** sea mayor o igual que el argumento que se pasa. Lo que se está comparando aquí es que el balance de la cuenta sea mayor o igual que cero.
4. Pueden agregarse la cantidad de verificaciones que se deseen en cada llamada de la función test, esto puede interpretarse como “Cada llamada a **expect** es un test a corroborar”.

Debe tomarse en cuenta que existen varias funciones de comparación, algunas para cadenas mediante expresiones regulares, numéricas enteras o de punto flotante y referentes al manejo de arrays y excepciones. Todas brindan una interfaz para la comparación de resultados retornado por la ejecución de una función. A continuación, se muestra una lista de las más relevantes.

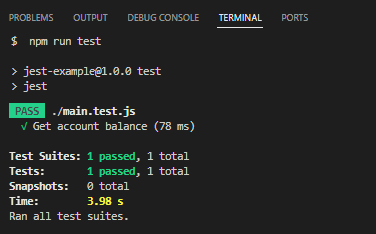


***Ejecución de tests***

Finalmente, el ultimo escaño de la guía, ejecutar el código de prueba. Como se recordará de la estructura de directorios, existe un archivo **package.json**, en el se encuentra una sección dedicada a los scripts.



Debemos utilizar **jest** como valor del elemento **“test”**. Con esto será tan simple como ejecutar el comando **npm run test**, para lanzar la batería de pruebas unitarias implementadas.



Luego de ejecutar el comando se mostrará un resumen con los resultados de cada caso prueba ejecutado.

Nota:

Para ejecutar de manera directa el ejemplo actual, no olvide lanzar el comando **npm install,** en la carpeta raíz del proyecto.