# Framework de pruebas jest

**Momento 1:** breve explicación y instalacion de framework de pruebas jest en node.js

**Jest** - Jest es un marco de prueba de JavaScript desarrollado por Facebook. Funciona de inmediato con una configuración mínima y tiene un corredor de pruebas incorporado, una biblioteca de afirmaciones y soporte para simulaciones.

**Supertest** : una biblioteca para probar servidores HTTP Node.js. Nos permite enviar solicitudes HTTP mediante programación, como GET, POST, PATCH, PUT, DELETE a servidores HTTP y obtener resultados.

## instalación de paquetes jest y supertest

**npm install --save-dev jest supertest**

## configuración de jest

Abra package.json y agregue el código a continuación.

```
"jest": {
        "testEnvironment": "node",
        "coveragePathIgnorePatterns": [
            "/node_modules/"
        ]
    },
```

Esa es la configuración básica que necesitamos para configurar jest  para probar nuestra API. Cualquier archivo que desee jestignorar se coloca dentro del "coveragePathIgnorePatterns". "coveragePathIgnorePatterns"especifica una expresión regular que coincide con el directorio que se excluirá, en nuestro caso queremos que ignore los node_modules directorios.

A continuación, agregamos el test script. Dentro de la script parte del package.json, agregue el siguiente script:

```
 "scripts": {
        "start-dev": "node index.js",
        "migrate": "npx sequelize-cli db:migrate",
        "migrate:reset": "npx sequelize-cli db:migrate:undo:all && npm
run migrate && npx sequelize-cli db:seed:all  ",
```

```
        "test": "cross-env NODE_ENV=test jest  --forceExit",
        "pretest": "cross-env NODE_ENV=test npm run migrate:reset"
    },
```

**pretest- pretest:**Es un script npm que se invoca automáticamente cuando se invoca el comando npm test. Enganchamos el comando para cambiar el entorno para probar y actualizar la base de datos antes de que se ejecute cada prueba.

**migrate:reset:** Este comando será responsable de actualizar la base de datos antes de que se ejecute cada prueba.
**cross-env-** un paquete independiente del sistema operativo para establecer variables de entorno. Lo usamos para configurar el NODE_ENV para que nuestra prueba pueda usar la base de datos de prueba. Ejecute el siguiente comando para instalar cross-env.

```
npm i -D cross-env
```

## Ejecución de scripts:

**Jest reconoce el archivo de prueba de tres formas:**
archivos que tienen extensión .test.js
archivos que tienen extensión .spec.js
Todos los archivos dentro de una  tests carpeta o directorio.

Ahora, comencemos a escribir pruebas para los puntos finales. Cree un archivo llamado login.test.js dentro del directorio de test

Copiamos el siguiente código en tests/test.test.js:

```
const request = require('supertest')
const app = require('../index')
describe('login Endpoints', () => {
    it('login user', async() => {
        const res = await request(app)
            .post('/api/usuario/login')
            .send({
                email: 'prueba@gmail.com',
                password: 'micontraseña',

            })
        expect(res.statusCode).toEqual(200)
        expect(res.body).toHaveProperty('tokenReturn');
    })

    it('can not login user with invalid password', async() => {
        const res = await request(app)
```

```
            .post('/api/usuario/login')
            .send({
                email: 'prueba@gmail.com',
                password: 'micontraseñaa',

            })
        expect(res.statusCode).toEqual(401)
    })

    it('can not login user with invalid username', async() => {
        const res = await request(app)
            .post('/api/usuario/login')
            .send({
                email: 'prueb@gmail.com',
                password: 'micontraseña',

            })
        expect(res.statusCode).toEqual(404)
    })
})
```

La **describe función** :se utiliza para agrupar pruebas relacionadas
El **it** es un alias de test la función que se ejecuta la prueba real.
La **expect función** prueba un valor usando un conjunto de matcher funciones.en este caso le pasamos la función toEqual que es lo queremos que nos devuelva la api con la propiedad accesstoken

para obtener un lista completa de métodos y funciones test visite la siguiente pagina
https://jestjs.io/docs/en/api.html

por último ejecutamos el comando:

```
C:\Users\andre\Desktop\prueba-sequelize-cli>npm test
```

el resultado se muestra como a continuación:

```
 PASS  test/login.test.js
  login Endpoints
    √ login user (211 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        2.678 s, estimated 3 s
```

como podemos ver en cada **it** ponemos los resultados que retorna la solicitud al endpoint login cuando se procesa correctamente y con el espect el resultado que debe responder dicha solicitud en esta caso un status.

en algunos casos tenemos rutas que tiene seguridad por medio de un middleware por ejemplo ,middleware de autenticación para testear este tipo de rutas debemos ejecutar la función beforeAll para obtener un token de autenticación :

```
const request = require('supertest')
const app = require('../index')

describe('Articulos Endpoints', () => {
    var token;

    beforeAll((done) => {
        request(app)
            .post('/api/usuario/login')
            .send({
                email: 'prueba@gmail.com',
                password: 'micontraseña',

            })
            .end((err, response) => {
                var result = JSON.parse(response.text);
                token = result.tokenReturn;
                console.log(token)
                done();
            });
    });
```

en este caso vemos que estamos haciendo el testeo a endpoint de artículos , pero esta tiene un middleware de autenticación lo que hacemos es llamar al método login con las credenciales de un usuario existente y en una variable token definida inicialmente almacenamos el resultado de dicha petición.

ahora para hacer uso del token de autenticación lo pasaremos por el método set del request que estamos creando :

```
it('agregar un nuevo articulo', async() => {
        const res = await request(app)
            .post('/api/articulo/add')
            .set('token', token)
            .send({
```

```
            nombre: 'articulo_test',
            descripcion: 'lorem limpsus',
            codigo: '2222',
            precio_venta: 2525,
            stock: 25,
            categoriaId: 1,

        })
    expect(res.statusCode).toEqual(200)
})
```

lo que hace  set es almacenar un objeto token en el header , que en el back la capturamos para dar acceso a la ruta que se le realiza la petición( se explicó en la semana 4).

**referencia:**

- https://dev.to/nedsoft/testing-nodejs-express-api-with-jest-and-supertest-1km6
- https://jestjs.io/