# Pruebas unitarias con marco de pruebas jest

## Marco teórico:

**Prueba de extremo a extremo** : un tipo de prueba que prueba que el flujo de una aplicación de principio a fin funciona como se espera. Esto también se conoce como prueba funcional. Un ejemplo de este tipo de prueba es probar un punto final o una ruta que implica probar todo lo necesario para que el punto final funcione, como la conexión de la base de datos, las dependencias, etc.

**Test Runner** : una biblioteca o herramienta que recoge el código fuente (pruebas) en un directorio o archivo determinado, ejecuta la prueba y escribe el resultado en la consola o en cualquier ubicación especificada, por ejemplo, Jest, Mocha.

**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.

**Objetivos:**

- instalación de paquetes jest y supertest
- configuración de jest
- ejecución los scripts

## 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 jes  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('../server')
describe('login Endpoints', () => {
    it('login user', async() => {
```

```
        const res = await request(app)
            .post('/api/auth/signin')
            .send({
                email: 'ejemplo@gmail.com',
                password: 'micontraseña',

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

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
```