Laboratório 5 – Configurando Testes com Jest

Este laboratório mostra como criar e configurar um projeto no Visual Studio Code com TypeScript, Node e framework de testes Jest com ts-jest.

1 Configurando projeto

1. Abra o Visual Studio Code e crie um novo diretório “typescriptnodejest_configuracao” dentro do seu repositório GIT criado anteriormente.

2. Utilize como base a configuração realizada no “Laboratório 1”. Iremos alterá-la durante o processo de configuração para a realização de testes.

3. Adicione as dependências para o framework de testes Jest (https://jestjs.io/) e ts-jest (https://kulshekhar.github.io/ts-jest/):

```
npm i jest @types/jest ts-jest -D
```

4. Crie um arquivo “jest.config.json” na raiz do projeto com as opções de configuração do Jest com o seguinte conteúdo para suportar módulos Ecmascript:

```
{
    "preset": "ts-jest/presets/default-esm",
    "moduleNameMapper": {
        "^(\\.{1,2}/.*)\\.js$": "$1"
    },
    "transform": {
        "^.+\\.tsx?$": [
            "ts-jest",
            {
                "useESM": true
            }
        ]
    }
}
```

5. Abra o arquivo “package.json” e localize a seção “scripts”. Iremos alterar essa seção para configurar os comandos de execução de testes e de relatórios de cobertura. Modifique o script atual de acordo com os seguintes itens:

```
"scripts": {
   "test": "node --experimental-vm-modules --no-warnings
node_modules/jest/bin/jest",
   "test:watch": "node --experimental-vm-modules --no-warnings
node_modules/jest/bin/jest --watch",
   "test:coverage": "node --experimental-vm-modules --no-warnings
node_modules/jest/bin/jest --coverage",
    …
  }
```

Garanta também que a seguinte propriedade esteja configurada logo no início do arquivo:

```
"type": "module"
```

6. Abra o arquivo “tsconfig.json” e realize as seguintes alterações:

Altere as seguintes configurações:

```
"target": "ESNext"
"module": "NodeNext"
"moduleResolution": "NodeNext"
```

Inclua as seguintes propriedades para evitar que o compilador do TypeScript processe os arquivos de teste ao se gerar uma compilação de produção:

```
"include": ["src/**/*"],
"exclude": ["node_modules", "**/*.spec.ts", "**/*.test.ts"]
```

Garanta também que a seguinte propriedade esteja configurada:

```
"ts-node": {
    "esm": true
}
```

7. Acrescente um arquivo “funcoes.ts” no diretório “src” com a seguinte implementação:

```
export function somar(a: number, b: number): number {
    return a+b;
}

export async function somarAsync(a: number, b: number): Promise<number> {
    return a + b;
}

export function stringBinParaNumber(binString: string): number {
    if (!/^[01]+$/.test(binString)) {
        throw new Error('Número binário inválido.');
    }
    return parseInt(binString, 2);
}
```

8. Acrescente um arquivo de teste “funcoes.test.ts” com a seguinte implementação para testar a função *somar* (importe o módulo “funções.js”):

```
describe('somar', () => {
    //test ou it
    test('deve retornar 3 para 1 + 2', () => {
        expect(somar(1,2)).toBe(3);
    });
    it('deve retornar 3 para 1 + 2', () => {
        expect(somar(1,2)).toBe(3);
    });
    //vários casos de teste
    test.each([[0,0,0],[1,0,1],[-1,-1,0],[0,-1,1]])
        ('deve retornar %i para %i + %i', (r,x,y) => {
            expect(somar(x,y)).toBe(r);
        });
});
```

9. Acrescente ao arquivo de teste “funcoes.test.ts” a seguinte implementação para testar a função *somarAsync*:

```
describe('somarAsync', () => {
    test('deve retornar 2 para 1 + 1', async () => {
        const resultado = await somarAsync(1,1);
        expect(resultado).toBe(2);
    });
    //ou
    test('deve retornar 2 para 1 + 1', async () => {
        expect(somarAsync(1,1)).resolves.toBe(2);
    });
});
```

10. Acrescente ao arquivo de teste “funcoes.test.ts” a seguinte implementação para testar a função *stringBinParaNumber*:

```
describe('stringBinParaNumber', () => {
    test('deve gerar exceção Error com "abc"', () => {
        //lançar uma exceção
        expect(() => stringBinParaNumber('abc')).toThrow();
        //lançar uma exceção com um tipo específico
        expect(() => stringBinParaNumber('abc')).toThrow(Error);
        //lançar uma exceção com uma mensagem específica
        expect(() => stringBinParaNumber('abc')).toThrow('Número binário
inválido.');
    });
});
```

11. Execute os seguintes comandos para realizar o teste e a análise de cobertura:

```
npm run test

npm run test:coverage
```