Laboratório 9 – Configurando Projeto Backend com Jest

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

1 Configurando projeto

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

2. Abra uma linha de comando dentro do novo diretório. Utilize CTRL+’ para abrir o terminal embutido do Visual Studio Code.

3. Inicialize o projeto Node com o seguinte comando para criar um arquivo “package.json”:

```
npm init -y
```

4. Adicione as dependências para uma configuração de um projeto de serviços backend com TypeScript, Node, Express e Mongoose:

```
npm install typescript @types/node ts-node nodemon -D

npm install mongoose --save

npm install @types/mongoose -D

npm install express body-parser --save

npm install @types/express @types/body-parser -D
```

5. Acrescente um arquivo “tsconfig.json” com as opções de compilação do TypeSript através do comando:

```
npx tsc --init
```

6. Abra o arquivo “tsconfig.json” e observe todas as opções disponíveis. Realize as seguintes alterações:

- Troque a opção “target” de “es5” para “es6”.
- Descomente a linha "sourceMap": true.
- Descomente a linha "outDir": "./". Troque o valor para "./dist".
- Descomente a linha "rootDir": "./". Troque o valor para "./src". Crie o novo diretório.
- Descomente a linha "moduleResolution": "node".

7. Abra o arquivo “package.json” e localize a seção “scripts”. Iremos alterar essa seção para configurar os comandos de compilação e execução do projeto via NPM. O resultado desejado será dois comandos para executar a aplicação “index” no Node em modo de desenvolvimento e de produção:

- “npm run dev” para iniciar a aplicação em modo de desenvolvedor via o ambiente ts-node com nodemon habilitado; nesse ambiente, qualquer alteração no código-fonte da aplicação será automaticamente refletido na execução.
- “npm run prod” para executar a aplicação com o código TypeScript já compilado para JavaScript.

```
"scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "tsc",
    "build:live": "nodemon src/index.ts",
    "dev": "npm run build:live",
    "start": "node dist/index.js",
```

```
    "prod": "npm run build && npm start"
  }
```

8. Adicione as dependências para o frameworkde testes Jest

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

9. 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": "jest --watch",
    "coverage": "jest --coverage",
    "build": "tsc",
    "build:live": "nodemon src/index.ts",
    "dev": "npm run build:live",
    "start": "node dist/index.js",
    "prod": "npm run build && npm start"
  }
```

10. Acrescente um arquivo “jest.config.js” com as opções de configuração do Jest através do comando:

```
npx ts-jest config:init
```

11. Acrescente um arquivo “funçoes.ts” no diretório “src” com a seguinte implementação:

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

12. Acrescente um arquivo de teste “funcoes.test.ts” com a seguinte implementação:

```
import { somar } from './funcoes';

test('somar 1+2 é igual a 3', () => {
    expect(somar(1,2)).toBe(3);
});
```

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

```
npm run test

npm run coverage
```