Objetivos:

I. Teste de integração;

II. Teste de integração usando Supertest.

Veja o vídeo se tiver dúvidas: https://youtu.be/j6gVQOhceJQ

Siga as instruções para criar o projeto para reproduzir os exemplos:

a) Crie uma pasta de nome servidor (ou qualquer outro nome sem caracteres especiais) no local de sua preferência do computador;

b) No terminal, execute o comando `npm init -y` para criar o arquivo fundamental de um projeto Node, arquivo package.json;

c) No terminal, execute o comando `npm i express dotenv jsonwebtoken` para instalar os pacotes de dependências;

d) No terminal, execute o comando `npm i -D jest supertest ts-jest ts-node ts-node-dev typescript` para instalar os pacotes das dependências de desenvolvimentos. O pacote supertest será usado em conjunto com o Jest para testar as requisições HTTP (https://www.npmjs.com/package/supertest);

e) No terminal, execute o comando `npm i -D @types/express @types/jest @types/jsonwebtoken @types/supertest` para instalar os pacotes que contêm as definições de tipos dos pacotes. Quando usamos um pacote é preciso ter acesso às declarações de tipo do pacote para que o TS saiba quais tipos de dados esperar do framework;

f) No terminal, execute o comando `tsc --init` para criar o arquivo de opções e configurações para o compilador TS (arquivo tsconfig.json);

g) Crie o arquivo .gitignore na raiz do projeto e coloque a linha para ignorar a pasta node_modules;

h) Crie o arquivo .env na raiz do projeto e coloque a seguinte variável de ambiente:

```
PORT = 3003
```

i) Coloque as seguintes propriedades no arquivo package.json:

```
"scripts": {
  "start": "ts-node ./src",
  "dev": "ts-node-dev ./src" ,
  "test": "jest"
```

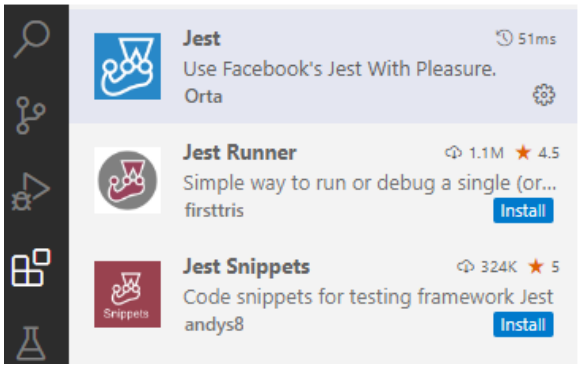
Estrutura de pastas e arquivos do projeto:

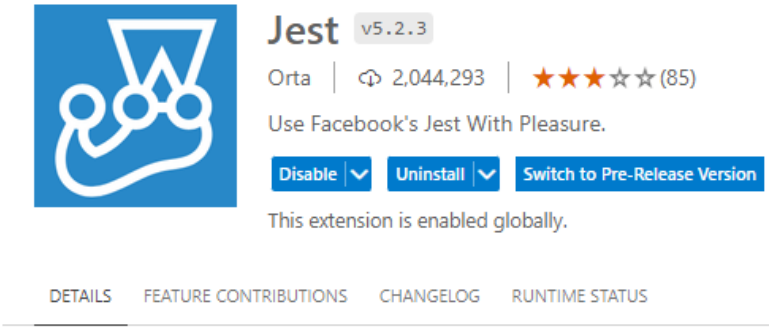
```
  },
  "jest": {
    "preset": "ts-jest",
    "testEnvironment": "node",
    "testMatch": [
      "<rootDir>/test/**/*.test.ts"
    ]
  },
```

A propriedade test, da scripts, usa o jest para rodar os arquivos de testes e a propriedade jest é usada para configurar o Jest:

- `preset`: especifica um conjunto predefinido de configurações do Jest para trabalhar com TS. No caso, o ts-jest é um preset que configura automaticamente o Jest para lidar com arquivos TS, incluindo a transpilação para JS antes da execução dos testes;
- `testEnvironment`: define o ambiente em que os testes serão executados. Isso é apropriado para testes de lógica de servidor ou código que depende de APIs específicas do Node.js;
- `testMatch`: define padrões que o Jest deve considerar como nomes de arquivos que contêm testes. Neste caso, <rootDir>/test/**/*.test.ts significa que o Jest procurará por arquivos dentro da pasta test que tenham a extensão .test.ts e os considerará como testes a serem executados.

Para auxiliar na execução dos testes no VS Code recomenda-se instalar a extensão vscode-jest ou alguma outra de sua preferência.





i. Teste de integração

Enquanto os testes unitários se concentram na verificação do comportamento de unidades individuais de código de forma isolada, os testes de integração se concentram na verificação do comportamento do sistema como um todo, incluindo a interação entre seus componentes.

Os testes unitários e os testes de integração são duas práticas distintas de teste de software, cada uma com seu próprio escopo e objetivo:

- Teste unitário:
  - É focado em testar unidades individuais de código, como funções, métodos ou classes, de forma isolada;
  - Uma unidade é a menor parte testável de um programa. No contexto do teste unitário, uma unidade geralmente se refere a uma função ou método;

  - O objetivo do teste unitário é garantir que cada unidade de código funcione conforme o esperado, isolando-a de suas dependências externas;
  - As dependências externas são geralmente simuladas ou substituídas por stubs, mocks ou doubles durante os testes unitários;
  - Os testes unitários são rápidos de escrever e executar, pois são focados em unidades de código pequenas e específicas.
- Teste de Integração:
  - É focado em testar a interação entre diferentes componentes ou módulos de um sistema. Em vez de testar unidades isoladas de código, o teste de integração verifica como os diferentes componentes se comportam juntos quando integrados em um sistema maior;
  - O objetivo do teste de integração é garantir que os diferentes componentes de um sistema funcionem corretamente quando combinados e se comuniquem uns com os outros;
  - Durante os testes de integração, os componentes são testados em conjunto, geralmente com todas as suas dependências externas;
  - Os testes de integração podem ser mais lentos e complexos do que os testes unitários, pois envolvem a execução de cenários de teste mais abrangentes que abordam a integração entre vários componentes.
  - Exemplos de ferramentas populares para teste de integração incluem Mocha (https://www.npmjs.com/package/mocha) e Chai (https://www.npmjs.com/package/chai).

Enquanto o teste unitário é de responsabilidade do próprio programador durante a codificação, o teste de integração pode ser responsabilidade de diferentes membros da equipe de desenvolvimento, dependendo da estrutura e das práticas de desenvolvimento adotadas pela organização. Alguns cenários comuns incluem:

- Equipe de desenvolvimento: membros da equipe de desenvolvimento podem ser responsáveis por escrever e executar os testes de integração, especialmente quando o processo de desenvolvimento segue práticas ágeis, como Desenvolvimento Guiado por Testes (TDD) ou Desenvolvimento Ágil de Software;
- Equipe de Qualidade (QA): em algumas organizações, uma equipe de QA separada pode ser encarregada de planejar, projetar e executar os testes de integração, além de outros tipos de testes, como testes de aceitação do usuário, testes de regressão e testes de desempenho. A equipe de QA pode ter conhecimentos especializados em testes e ferramentas de automação de testes;
- Engenheiros de testes de software: algumas organizações têm engenheiros de testes de software dedicados que são responsáveis por projetar e implementar estratégias de teste, incluindo testes de integração. Esses profissionais podem colaborar estreitamente com os desenvolvedores para garantir uma cobertura abrangente de testes em todos os níveis do sistema;
- Equipe de Operações (Ops): em ambientes DevOps, onde há integração contínua e entrega contínua (CI/CD), a equipe de operações pode desempenhar um papel importante nos testes de integração, especialmente na automação de testes em ambientes de pré-produção e produção.

Independentemente de quem seja responsável pelos testes de integração, é fundamental que haja uma colaboração estreita entre os desenvolvedores, os testadores e outras partes interessadas para garantir uma estratégia de teste eficaz e abrangente, que ajude a identificar e corrigir problemas de integração antes que eles afetem os usuários finais.

ii. Teste de integração usando Supertest

O Supertest é uma biblioteca usada para testar APIs HTTP em conjunto com frameworks de teste como o Jest, Mocha, Jasmine (https://www.npmjs.com/package/jasmine), entre outros. O Supertest fornece uma interface simplificada para fazer solicitações HTTP e verificar as respostas, facilitando o teste de rotas e endpoints de uma aplicação web.

No entanto, o Supertest em si não é um framework de teste completo. Ele é projetado para ser usado em conjunto com um framework de teste como o Jest. Isso significa que precisamos configurar um ambiente de teste usando o Jest (ou outro framework de teste) para escrevermos e executarmos nossos testes, e então usar o Supertest dentro desses testes para testar as rotas HTTP.

O Supertest é frequentemente usado para teste de integração, pois permite testar o comportamento de toda a aplicação em execução, incluindo a lógica de roteamento, a interação com o BD e outros sistemas externos.

Como exemplo considere os seguintes arquivos para criar um projeto com controladores, middleware e rotas.

Arquivo: src/controllers/Dados.ts

```
import { Request, Response } from "express";

class Dados {
  public async carro(_: Request, res: Response): Promise<Response> {
    return res.json([{marca:"VW", modelo:"Gol"}, {marca:"Fiat", modelo:"Uno"}]);
  }

  public async moto(_: Request, res: Response): Promise<Response> {
    return res.json([{modelo:"CG", ano:2010}, {modelo:"DT", ano:1990}]);
  }
}

export default new Dados();
```

Arquivo: src/controllers/Login.ts

```
import { Request, Response } from "express";
import { gerarToken } from "../middlewares";

export default async function logar(req: Request, res: Response) {
  const { mail, senha } = req.body;

  if (mail == "abc@teste.com" && senha == "123") {
    const token = await gerarToken({ nivel: "um" });
    return res.json({ token });
  } else if (mail == "xyz@teste.com" && senha == "abc") {
    const token = await gerarToken({ nivel: "dois" });
    return res.json({ token });
  } else {
    return res.json({ error: "Dados não conferem" });
  }
}
```

Arquivo: src/controllers/Matematica.ts

```
import { Request, Response } from "express";

class Matematica {
  public async somar(req: Request, res: Response): Promise<Response> {
    let { x, y } = req.body;
    const r = parseFloat(x) + parseFloat(y);
    if (isNaN(r)) {
      return res.json({ error: "Parâmetros incorretos" });
    }
    return res.json({ r });
  }

  public async subtrair(req: Request, res: Response): Promise<Response> {
    let { x, y } = req.params;
    const r = parseFloat(x) - parseFloat(y);
    if (isNaN(r)) {
      return res.json({ error: "Parâmetros incorretos" });
    }
    return res.json({ r });
  }
}

export default new Matematica();
```

Arquivo: src/middlewares/index.ts

```
import { Request, Response, NextFunction } from "express";
import jwt from "jsonwebtoken";
import dotenv from "dotenv";
dotenv.config();

const chavePrivada = "12345";
// cria um token usando os dados de entrada e a chave secreta
const gerarToken = async (entrada: any) => jwt.sign(entrada, chavePrivada);

const validarToken = (req: Request, res: Response, next: NextFunction) => {
  // pega o token enviado pelo cliente no header da requisição
  const authorization: any = req.headers.authorization;

  try {
    // autorização no formato Bearer token
    const [, token] = authorization.split(" ");
    const decodificado = <any>jwt.verify(token, chavePrivada);
    if (decodificado) {
      res.locals = decodificado;
      next();
```

```
    } else {
      res.status(401).send({ error: "Não autorizado" });
    }
  } catch (e: any) {
    res.status(401).send({ error: "Não autorizado" });
  }
};

const checarNivel = (_: Request, res: Response, next: NextFunction) => {
  const {nivel} = res.locals;
  if( nivel == "dois" ){
    next();
  }
  else{
    res.status(401).send({ error: "Acesso negado" });
  }
};

export { gerarToken, validarToken, checarNivel };
```

Arquivo: src/routes/dados.ts

```
import { Router, Request, Response } from "express";
import controlador from "../controllers/Dados";
import { checarNivel } from "../middlewares";

const routes = Router();
routes.get("/carro", controlador.carro);
routes.get("/moto", checarNivel, controlador.moto);

//aceita qualquer método HTTP ou URL
routes.use((_: Request, res: Response) =>
  res.json({ error: "Requisição desconhecida" })
);

export default routes;
```

Arquivo: src/routes/index.ts

```
import { Router, Request, Response } from "express";
import dados from "./dados";
import matematica from "./matematica";
import logar from "../controllers/Login";
import { validarToken } from "../middlewares";

const routes = Router();
routes.use("/dados", validarToken, dados);
routes.post("/login", logar)
routes.use("/matematica", matematica);
```

```
//aceita qualquer método HTTP ou URL
routes.use((_: Request, res: Response) =>
  res.status(404).json({ error: "Requisição desconhecida" })
);

export default routes;
```

Arquivo: src/routes/matematica.ts

```
import { Router, Request, Response } from "express";
import controlador from "../controllers/Matematica";

const routes = Router();
routes.get("/", controlador.somar);
routes.post("/:x/:y", controlador.subtrair);

//aceita qualquer método HTTP ou URL
routes.use((_: Request, res: Response) =>
  res.status(404).json({ error: "Requisição desconhecida" })
);

export default routes;
```

Arquivo: src/index.ts

Observação: será necessário exportar a variável app para podermos usar ela nos testes.

```
import express, {Express} from "express";
import dotenv from "dotenv";
import routes from "./routes";
dotenv.config();

const PORT = process.env.PORT || 3000;
const app:Express = express(); // a aplicação é um objeto do tipo Express
app.use(express.json());

app.listen(PORT, () => {  console.log(`Rodando na porta ${PORT}`);   });

// define a rota para o pacote /routes
app.use(routes);

export { app };
```

Arquivo: test/controlador.test.ts

```
import request from "supertest";
import {app} from "../src";

describe("Rota principal", () => {
```

```
  test("URL desconhecida", async () => {
    const response = await request(app).get("/");
    expect(response.status).toBe(404);
    expect(response.body).toEqual({ error: "Requisição desconhecida" });
  });
});

describe("Rota matemática", () => {
  test("URL desconhecida", async () => {
    const response = await request(app).delete("/matematica");
    expect(response.status).toBe(404);
    expect(response.body).toEqual({ error: "Requisição desconhecida" });
  });

  test("Soma com sucesso", async () => {
    const response = await request(app).get("/matematica").send({ x: 2, y: 3 });

    expect(response.body).toEqual({ r: 5 });
  });

  test("Soma com falha", async () => {
    const response = await request(app).get("/matematica").send({});

    expect(response.body).toEqual({ error: "Parâmetros incorretos" });
  });

  test("Subtração com sucesso", async () => {
    const response = await request(app).post("/matematica/2/3");

    expect(response.body).toEqual({ r: -1 });
  });

  test("Subtração com falha", async () => {
    const response = await request(app).post("/matematica/a/b");

    expect(response.body).toEqual({ error: "Parâmetros incorretos" });
  });
});

describe("Rota login", () => {
  test("Dados não conferem", async () => {
    const response = await request(app).post("/login");

    expect(response.body).toEqual({ error: "Dados não conferem" });
  });

  test("Login nível um", async () => {
    const response = await request(app).post("/login")
                                       .send({mail:"abc@teste.com", senha:"123"});
```

```
    expect(response.body.token.length).toEqual(123);
  });

  test("Login nível dois", async () => {
    const response = await request(app).post("/login")
                                       .send({mail:"xyz@teste.com", senha:"abc"});

    expect(response.body.token.length).toEqual(125);
  });
});

describe("Rota dados", () => {
  let tokenUm:string, tokenDois:string;

  beforeAll( async() => {
    // Efetua o login
    const responseUm = await request(app).post("/login")
                                         .send({mail:"abc@teste.com", senha:"123"});
    tokenUm = responseUm.body.token; // Obtém o token
    const responseDois = await request(app).post("/login")
                                           .send({mail:"xyz@teste.com", senha:"abc"});
    tokenDois = responseDois.body.token; // Obtém o token
  });

  it("Requisição desconhecida", async () => {
    const response = await request(app).post("/dados")
                                       .set("Authorization", `Bearer ${tokenUm}`);

    expect(response.body).toEqual({ error: "Requisição desconhecida" });
  });

  it("Lista carros", async () => {
    const response = await request(app).get("/dados/carro")
                                       .set("Authorization", `Bearer ${tokenUm}`);

    expect(response.body).toEqual([
        {"marca": "VW", "modelo": "Gol"},
        {"marca": "Fiat", "modelo": "Uno"}
      ]);
  });

  it("Lista motos", async () => {
    const response = await request(app).get("/dados/moto")
                                       .set("Authorization", `Bearer ${tokenDois}`);
    expect(response.body).toEqual([
      {"modelo": "CG","ano": 2010},
      {"modelo": "DT","ano": 1990}
    ]);
```

```
  });

  it("Lista motos nível um", async () => {
    const response = await request(app).get("/dados/moto")
                                       .set("Authorization", `Bearer ${tokenUm}`);

    expect(response.status).toBe(401);
    expect(response.body).toEqual({ error: "Acesso negado" });
  });
});
```

Observação: se o servidor estiver rodando o 1º teste apresentará a seguinte mensagem de erro:

Error: listen EADDRINUSE: address already in use :::3003Jest

Esse erro indica que há um problema ao tentar iniciar o servidor na porta 3003 pelo fato dessa porta já está sendo usada por outro processo.

Na construção dos testes o supertest é usado para fazer a requisição HTTP, veja no exemplo a chamada da função `request`. Ela recebe como parâmetro o objeto que está na variável `app` (objeto do tipo Express) que define a aplicação no arquivo src/index.ts.

```
import request from "supertest";

test("URL desconhecida", async () => {
  const response = await request(app).get("/");
  expect(response.status).toBe(404);
  expect(response.body).toEqual({ error: "Requisição desconhecida" });
});
```

O método `get` é usado para fazer requisições do tipo HTTP GET. Se a requisição fosse para o método HTTP POST deveríamos ter chamado o método `post`, por exemplo,

```
await request(app).post("/");
```

como a requisição devolve uma Promise, então devemos preceder a chamada com `await`.

Os testes podem ser definidos usando as funções `test` ou `it` do Jest, e as asserções são construídas usando a função `expect` e os matchers do Jest.

Resultado dos testes:

```
PASS test/controlador.test.ts
  Rota principal
    √ URL desconhecida (47 ms)
  Rota matemática
    √ URL desconhecida (5 ms)
    √ Soma com sucesso (11 ms)
    √ Soma com falha (3 ms)
    √ Subtração com sucesso (4 ms)
    √ Subtração com falha (2 ms)
  Rota login
    √ Dados não conferem (2 ms)
    √ Login nível um (5 ms)
    √ Login nível dois (4 ms)
  Rota dados
    √ Requisição desconhecida (4 ms)
    √ Lista carros (3 ms)
    √ Lista motos (4 ms)
    √ Lista motos nível um (2 ms)
```

O Supertest fornece também o método `send` para passarmos dados pelo body da requisição:

```
const response = await request(app).post("/login")
                                   .send({mail:"xyz@teste.com", senha:"abc"});
```

O método set pode ser usado para passarmos parâmetros pelo header da requisição:

```
const response = await request(app).get("/dados/moto")
                                   .set("Authorization", `Bearer ${tokenUm}`);
```