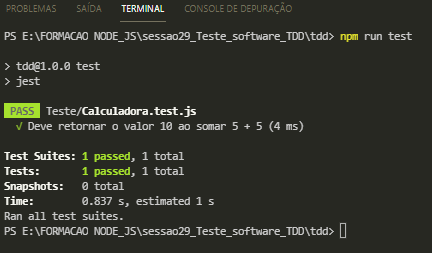
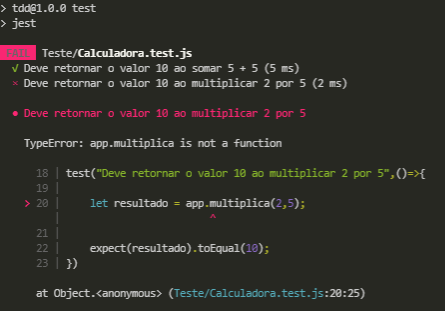
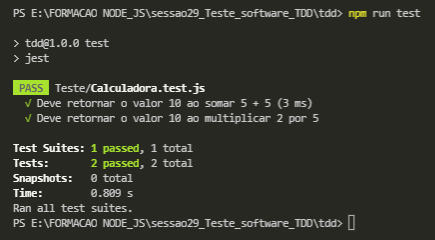
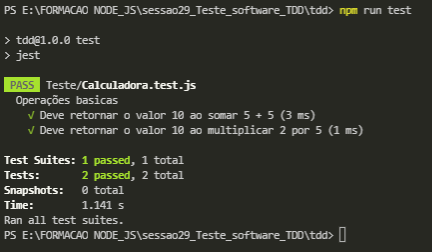
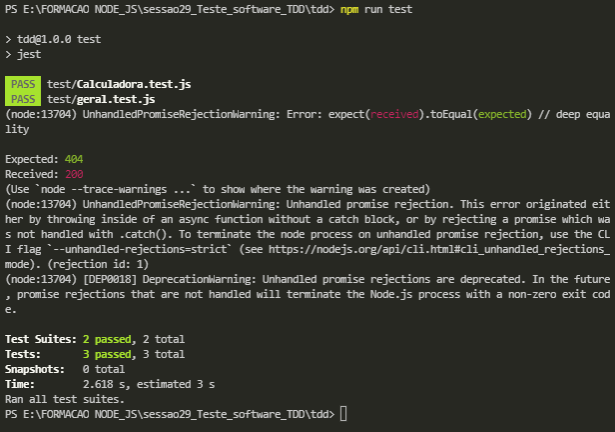
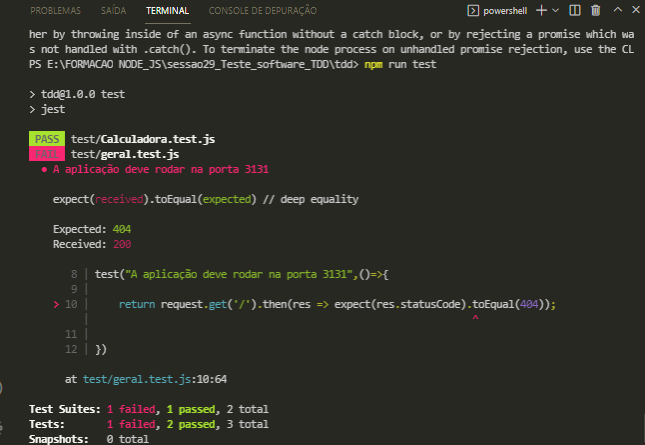
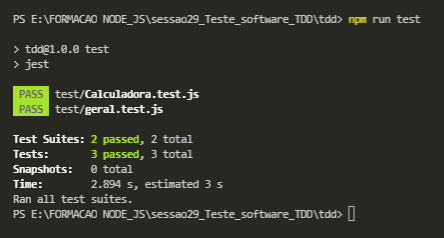
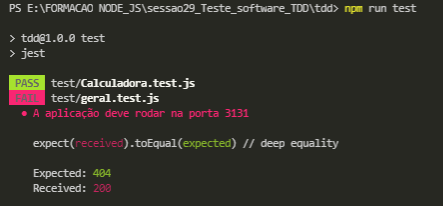
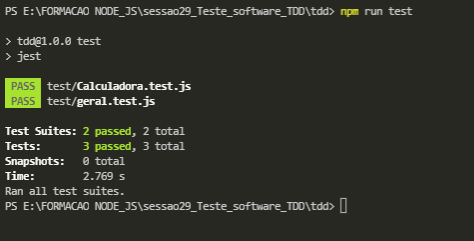
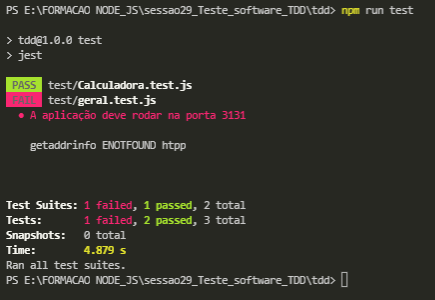
**REALIZANDO TESTES DE SOFTWARE**

1. Inicie um novo projeto digite npm init.
2. Para realizar os testes vamos instalar a biblioteca Jest que é a principal biblioteca de testes do Javascript, digite npm install jest.
3. Crie um arquivo app.js na pasta TDD
4. Crie uma pasta chamada Teste
5. Abra o arquivo package.json e no script acrescente: “guia”:”node app.js”
6. Agora se formos no terminal e digitarmos “npm run guia”, automaticamente ele irá executar o node no app.js
7. Com essa funcionalidade podemos rodar nosso teste, basta somente, apagarmos o conteudo que está dentro de “test” e acrescentarmos a biblioteca “jest” isso fará que o teste seja executado com essa biblioteca.
8. {
9. *"dependencies"*: {
10. *"jest"*: "^27.4.7"
11. },
12. *"name"*: "tdd",
13. *"version"*: "1.0.0",
14. *"main"*: "index.js",
15. *"devDependencies"*: {},
16. *"scripts"*: {
17. *"test"*: "jest",
18. *"guia"*: "node app.js"
19. },
20. *"author"*: "",
21. *"license"*: "ISC",
22. *"description"*: ""
23. }
24. Para corrigir erros de autocomplete do jest digite:
25. Npm install @types/jest

**CALCULADORA.TEST**

1. Vamos criar um arquivo chamado calculadora.test.js
2. Vamos criar o teste chamando a função test que possui como parâmetros oque o teste irá fazer seguido do arrow function que será executada no teste.
3. Dentro desta arrow function criamos uma variável que recebera a execução do método ou função, lembrando que temos que importar o arquivo que será testado.
4. Temos também um expect que é o resultado esperado do teste.
5. //importando o arquivo onde está a função soma
6. *let* app = require("../app");
7. /\*\*Criando o teste de uma função de soma \*/
8. test("Deve retornar o valor 10 ao somar 5 + 5",()*=>*{
10. *let* resultado = app.soma(5,5);
11. //para ver se o valor está correto ou não
12. expect(resultado).toEqual(10);
13. })
14. .
15. No arquivo app.js crie agora a função soma
16. *module*.*exports*={
17. soma: *function*(*a* , *b*){
18. return *a* + *b*;
19. }
20. }
21. Agora execute o teste: npm run test
22. O jest irá realizar o teste e dará o resultado.
23. ****
24. C
25. C
26. Vamos criar outro teste para uma função de multriplicação.
27. //tese de multipliação
28. test("Deve retornar o valor 10 ao multiplicar 2 por 5",()*=>*{
29. *let* resultado = app.multiplica(2,5);
30. expect(resultado).toEqual(10);
31. })
32. C
33. Agora execute o teste para ver o resultado.
34. ****
35. Vemos que o teste falhou pois a função multiplica ainda não foi criada no arquivo app.
36. Vamos criar a função multiplica no app.js
37. multiplica: *function*(*a*, *b*){
38. return *a* \* *b*;
39. }
40. }
41. agora vamos novamente realizar o teste e ver o resultado.
42. ****
43. Agora o teste passou.
44. Quando temos um projeto que terá vários testes, podemos organizar a suíte de testes em categorias, para isso iremos utilizar a função describe(), onde daremos o nome da categoria.
45. Desta forma o resultado ficará mais legível
46. ****

**INTEGRANDO O JEST COM O EXPRESS**

1. Crie a pasta src e crie todo o código da aplicação nesta pasta.
2. Vamos primeiramente fazer um teste para verificar se a aplicação está rodando em uma porta ou não.
3. Vamos criar uma nova suíte de teste dentro da pasta test.
4. crie o arquivo geral.test.js
5. vamos instalar primeiramente a biblioteca supertest para testar as rotas da aplicação.
6. Npm install supertest
7. dentro do arquivo importe o supertest,
8. crie um objecto request para ele ficar fazendo requisição dentro da aplicação para podermos testar.
9. Crie o test fazendo uma requisição e esperando como resposta de statusCode igual à 404.
10. //importando a biblioteca de teste de rotas
11. *let* supertest = require('supertest');
12. //criando um objeto para criar requisição dentro do test
13. *let* request = supertest("www.google.com");
14. //teste para verificar se a aplicação está rodando em uma determinada porta
15. test("A aplicação deve rodar na porta 3131",()*=>*{
16. request.get('/').then(*res* *=>* expect(*res*.statusCode).toEqual(404));
17. })
18. ao executarmos o teste veremos que dará um erro, porém o teste passa,
19. ****
20. Isso ocorre porque a promisse do teste deve ser precedida de um return.
21. //importando a biblioteca de teste de rotas
22. *let* supertest = require('supertest');
23. //criando um objeto para criar requisição dentro do test
24. *let* request = supertest("www.google.com");
25. //teste para verificar se a aplicação está rodando em uma determinada porta
26. test("A aplicação deve rodar na porta 3131",()*=>*{
27. return request.get('/').then(*res* *=>* expect(*res*.statusCode).toEqual(404));
28. })
29. Corrigindo e fazendo o teste novamente veremos que o teste dá certo.
30. ****
31. Vemos que o teste geral não passa porque ao realizarmos o request para a página do goolgle nos recebemos como resposta o statusCode 200 e não 404 que estava sendo esperado no teste.
32. Vamos alterar o statusCode para 200..
33. //teste para verificar se a aplicação está rodando em uma determinada porta
34. test("A aplicação deve rodar na porta 3131",()*=>*{
35. return request.get('/').then(*res* *=>* expect(*res*.statusCode).toEqual(200));
36. })
37. Execute o teste novamente e verá que não irá ocorrer nenhum erro.
38. ****
39. Outra alternativa que temos é trabalhar com await ao invez de uma promisse.
40. //teste para verificar se a aplicação está rodando em uma determinada porta
41. test("A aplicação deve rodar na porta 3131",async()*=>*{
42. // return request.get('/').then(res => expect(res.statusCode).toEqual(200));
43. *let* res = await request.get('/')/
44. expect(res.statusCode).toEqual(404);
45. })
46. Rodando o teste vermos novamente o erro devido o status code que estamos esperando ser 404 e não 200.
47. ****
48. Ao mudarmos o statuscode que estamos esperando para 200
49. //teste para verificar se a aplicação está rodando em uma determinada porta
50. test("A aplicação deve rodar na porta 3131",async()*=>*{
51. // return request.get('/').then(res => expect(res.statusCode).toEqual(200));
52. *let* res = await request.get('/');
53. expect(res.statusCode).toEqual(200);
54. })
55. Veremos que não ocorrerá erro e o teste passou
56. ****
57. para não termos que ficar utilizando try catch nos await dos testes, é melhor utilizar as promisses para os testes.
58. Agora altere o endereço do request para verificarmos se a aplicação está rodando na porta 3131.
59. //alterando o endereço para o endereço da aplicação
60. *let* request = supertest("htpp://localhost:3131");
61. Vamos rodar o teste e vamos ver que o teste não irá passar, poia a aplicação ainda não existe.
62. ****
63. Após escrever o teste e fazer o teste falhar vamos programar a aplicação.
64. Vamos instalar o express :
65. Npm install express