INFOESTE 2024

Introdução aos Testes Automatizados: Desvendando o Poder da Automação

Vinicius Vasconcelos

Professor | Desenvolvedor Back-end

Objetivo

Métricas de sucesso...

- Oferecer um curso rápido e introdutório;
- Ampliar o conhecimento das pessoas participantes;
- Demonstrar temas relevantes que não dependem de ferramentas específicas;
- Promover a interação entre as pessoas participantes de forma descontraída, educativa e envolvente;
- Incentivar o networking.

Motivação

Porque testes são importantes?

- Melhora a escrita de código, diminui erros e aumentam a produtividade;
- Aperfeiçoa o raciocínio lógico para a escrita de código;
- Muitas vagas pedem conhecimento em teste de software;
- Em entrevistas/testes técnicos para júnior, é um diferencial;
- Durante minha jornada como docente, sentia dificuldade/preguiça por parte das pessoas em realizar os testes kkkk.

Ferramentas

Tecnologias utilizadas no curso!

- Express.js;
- Mocha;
- Chai.JS;
- Sinon.JS;
- Chai HTTP;
- Istanbul/NYC.

Express.js

Framework para Node.js que facilita a criação de APIs e aplicativos web. Ele fornece um conjunto robusto de recursos para o desenvolvimento, assim facilitando alguns processos dentro da criação dos softwares.

Mocha

Framework de testes para Node.js. Ele ajuda a organizar e estruturar testes de uma maneira flexível, compreensível e modular.

Chai.js

Uma biblioteca de asserção para Node.js. Ela pode ser usada com qualquer framework de testes JavaScript, assim ajudando a expressar testes de uma forma que seja fácil de entender e manter.

Sinon.JS

Uma biblioteca que fornece funções para mocks, stubs e spies. Essencial para testar chamadas de funções e comportamentos isolados.

Chai HTTP

Uma extensão do Chai para testar requisições HTTP. Facilita a verificação das respostas de APIs em seus testes.

Istanbul/NYC

Ferramentas para medir a cobertura de código dos testes. Elas ajudam a garantir que todas as partes do seu código estão sendo testadas, identificando quais linhas de código foram executadas durante os testes.

Testes

Por que realizar testes automatizados?

- Diminuição de falhas/bugs;
- Evitar testes viciados;
- Garantir segurança para alterações e refatorações;
- Proporcionar segurança ao fazer deploy de grandes partes do sistema;
- E muito mais...

Testes de unidade vs Testes de integração

- Testes de unidade:
  - Rápidos;
  - Baratos;
  - Geralmente mais fáceis;

- Testes de integração:
  - Maior abrangência em menor tempo;
  - Em alguns casos, o único que dá para ser feito.

Test double o que é?

Termo genérico usado para qualquer objeto que substitui um componente real no sistema que está sendo testado. Eles são especialmente úteis quando se precisa isolar parte do sistema que está sendo testada.

Test doubles são fundamentais em testes automatizados, pois permitem simular e controlar o comportamento de dependências.

Alguns exemplos...

- **Fake:** Implementações que funcionam, porém mais simplificadas em relação a um produto real (como um banco de dados em memória);

- **Mock:** Objetos pré-programados com expectativas idênticas às que devem ser recebidas;

- **Stub:** Proporciona respostas predefinidas a chamadas feitas durante o teste, geralmente não respondendo a nada mais fora do que foi programado;

...

- **Spy:** Similar aos stubs, mas eles também registram informações sobre como foram chamados;

- **Dummy:** Objetos que são passados como argumentos, mas nunca são realmente usados. Geralmente são utilizados apenas para preencher requisitos de assinatura de métodos.

Ou seja...

- **Fake:**
  - Implementações reais; (ex: uso em memória);
  - Diminuição de complexidade para realizar os testes;
  - Não há lógica, retorna o desejado;

- **Mock:**
  - Expectativa vs falha;
  - Realiza testes de interação entre métodos;

- **Spy:**
  - Age como um espião; (ex: verifica se função foi chamada, quantidade de vezes chamada);
  - Bem parecido com o mock, mas chama implementações reais;

...

- **Stub:**
  - Parecido com o fake e mock, mas altera um comportamento;

- **Dummy:**
  - Dados que substituem dados, mas que não são usados nos testes.

 Cobertura de Testes

O que é e como funciona?

Cobertura de testes é uma métrica que mede a quantidade de código em um programa que está sendo testada por testes automatizados, assim ajudando a entender se todas as partes estão sendo verificadas.

Métricas Comuns:

- **Cobertura de Instruções:** Mede a porcentagem de linhas de código executadas durante os testes;
- **Cobertura de Ramificações:** Mede se todas as decisões (como ifs, for e etc..) no código foram testadas;
- **Cobertura de Funções:** Mede quantas funções no código foram chamadas durante os testes.

Arquivo de configuração do NYC

O arquivo **nycrc.json** é usado pela biblioteca nyc, que é um instrumento de cobertura de código para Node.js:

- **"extension": [".test.js"]:** Define a extensão dos arquivos de teste;
- **"include": ["src"]:** Inclui a pasta src no relatório de cobertura;
- **"exclude": ["tests"]:** Exclui a pasta tests do relatório de cobertura.

Cobertura de testes usando NYC

A cobertura de testes apresentada pelo nyc fornece uma visão detalhada sobre como os testes estão cobrindo o código do seu projeto.

Colunas da Cobertura de Testes

- **File:** Indica o arquivo ou diretório que está sendo analisado;

- **% Stmts:** Percentual de instruções (statements) do arquivo que foram executadas durante os testes;

- **% Branch:** Percentual de ramificações (branches) do arquivo que foram testadas. Isso se refere a estruturas condicionais, como if, else, switch, etc;

...

- **% Funcs:** Percentual de funções do arquivo que foram chamadas durante os testes;

- **% Lines:** Percentual de linhas de código do arquivo que foram executadas;

- **Uncovered Line #s:** Número das linhas que não foram cobertas pelos testes.

Obrigado

 GitHub: vinicius-vasconcelos

 LinkedIn: vinicius-vasconcelos-8828a416b