Testes Automatizados com Jest

Daniel de Godoy Carolino
daniel.carolino@gmail.com
daniel.carolino@fatec.sp.gov.br

Resumo

Esse capítulo mostra como o TDD (Test Driven Development) pode ser usado junto com o Jest, uma biblioteca super popular de testes no mundo JavaScript. A ideia é mostrar como essa dupla pode melhorar muito a qualidade do seu código e deixar o processo de desenvolvimento mais eficiente e tranquilo.

Além de ser fácil de configurar e rodar, o Jest funciona muito bem com frameworks como React, Node, Angular e Vue. Ele tem vários recursos úteis como cobertura de código, mocks automáticos e testes assíncronos, o que ajuda a encontrar erros mais rápido e com mais precisão. Com tudo isso, o Jest acaba incentivando um desenvolvimento mais organizado, que facilita a manutenção e o crescimento do sistema.

Palavras-chave: *Testes Automatizados, Jest, TDD, JavaScript, Qualidade de Software, Cobertura de Código, Desenvolvimento Ágil, Integração Contínua, Mocks, Testes Unitários, Engenharia de Software.*

Introdução

Compartilhando sobre como usar o TDD com o Jest, que é uma das ferramentas mais populares pra quem desenvolve com JavaScript. A ideia aqui não é só jogar uns comandos ou mostrar exemplos soltos, mas ajudar a entender de verdade por que vale a pena testar seu código e como isso muda o jeito que a gente desenvolve software.

Tem muita gente que acha que escrever testes é perda de tempo ou que dá mais trabalho do que resultado. Mas a real é que, quando a gente pega o jeito e começa a usar ferramentas como o Jest, tudo fica mais simples. Os testes passam a fazer parte do processo natural de escrever código, não como uma obrigação chata, mas como uma ferramenta que realmente ajuda no dia a dia.

Além disso, o Jest foi feito pra ser fácil de usar. Ele funciona direto com Node e se integra super bem com frameworks como React, Vue e Angular. Você não precisa passar horas configurando — é instalar e começar a testar. Ele já vem com um monte de coisa pronta, como cobertura de código, suporte a testes assíncronos e até recursos de mocks que ajudam muito quando o código depende de APIs externas ou serviços que ainda não existem.

Outra coisa legal é que quando a gente aplica o TDD, que é a ideia de escrever os testes antes do código, a gente muda a forma de pensar. Em vez de sair codando no improviso, a gente para pra pensar no que a função ou o componente realmente precisa fazer. Isso deixa o código mais limpo, mais fácil de manter e com menos chances de ter bugs escondidos.

Também é importante lembrar que testes não são só pra quem tá em projeto grande. Mesmo em aplicações pequenas ou pessoais, ter testes automatizados pode salvar tempo e dor de cabeça, principalmente quando você volta no projeto depois de um tempo ou quando começa a adicionar novas funcionalidades. E se o projeto for em equipe, melhor ainda: os testes servem como uma documentação viva do sistema, que mostra claramente o que cada parte deveria fazer.

Então, se você nunca testou nada, nunca usou Jest ou nunca experimentou escrever os testes antes do código, esse capítulo é uma chance legal de conhecer essas práticas de um jeito leve, com exemplos simples e direto ao ponto.

Desenvolvimento

Quando a gente junta TDD com Jest, a gente tem uma forma muito eficiente e moderna de desenvolver código JavaScript. O TDD propõe que os testes venham antes do código real — isso faz a gente pensar melhor no que o sistema precisa fazer. E o Jest é uma baita ferramenta que facilita esse processo todo.

O que é TDD?

O TDD funciona em três passos simples:

1. Escreve um teste que vai falhar (fase "vermelha").
2. Escreve o código só pra fazer esse teste passar (fase "verde").

3. Dá uma melhorada no código, mantendo o teste passando (fase "refatoração").

Fazendo isso sempre, a gente acaba criando um código mais limpo, organizado e com menos chances de dar problema lá na frente.

Por que usar o Jest?

O Jest é bem direto ao ponto:

- Você instala rapidinho com ***npm install jest.***
- Já vem pronto pra usar, sem precisar ficar quebrando a cabeça com configuração.
- Dá pra testar coisas assíncronas, fazer mocks, checar cobertura de código e até usar snapshots.

Exemplos na prática

Vamos supor que a gente quer testar uma função de soma. Com o Jest, isso aqui já funciona:

```
function soma(a, b) {
  return a + b;
}

test('soma de 2 + 3 deve ser 5', () => {
  expect(soma(2, 3)).toBe(5);
});
```

Se quiser testar validações, por exemplo de e-mail:

```
test('valida e-mail com formato correto', () => {
  expect(validarEmail('teste@exemplo.com')).toBe(true);
});
```

Depois você implementa a função ***validarEmail***, e assim vai seguindo o ciclo do TDD.

Teste de API com Jest e Node:

```
const fetch = require('node-fetch');

// Teste para verificar se a PokeAPI retorna os dados do Pikachu
test('deve retornar os dados do Pikachu da PokeAPI', async () => {
  const response = await fetch('https://pokeapi.co/api/v2/pokemon/pikachu');
  const data = await response.json();

  expect(data.name).toBe('pikachu');
  expect(data.id).toBe(25);
  expect(data.types[0].type.name).toBe('electric');
});
```

Vantagens reais no dia a dia

Usar TDD com Jest faz muita diferença no mundo real:

- Você pega erro antes que ele cause problema.
- Refatorar vira algo seguro.
- O time inteiro entende melhor o que cada parte do sistema faz.
- Dá pra usar integração contínua e automatizar testes em cada push no Git.

Empresas grandes como Meta, Spotify e Twitter (atualmente X) usam Jest em projetos grandes, então a gente sabe que dá certo até nos ambientes mais exigentes.

Considerações Finais

Trabalhar com testes automatizados usando o Jest, junto com o TDD, muda totalmente a forma como a gente desenvolve software. Não é só uma técnica legal — é um jeito diferente de pensar: colocar a qualidade como prioridade desde o começo.

O Jest ajuda muito nesse processo porque automatiza um monte de coisa chata, pega erro logo no início e mantém o código organizado, mesmo em projetos grandes e com várias pessoas trabalhando junto. Já o TDD dá uma direção mais clara pro que precisa ser feito, deixando tudo mais planejado e com menos retrabalho.

No fim das contas, usar Jest e TDD juntos não é só uma boa prática — é quase uma necessidade se você quer criar aplicações modernas, que sejam fáceis de manter, crescer e confiar no que está funcionando.

Bibliografia

- FATEC ARARAS. *Faculdade de Tecnologia de Araras*. Disponível em: https://fatecararas.cps.sp.gov.br. Acesso em: 27 maio 2025.

- CAROLINO, Daniel. *ebook_qualidade_teste_software*. GitHub. Disponível em: https://github.com/DanielCarolino89/ebook_qualidade_teste_software

- SARAIVA JUNIOR, Orlando. *ebook_qualidade_teste_software*. GitHub. Disponível em: https://github.com/orlandosaraivajr/ebook_qualidade_teste_software. Acesso em: 27 maio 2025.

- JESTJS. *Getting Started – Jest Documentation*. Disponível em: https://jestjs.io/docs/getting-started. Acesso em: 03 junho 2025.

- BECK, Kent. *Test-Driven Development: By Example*. Addison-Wesley, 2002. (Informações complementares sobre TDD, frequentemente citadas em cursos e práticas com Jest).

- POKEAPI. *The RESTful Pokémon API*. Disponível em: https://pokeapi.co. Acesso em: 03 junho 2025.