



**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS**  
**ENGENHARIA DE SOFTWARE**  
**TESTE DE SOFTWARE**

**Jest: Explorando a Profundidade de um Framework de Teste**  
**Moderno**

Gabriel de Moura e Souza  
Gabriel Estevão Nogueira Sobrinho  
Luana Fleury Braz  
Lucas Paixao Soares Ribeiro  
Rafael Ferraz Barra

## 1. INTRODUÇÃO

No atual cenário tecnológico, observa-se uma crescente demanda por soluções de software que sejam simultaneamente inovadoras, de elevada qualidade e dotadas de conveniência operacional. Diante deste panorama, torna-se imperativo que empresas priorizem a entrega de produtos de software que primam pela confiabilidade, especialmente em tempos de implementações aceleradas. Os frameworks de teste, constituídos por um conjunto coeso de diretrizes, convenções e ferramentas, surgem como instrumentos cruciais para auxiliar os profissionais de tecnologia na criação e execução de testes de software de maneira otimizada e estruturada. A importância desses frameworks é amplificada quando consideramos a crescente ênfase nas práticas de integração contínua e entrega contínua (CI/CD). Estes, ao facilitarem uma detecção e correção mais eficazes de erros, não apenas consolidam a robustez do software, mas também contribuem significativamente para a mitigação dos custos associados a eventuais adversidades ou comprometimentos operacionais.

Em concordância com essa perspectiva, dados indicam que o mercado de testes automatizados, avaliado em 20 bilhões de dólares em 2022, projeta uma taxa de crescimento anual composta superior a 15% entre 2023 e 2032. Ferramentas de teste avançadas são caracterizadas pela capacidade de suporte a uma vasta gama de linguagens de programação e ambientes de teste, têm sua indispensabilidade ratificada. Estas proporcionam um grau reduzido de intervenções manuais, geram análises acionáveis e, de forma subsequente, aceleram o tempo de entrada no mercado, culminando em uma valorização empresarial acentuada. Dentro deste contexto acadêmico e de pesquisa, destaca-se o framework Jest. A seguir, faremos uma análise das características específicas do framework de testes Jest, abordando sua origem e concepção, e demonstraremos sua aplicação, com foco especial na facilitação de testes end-to-end.

## 2. O QUE É O JEST?

O Jest é um dos frameworks de teste JavaScript mais populares atualmente. Ele é amplamente reconhecido por sua capacidade de trabalhar com várias tecnologias, como TypeScript, Node, React, Angular e Vue. Inicialmente, o Jest foi projetado para realizar testes unitários em componentes React. No entanto, devido à sua eficiência e recursos, rapidamente ganhou popularidade e passou a ser amplamente utilizado em aplicações JavaScript, tanto no front-end quanto no back-end. Além dos testes unitários, também pode ser usado para testes de componentes, permitindo testar diferentes partes de uma aplicação. Uma das características distintivas é sua CLI (Interface de Linha de Comando), que permite a execução de comandos relevantes diretamente do terminal. Além disso, oferece uma biblioteca de asserções, um test runner e suporte para técnicas de simulação, entre outros recursos.

O Jest foi desenvolvido pela empresa Meta, anteriormente conhecida como Facebook. O principal responsável pelo desenvolvimento é Christoph Nakazawa que teve como objetivo principal criar um framework que fornecesse suporte e simplicidade para testar aplicações web complexas. A robustez do Jest também pode ser observada em sua independência, por não depender de aplicativos ou softwares de terceiros, o que o coloca em vantagem em relação a outros frameworks de teste de automação populares.

Com sua ampla compatibilidade com diferentes tecnologias e sua capacidade de oferecer suporte a testes em várias partes de uma aplicação, o Jest se destaca como uma solução abrangente e eficiente para a realização de testes em ambientes JavaScript. Sua CLI intuitiva facilita a execução de comandos relevantes diretamente do terminal, tornando o processo de teste mais prático e eficaz. Além disso, a biblioteca de asserções e o test runner fornecidos pelo Jest contribuem para simplificar o processo de escrita e execução de testes.

No geral, o Jest é amplamente reconhecido como um dos melhores frameworks de teste unitário disponíveis atualmente. Sua versatilidade, eficiência e recursos avançados o tornam uma escolha popular entre os desenvolvedores, permitindo testar e garantir a qualidade de aplicações JavaScript em diferentes contextos e tecnologias. Seja para testes unitários ou de componentes, oferece uma solução completa e confiável para a realização de testes JavaScript.

### **3. POR QUE MUITAS EQUIPES DE CONTROLE DE QUALIDADE ESCOLHEM O JEST?**

Ao testar uma aplicação, equipes de controle de qualidade buscam eliminar abordagens desordenadas para diminuir a probabilidade de erros e evitar confusões. O Jest se destaca como uma escolha popular entre muitas equipes de QA por várias razões, primeiramente, ele conta com uma documentação constantemente atualizada, fácil de usar e repleta de exemplos típicos, tornando a curva de aprendizado suave para novos usuários.

Um dos recursos notáveis do Jest é o seu modo interativo, que executa automaticamente todos os testes afetados após qualquer mudança no último commit. Esta funcionalidade, combinada com uma sintaxe que permite testar ou pular testes específicos, oferece uma grande flexibilidade às equipes. Além disso, ele traz facilidades como mocking, permitindo a simulação de qualquer objeto fora do escopo do teste, e a capacidade de gerar cobertura de código, dando visão sobre partes do código que necessitam de mais testes.

Recursos como execução rápida de testes em paralelo, snapshot testing e suporte a TypeScript são outros fatores que tornam o Jest atraente. A fácil migração para o Jest, graças ao módulo code mods, e uma API completa e robusta, também são aspectos cruciais que muitas equipes de controle de qualidade consideram ao escolher o Jest como seu framework de testes em JavaScript.

Contudo, vale ressaltar que, para projetos menores ou de natureza mais simples, o Jest, com sua vasta gama de funcionalidades, pode ser percebido como um overhead. Isto é, algumas equipes podem sentir que o Jest introduz uma complexidade adicional que não é necessariamente exigida pelo escopo do projeto em questão. Esse aspecto pode ser particularmente relevante para equipes que buscam soluções mais leves e menos intrusivas, ou que têm preocupações sobre a eficiência e a otimização do tempo de teste.

## 4. INSTALAÇÃO E CONFIGURAÇÃO

Adotar o Jest como framework de testes em seu projeto pode ser uma decisão transformadora, especialmente com sua gama de funcionalidades e flexibilidade. Felizmente, o processo de instalação é projetado para ser simples, mesmo para aqueles que são relativamente novos no ecossistema JavaScript. Para garantir uma transição suave, elaboramos um guia detalhado para ajudá-lo na instalação e configuração do Jest.

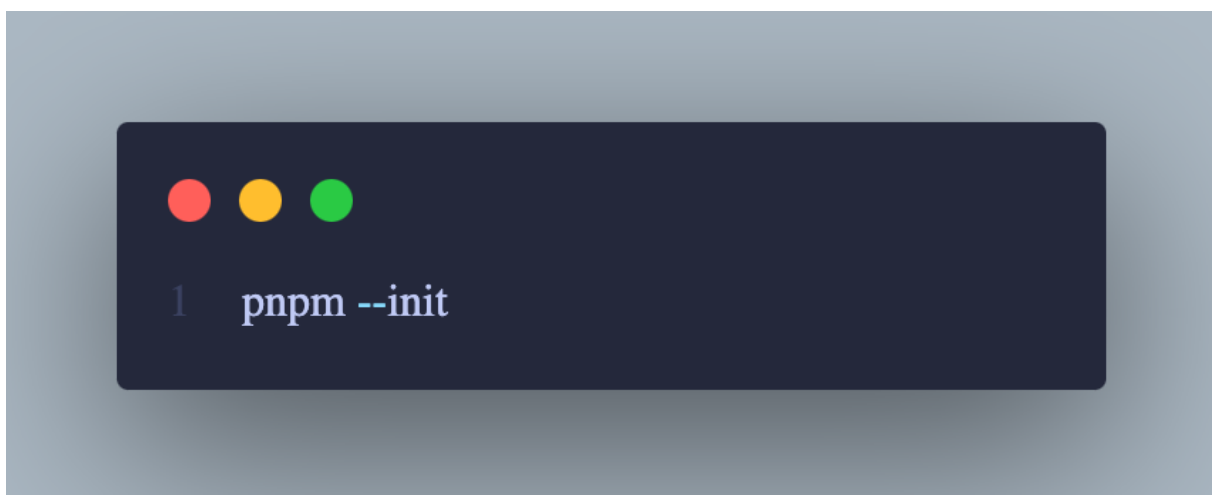
### Pré-requisitos:

Primeiro e acima de tudo, é fundamental ter o *Node.js* e o *npm* (Node Package Manager) previamente instalados em seu sistema. Se você está partindo do zero, o site oficial do Node.js fornece um guia claro para fazer isso.

### Passo a Passo para a Instalação e Configuração:

#### 4.1 Iniciando um Novo Projeto Node.js

Se você está começando um projeto do zero, é aconselhável inicializar um ambiente Node.js. Usando o terminal ou prompt de comando, insira:



Isso automaticamente gerará um arquivo `package.json`, que servirá como o coração da configuração do seu projeto.

## 4.2. Adicionando o Jest ao seu Projeto

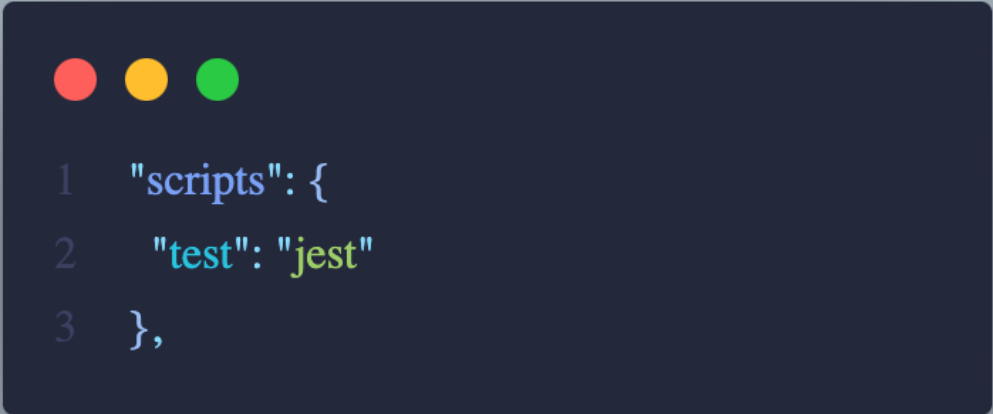
Uma vez que você tenha seu projeto Node.js em andamento, a instalação do Jest é apenas um comando de distância. No terminal, execute:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays a single command on line 1.

```
1 pnpm install --save-dev jest
```

## 4.3. Personalizando as Configurações de Teste

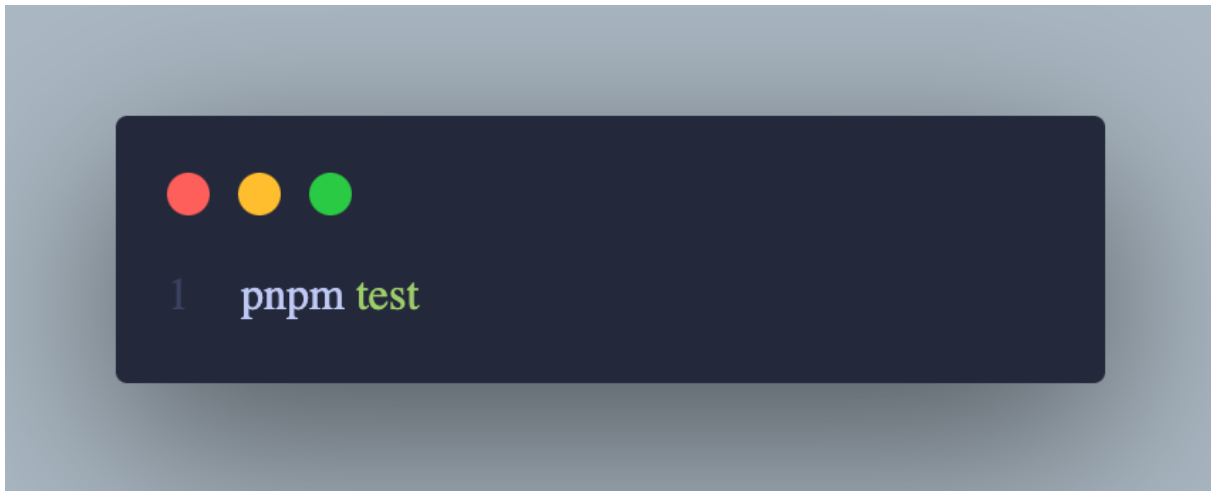
Com o Jest agora parte de seu projeto, é hora de configurá-lo para atender às suas necessidades. Dentro do arquivo `package.json`, você encontrará uma seção `scripts`. Modifique-a para incluir o Jest como sua ferramenta de teste padrão:

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. It displays a JSON configuration snippet on three lines.

```
1 "scripts": {  
2   "test": "jest"  
3 },
```

#### 4.4. Realizando Testes Iniciais

Com a instalação completa e a configuração em vigor, você pode querer executar seus primeiros testes para garantir que tudo está funcionando como esperado. Simplesmente digite:



#### 4.5. Configurações Adicionais:

O Jest é conhecido por sua adaptabilidade. Você pode querer explorar mais opções de configuração, como definir um ambiente de teste específico, configurar mocks ou especificar padrões de arquivos de teste. A documentação oficial do Jest é um recurso valioso para essas personalizações avançadas.

### 5. CONCORRÊNCIA EM TESTES UNITÁRIOS (CONCURRENT)

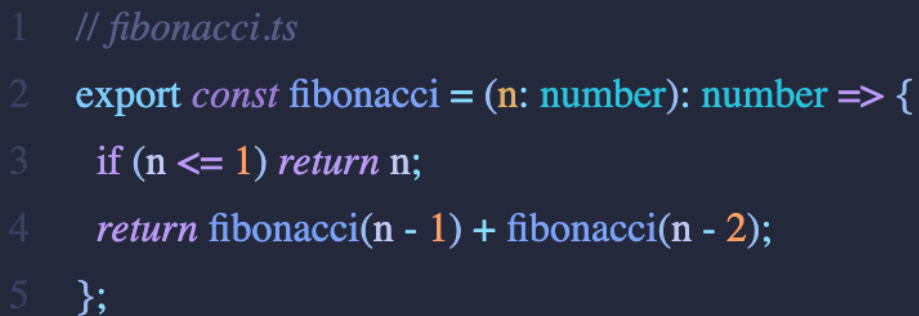
Quando testes são executados em concorrência, eles são executados em paralelo em diferentes threads ou processos. Isso significa que múltiplos testes podem ser executados simultaneamente, aproveitando os múltiplos núcleos da CPU para acelerar a execução total dos testes. Em termos práticos, isso é especialmente útil quando se tem muitos testes que são independentes e não têm dependências entre si.

Testes unitários, em sua natureza, devem ser isolados e não devem depender de estados globais, APIs externas ou serviços. Esta independência é fundamental para garantir que cada teste unitário avalie uma unidade específica de código em isolamento. Dado que esses testes são independentes, faz sentido executá-los em concorrência para acelerar a execução.

Entretanto, em alguns casos, como testes de integração, pode haver uma necessidade de preservar uma ordem específica de execução ou de manter algum estado entre testes. Nesses cenários, os testes devem ser executados em sequência (ou serialmente). Esta sequencialidade garante que os testes são executados em uma ordem específica, e que o estado ou os recursos compartilhados são acessados de maneira controlada.

No Jest, a concorrência não é o comportamento padrão. Em vez disso, testes são executados em sequência por padrão. Se você quiser que os testes sejam executados em concorrência, você precisa especificá-lo explicitamente, com `jest.concurrent` ou `test.concurrent`. Isso dá aos desenvolvedores a flexibilidade de escolher entre concorrência e execução sequencial, com base nas necessidades específicas de seus testes.

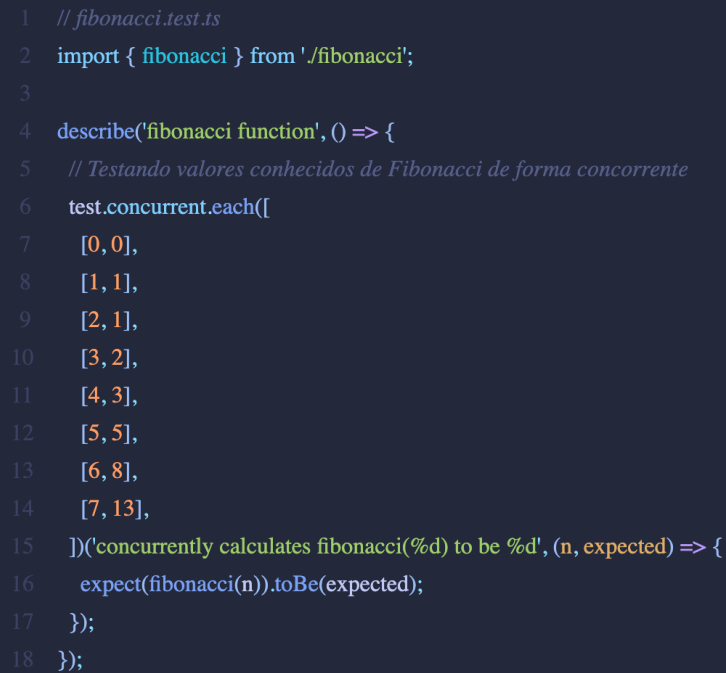
## 5.1 EXEMPLOS DE CÓDIGO



```
1 // fibonacci.ts
2 export const fibonacci = (n: number): number => {
3   if (n <= 1) return n;
4   return fibonacci(n - 1) + fibonacci(n - 2);
5 };
```



### 5.1.1 Teste 1 - Verificar valores conhecidos de Fibonacci



```
1 // fibonacci.test.ts
2 import { fibonacci } from './fibonacci';
3
4 describe('fibonacci function', () => {
5   // Testando valores conhecidos de Fibonacci de forma concorrente
6   test.concurrent.each([
7     [0, 0],
8     [1, 1],
9     [2, 1],
10    [3, 2],
11    [4, 3],
12    [5, 5],
13    [6, 8],
14    [7, 13],
15  ])('concurrently calculates fibonacci(%d) to be %d', (n, expected) => {
16    expect(fibonacci(n)).toBe(expected);
17  });
18 });
```

#### Neste teste:

- Utilizamos `jest.concurrent.each` para rodar testes de forma concorrente com múltiplos conjuntos de dados.
- Cada item no array (por exemplo, `[0, 0]` ou `[7, 13]`) representa um conjunto de dados, onde o primeiro valor é a entrada (`n`) e o segundo valor é o valor esperado da função `fibonacci(n)`.
- Para cada conjunto de dados, o teste verifica se a saída da função `fibonacci` para o valor de entrada `n` é igual ao valor esperado. Por exemplo, para o conjunto de dados `[7, 13]`, o teste verifica se `fibonacci(7)` retorna 13.
- A string `'concurrently calculates fibonacci(%d) to be %d'` é o nome do teste, onde `%d` é um placeholder que é substituído pelos valores em cada conjunto de dados. Portanto, para o conjunto `[7, 13]`, o nome do teste será `'concurrently calculates fibonacci(7) to be 13'`.

## 6 CONCLUSÃO

Ao longo deste artigo, exploramos a profundidade e a amplitude do framework de teste Jest, elucidando sua crescente popularidade no mundo moderno do desenvolvimento de software. Mergulhamos nas características intrínsecas que tornam o Jest uma ferramenta indispensável para muitas equipes de controle de qualidade.

O Jest se destaca não apenas por sua robustez e adaptabilidade, mas também pela maneira intuitiva como aborda os testes. Desde sua documentação exemplar até sua flexível API e fácil configuração, cada aspecto do Jest parece refletir um compromisso com a excelência e a otimização do processo de teste. Porém, como qualquer ferramenta, é essencial reconhecer que, embora o Jest seja excepcionalmente adequado para muitos projetos, ele pode não ser a escolha ideal para todos. No entanto, para aqueles que buscam um framework que é moderno, eficiente e altamente personalizável, o Jest emerge como uma opção formidável.

Em conclusão, ao considerar ferramentas de teste para projetos JavaScript, o Jest certamente merece sua atenção. Sua capacidade de transformar e melhorar processos de teste é inegável, e sua contínua evolução promete manter sua posição na vanguarda dos frameworks de teste por muitos anos vindouros.

## REFERÊNCIAS

JEST. API documentation: `test.concurrent(name, fn, timeout)`. Jest, [s.d.]. Disponível em: <https://jestjs.io/docs/api#testconcurrentname-fn-timeout>. Acesso em: 25 de Outubro de 2023.

CIRCLECI. API Testing with Jest. CircleCI Blog, [s.d.]. Disponível em: <https://circleci.com/blog/api-testing-with-jest/>. Acesso em: 25 de Outubro de 2023.

TESTOMAT.IO. Tutorial on how to organize an advanced Jest testing framework. Testomat.io Blog, [s.d.]. Disponível em: <https://testomat.io/blog/tutorial-on-how-to-organize-an-advanced-jest-testing-framework/>. Acesso em: 25 de Outubro de 2023