UNIME – LAURO DE FREITAS
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

JEAN CARLOS ROCHA NASCIMENTO

DESENVOLVIMENTO COM REACT: CÁLCULO DE SOMA DE NÚMEROS UTILIZANDO NEXT.JS

Lauro de Freitas
2024

JEAN CARLOS ROCHA NASCIMENTO

DESENVOLVIMENTO COM REACT: DOCUMENTAÇÃO DA CRIAÇÃO DA PÁGINA WEB PARA SOMA DE NÚMEROS

Trabalho apresentado à disciplina Desenvolvimento com React do Curso Bacharel em Sistemas de Informação da UNIME - Campus Lauro de Freitas, como requisito parcial para a obtenção de nota.

Orientadores:

Prof. Gian Carlo Decarli
Prof. Gian Mauricio Fritsche

Lauro de Freitas

2024

RESUMO

Este trabalho aborda a configuração e utilização do Jest e da React Testing Library para a realização de testes unitários em componentes React. O objetivo principal é capacitar os desenvolvedores a escrever testes que verifiquem a renderização correta dos componentes e a resposta adequada aos eventos do usuário, garantindo a funcionalidade da aplicação.

Para atingir esse objetivo, o primeiro passo consiste na instalação e configuração do Jest e da React Testing Library em um projeto React existente. Em seguida, são apresentados exemplos práticos de como escrever testes unitários, abrangendo desde a renderização de componentes até a simulação de interações do usuário, como cliques e alterações de entrada.

Os resultados obtidos demonstram a eficácia dos testes automatizados na identificação de falhas antes da produção, aumentando a confiabilidade do software. Além disso, enfatizamos a importância de incorporar uma cultura de testes no desenvolvimento de software, que não só melhora a qualidade do código, mas também fornece uma experiência mais robusta para o usuário final.

Conclui-se que a adoção da Biblioteca de Testes Jest e React é fundamental para garantir a integridade dos componentes em projetos React, promovendo práticas de desenvolvimento que priorizem a qualidade e a manutenibilidade do software.

Palavras-chave: Jest, React Testing Library, Testes Unitários, Desenvolvimento de Software, Componentes React.

ABSTRACT

This work addresses the configuration and use of Jest and the React Testing Library to perform unit tests on React components. The main objective is to enable developers to write tests that verify the correct rendering of components and an adequate response to user events, guaranteeing the functionality of the application. To achieve this goal, the first step is to install and configure Jest and the React Testing Library in an existing React project. Next, practical examples of how to write unit tests are presented, ranging from rendering components to simulating user interactions such as clicks and input changes. The results obtained demonstrate the effectiveness of automated tests in identifying faults before production, increasing software reliability. Furthermore, we emphasize the importance of incorporating a testing culture into software development, which not only improves code quality but also provides a more robust experience for the end user. It is concluded that the adoption of the Jest and React Test Library is essential to guarantee the integrity of components in React projects, promoting development practices that prioritize the quality and maintainability of the software.

Keywords: Jest, React Testing Library, Unit Testing, Software Development, React Components.

SUMÁRIO

1- INTRODUÇÃO

No desenvolvimento moderno de software, a garantia de qualidade é uma preocupação primordial que se reflete em diversas práticas de ocorrência. À medida que as aplicações se tornam mais complexas e dinâmicas, a necessidade de implementação de testes automatizados se torna ainda mais evidente. Testes unitários, em particular, desempenham um papel fundamental na identificação de falhas, na validação de funcionalidades e na promoção da confiança na base de código.

O React, uma das bibliotecas JavaScript mais populares para a construção de interfaces de usuário, oferece uma maneira eficiente de desenvolver componentes reutilizáveis. No entanto, garantir que esses componentes funcionem conforme o esperado, especialmente em resposta a eventos do usuário, requer uma abordagem metódica. Nesse contexto, o Jest e a React Testing Library surgem como ferramentas indispensáveis para os desenvolvedores que buscam adotar práticas de teste em seus projetos.

O Jest, um framework de testes de JavaScript, é extremamente reconhecido por sua simplicidade e eficiência. Ele fornece uma configuração inicial fácil e uma série de funcionalidades que facilitam a escrita de testes. Por outro lado, a React Testing Library complementa o Jest ao focar na forma como os usuários interagem com a aplicação, incentivando a escrita de testes que se assemelham à experiência real do usuário.

Esta introdução apresenta a importância de configurar e utilizar Jest e React Testing Library em projetos React, destacando como essas ferramentas importantes para a criação de aplicações mais robustas e confiáveis. Ao longo deste trabalho, serão explorados os passos necessários para a configuração dessas bibliotecas, bem como exemplos práticos que ilustram a escrita de testes unitários eficazes. Dessa forma, os desenvolvedores estarão melhor preparados para enfrentar os desafios do desenvolvimento de software, garantindo a qualidade e a usabilidade de suas aplicações.

2- OBJETIVO

O objetivo deste trabalho é apresentar um guia abrangente sobre a configuração e utilização do Jest e da React Testing Library em projetos React, capacitando desenvolvedores a implementar testes unitários eficazes para seus componentes. Especificamente, procure-se:

1. **Demonstrar a configuração:** Orientar os desenvolvedores na instalação e configuração inicial do Jest e da React Testing Library, garantindo que possam integrá-los de forma eficiente em seus projetos existentes.
2. **Ensinar a Escrita de Testes Unitários:** Proporcionar exemplos práticos de como escrever testes unitários que verificam a renderização correta dos componentes React, bem como sua resposta a interações do usuário, como cliques e alterações de entrada.
3. **Destacar a Importância dos Testes Automatizados:** Discuta a relevância dos testes automatizados no ciclo de desenvolvimento de software, enfatizando como eles são destacados para a melhoria da qualidade do código e a redução de erros na produção.
4. **Promover Boas Práticas de Desenvolvimento:** Incentivar a adoção de uma cultura de testes entre desenvolvedores, estabelecendo a importância de incluir testes como parte integrante do processo de desenvolvimento para garantir a confiabilidade e manutenibilidade das aplicações.

Ao atingir esses objetivos, pretendo oferecer um recurso valioso para desenvolvedores que desejam aprimorar suas habilidades em testes e garantir a robustez de suas aplicações React.

3- DESENVOLVIMENTO

O código apresentado é um conjunto de testes unitários para o componente `Counter`, utilizando a biblioteca Testing Library e o Jest. Cada teste tem um propósito específico e visa validar diferentes funcionalidades do componente. Abaixo, está a explicação detalhada de cada um desses testes:

1. Teste de Renderização Inicial

```
test('deve renderizar o contador inicial como 0', () => {
    render(<Counter />);
    const counterText = screen.getByText(/contador: 0/i);
    expect(counterText).toBeInTheDocument();
});
```

Propósito: Verifique se o contador está renderizado corretamente com o valor inicial de 0.

O que verifica: Este teste garante que, ao renderizar o componente Counter, o texto exibido pelo contador corresponde ao valor inicial esperado. Ele utiliza a função getByTextpara procurar por uma string que contenha "contador: 0". A verificação expect(counterText).toBeInTheDocument()garante que esse texto esteja apresentado no documento, confirmando que o componente inicia com o valor correto.

2. Teste de Incremento

```
test('deve incrementar o contador ao clicar no botão', () => {
    render(<Counter />);
    const incrementButton = screen.getByText(/incrementar/i);
    fireEvent.click(incrementButton);
    const counterText = screen.getByText(/contador: 1/i);
    expect(counterText).toBeInTheDocument();
});
```

Propósito: Validar se o contador aumenta em 1 quando o botão de incrementar é clicado.

O que verifica: Neste teste, após renderizar o componente, ele localiza o botão "incrementar" e simula um clique nesse botão usando fireEvent.click(). Em seguida, verifique se o texto do contador foi atualizado para "contador: 1". A verificação confirma que a funcionalidade de incremento está funcionando corretamente, assegurando que o estado do componente muda como esperado após a interação do usuário.

3. Teste de Decremento

```
test('deve decrementar o contador ao clicar no botão', () => {
    render(<Counter />);
    const incrementButton = screen.getByText(/incrementar/i);
    fireEvent.click(incrementButton);
    fireEvent.click(incrementButton);
    const decrementButton = screen.getByText(/decrementar/i);
    fireEvent.click(decrementButton);
    const counterText = screen.getByText(/contador: 1/i);
    expect(counterText).toBeInTheDocument();
});
```

Propósito: Garantir que o contador diminua em 1 ao clicar no botão de decrementar.

O que verifica: Este teste começa renderizando o componente Countere, inicialmente, simula dois cliques no botão de incrementar, o que aumenta o

contador para 2. Em seguida, localize o botão de decrementar e simule um clique nele. Após essa interação, verifique se o texto do contador foi atualizado para "contador: 1". Essa verificação garante que a funcionalidade de decremento seja inovadora corretamente e que o estado do contador reflita a interação do usuário.

4- CONCLUSÃO

Esses testes oferecem uma cobertura essencial para as funcionalidades básicas do componente `Counter`, garantindo que ele funcione conforme o esperado em diferentes situações. Através da renderização inicial, do incremento e do decremento, os testes validam a lógica interna do componente e garantem que ele responda especificamente às interações do usuário. A utilização da Jest e React Testing Library nesse contexto não só melhora a confiabilidade do código, mas também contribui para uma experiência do usuário mais consistente e sem erros.