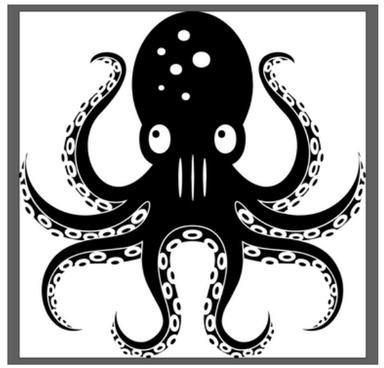
Testes unitários no React

Com React Testing Library e Jest



A ideia por trás dos testes unitários no front é renderizar, em memória, o componente que será testado e verificar se os elementos que deveriam aparecer na tela/dom estarão lá. É possível simular interações com o usuário, como por exemplo: click, mouse hover e input de texto.

React Testing Library

React Testing Library é um conjunto de utilitários que permitem testar componentes React sem depender dos detalhes de implementação.

Princípios do React Testing Library

Tentamos expor apenas métodos e utilitários que o encorajem a escrever testes que se assemelhem a como suas páginas da web são usadas.

Jest

Jest é um test runner JavaScript que permite a você acessar o DOM através do jsdom. Mesmo o jsdom sendo apenas uma aproximação de como um navegador funciona, é bom o suficiente para testar componentes React.

Exemplo

Neste exemplo é renderizado um componente e foi feita uma verificação simples se os elementos estão mesmo na tela

```
it('Deve mostrar as opções quando for aberto', () => {
    const { getByText } = render(
        <Dropdown
            title={title}
            options={options}
        />
    );

    const openButton = getByText(title);
    userEvent.click(openButton);

    expect(screen.getByText(options[0])).toBeInTheDocument();
    expect(screen.getByText(options[1])).toBeInTheDocument();
    expect(screen.getByText(options[2])).toBeInTheDocument();
    expect(screen.getByText(options[3])).toBeInTheDocument();
    expect(screen.getByText(options[4])).toBeInTheDocument();
});
```

Exemplo

Neste exemplo, é verificado que um conteúdo NÃO está na tela.

```
it('Deve começar fechado', () => {
    render(
        <Dropdown
            title={title}
            options={options}
        />
    );

    expect(screen.queryByText(options[0])).not.toBeInTheDocument();
    expect(screen.queryByText(options[1])).not.toBeInTheDocument();
    expect(screen.queryByText(options[2])).not.toBeInTheDocument();
    expect(screen.queryByText(options[3])).not.toBeInTheDocument();
    expect(screen.queryByText(options[4])).not.toBeInTheDocument();
});
```

A diferença entre queryBy, findBy e getBy

findBy:

- **Quando a correspondência é encontrada: retorna uma resolved promise.**
- **Quando a correspondência não é encontrada: retorna uma rejected promise.**
- **Útil quando se deseja testar elementos que não estejam no document.**

getBy:

- **Quando a correspondência é encontrada: Retorna o nó HTML correspondente a busca.**
- **Quando a correpondência não é encontrada: Retorna um erro/excessão.**

queryBy:

- **Quando a correspondência é encontrada: Retorna o nó HTML correspondente a busca.**
- **Quando a correpondência não é encontrada: Retorna null.**

Mockando retorno de API

Foi utilizado um pacote auxiliar chamado nock. O nock intercepta as requisições para a url especificada e permite que o usuário forneça um retorno customizado.

```
it('Mockando a lista renderizada (mockando chamada de api)', async () => {
    nock('https://pokeapi.co/api/v2')
        .defaultReplyHeaders({
            'access-control-allow-origin': '*',
        })
        .get('/pokemon?limit=151')
        .reply(200, requestMock);

    render(<PokemonList />);

    const liPokemon = await waitFor(async () => screen.findByTestId('pokemon-id-0'));

    expect(liPokemon).toBeInTheDocument();
    expect(liPokemon).toHaveTextContent('Rafael');
});
```