React Testing Library

Bartosz Potrykus 3fT

Co to jest?

- React Testing Library to biblioteka do testowania jednostkowego aplikacji napisanych w React.js. Jest ona zaprojektowana tak, aby pomagać programistom testować komponenty React w sposób, który odzwierciedla interakcje użytkownika z aplikacją.

Aplikacja

- Do zrozumienia RTL przysłuży nam przygotowana przez ze mnie aplikacja znajdujący się w linku poniżej:
- https://github.com/BartoszPotrykuss/ReactTestingLibrary-kalkulator

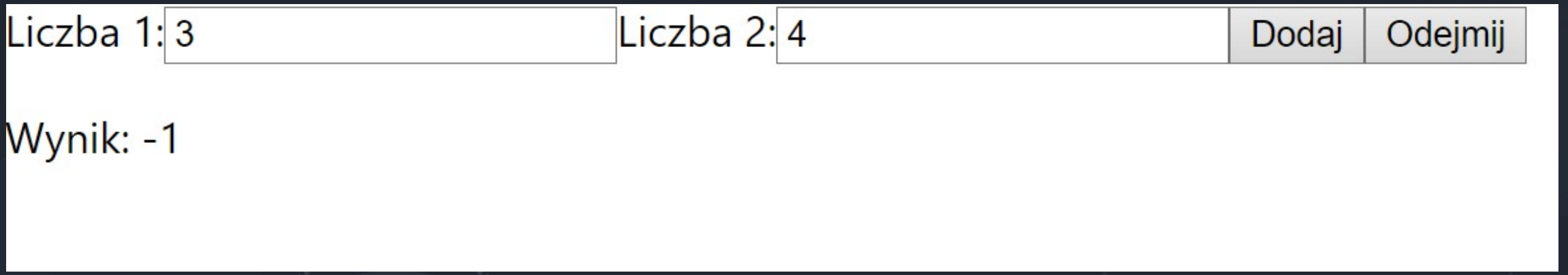
Instalacja

Aby zainstalować RTL do swojej aplikacji należy otworzyć terminal lub powershell w swoim projekcie, a następnie wpisać poniższą komendę:

```
npm install --save-dev @testing-library/react
```

Konfiguracja

- W pliku setupTests.js należy zaimportować bibliotekę wklejając poniższy kod:
- import '@testing-library/jest-dom';



Aplikacja

- Nasza aplikacja to prosty kalkulator z dwoma inputami typu number i dwoma buttonami, których zadaniem jest dodać lub odjąć liczby.

```
import React from 'react';
import { act, render, screen, fireEvent, waitFor } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import App from './App';
```

Importy

- Przed stworzeniem testów zaimportujmy potrzebne rzeczy do pliku App.test.js, w którym będziemy tworzyć testy.

```
// Testuje, czy kalkulator renderuje się z domyślnymi wartościami
test('renders calculator with initial values', () => {
  render(<App />);

  // Pobiera elementy z drzewa DOM na podstawie ich ról i etykiet
  const liczba1Input = screen.getByRole('spinbutton', { name: /liczba 1/i });
  const liczba2Input = screen.getByRole('spinbutton', { name: /liczba 2/i });
  const dodajButton = screen.getByRole('button', { name: /dodaj/i });
  const odejmijButton = screen.getByRole('button', { name: /odejmij/i });
  const wynikText = screen.getByText(/wynik/i);

  // Sprawdza, czy elementy są widoczne w drzewie DOM
  expect(liczba1Input).toHaveValue(0);
  expect(liczba2Input).toHaveValue(0);
  expect(dodajButton).toBeInTheDocument();
  expect(odejmijButton).toBeInTheDocument();
  expect(wynikText).toHaveTextContent('Wynik: 0');
});
```

Pierwszy test- domyślne wartości

- Render komponentu App
- Użycie screen.getByRole by pobrać elementy z drzew DOM na podstawie ich ról i etykiet.
- Użycie screen.getByText by pobrać element zawierający tekst: "wynik" ignorując wielkie litery za pomocą "i".
- Sprawdzenie czy dwa pierwsze inputy mają domyślnie wartość zero
- Sprawdzenie czy obydwa buttony są widoczne w drzewie DOM
- Sprawdzenie czy paragraf z wynik posiada domyślnie tekst: "Wynik: 0".

Drugi test – test odejmowania- fireEvent

```
// Testuje, czy odejmowanie działa poprawnie
test('performs subtraction correctly', async () => {
  render(<App />);

  // Pobiera elementy z drzewa DOM na podstawie atrybutów data-testid
  const liczba1Input = screen.getByTestId("liczba1");
  const liczba2Input = screen.getByTestId("liczba2");
  const odejmijButton = screen.getByTestId("odejmij");

  // Symuluje zmiany wartości i kliknięcie przycisku
  fireEvent.change(liczba1Input, { target: { value: '10' } });
  fireEvent.change(liczba2Input, { target: { value: '4' } });
  fireEvent.click(odejmijButton);

  const wynikText = screen.getByText(/wynik/i);

  // Oczekuje na zmiany i sprawdza, czy wynik jest poprawny
  await waitFor(() => {
    expect(wynikText).toHaveTextContent('Wynik: 6');
  });
});
```

- Render komponentu App
- Pobranie wartości za pomocą screen.getByTestId.

•Zwróć uwagę na to że w pliku App.js do inputów są przypisane data-testid.

•c) Za pomocą fireEvent.change zmieniamy wartości elementów podanych jako pierwszy argument funkcji. Drugi argument to zmiana wartości za pomocą target oraz value.

•d) Za pomocą fireEvent.click symulujemy kliknięcie przycisku.

•e) Pobieramy element z wynikiem.

•f) Za pomocą asynchronicznej funkcji (dlatego dodajemy await przed funkcją oraz async w definicji
test) waitFor czekamy na zmiany.

•g) Sprawdzamy czy odejmowanie zostało zrobione prawidłowo.

```
// Testuje, czy dodawanie działa poprawnie
test('performs addition correctly', async () => {
  render(<App />);

  // Pobiera elementy z drzewa DOM na podstawie atrybutów data-testid
  const liczba1Input = screen.getByTestId("liczba1");
  const liczba2Input = screen.getByTestId("liczba2");
  const dodajButton = screen.getByTestId("dodaj");

  // Symuluje interakcję użytkownika przy użyciu userEvent
  await act(async () => {
    userEvent.type(liczba1Input, '5');
    userEvent.type(liczba2Input, '3');
    userEvent.click(dodajButton);
  });

  const wynikText = screen.getByText(/wynik/i);

  // Oczekuje na zmiany i sprawdza, czy wynik jest poprawny
  await waitFor(() => {
    expect(wynikText).toHaveTextContent('Wynik: 8');
  });
});
```

Trzeci test – test dodawania - userEvent

- a) Render komponentu App
- b) Pobranie wartości za pomocą screen.getByTestId.
- c) Definicja metody act, która pomaga w zarządzaniu asynchronicznymi operacjami w testach
- c) Za pomocą userEvent.type zmieniamy wartości elementów podanych jako pierwszy argument funkcji. Drugi argument to wartość.
- d) Za pomocą userEvent.click symulujemy kliknięcie przycisku.
- e) Pobieramy element z wynikiem.
- f) Za pomocą asynchronicznej funkcji (dlatego dodajemy await przed funkcją oraz async w definicji test) waitFor czekamy na zmiany.
- g) Sprawdzamy czy odejmowanie zostało zrobione prawidłowo.

Jak sprawdzić testy?

W terminalu wpisujemy: "npm test"

```
PASS src/App.test.js
  √ renders calculator with initial values (79 ms)
  √ performs subtraction correctly (12 ms)
  √ performs addition correctly (23 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.634 s, estimated 2 s
Ran all test suites related to changed files.
```

Inne możliwości szukania elementów

- getByText
- getByRole
- getByLabelText
- getByPlaceholderText
- getByAltText
- getByDisplayValue

- queryByText
- queryByRole
- queryByLabelText
- queryByPlaceholderText
- queryByAltText
- queryByDisplayValue

GetBy vs queryBy

Reakcja na nieznaleziony element:

- **getBy** rzuca błąd, jeśli element nie zostanie znaleziony.
- **queryBy** zwraca **null**, jeśli element nie zostanie znaleziony, bez wywoływania błędu.

Użycie w testach:

- **getBy** jest stosowane, gdy oczekujemy, że element istnieje i chcemy, aby jego nieobecność spowodowała awarię testu.
- **queryBy** jest używane, gdy chcemy sprawdzić, czy element istnieje, ale jego nieobecność nie powinna skutkować awarią testu.

Inne funkcje sprawdzające

- toBeDisabled
- toBeEnabled
- toBeEmpty
- toBeEmptyDOMElement
- toBeInTheDocument
- toBeInvalid
- toBeRequired
- ToBeValid
- toContainElement
- toContainHTML
- toHaveDescription
- toHaveAttribute
- toHaveClass
- toHaveFocus
- toHaveFormValues
- toHaveStyle
- toHaveTextContent
- toHaveValue
- toHaveDisplayValue
- toBeChecked
- ToBePartiallyChecked
- toBeVisible

fireEvent vs userEvent

- userEvent dostarcza wygodne metody do symulowania interakcji użytkownika, a fireEvent stanowi podstawowy zestaw narzędzi do ogólnego celu. Wybór między nimi zależy od potrzeb testów oraz preferencji programisty. W wielu przypadkach userEvent jest bardziej czytelny i łatwiejszy do utrzymania, zwłaszcza dla bardziej skomplikowanych scenariuszy interakcji.