JS-Session

Testing React Components

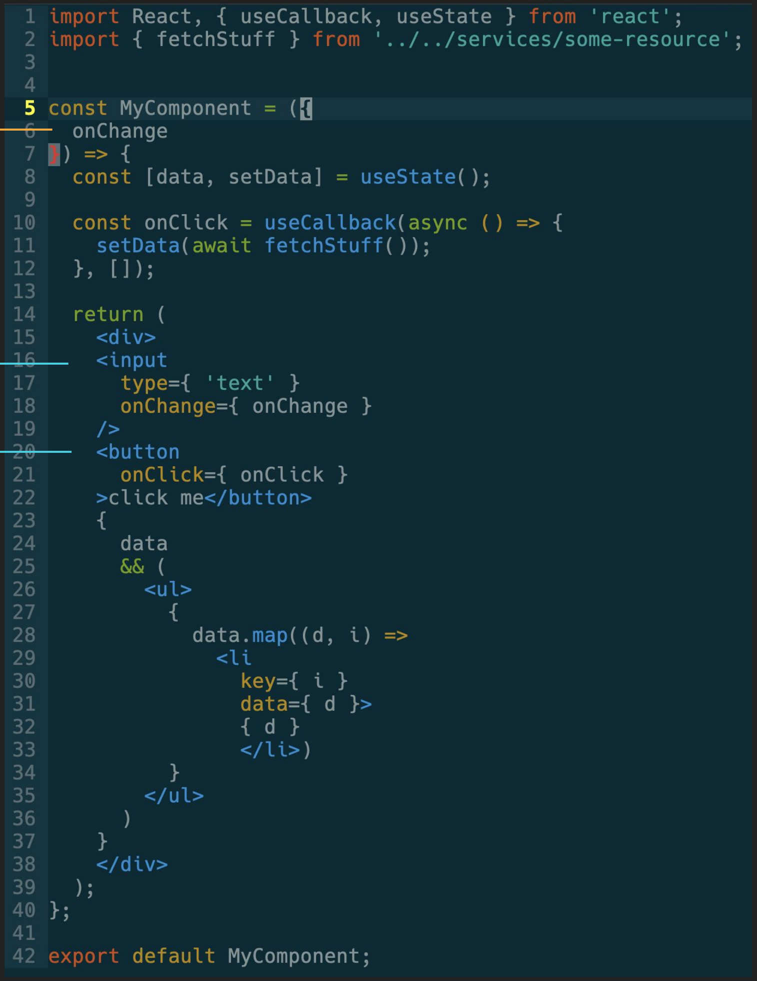
What to test?

Treat the component like a black box.

→ Test only public interface

Most components provide two interfaces:

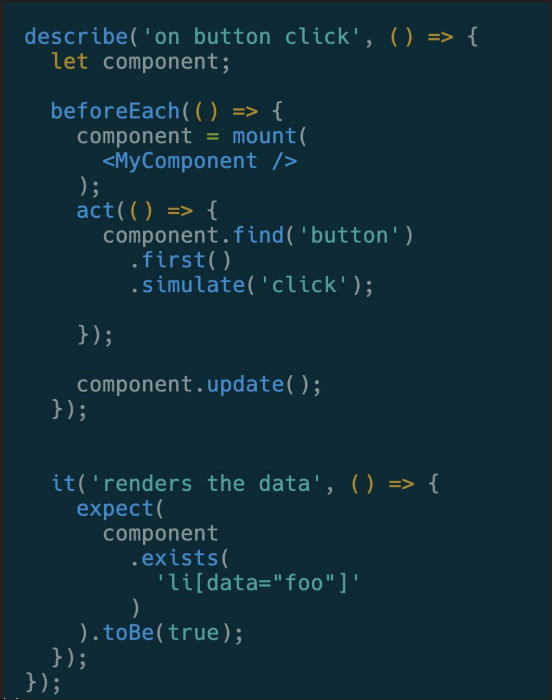
- Props
  - Allow the parent to interact with the component
- Interactive UI elements
  - Allow the user to interact with the component



```
import React, { useCallback, useState } from 'react';
import { fetchStuff } from '../../services/some-resource';


const MyComponent = ({
  onChange
}) => {
  const [data, setData] = useState();

  const onClick = useCallback(async () => {
    setData(await fetchStuff());
  }, []);

  return (
    <div>
    <input
      type={ 'text' }
      onChange={ onChange }
    />
    <button
      onClick={ onClick }
    >click me</button>
    {
      data
      && (
        <ul>
          {
            data.map((d, i) =>
              <li
                key={ i }
                data={ d }>
                { d }
                </li>)
          }
        </ul>
      )
    }
    </div>
  );
};

export default MyComponent;
```

Testing user interactions → `simulate()`

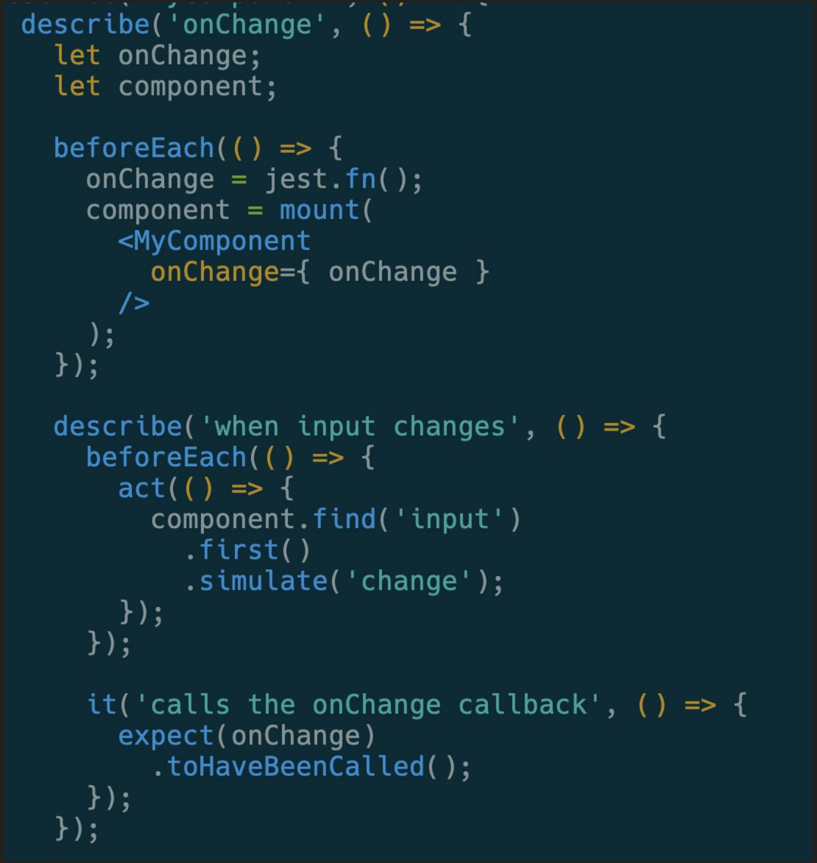
- Emulate user actions and assert on resulting actions.
  - Two common scenarios:
    - Interaction triggers visual changes
      → use snapshots
      → use `find()`, `exists()`, etc. to verify desired UI changes
    - Interaction triggers background task (e.g. API call).
      → use mocks
      → use `toHaveBeenCalled()`, etc. to verify expected functions/services/… were invoked.



```
describe('on button click', () => {
  let component;

  beforeEach(() => {
    component = mount(
      <MyComponent />
    );
    act(() => {
      component.find('button')
        .first()
        .simulate('click');

    });

    component.update();
  });


  it('renders the data', () => {
    expect(
      component
        .exists(
          'li[data="foo"]'
        )
    ).toBe(true);
  });
});
```

Testing props

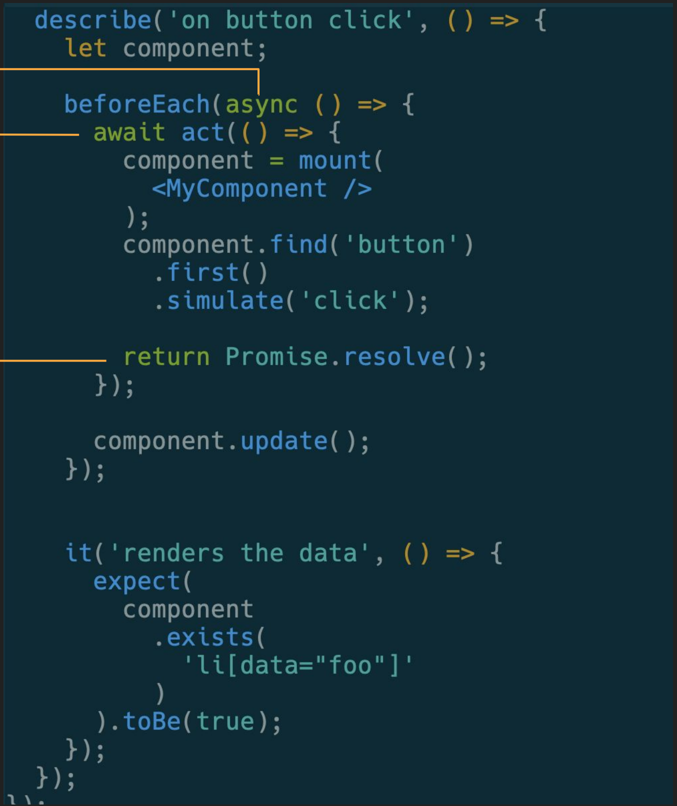
- Test every prop your component offers.
- Use mock functions to test callbacks.



```
describe('onChange', () => {
  let onChange;
  let component;

  beforeEach(() => {
    onChange = jest.fn();
    component = mount(
      <MyComponent
        onChange={ onChange }
      />
    );
  });

  describe('when input changes', () => {
    beforeEach(() => {
      act(() => {
        component.find('input')
          .first()
          .simulate('change');
      });
    });

    it('calls the onChange callback', () => {
      expect(onChange)
        .toHaveBeenCalled();
    });
  });
```

Testing asynchronous code → async/await

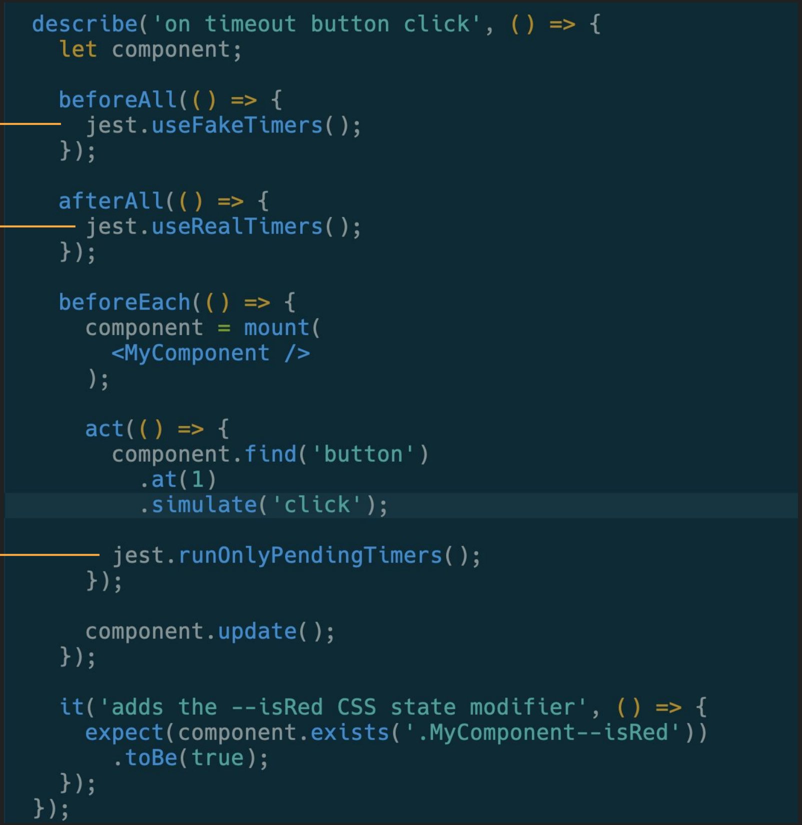
- async(hronous) test setup
- (a)wait for (inter)act(ions)
- If act(ions) are not asynchronous by nature return a Promise



```
describe('on button click', () => {
  let component;

  beforeEach(async () => {
    await act(() => {
      component = mount(
        <MyComponent />
      );
      component.find('button')
        .first()
        .simulate('click');

      return Promise.resolve();
    });

    component.update();
  });


  it('renders the data', () => {
    expect(
      component
        .exists(
          'li[data="foo"]'
        )
    ).toBe(true);
  });
});
```

Testing asynchronous code with timers

- Use Jest timer mocks
- Don't forget to cleanup timer-mocks, else they may affect other tests.
- Use `runOnlyPendingTimers()` to progress timers.



```
describe('on timeout button click', () => {
  let component;

  beforeAll(() => {
    jest.useFakeTimers();
  });

  afterAll(() => {
    jest.useRealTimers();
  });

  beforeEach(() => {
    component = mount(
      <MyComponent />
    );

    act(() => {
      component.find('button')
        .at(1)
        .simulate('click');

      jest.runOnlyPendingTimers();
    });

    component.update();
  });

  it('adds the --isRed CSS state modifier', () => {
    expect(component.exists('.MyComponent--isRed'))
      .toBe(true);
  });
});
```

Resources

- Jest Documentation: https://jestjs.io/docs/en/getting-started
- Enzyme Documentation: https://enzymejs.github.io/enzyme/docs/api/
- Code samples: https://github.com/haensl/js-session-testing