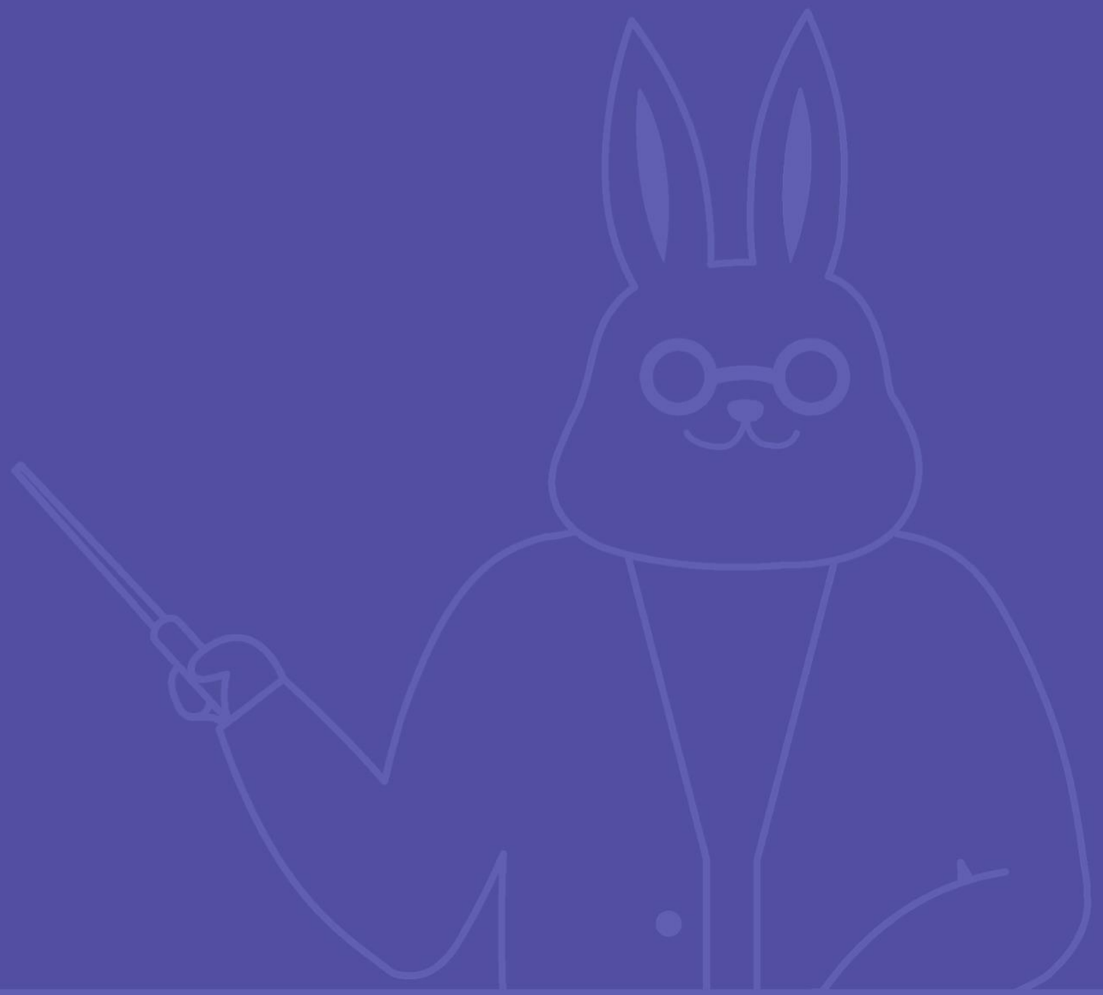
React 심화 II

02 React 테스팅

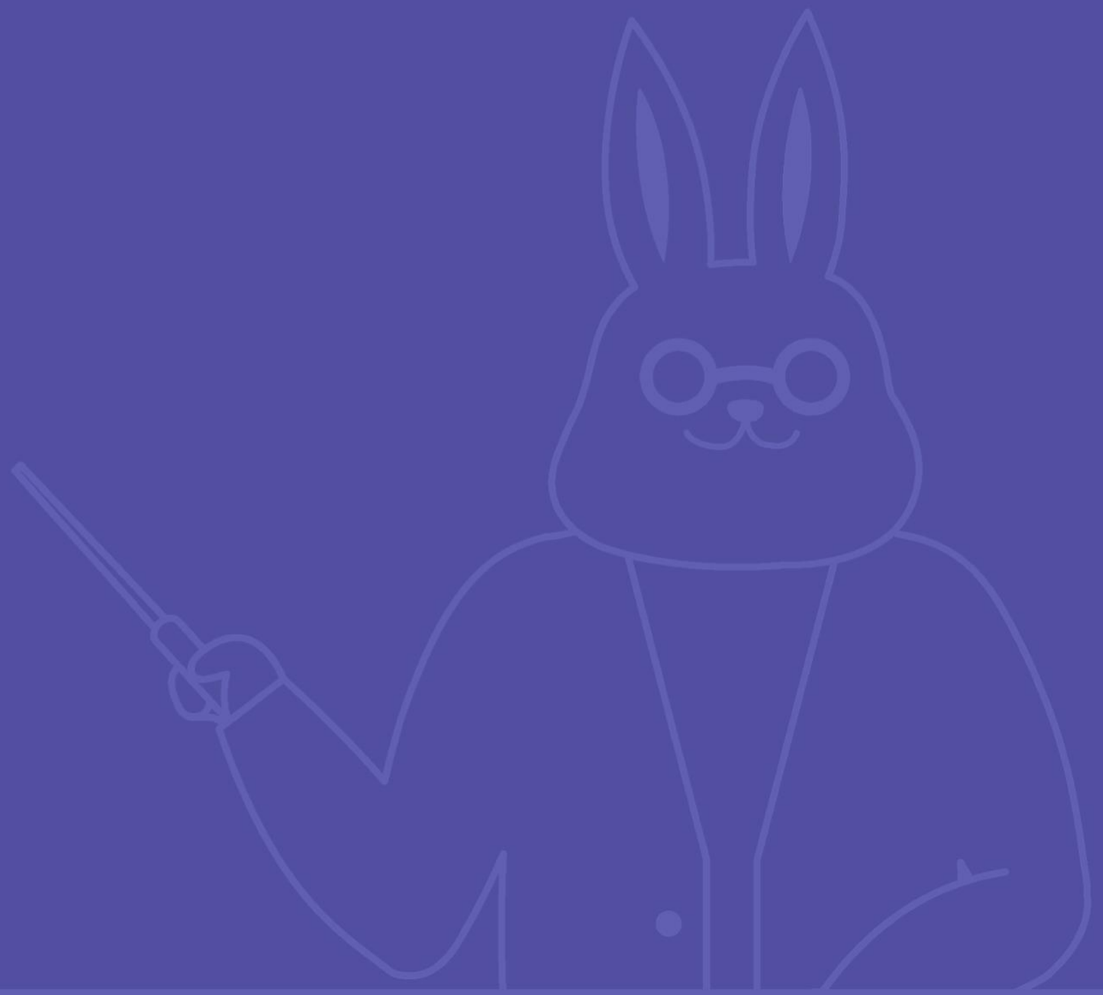


/* elice */

목차

01

React 테스팅



코드 테스트가 필요한 경우

- 코드를 작성하고 나면, 원하는 대로 동작하는지 알기 위해 테스트를 함.
- 코드에 버그가 있으면, 어떤 상황에서 버그가 발생하는지를 알기 위해 테스트를 함.
- 코드를 리팩토링하면, 원래대로 동작하는지 테스트함.

코드 테스트가 필요한 경우

- 리액트 앱의 컴포넌트가 늘어날수록, 컴포넌트끼리 서로 영향을 미치는 경우가 많아짐.
- 특정 코드가 수정되면, 어떤 컴포넌트에 에러가 발생할 수 있음.

테스팅 코드 작성의 이점

- 언급한 상황들에 대한 테스팅 코드를 작성하여, 미연의 에러를 방지.
- TDD(Test Driven Development) 등의 방법론을 적용하여 생산성을 향상.
- 테스트가 늘어나면서 테스트 코드가 구현 코드에 대한 문서화가 됨.
- 테스트가 용이하게 코드를 작성하여, 코드 품질과 신뢰성을 높임.

테스터블한 코드 작성하기

- 코드가 영향을 미치는 범위를 최대한 줄인다. 사이드 이펙트를 줄임.
- 하나의 함수가 너무 많은 일을 하지 않게 함.
- 기능들을 작게 분리함.

End-to-end Testing

- 유저가 어떤 시나리오를 가지고 그 시나리오의 end-to-end로 잘 동작하는지 테스트함.
- 필요한 경우 웹서버, 데이터베이스를 실행함.
- 범위가 너무 넓어서 에러가 발생했을 때 특정 기능이 안 된다는 것은 알 수 있지만, 정확히 어떤 부분에 문제가 생겼는지는 알기 힘듦.
- ex) 유저가 회원가입 후, 로그인하여 유저 정보 페이지를 볼 수 있는지 테스트하는 경우.

주요 테스팅 용어

- Mocking - 특정 동작을 흉내 내는 것.
  ex) 실제 API를 호출하는 게 아니라, 가짜 payload를 반환하는 mocking function을 만듦.
- Stubbing - 더미를 채워 넣는 것.
  ex) Child 컴포넌트를 렌더링하지 않고, 대신 그 자리에 등을 채워 넣음.

테스팅의 구성

- setup - 테스트하기 위한 환경을 만든다. mock data, mock function 등을 준비함.
- expectation – 원하는 테스트 결과를 만들기 위한 코드를 작성함.
- assertion – 정말 원하는 결과가 나왔는지를 검증함.

테스팅의 구성

Code

```
const transformUser = (user) => {
  const { name, age, address } = user
  return [name, age, address]
}
```

테스팅의 구성

Code

```
test('Test transformUser', () => {
  // setup
  const mockUser = { name : 'testName', age: 20, address: 'testAddress' }

  // expectation
  const result = transformUser(mockUser)

  // assertion
  expect(result).toBe(['testName', 20, 'testAddress'])
})
```

React 화이트박스 테스팅, 블랙박스 테스팅

- 화이트박스 테스팅은 컴포넌트 내부 구조를 미리 안다고 가정하고 테스트 코드를 작성.
- 블랙박스 테스팅은 컴포넌트 내부 구조를 모른 채 어떻게 동작하는지에 대한 테스트 코드를 작성.

테스팅의 범위에 따른 분류

- Unit 테스팅
- Integration 테스팅
- end-to-end(e2e) 테스팅

Unit Testing

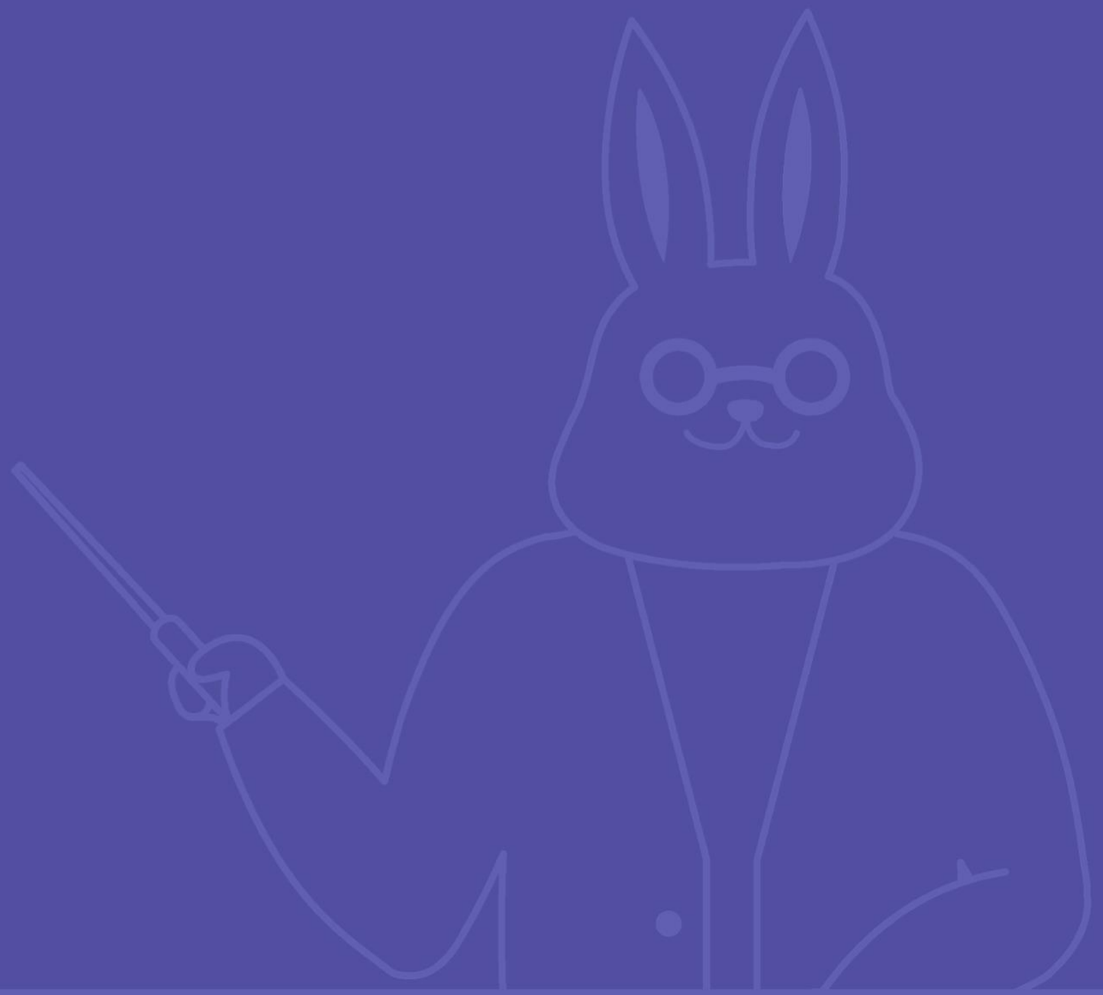
- 다른 부분과 분리된 작은 코드를 만들고 그것을 테스트함.
- 작은 코드는 function, module, class 등을 의미.
- 각 부분이 원하는 대로 동작함을 보장하기 위함.
- 테스트는 서로 분리되어야 함.
- ex) 특정 컴포넌트가 데이터에 따라 잘 렌더링되는지를 테스트하는 경우.

Integration Testing

- 앱의 특정 부분이 동작하는지 테스트함.
- ex) 여러 컴포넌트가 한꺼번에 동작하거나, 어떤 페이지의 부분이 잘 동작하는지를 테스트하는 경우.
- ex) react-router, redux 등이 특정 컴포넌트와 함께 잘 동작하는지를 테스트하는 경우.

02

jest



jest

- facebook에서 오픈소스화한 테스팅 프레임워크.
- assertion 함수들, test runner, mock 라이브러리 등을 모두 제공.
- create-react-app에서 기본적으로 사용됨.
- 사용성이 좋고, 가장 인기 있는 프로젝트.

jest의 핵심 기능들

- assertion matchers
- async assertion
- mock functions
- testing lifecycle functions
- grouping
- snapshot testing

Assertion matchers

- jest는 풍부한 matcher를 제공하여, 여러 상황에서 match를 체크함.
- expect()는 expectation object를 리턴한다. 이 object의 메서드를 이용해 여러 매칭 상황을 assert함.

Async Assertion

- 비동기 상황의 테스트를 처리할 수 있는 여러 방법을 제공함.
- callback, promise, async/await을 모두 활용할 수 있음.

Mock functions

- mock function을 만듦.
- 모듈 전체를 mocking 함.
- 라이브러리 전체를 mocking 함.

Lifecycle functions

- 각 테스트의 시작과 끝, 전체 테스트의 시작과 끝에 원하는 작업을 할 수 있음.
- beforeEach, afterEach, beforeAll, afterAll 함수를 활용함.
- describe 블록 안에 사용하면 별도의 scope를 가짐.

Grouping

- describe 함수를 이용해 여러 test() 함수를 논리적으로 나눔.
- describe 함수 안에 describe 함수가 중첩될 수 있음.

Snapshot Testing

- 특정 함수, 모듈, 컴포넌트 등의 결과를 serializable한 형태의 snapshot으로 저장하고, 추후 변경이 발생했을 때 이전의 snapshot과 새로운 snapshot을 비교하여 변경이 발생했는지를 추측함.
- jest의 주요 기능으로, 코드의 변경이 컴포넌트의 렌더링 결과에 영향을 미치는지를 파악하기에 적합함.

jest 함수의 실행 순서

- beforeAll, beforeEach, afterEach, afterAll의 순서로 Lifecycle 함수들이 실행됨.
- 다만, describe 블록 안에 있는 before-*, after-* 함수는 해당 블록의 범위 안에서 실행됨.
- describe 함수는 모든 test() 함수 이전에 실행된다. 따라서 test() 함수들은 순차적으로 한꺼번에 실행됨.

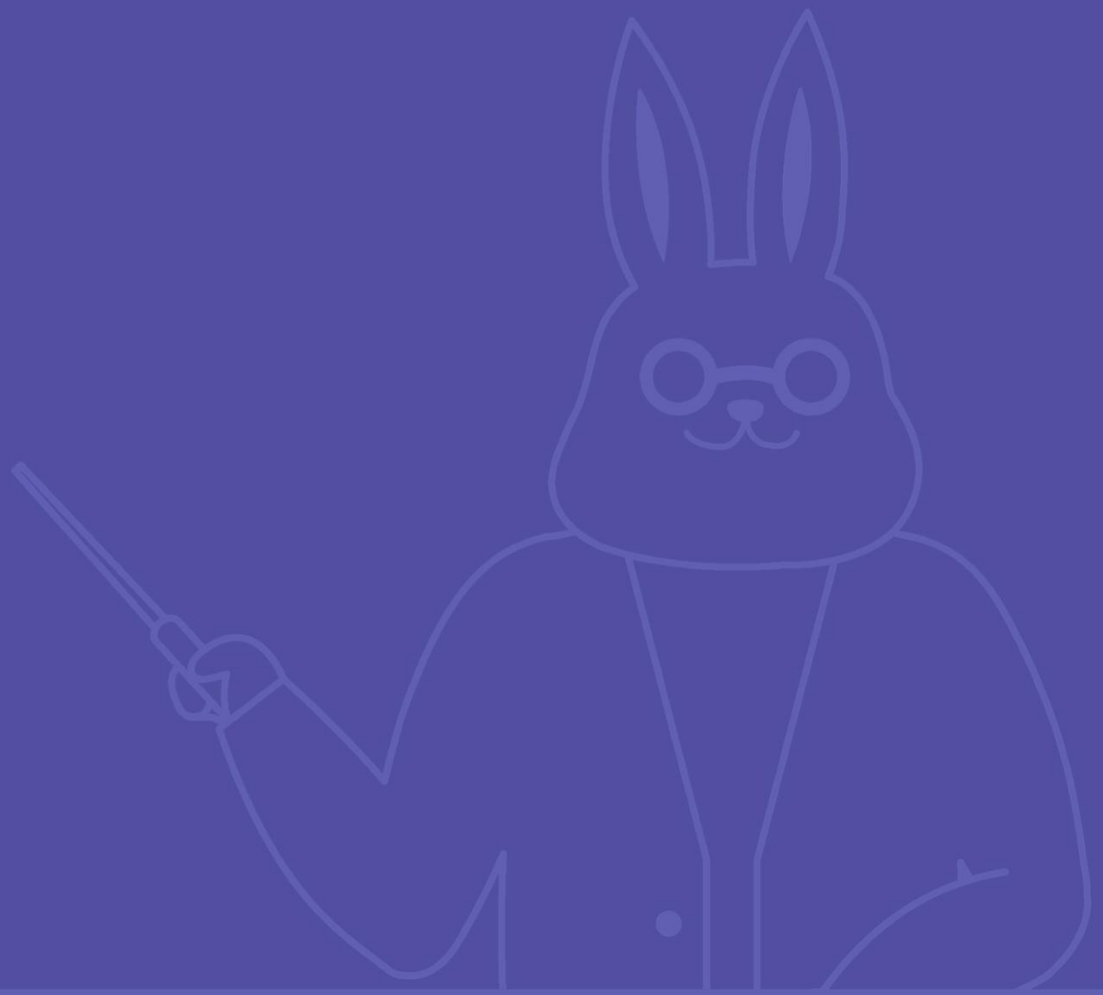
테스팅의 구성

Code

```
test('User component', () => {
  const mockProps = {
    name: 'test-username',
    age: 20
  }
  const { container } = render(<User {...mockProps} />)
  expect(container.firstChild).toMatchSnapshot()
})
```

03

jest 활용



Assertion Matchers 활용

- toBe()
- toEqual()
- toContain()
- toMatch()
- toThrow()

Assertion Matchers 활용

Code

```
function isPythagorean(a, b, c) {
  return a * a + b * b === c * c
}

function createTodo(id, title, content) {
  return { id, title, content }
}

function transformUser(user) {
  const { name, age, address } = user
  return [name, age, address]
}
```

Assertion Matchers 활용

Code

```
test('Should 3, 4, 5 pythagorean', () => {
  expect(isPythagorean(3, 4, 5)).toBe(true)
})

test('Should 3, 4, 6 not pythagorean', () => {
  expect(isPythagorean(3, 4, 6)).toBe(false)
})
```

Assertion Matchers 활용

Code

```
test('Should create user', () => {
  const id = 1, title = "Test todo", content = "Test content";
  expect(createUser(id, title, content)).toEqual({ id, title, content })
})

test('Should create user', () => {
  const id = 1, title = "Test todo", content = "Test content";
  expect(createUser(id, title, content)).title).toMatch("Test todo")
})
```

Assertion Matchers 활용

Code

```
test('Should contain name after transformUser', () => {
  const user = { name: "test name", age: 20, address: "test address" }
  expect(transformUser(user)).toContain("test name")
})

test('Should contain name after transformUser', () => {
  const user = { name: "test name", age: 20, address: "test address" }
  expect(transformUser(user)).not.toContain(30)
})
```

Async assertion

- callback 패턴의 경우, test()함수가 제공하는 done() 함수를 활용하여 콜백이 끝나고 done()을 호출. 에러가 발생하면 done()의 인자로 에러를 넘김.
- Promise 패턴의 경우 async/await을 활용하거나, Promise를 리턴.

Async assertion

Code

```
function isPythagoreanAsync(a, b, c) {
  return new Promise(resolve => {
    setTimeout(() => {
      const result = isPythagorean(a, b, c)
      if (result) return resolve(result)
      reject(new Error("Not pythagorean"))
    }, 500)
  })
}
```

Async assertion

Code

```
test('Should 3, 4, 5 be pythagorean async', (done) => {
  isPythagoreanAsync(3, 4, 5).then(done).catch(done)
})

test('Should 3, 4, 5 be pythagorean async', () => {
  return expect(isPythagoreanAsync(3, 4, 5)).resolves.toBe(true)
})

test('Should 3, 4, 6 be not pythagorean async', () => {
  return expect(isPythagoreanAsync(3, 4, 5)).rejects.toBe("Not pythagorean")
})
```

Mock functions

- jest.fn()을 활용하여, mock function 객체를 만듦.
- mockReturnValueOnce() 등으로 리턴하는 값을 임의로 조작한다. 여러번 호출하면, 순서대로 세팅된 값을 반환함.
- mockResolvedValue() 로 promise가 resolve하는 값을 조작함.
- jest.mock()으로 특정 모듈을 mocking함.

Mock functions - assertion

- toHaveBeenCalled - 이 함수가 호출되었는지 검증.
- toHaveBeenCalledWith(arg1, arg2, ...) - 이 함수가 특정 인자와 함께 호출되었는지 검증.
- toHaveBeenLastCalledWith(arg1, arg2, ...) - 마지막으로 특정 인자와 함께 호출되었는지 검증.

Mock functions

Code

```
test('Should 3, 4, 5 be pythagorean async', (done) => {
  isPythagoreanAsync(3, 4, 5).then(done).catch(done)
})

test('Should 3, 4, 5 be pythagorean async', () => {
  return expect(isPythagoreanAsync(3, 4, 5)).resolves.toBe(true)
})

test('Should 3, 4, 6 be not pythagorean async', () => {
  return expect(isPythagoreanAsync(3, 4, 5)).rejects.toBe("Not pythagorean")
})
```

Lifecycle functions

- beforeEach
- afterEach
- beforeAll
- afterAll

Lifecycle functions

Code

```
beforeEach(() => {
  setupMockData()
})

afterEach(() => {
  clearMockData()
})
```

Grouping

- describe()
- test()

Grouping

Code

```
describe('This is group 1', () => {
  describe('This is inner group 1', () => {
    test('Test 1', () => {})
  })

  describe('This is inner group 2', () => {
    test('Test 2', () => {})
  })
})
```

Snapshot testing

- toMatchSnapshot()을 호출하면, 기존에 스냅샷이 없었을 경우 .snap 파일을 만듦.
- 기존 스냅샷이 있을 경우, 새로운 스냅샷과 비교하여 변경 사항이 있으면 테스트는 실패함.
- toMatchInlineSnapshot()을 호출하면 별도의 스냅샷 파일을 만들지 않음.
  이 경우, 어떻게 스냅샷이 쓰였는지를 하나의 파일 안에서 알 수 있게 됨.

Snapshot testing

Code

```
test('Snapshot test form', () => {
  const { container } = render(<MyForm />)
  expect(container.firstChild).toMatchSnapshot()
})

test('Snapshot test form', () => {
  const { container } = render(<MyForm />)
  expect(container.firstChild).toMatchInlineSnapshot()
})
```

Snapshot testing

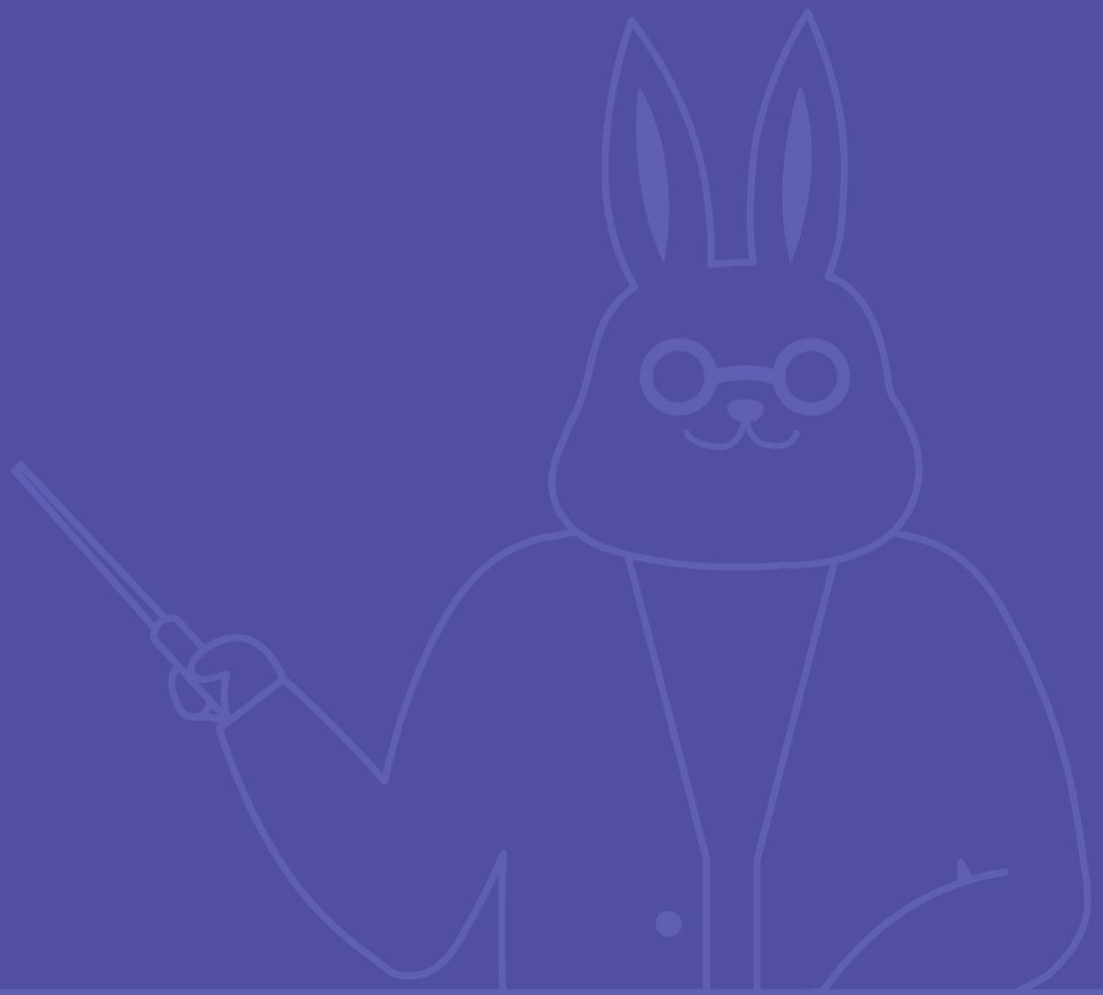
Code

```
expect(container.firstChild).toMatchInlineS
napshot(`
  <div>
    <div>
      <img
        alt="Placeholder"
      src="https://via.placeholder.com/150"
      />
    </div>
    <form
      id="programming-form"
    >
```

```
      <fieldset>
        <legend>
          Basic Info
        </legend>
        <label
          for="username"
        >
          Username
        </label>
        ...
```

04

react-testing-library



react-testing-library

- 테스트가 소프트웨어가 사용되는 모습을 닮을수록, 테스트를 더욱 신뢰할 수 있게 됨.
- The more your tests resemble the way your software is used, the more confidence they can give you.

react-testing-library

- React 컴포넌트의 특정 메서드나 상태를 테스트하는 게 아니라,
  실제 유저가 사용하는 방식대로 테스트하는 접근.
- 유저가 페이지에서 어떤 DOM 요소에 접근하는 방법을 흉내냄.
- 이 방식으로 테스트 코드를 작성하면, 내부 구현이 바껴도 테스트가 깨지지 않음.

react-testing-library

- React 컴포넌트가 렌더링한 결과에 대한 접근만 가능함.
- 쿼리는 내부 상태나 내부 메서드에 접근할 수 없게 설계됨.

react-testing-library의 쿼리 - get

- getBy 관련 쿼리는 원하는 요소를 찾지 못할 경우나 여러개의 요소를 찾을 경우 에러를 던짐.
- getAllBy 관련 쿼리는 여러 요소를 찾아 배열을 반환한다. 원하는 요소를 찾지 못할 경우 에러를 던짐.
- 원소가 반드시 페이지에 존재해야만 하는 경우 활용함.

react-testing-library의 쿼리 - find

- findBy 관련 쿼리는 원하는 원소가 없더라도 비동기적으로 기다림.
- 여러 원소를 찾거나, 정해진 timeout동안 찾지 못하면 에러를 던짐.
- findAllBy 관련 쿼리는 여러 원소를 검색해 배열을 반환한다. 정해진 timeout동안 찾지 못하면 에러를 던짐.
- Promise를 리턴하며, 실패시 reject, 성공시 resolve.
- 어떤 유저의 동작 후에 등장하는 원소 등을 테스트 하고자할 때 활용함.

react-testing-library의 쿼리 - query

- queryBy 관련 쿼리는 getBy와 비슷하게 원소를 찾아 반환하나, 못찾을 경우 에러를 던지지 않고 null을 반환함. 여러 원소를 찾으면 에러를 던짐.
- queryAllBy 관련 쿼리는 getAllBy와 비슷하게 여러개의 원소를 찾아 배열로 반환하나, 하나도 찾지 못하면 에러 대신 빈 배열을 반환함.
- 특정 요소를 찾을 수 없음을 assertion의 기준으로 둘 때 활용됨.

react-testing-library의 쿼리 - container

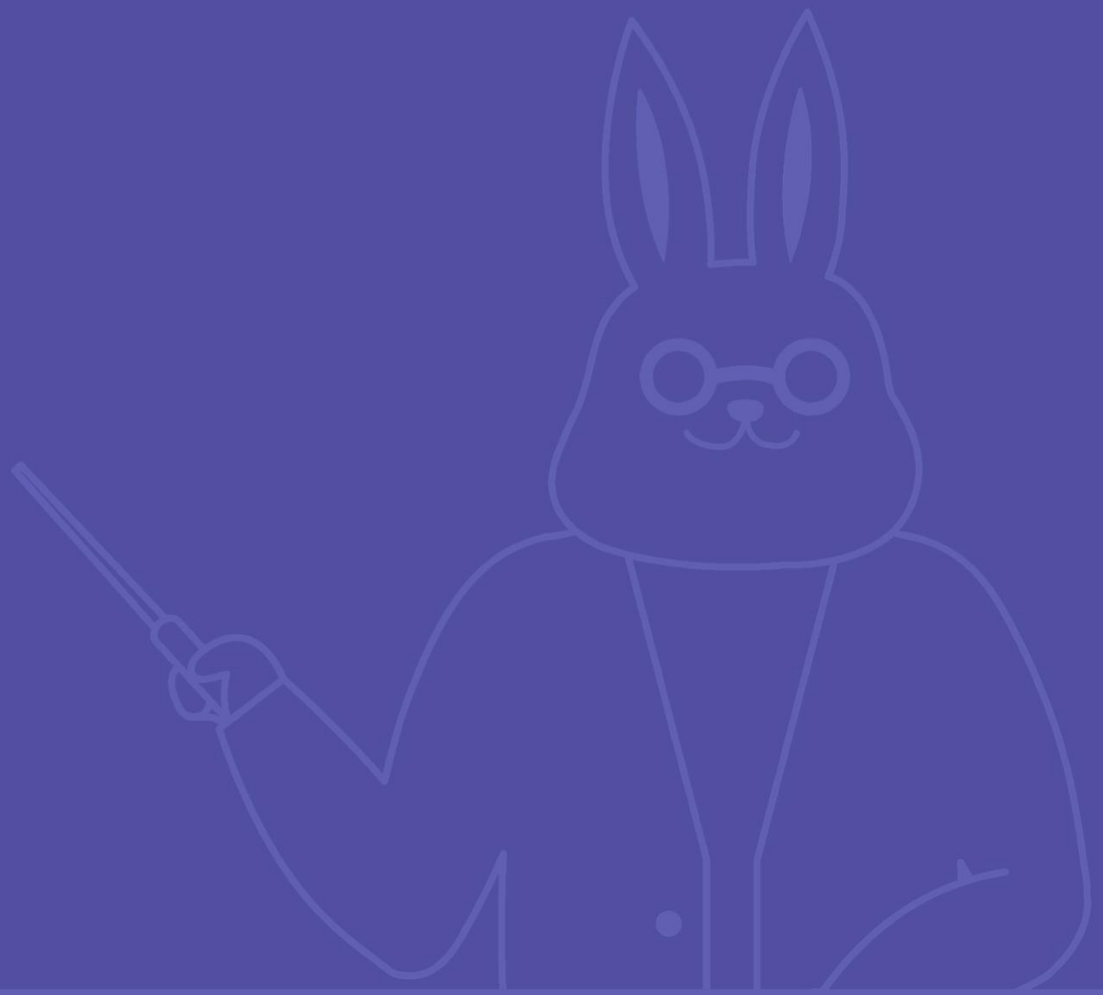
- container는 컴포넌트를 렌더한 결과를 감싸는 원소.
- queryselector(), querySelectorAll()을 이용해 selector 문법으로 원소를 선택할 수 있음.

jest-dom

- react-testing-library는 jest를 확장하여, 좀더 쓰기 편한 assertion을 제공함.
- toBeInTheDocument(), toHaveValue(), toBeDisabled(), toBeVisible() 등, DOM 테스팅에 특히 유용한 assertion 메서드를 제공함.

05

쿼리의 우선순위



쿼리의 우선순위

- 유저가 페이지를 이동하는 방식에 가까운 쿼리일수록 우선순위가 높음.
- 접근성 높은 HTML을 작성할수록 테스트가 용이한 코드가 됨.

쿼리의 우선순위 - ByRole

- accessibility tree에 있는 요소들을 기준으로 원소를 찾음.
- 유저가 웹 페이지를 사용하는 방식을 가장 닮은 쿼리.
- 동일한 role을 가진 경우, accessible name을 이용해 원소를 구별함.
- 임의로 role 혹은 aria-*을 부여하는 것을 지양함.

쿼리의 우선순위 - ByRole

- 자주 사용되는 Role - button, checkbox, listitem, heading, img, form, textbox, link
- 자주 사용되는 accessible name

  button - 텍스트

  label - 텍스트

  a - 텍스트

  img - alt 텍스트

쿼리의 우선순위 - ByRole

TestForm.jsx

```
function TestForm() {
  const formRef = useRef();
  const handleSubmit = (e) => {
    e.preventDefault();
    formRef.current.reset();
  };
  return (
    <form onSubmit={handleSubmit} ref={formRef}>
      <label htmlFor="username">Username</label>
      <input id="username" type="text" name="username" />

      <input type="submit" value="Submit" />
    </form>
  )
}
```

쿼리의 우선순위 - ByRole

Code

```
test('제출 버튼을 찾아 클릭하면, Username 인풋이 비워진다.', () => {
  const { getByRole } = render(<TestForm />)
  const usernameInput = getByRole('textbox', { name : 'Username' })
  const submitButton = getByRole('button', { name : 'Submit' })
  userEvent.type(usernameInput, "test username")
  userEvent.click(submitButton)

  expect(usernameInput).toHaveValue("")
})
```

쿼리의 우선순위 - Text

- 유저가 볼 수 있는 Text 값을 기준으로 쿼리를 찾음.
- ByLabelText – label과 연관된 원소를 찾음.
- ByPlaceholderText – placeholder와 연관된 원소를 찾음.
- ByText – 주어진 Text와 연관된 원소를 찾음.
- ByDisplayValue – input, textarea, select 등의 value를 기준으로 원소를 찾음.

쿼리의 우선순위 - Text

Code

```
test('제출 버튼을 찾아 클릭하면, Username 인풋이 비워진다.', () => {
    const { getByLabelText, getByText } = render(<SimpleTestForm />);
    const usernameInput = getByLabelText("Username");
    const submitButton = getByText("Submit");

    userEvent.type(usernameInput, "test username");
    userEvent.click(submitButton);

    expect(usernameInput).toHaveValue("");
})
```

쿼리의 우선순위 - semantic queries

- 유저에게 보이지 않지만, 접근성 스펙에 적합한 alt, title을 이용하여 원소를 검색함.
- ByAltText – img, area, input등의 alt 속성으로 원소를 검색함.
- ByTitle – title 속성으로 원소를 검색함.

쿼리의 우선순위 - Test ID

- data-testid 속성을 원하는 원소에 지정하고, 쿼리를 이용해 찾음.
- 유저가 해당 속성을 기반으로 화면의 요소를 찾는 게 아니므로 우선순위가 낮음.
- 다른 쿼리로 테스트를 작성할 수 없을 때 이 쿼리를 백도어로 활용함.

쿼리의 우선순위 - Test ID

Code

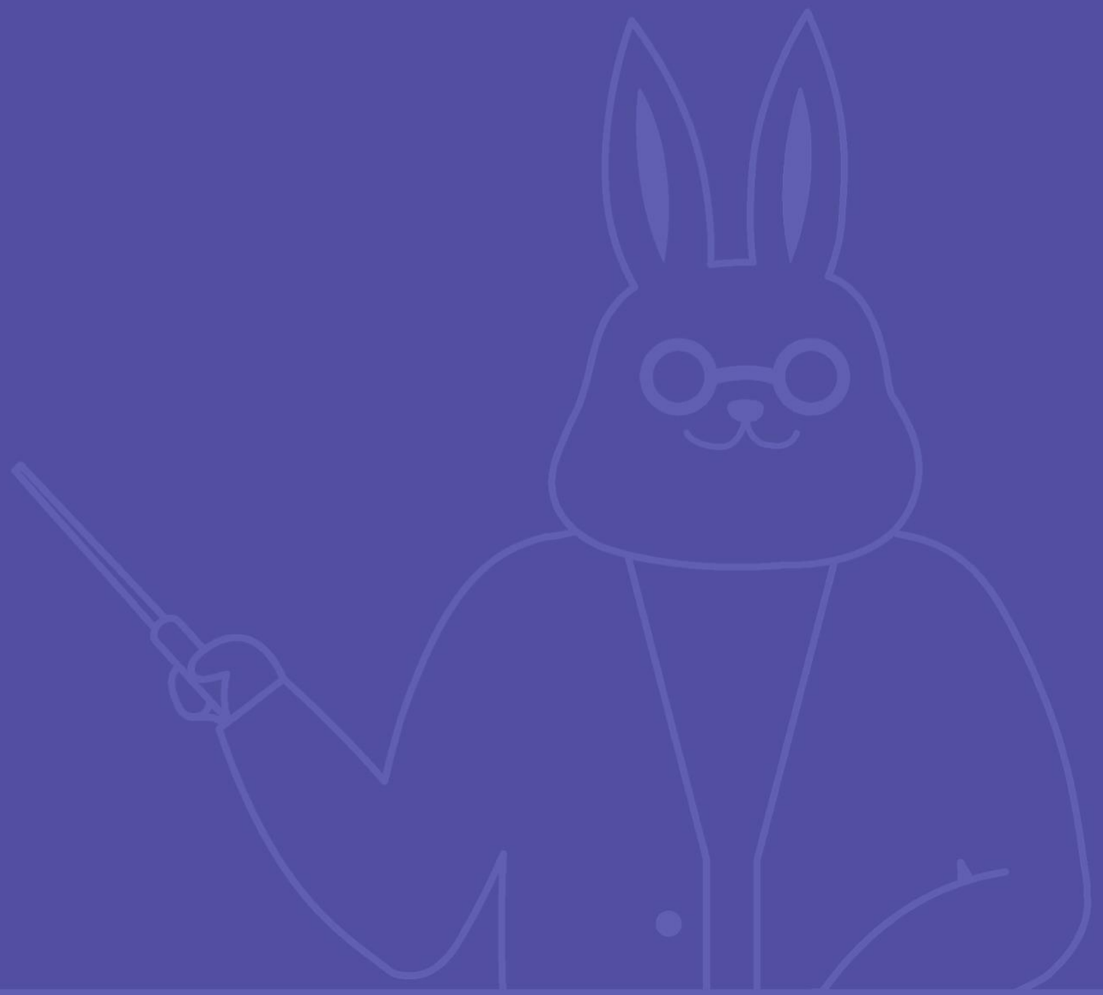
```
test('제출 버튼을 찾아 클릭하면, Username 인풋을 비운다.', () => {
  const { getByTestId } = render(<SimpleTestForm />);
  const usernameInput = getByTestId("username-input");
  const submitButton = getByTestId("submit-button");

  userEvent.type(usernameInput, "test username");
  userEvent.click(submitButton);

  expect(usernameInput).toHaveValue("");
})
```

06

유저 이벤트



user-event

- 내장 이벤트 함수인 fireEvent, createEvent를, 좀더 직관적이고 범용적으로 사용할 수 있도록 만든 라이브러리.
- click, type, keyboard, upload, hover, tab 등 유저가 실제로 웹페이지를 사용하며 만드는 이벤트를 메서드로 제공함.

user-event - click

Code

```
userEvent.click(submitButton)

userEvent.click(submitButton, { shiftKey: true })

userEvent.click(submitButton, { shiftKey: true }, { clickCount: 5 })
```

user-event - click

Code

```
test('숨은 텍스트를 보여준다.', () => {
  const text = "Hidden text!";

  const { getByRole, queryByText } = render(<Expansion text={text} />);
  expect(queryByText("Hidden text!")).toBe(null);

  const showButton = getByRole("button", { name: "Expand" });
  expect(showButton).toBeInTheDocument();

  userEvent.click(showButton);
  expect(queryByText(text)).toBeInTheDocument();
})
```

user-event - click

Code

```
function Expansion({ text }) {
  const [expanded, setExpanded] = useState(false);

  return (
    <div>
      <button onClick={() => setExpanded((b) => !b)}>Expand</button>
      {expanded && <div>{text}</div>}
    </div>
  );
}
```

user-event - type

Code

```
userEvent.type(inputElement, 'react advanced')

userEvent.type(inputElement, 'react{space}advanced{enter}')

await userEvent.type(inputElement, 'react advanced', { delay: 300 })
```

user-event - type

Code

```
test("Typeahead에서 쿼리에 따른 검색 결과를 보인다", () => {
  const mockSearchData = ["kim", "song", "shim", "park", "yoon"];
  const { getByPlaceholderText, getAllByRole } = render(
    <Typeahead searchData={mockSearchData} />
  );

  const inputElement = getByPlaceholderText("Type name...");
  userEvent.type(inputElement, "s");

  expect(getAllByRole("listitem").length).toBe(2);

  userEvent.clear(inputElement);
  expect(getAllByRole("listitem").length).toBe(mockSearchData.length);
});
```

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
김일식

강사
김일식

감수자
-

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io