

# **ALL ABOUT ZEST**



### **'테스트 코드'란 무엇일까?**

코드가 안정적이고 제대로 동작하는지 확인하기 위해 작성하는 코드이다.  
이를 통해 특정 동작을 하는 코드가 예상과 동일하게 결과를 잘 내는지 품질 검사에 사용

### **'테스트 코드'를 작성하면 뭐가 좋을까?**

1. 프로그램의 안정성을 보장
2. 스펙이 추가되어 코드를 수정 시, 테스트 코드가 기존에 존재한다면 기존의 기능과 같은 지 확인 가능
3. 개발 테스트 자동화 / 수동 테스트 최소화

## JEST 란 무엇일까?

All-in-one 테스트 라이브러리 (In JS)

Jest 이전에는 자바스크립트 코드 테스트는 여러가지 테스트 라이브러리를 조합하여 사용

Jest는 라이브러리 하나에, **Test Runner**와 **Test Matcher** 그리고 **Test Mock** 프레임워크까지 제공

## Test Runner

테스트 실행을 조율하고 테스트 결과를 사용자에게 제공하는 역할.

```
test('테스트 코드명', () => {  
  expect(실행값).toBe(기댓값);  
})
```

```
describe("plus test", () => {  
  it("1+2", done => {  
    expect(1 + 2).toEqual(3);  
    done();  
  });  
});
```

## TEST MATCHER

테스트 표현식을 작성할 때 좀 더 문맥적으로 자연스럽게 우아한 문장을 만들 수 있게 보조  
Matcher는 어떤 값들의 상호 일치 여부나 특정한 규칙 준수 여부 등을 판별하기 위해 만들어진 메소드나 객체

- `toEqual()`: 두 값이 같은지 비교하는 Matcher이다. object가 정확히 같은지 판단할 수 있다. javascript에서 `object.is` 와 같다.

```
expect(response).toEqual(value);
```

- `toBeTruthy()`, `toBeFalsy()`: 값이 false인지 true인지 확인하는 Matcher이다.

```
expect(response).toBeTruthy(); // response가 true인지 확인
```

- `toHaveLength()`: object의 길이를 확인하는 Matcher이다.

```
expect(response).toHaveLength(3);
```

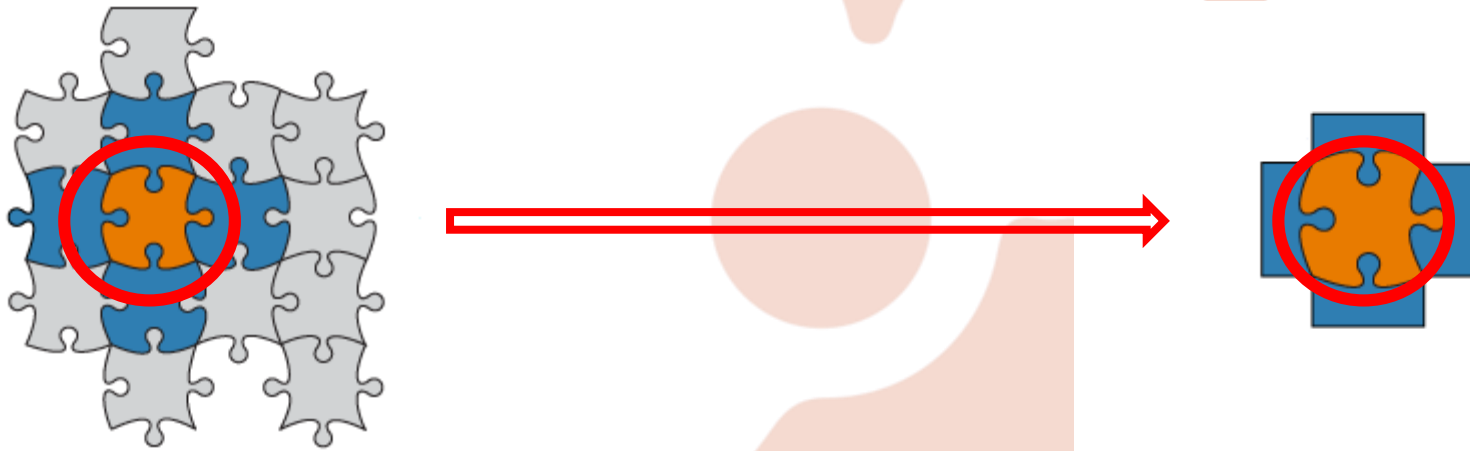
- `toContain()`: array에 포함하는지 확인하는 Matcher이다.

```
expect(response).toContain('hello');
```

- `toThrow()`: 에러를 throw()하는지 확인하는 Matcher이다.

```
expect(() => {  
    drinkFlavor('octopus');  
}).toThrow();
```

## TEST MOCK



우리가 실제 쓰는 프로그래밍

테스트에 필요한 MOCK 객체

Mock?

시간, 비용 등의 Cost가 높거나 객체 서로 간의 의존성이 강해 구현하기 힘들 경우  
가짜 객체를 만들어 사용하는 방법 (상위 개념으로 **테스트 더블**)

## TEST MOCK 용어

### Mock에 대한 기본적인 분류 개념, 테스트 더블

#### 1. 테스트 더블

- 테스트를 진행하기 어려운 경우 이를 대신해 테스트를 진행 할 수 있도록 만들어주는 객체를 말한다.
- Mock 객체와 유사한 의미를 가지며 테스트 더블이 좀더 상위 의미로 사용된다.

#### 2. 더미객체(Dummy Object)

- 단순히 인스턴스화될 수 있는 수준으로만 객체를 구현한다.
- 인스턴스화된 객체가 필요할 뿐 해당 객체의 기능까지는 필요하지 않은 경우에 사용한다.

#### 3. 테스트 스텝(Test Stub)

- 더미 객체 보다 좀더 구현된 객체로 더미 객체가 마치 **실제로 동작하는 것처럼 보이게 만들어 놓은 객체**이다.
- 객체의 특정 상태를 가정해서 만들어 특정 값을 리턴해 주거나 특정 메시지를 출력해 주는 작업을 한다.
- 특정 상태를 가정해서 하드코딩된 형태**이기 때문에 로직에 따른 값의 변경은 테스트 할 수 없다.
- 즉, 어떤 행위가 호출됐을 때 특정 값으로 리턴시켜주는 형태가 Stub이다.

#### 4. 페이크 객체(Fake Object)

- 어려운 상태를 대표할 수 있도록 구현된 객체로 **실제 로직이 구현된 것처럼 보이게 한다.**
- 실제로 DB에 접속해서 비교할 때와 동일한 모양이 보이도록 객체 내부에 구현 할 수 있다.
  - 테스트케이스 작성을 위해서 다른 객체들과의 의존성을 제거하기 위해 사용한다.
  - 페이크 객체를 만들 때 복잡도로 인해서 노력이 많이 들어 갈 경우 적절한 수준에서 구현하거나, Mock 프레임 워크를 사용한다.
  - 페이크 객체를 생성하기 위한 노력이 많이 필요한 경우 실제 객체를 가져와 테스트 한다.

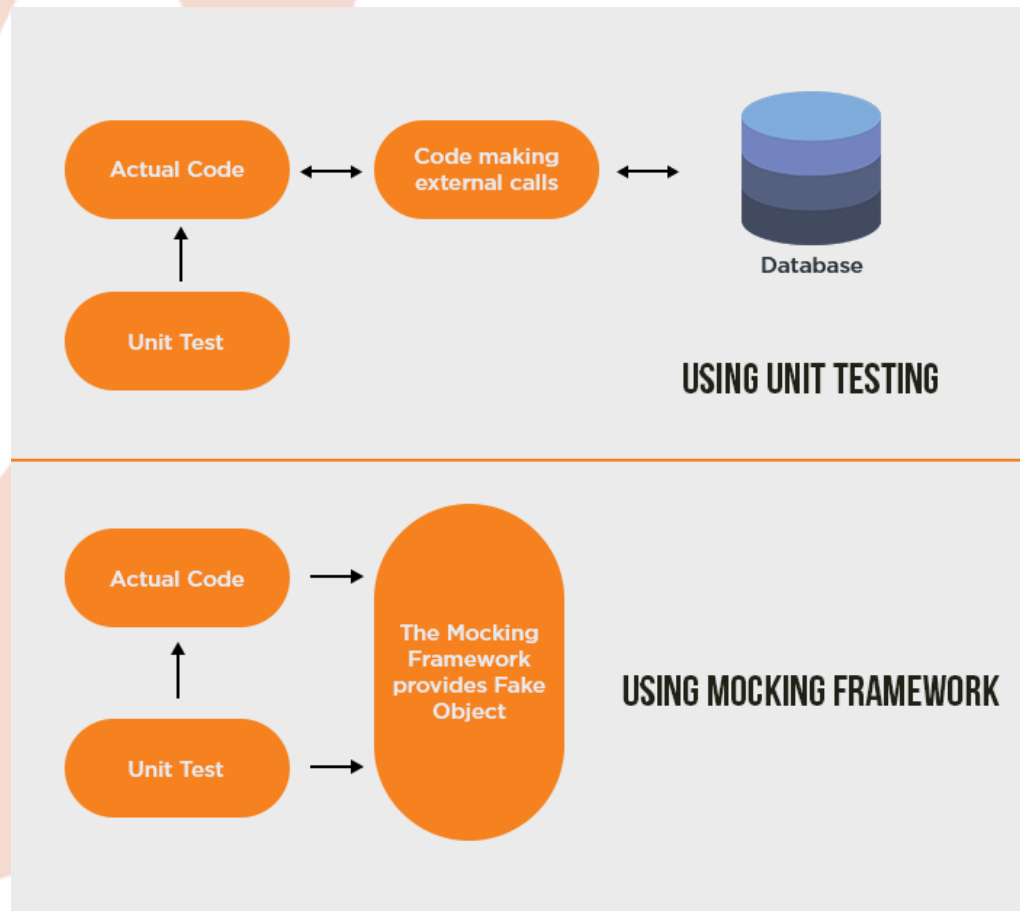
#### 5. 테스트 스파이(Test Spy)

- 테스트에 사용되는 객체, 메소드의 사용 여부 및 정상 호출 여부를 기록하고 요청시 알려준다.
- 테스트 더블로 구현된 객체에 자기 자신이 호출 되었을 때 확인이 필요한 부분을 기록하도록 구현한다.
- 특정 테스트 메서드가 몇번 호출 되었는지 필요한 경우 전역 변수로 카운트를 설정하고 특정 테스트 메서드에 카운트를 올리는 부분을 추가한 후 이 카운트를 가져오는 메서드를 추가한다.
- 특정 메소드가 호출 되었을 때 또 다른 메서드가 실행이 되어야 한다면 같은 행위 기반 테스트가 필요한 경우 사용한다.

#### 6. Mock 객체(Mock Object)

- 행위를 검증하기 위해 사용되는 객체를 지칭하며 수동으로 만들 수도 있고 프레임워크를 통해 만들 수 있다.
- 행위 기반 테스트는 복잡도나 정확성등 작성하기 어려운 부분이 많기 때문에 상태 기반 테스트가 가능하다면 만들지 않는다.
- Mock 객체는 테스트 더블 하위객체로 써의 좁은 의미와 테스트 더블을 포함한 넓은 의미 2가지로 사용 될 수 있다.

이때 Mock Object는 **행위 검증(behavior verification)**에 사용하고, Stub은 **상태 검증(state verification)**에 사용하는 것이다.



## SAMPLE CODE

```
function sum(a, b) {  
  return a + b;  
}  
  
test('1 + 2 = 3', () => {  
  expect(sum(1, 2)).toBe(3);  
});
```



## TEST 대신 IT

이 키워드 말고 it 이라는 키워드를 사용 할 수도 있습니다.  
작동방식은 완전히 똑같습니다.

```
function sum(a, b) {  
  return a + b;  
}  
  
test('1 + 2 = 3', () => {  
  expect(sum(1, 2)).toBe(3);  
});
```

```
it('calculates 1 + 2', () => {  
  expect(sum(1, 2)).toBe(3);  
});
```

## IT을 묶어 DESCRIBE

Describe를 이용해서 여러 테스트 케이스를 묶을 수 있습니다

```
it('calculates 1 + 2', () => {  
  expect(sum(1, 2)).toBe(3);  
});
```

```
describe('sum', () => {  
  it('calculates 1 + 2', () => {  
    expect(sum(1, 2)).toBe(3);  
  });  
  
  it('calculates all numbers', () => {  
    const array = [1, 2, 3, 4, 5];  
    expect(sumOf(array)).toBe(15);  
  });  
});
```

## Express와 연계한 JEST

NPM 모듈인 supertest를 설치하고 적용

```
npm install -D supertest

const app = require("./app");
const request = require("supertest");

describe("express test", () => {
  it("hello world Test", done => {
    request(app)
      .get("/hello")
      .then(res => {
        expect(res.text).toEqual("world");
        done();
      });
  });
});
```

## Express와 연계한 JEST

```
npm install -D supertest
```

```
const app = require("./app");
const request = require("supertest");

describe("express test", () => {
  it("hello world Test", done => {
    request(app)
      .get("/hello")
      .then(res => {
        expect(res.text).toEqual("world");
        done();
      });
  });
});
```

```
> jestTest@1.0.0 test /Users/jake/projects/jestTest
> jest ./*.test.js
```

```
PASS ./app.test.js
  express test
    ✓ hello world Test (6ms)
```

```
Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.109s
Ran all test suites matching /\.\/app.test.js/i.
```

## Express와 연계한 JEST

Api 여러 개 적용 예시

```
describe("express test", () => {
  it("hello world Test", done => {
    request(app)
      .get("/hello")
      .then(res => {
        expect(res.text).toEqual("world");
        done();
      });
  });

  it("bye world Test", done => {
    request(app)
      .get("/bye")
      .then(res => {
        expect(res.text).toEqual("world");
        done();
      });
  });
});
```

## Express와 연계한 JEST

```
describe("express test", () => {  
  it("hello world Test", done => {  
    request(app)  
      .get("/hello")  
      .then(res => {  
        expect(res.text).toEqual("world");  
        done();  
      });  
  });  
  
  it("bye world Test", done => {  
    request(app)  
      .get("/bye")  
      .then(res => {  
        expect(res.text).toEqual("world");  
        done();  
      });  
  });  
});
```

```
> jestTest@1.0.0 test /Users/jake/projects/jestTest  
> jest ./*.test.js  
  
PASS ./app.test.js  
  express test  
    ✓ hello world Test (35ms)  
    ✓ bye world Test (8ms)  
  
Test Suites: 1 passed, 1 total  
Tests:      2 passed, 2 total  
Snapshots:  0 total  
Time:       1.214s  
Ran all test suites matching /\.\/app.test.js/i.
```

## Front-End 예시

원본 코드

```
// displayUser.js
'use strict';

const $ = require('jquery');
const fetchCurrentUser = require('./fetchCurrentUser.js');

$('#button').click(() => {
  fetchCurrentUser(user => {
    const loggedInText = 'Logged ' + (user.loggedIn ? 'In' : 'Out');
    $('#username').text(user.fullName + ' - ' + loggedInText);
  });
});
```

## Front-End 예시

테스트 코드

```
test('displays a user after a click', () => {

  document.body.innerHTML =
    '<div>' +
    '  <span id="username" />' +
    '  <button id="button" />' +
    '</div>';

  fetchCurrentUser.mockImplementation(cb => {
    cb({
      fullName: 'Johnny Cash',
      loggedIn: true,
    });
  });

  // Use jquery to emulate a click on our button
  $('#button').click();
  expect(fetchCurrentUser).toBeCalled();
  expect($('#username').text()).toEqual('Johnny Cash - Logged In');
});
```



## Front-End 예시

```
// displayUser.js
'use strict';

const $ = require('jquery');
const fetchCurrentUser = require('./fetchCurrentUser.js');

$('#button').click(() => {
  fetchCurrentUser(user => {
    const loggedInText = 'Logged ' + (user.loggedIn ? 'In' : 'Out');
    $('#username').text(user.fullName + ' - ' + loggedInText);
  });
});
```

```
test('displays a user after a click', () => {

  document.body.innerHTML =
    '<div>' +
    '  <span id="username" />' +
    '  <button id="button" />' +
    '</div>';

  fetchCurrentUser.mockImplementation(cb => {
    cb({
      fullName: 'Johnny Cash', 1. 임의의 값 지정
      loggedIn: true,
    });
  });

  // Use jquery to emulate a click on our button
  $('#button').click(); 2. 버튼 클릭
  expect(fetchCurrentUser).toBeCalled();
  expect($('#username').text()).toEqual('Johnny Cash - Logged In');
}); 3. 데이터 검증
```

## How To Make Unit Test

### 단위테스트 규칙

- **독립적(Independent)**이어야 한다. 어떤 테스트도 다른 테스트에 의존하지 않아야 한다.
- **격리(Isolation)**되어야 한다. Ajax, LocalStorage, UI Event 등 테스트 대상이 의존하는 것을 다른 것으로 대체한다. (= 테스트 더블)
- given, when, then 단계에 따라 테스트 코드를 작성한다.

## How To Make Unit Test

Story. 사용자는 **Swiper**를 생성할 수 있다.

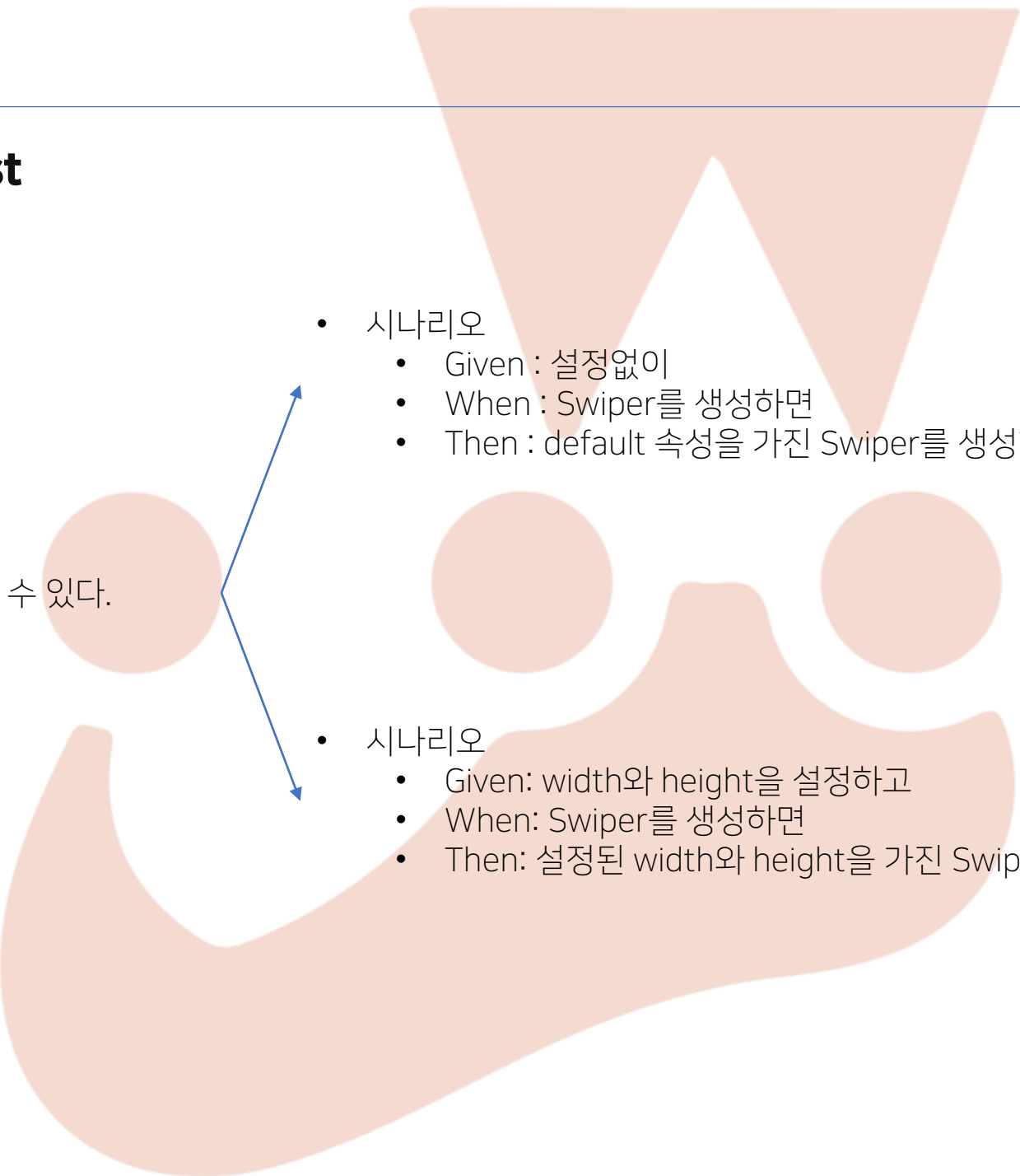
Story. 사용자는 다음 슬라이드로 이동할 수 있다.

Story. 사용자는 이전 슬라이드로 이동할 수 있다.

## How To Make Unit Test

- 사용자는 Swiper를 생성할 수 있다.

## How To Make Unit Test

- 사용자는 Swiper를 생성할 수 있다.
  - 시나리오
    - Given : 설정없이
    - When : Swiper를 생성하면
    - Then : default 속성을 가진 Swiper를 생성한다.
  - 시나리오
    - Given: width와 height를 설정하고
    - When: Swiper를 생성하면
    - Then: 설정된 width와 height를 가진 Swiper를 생성한다.
- 

## How To Make Unit Test

- 사용자는 Swiper를 생성할 수 있다.
  - 시나리오
    - Given : 설정없이
    - When : Swiper를 생성하면
    - Then : default 속성을 가진 Swiper를 생성한다.
  - 시나리오
    - Given: width와 height를 설정하고
    - When: Swiper를 생성하면
    - Then: 설정된 width와 height를 가진 Swiper를 생성한다.
- 사용자는 다음 슬라이드로 이동할 수 있다.
  - 시나리오
    - Given: 다음 슬라이드가 있을 때
    - When: 다음 슬라이드를 요청하면,
    - Then: 다음 슬라이드로 이동한다..
  - 시나리오
    - Given: 다음 슬라이드가 없을 때
    - When: 다음 슬라이드를 요청하면,
    - Then: 현재 슬라이드를 유지한다

## How To Make Unit Test

```
describe('Swiper 컴포넌트', () => {
  describe('사용자는 Swiper를 생성할 수 있다.', () => {
    it('설정없이 Swiper를 생성하면, default 속성을 가진 Swiper를 생성한다.', () => {
      const swiper = shallow(<Swiper />);
      const expected = { slides: [], index: 0, width: null, height: null, duration: 300 };
      expect(swiper.state()).toEqual(expected);
    });

    it('width, height를 설정하고 Swiper를 생성하면, 설정된 width와 height를 가진 Swiper를 생성한다.', () => {
      const swiper = shallow(<Swiper width={500} height={300} />);
      expect(swiper.state().width).toEqual(500);
      expect(swiper.state().height).toEqual(300);
    });
  });

  describe('사용자는 다음 슬라이드로 이동할 수 있다.', () => {
    it('다음 슬라이드가 있을 때 다음 슬라이드를 요청하면, 다음 슬라이드로 이동한다.', () => {
      const slides = ['slide1', 'slide2'];
      const swiper = shallow(<Swiper slides={slides} />);
      swiper.instance().next();
      expect(swiper.state().index).toEqual(1);
    });

    it('다음 슬라이드가 없을 때 다음 슬라이드를 요청하면, 현재 슬라이드를 유지한다.', () => {
      const swiper = shallow(<Swiper />);
      swiper.instance().next();
      expect(swiper.state().index).toEqual(0);
    });
  });

  describe('사용자는 이전 슬라이드로 이동할 수 있다.', () => {
    it('이전 슬라이드가 있을 때 이전 슬라이드를 요청하면, 이전 슬라이드로 이동한다.', () => {
      const slides = ['slide1', 'slide2'];
      const swiper = shallow(<Swiper slides={slides} index={1} />);
      swiper.instance().prev();
      expect(swiper.state().index).toEqual(0);
    });

    it('이전 슬라이드가 없을 때 이전 슬라이드를 요청하면, 현재 슬라이드를 유지한다.', () => {
      const swiper = shallow(<Swiper />);
      swiper.instance().prev();
      expect(swiper.state().index).toEqual(0);
    });
  });
});
```

### MORE ABOUT

Jest로 마이그레이션 가능한 타 라이브러리

Jasmin이나 Mocha를 사용 중이라면, Jest는 대부분 호환 가능하므로 마이그레이션이 덜 복잡합니다

AVA, Expect.js (by Automattic), Jasmine, Mocha, proxyquire, Should.js, Tape, Chai -> Jest 마이그레이션 가능

- 출처

<https://velog.io/@stampid/JEST-%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8-%EC%9C%A0%EB%8B%9B-%ED%85%8C%EC%8A%A4%ED%8A%B8-ykk5krj31z>

<https://medium.com/@jinseok.choi/jest%EB%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%9C-unit-test-%EC%A0%81%EC%9A%A9%EA%B8%B0-420049c16cc8>

<https://hees-dev.tistory.com/57>

<https://jestjs.io/>

<https://velog.io/@velopert/%EC%9E%90%EB%B0%94%EC%8A%A4%ED%81%AC%EB%A6%BD%ED%8A%B8-%ED%85%8C%EC%8A%A4%ED%8C%85%EC%9D%98-%EA%B8%B0%EC%B4%88>

<https://www.crocus.co.kr/1555>