Date0629

마다 React 진행

- Service Function
  - 일정표
  - Calender
  - To do list
  - Customize
    - 커스터마이징 요소
- BM
  - 기본 & 월정액
- Page
  - Home
  - Daily Page
  - Calender
  - Customizing
  - My
- HTTPS 접근
  - Amazon RSA 2048 M01를 사용
- 배포

    - 결국 React build해서 express에 배포한다.

Git & Github 공부

- Git action

Testing

- 리액트 프로젝트에서 TDD(Test Driven Development) : 테스트 주도 개발
- 테스트
    - 직접 일일히 => 힘들어 => 테스트 자동화
    - Unit Test
        - 조그마한 단위로 작성
            - 컴포넌트가 잘 렌더링 된다.
            - 컴포넌트의 특정 함수를 실행하면 상태가 우리가 원하는 형태로 바뀐다
            - 리덕스의 액션 생성 함수가 액션 객체를 잘 만들어 낸다.
            - 리덕스의 리듀서에 상태와 액션객체를 넣어 호출하면 새로운 상태를 잘 만들어 준다
        - Integrated Test
            - 기능들이 전체적으로 잘 동작하는지 확인하기 위해서 사용.
                - 여러 컴포넌트들 렌더링, 서로 상호 작용 확인
                - DOM 이벤트를 발생, 우리의 UI 원하는 변화 잘 발생
                - 리덕스와 연동된 컨테이너 컴포넌트의 DOM에 특정 이벤트 발생했을 때 우리가 원하는 액션이 잘 디스패치

javascript testing

- 테스팅 도구 차이점
    - karma
    - Jasmine
    - Jest (CRA에 자동으로 적용되어 있다)
    - Chai
    - Mocha

TDD

- 실패 => 성공 => 리팩토링
- **실습1**

stats.test.js

```
// stats.js 코드를 test하는 파일 with jest
const stats = require('./stats');
```

```
describe('stats', () => { // describe로 test를 모으자
    if('gets maximum value', ()=>{
        expect(stats.max([1,2,3,4])).toBe(4);
    });
});
```

stats.js

```
exports.max = numbers => {
    let result = numbers[0];
    numbers.forEach(n => {
        if(n > result){
            result = n;
        }
    });
    return result;
};

=> 리팩토링

expors.max = numbers => Math.max(...numbers);
```

- **실습2**

stats.test.js

```
const stats = require('./stats');

describe('stats', () =>{
    it('gets maximum value', () =>{ // test 대신에 it으로 test 진행
        expects(stats.max([1,2,3,4])).toBe(4);
    });
    it('gets minimum value', () =>{
        expect(stats.min([1,2,3,4])).toBe(1);
    });
});
```

stats.js

```
exports.max = numbers => Math.max(...numbers);
exports.min = numbers => Math.min(...numbers);
```

- **실습3**

stats.test.js

```
const stats = require('./stats');

describe('stats', ()=>{
    it('gets maximum value', () =>{
        expect(stats.max([1,2,3,4])).toBe(4);
    });

    it('gets minimum value', () => {
        expect(stats.min([1,2,3,4])).toBe(1);
    });
    if('gets average value', () =>{
        expect(stats.avg([1,2,3,4,5])).toBe(3);
    });
});
```

stats.js

```
exports.max = numbers => Math.max(...numbers);
exports.min = numbers => Math.min(...numbers);
exports.avg = numbers => {
    const sum = numbers.reduce((acc, current) => acc + current, 0);
    return sum/numbers.length;
};
```

- **실습4**

stats.test.js

```
const stats = require('./stats');

describe('stats', () => {
  it('gets maximum value', () => {
    expect(stats.max([1, 2, 3, 4])).toBe(4);
  });
  it('gets minimum value', () => {
    expect(stats.min([1, 2, 3, 4])).toBe(1);
  });
  it('gets average value', () => {
    expect(stats.avg([1, 2, 3, 4, 5])).toBe(3);
  });
  describe('median', () => {
    it('sorts the array', () => {
      expect(stats.sort([5, 4, 1, 2, 3])).toEqual([1, 2, 3, 4, 5]);
    });
    it('gets the median for odd length', () => {
      expect(stats.median([1, 2, 3, 4, 5])).toBe(3);
    });
    it('gets the median for even length', () => {
      expect(stats.median([1, 2, 3, 4, 5, 6])).toBe(3.5);
```

```
    });
  });
  describe('mode', () => {
    it('has one mode', () => {
      expect(stats.mode([1, 2, 2, 2, 3])).toBe(2);
    });
    it('has no mode', () => {
      expect(stats.mode([1, 2, 3])).toBe(null);
    });
    it('has multiple mode', () => {
      expect(stats.mode([1, 2, 2, 3, 3, 4])).toEqual([2, 3]);
    });
  });
});
```

stats.js

```
exports.max = numbers => Math.max(...numbers);
exports.min = numbers => Math.min(...numbers);
exports.avg = numbers =>
  numbers.reduce(
    (acc, current, index, { length }) => acc + current / length,
    0
  );

exports.sort = numbers => numbers.sort((a, b) => a - b);
exports.median = numbers => {
  const { length } = numbers;
  const middle = Math.floor(length / 2);
  return length % 2
    ? numbers[middle]
    : (numbers[middle - 1] + numbers[middle]) / 2;
};

exports.mode = numbers => {
  const counts = new Map();
  numbers.forEach(n => {
    const count = counts.get(n) || 0;
    counts.set(n, count + 1);
  });
  const maxCount = Math.max(...counts.values());
  const modes = [...counts.keys()].filter(
    number => counts.get(number) === maxCount
  );

  if (modes.length === numbers.length) {
    // 최빈값이 없음
    return null;
  }

  if (modes.length > 1) {
    // 최빈값이 여러개
```

```
    return modes;
  }

  // 최빈값이 하나
  return modes[0];
};
```

리액트 컴포넌트 테스트

- React 공식 문서에서 권장하는 방법 : react-testing-library
- 대처 방안 : Enzyme
- 2018년부터 react-testing-library가 좋다 : 렌더링 결과에 더 집중
- Enzyme이 사용률이 더 높다

리액트 프로젝트 만들기

```
$ yarn create-react-app react-enzyme-test
# or $ npx create-react-app react-enzyme-test
```

- VS code 사용하는 경우 IDE 지원을 위해 **@types/jest** 설치
- 다음 라이브러리들 설치

```
$ yarn add enzyme enzyme-adapter-react-16
# 또는 $ npm install --save enzyme enzyme-adapter-react-16
```

- src/setupTest.js 설치

src/setupTest.js

```
import {configure} from 'enzyme';
import Adapter from 'enzyme-adapter-react-16';

configure({adapter: new Adapter() });
```

src/Profile.js

```
import React from 'react';

const Profile = ({username, name}) => {
  return (
    <div>
```

```
      <b>{username}</b>  
      <span>({name})</span>
    </div>
  );
};

export default Profile;
```

App.js

```
import React from 'react';
import Profile from './Profile';

function App(){
  return (
    <div>
      <Profile username = "veloport" name = "김민준" />
    </div>
  );
}

export default App;
```

Enzyme 사용법

1. 스냅샷 테스팅

- **스냅샷 테스팅 : ** 렌더링된 결과가 이전에 렌더링한 결과와 일치하는지 확인하는 작업

```
$ yarn add enzyme-to-json # enzyme-to-json 이라는 라이브러리 설치
```

- package.json 파일로 **'jest'** 설정 추가

package.json

```
{
  "name": "react-enzyme-test",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@types/jest": "^24.0.13",
    "enzyme": "^3.9.0",
    "enzyme-adapter-react-16": "^1.13.1",
    "enzyme-to-json": "^3.3.5",
    "react": "^16.8.6",
    "react-dom": "^16.8.6",
```

```
    "react-scripts": "3.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {
    "production": [">0.2%", "not dead", "not op_mini all"],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "jest": { // 이렇게 추가
    "snapshotSerializers": ["enzyme-to-json/serializer"]
  }
}
```

- **Profile.test.js** 작성

Profile.test.js

```
import React from 'react';
import { mount } from 'enzyme';
import Profile from './Profile';

describe('<Profile />', () =>{
  it('matched snapshot', () => {
    const wrapper = mount(<Profile username="veloport" name="김민준"/>);
    expect(wrapper).toMatchSnapshot();
  })
})

// **mount 함수 : ** Enzyme을 통해서 리액트 컴포넌트 렌더링
// wrapper를 통해서 props 조회, DOM 조회, state 조회 등 가능
// mount 외에도 shallow도 있다.
```

- 테스트 실행

```
$ yarn test
```

- 결과

- 다음과 같은 스냅샷 파일 생성 : \src__snapshots__/Profile.test.js.snap/

__snapshots__/Profile.test.js.snap/

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

  exports[`<Profile /> matches snapshot 1`] = `
  <Profile
    name="김민준"
    username="velopert"
  >
    <div>
      <b>
        velopert
      </b>

      <span>
        (
        김민준
        )
      </span>
    </div>
  </Profile>
  `;
```

- 컴포넌트 수정 시 다음의 스냅샷이 일치하지 않아서 테스트 실패
- u를 눌러서 스냅샷을 업데이트 가능

props 접근

- 컴포넌트 인스턴스에 접근 가능

src/Profile.test.js

```
import React from 'react';
import {mount} from 'enzyme';
import Profile from './Profile';

describe('<Profile>', () => {
  it('matches snapshot', () =>{
    const wrapper = mount(<Profile username="velopert" name="김민준" />);
    expect(wrapper).toMatchSnapshot();
  });
  it('renders username and name', () =>{
    const wrapper = mount(<Profile username="velopert" name="김민준" />);
    expect(Wrapper.props().username).toBe('velopert');
    expect(Wrapper.props().name).toBe('김민준');
  });
});
```

DOM 접근

src/Profile.test.js

```
import React from 'react';
import {mount} from 'enzyme';
import Profile from './Profile';

describe('<Profile />', ()=>{
  it('matches snapshot', ()=>{
    const wrapper = mount(<Profile username="velopert" name="김민준" />);
    expect(wrapper).toMatchSnapshot();
  });
  it('renders username & name', () =>{
    const wrapper = mount(<Profile username="velopert" name="김민준" />);

    expect(wrapper.props().username).toBe('velopert');
    expect(wrapper.props().name).toBe('김민준');

    const boldElement = wrapper.find('b');
    expect(boldElement.contains('velopert')).toBe(true);
    const spanElement = wrapper.find('span');
    expect(spanElement.text()).toBe('(김민준)');
  });
});
```

- **find** 함수를 사용하면 특정 DOM을 선택할 수 있다.
- browser의 **querySelector**와 같다.
- CSS 클래스는 **find('.my-class')**, id는 **find('#myid')**, 태그는 **find('span')**으로 조회
- 특정 컴포넌트의 인스턴스 **find('MyComponent)**

Class component의 테스팅

src/Counter.js

```
import React, {Component} from 'react';

class Counter extends Component{
  state = {
    number: 0
  };
  handleIncrease = () =>{
    this.setState({
      number: this.state.number + 1
    });
  };
  handleDecrease = () =>{
    this.setState({
      number: this.state.number - 1
```

```
    });
  };
  render(){
    return(
      <div>
        <h2>{this.state.number}</h2>
        <button onClick = {this.handleIncrease}>+1</button>
        <button onClick = {this.handleIncrease}>-1</button>
      </div>
    );
  }
}

export default Counter;
```

src/Counter.test.js

```
import React from 'react';
import {shallow} from 'enzyme';
import Counter from './Counter';

describe('<Counter />', () =>{
  it('matches snapshot', () =>{
    const wrapper = shallow(<Counter />);
    expect(wrapper).toMatchSnapshot();
  });
  it('has initial number', ()=>{
    const wrapper = shallow(<Counter />);
    expect(wrapper.state().number).toBe(0);
  });
  it('increases', ()=>{
    const wrapper = shallow(<Counter />);
    wrapper.instance().handleIncrease();
    expect(wrapper.state().number).toBe(1);
  });
  it('decreases', ()=>{
    const wrapper = shallow(<Counter />);
    wrapper.instance().handleDecrease();
    expect(wrapper.state().number).toBe(-1);
  });
});
```

- **mount** 대신 **shallow** 함수 사용.
- **shallow**는 컴포넌트 내부에 다른 리액트 컴포넌트가 있다면 이를 렌더링하지 않는다.
  - **shallow**는 컴포넌트의 내부를 렌더링하지 않고 지나간다
- **state()** : 컴포넌트의 state 조회
- **instance() : ** 내장 메서드를 호출한다

DOM 이벤트 시뮬레이트

- 내장 메서드를 직접 호출하는 것이 아닌, 버튼 클릭 이벤트를 시뮬레이트해서 기능이 잘 작동하는지 확인

Counter.test.js

```
import React from 'react';
import {shallow} from 'enzyme';
import Counter from './Counter';

describe('<Counter />', ()=>{
  it('matches snapshot', ()=>{
    const wrapper = shallow(<Counter />);
    expect(wrapper).toMatchSnapshot();
  });
  it('has initial number', ()=>{
    const wrapper = shallow(<Counter />);
    expect(wrapper.state().number).toBe(0);
  });
  it('increases', ()=>{
    const wrapper = shallow(<Counter />);
    wrapper.instance().handleIncrease();
    expect(wrapper.state().number).toBe(1);
  });
  it('decreases', ()=>{
    const wrapper = shallow(<Counter />);
    wrapper.instance().handleDecrease();
    expect(wrapper.state().number).toBe(-1);
  });
  it('calls handleIncrease', ()=>{
    // 클릭 이벤트를 시뮬레이트 하고, state 확인
    const wrapper = shallow(<Counter />);
    const plusButton = wrapper.findWhere(
      node => node.type() === 'button' && node.text() === '+1'
    );
    plusButton.simulate('click');
    expect(wrapper.state().number).toBe(1);
  });
  it('calls handleDecrease', ()=>{
    // 클릭 이벤트를 시뮬레이트하고, h2 태그의 텍스트 확인
    const Wrapper = shallow(<Counter />);
    const minusButton = wrapper.findWhere(
      node => node.type() === 'button' && node.text() === '-1'
    );
    minusButton.simulate('click');
    const number = wrapper.find('h2');
    expect(number.text()).toBe(-1);
  });
});
```

- **findWhere** 원하는 버튼 태그를 선택할 수 있다.

- **simulate** 버튼에 이벤트를 시뮬레이트할 때 원하는 엘리먼트를 찾아서 사용
    - (<이벤트 이름>, <이벤트 객체>)

```
input.simulate('change', {
  target:{
    value: 'hello world'
  }
});
```

- 값이 잘 update 되었는지 확인하기 위해서 두 가지 방법을 사용하였다.
    - 첫 번째, state를 직접 조회하는 것
    - 두 번째, h2 태그를 조회해서 값을 확인.
    - 실제 테스트 코드는 둘 다 가능!

함수형 컴포넌트와 Hooks 테스팅

src/HookCounter.js

```
import React, {useState, useCallback} from 'react';

const HookCounter = () =>{
  const [number, setNumber] = useState(0);
  const onIncrease = useCallback(()=>{
    setNumber(number + 1);
  }, [number]);
  const onDecrease = useCallback(()=>{
    setNumber(number - 1);
  }, [number]);

  return(
    <div>
      <h2>{number}</h2>
      <button onClick={onIncrease}>+1</button>
      <button onClick={onDecrease}>-1</button>
    </div>
  );
};

export default HookCounter;
```

src/App.js

```
import React from 'react';

import HookCounter from './HookCounter';
```

```
function App() {
  return (
    <div>
      <HookCounter />
    </div>
  );
}

export default App;
```

- **Hook 사용하는 경우 꼭 !!!!!! shallow 아닌 mount 사용**
- useEffect Hook은 shallow에서 작동하지 않고, 버튼 엘리먼트에 연결되어 있는 함수가 이전 함수를 가리키고 있기 때문에,
- 에를 들어 +1 버튼의 클릭 이벤트를 두 번 시뮬레이트해도 결과값이 2가 되는 것이 아닌 1이 된다.

HookCounter.test.js

```
import React from 'react';
import {mount} from 'enzyme';
import HookCounter from './HookCounter';

describe('<HookCounter />', () =>{
  it('matches snapshot', () =>{
    const wrapper = mount(<HookCounter />);
    expect(wrapper).toMatchSnapshot();
  });
  it('increases', ()=>{
    const wrapper = mount(<HookCounter />);
    let plusButton = wrapper.findWhere(
      node => node.type() === 'button' && node.text() === '+1'
    );
    plusButton.simulate('click');
    plusButton.simulate('click');

    const number = wrapper.find('h2');

    expect(number.text()).toBe('2');
  });
  it('decreases', ()=>{
    const wrapper = mount(<HookCounter />);
    let decreaseButton = wrapper.findWhere(
      node =>node.type() === 'button' && node.text() === '-1'
    );
    decreaseButton.simulate('click');
    decreaseButton.simulate('click');

    const number = wrapper.find('h2');

    expect(number.text()).toBe('-2');
```

```
  });
});
```

react-testing-library

- Enzyme와 다르게 모든 테스트를 DOM 위주로 진행
- 컴포넌트의 props나 state를 조회하지 않는다.
- 실제 리팩토링할 때는 내부 구조 및 네이밍은 많이 바뀔 수 있어도 실제 작동 방식은 크게 안 바뀐다.
- **react-testing-library** : 컴포넌트의 기능이 똑같이 작동한다면 컴포넌트의 내부 구현 방식이 많이 바뀌어도 테스트가 실패하지 않도록 작동
- react-testing-library : 필요한 기능만 지원해서 가볍고, 일관성있게 작성 가능

1. 리액트 프로젝트 만들기

```
$ yarn create rlt-tutorial
# or npx create-react-app trl-tutorial
```

2. 설치

```
$ yarn add react-testing-library jest-dom
# or npm install --save react-testing-library jest-dom
```

- jest-dom은 jest 확장으로서, DOM에 관련된 matcher를 추가
- VS code 사용 시 @types/jest 패키지도 설치

src/setupTests.js

```
import 'react-testing-library/cleanup-after-each';
import 'jest-dom/extend-expect';
```

- react-testing-library는 리액트에서 DOM을 위한 JSDOM를 사용
- document.body 리액트 컴포넌트 렌더링
- clean-up-after-each를 부르면, 끝날 때까지 기존의 가상 화면에 남은 UI 정리
- 추가적으로 'jest-dom/extend-expect'를 불러서 jest에서 DOM 관련 matcher 사용 가능

3. 1st 테스트 코드

src/Profile.js

```
import React from 'react';
```

```
const Profile = ({username, name}) =>{
  return(
    <div>
      <b>{username}</b> 
      <span>({name})</span>
    </div>
  );
};

export default Profile;
```

src/App.js

```
import React from 'react';
import Profile from './Profile';

const App = () =>{
  return <Profile username="veloprot" name="김민준" />;
};

export default App;
```

src/Profile.test.js

```
import React from 'react';
import {render} from 'react-testing-library';
import Profile from './Profile';

describe('<Profile />', () =>{
  it('matches snapshot', () => {
    const utils = render(<Profile username="velopert" name="김민준" />);
    expect(utils.container).toMatchSnapshot();
  });
  it('shows the props correctly', ()=>{
    const utils = render(<Profile username="velopert" name="김민준' />);
    utils.getByText('velopert'); // velopert라는 텍스트를 가진 엘리먼트
    utils.getByText('(김민준)'); // (김민준)이라는 텍스트를 가진 엘리먼트
    utils.getByText(/김/); // 정규식 /김/을 통과하는 엘리먼트가 있는지 확인
  });
});
```

- **yarn test** 명령어를 통해서 확인
- 렌더링을 사용할 때는 **render()** 함수 사용
- **container**는 해당 컴포넌트의 최상위 **DOM**을 가리킨다. => 스냅샷 테스팅도 가능
- **getText** : 쿼리 함수. 원하는 DOM을 가리킬 수 있다.

스냅샷 테스팅

- **스냅샷 테스팅** : 렌더링된 결과가 이전에 렌더링한 결과와 일치하는지 확인하는 작업
- 코드 저장 시 **src/snapshots/Profile.test.js.snap** 생성

```
// Jest Snapshot v1, https://goo.gl/fbAQLP

exports[`<Profile /> matches snapshot 1`] = `
<div>
  <div>
    <b>
      velopert
    </b>

    <span>
      (
      김민준
      )
    </span>
  </div>
</div>
`;
```

- 컴포넌트가 렌더링되었을 때 이 스냅샷과 일치하지 않으면 테스트가 실패.
- 스냅샷을 업데이트하고 싶다면 테스트 실행하는 콘솔 창에서 'u' 키를 사용

어떤 쿼리 사용?

- 다음 우선순위를 따라서 사용하는 것을 권장

1. getByLabelText
    - label이 있는 input의 label 내용으로 input을 선택

```
<label for="username-input">아이디</label>
<input id="username-input" />

const inputNode = getByLabelText('아이디');
```

2. getByPlaceholderText
    - placeholder 값으로 input 및 textarea 선택

```
<input placeholder="아이디" />;
const inputNode = getByPlaceholderText('아이디');
```

3. getByText
    - 엘리먼트가 가지고 있는 텍스트 값으로 DOM을 선택

```
<div>Hello World!</div>

const div = getByText('Hello World!');
```

4. getByDisplayValue
    - input, textarea, select가 지니고 있는 현재 값을 가지고 엘리먼트를 선택한다.

```
<input value="text" />

const input = getByDisplayValue('text');
```

5. getByAltText
    - alt 속성을 가지고 있는 엘리먼트(주로 img)를 선택

```
<img src = "/awesome.png" alt="awesome image" />;
const imgAwesome = getByAltText('awesome');
```

6. getByTitle
    - title 속성을 가지고 있는 DOM 혹은 title 엘리먼트를 지니고 있는 SVG 선택할 때 사용

```
<p>
  <span title="React">리액트</span>는 멋있는 라이브러리.
</p>

<svg>
  <title>Delete</title>
  <g><path /></g>
</svg>

const spanReact = getByTitle('React');
const svgDelete = getByTitle('Delete');
```

7. getByRole
    - 특정 role 값을 지니고 있는 엘리먼트 선택

```
<span role ="button">삭제</span>

const spanRemove = getByRole('button');
```

8. getByTestId
    - 특정 DOM에 직접 사용할 때 사용할 id를 달아서 선택

```
  <div data-testid = "commondiv">흔한 div</div>;
  const commonDiv = getByTestId('commondiv');
```

Counter 컴포넌트 테스트 코드 작성

src/Counter.js

```
import React, {useState, useCallback} from 'react';

const Counter = () =>{
  const [number, setNumber] = useState(0);

  const onIncrease = useCallback(()=>{
    setNumber(number + 1);
  }, [number]);

  const onDecrease = useCallback(()=>{
    setNumber(number -1);
  }, [number]);

  return(
    <div>
      <h2>{number}</h2>
      <button onClick = {onIncrease}>+1</button>
      <button onClick = {onDecrease}>-1</button>
    </div>
  );
};

export default Counter;
```

src/App.js

```
import React from 'react';
import Counter from './Counter';

const App = () =>{
  return <Counter />;
};

export default App;
```

src/Counter.test.js

```
import React from 'react';
import {render, fireEvent} from 'react-testing-library';
```

```
import Counter from './Counter';

describe('<Counter />', ()=>{
  it('matches snapshot', ()=>{
    const utils = render(<Counter />);
    expect(utils.container).toMatchSnapshot();
  });
  it('has a number and two buttons', ()=>{
    const utils = render(<Counter />);
    // 버튼과 숫자가 있는지 확인
    utils.getByText('0');
    utils.getByText('+1');
    utils.getByText('-1');
  });
  it('increases', ()=>{
    const utils = render(<Counter />);
    const number = utils.getByText('0');
    const plusButton = utils.getByText('+1');

    // 클릭 이벤트를 두 번 발생
    fireEvent.click(plusButton);
    fireEvent.click(plusButton);
    expect(number).toHaveTextContent('2'); // jest-dom의 확장 matches
    expect(number.textContext).toBe('2'); //textContext 직접 비교
  });
  it('decreases', ()=>{
    const utils = render(<Counter />);
    const number = utils.getByText('0');
    const plusButton = utils.getByText('-1');

    // 클릭 이벤트 두 번 발생
    fireEvent.click(plusButton);
    fireEvent.click(plusButton);
    expect(number).toHaveTextContent('-2'); // jest-dom의 확장 matches
  });
});
```

- 이벤트 다루기

```
fireEvent.이벤트이름(DOM, 이벤트 객체);

fireEvent.change(myInput, { target: { value: 'hello world' } });
```

알고리즘 공부

Velog & 백준 & Commit