

Case1 : Context Menu

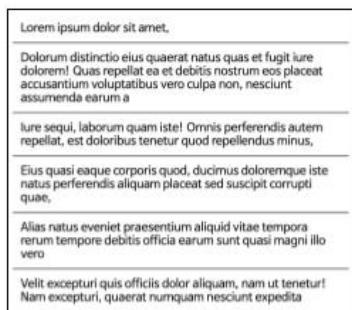
케이스 주제

Q. 버튼을 클릭하면 컨텍스트 메뉴가 나타나고, 메뉴를 선택하거나 그 외의 부분을 클릭하면 사라지는 팝오버 컴포넌트를 구현하세요

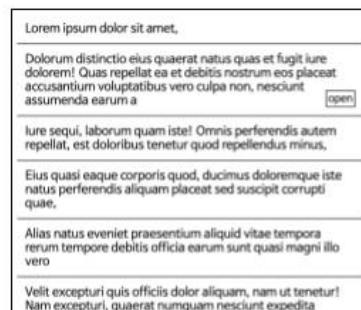
기능 요구사항

- 임의의 메뉴 목록에 대하여 각 메뉴를 클릭하면 해당 메뉴에 관련된 문구가 떠있는 형태(팝오버)로 나타난다.
- 팝오버 상태에서 해당 문구를 클릭하면 해당 메뉴가 사라진다.
- 팝오버 상태에서 다른 메뉴를 클릭하면 해당 메뉴의 팝오버가 나타난다. 이 때 기존의 팝오버는 사라진다.

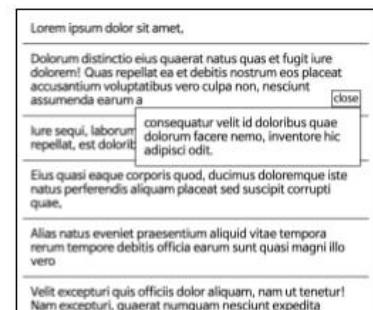
기능 작동 이미지



1) 최초상태.



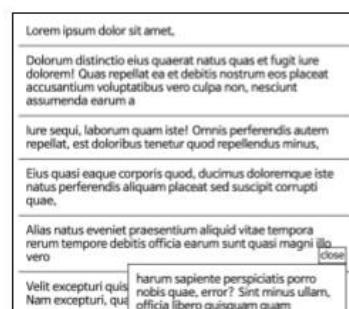
2) 메뉴 하나에 마우스 오버



3) 메뉴 클릭시



4) 다른 메뉴에 마우스 오버



5) 다른 메뉴 클릭시

문제

- q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요
- q2. 문제 상황에 대하여 jquery로 동작을 구현시킬 수 있는 코드를 작성해보세요
- q3. 문제 상황에 대하여 HTML(detail 태그)로 동작을 구현시킬 수 있는 코드를 작성해보세요
- q4. 문제 상황에 대하여 React로 동작을 구현시킬 수 있는 코드를 작성해보세요
- q5. 문제 상황에 대하여 React-CreatePortal 기능으로 동작을 구현시킬 수 있는 코드를 작성해보세요

주요 학습 키워드

이벤트 리스너, 이벤트 위임 구현 원리, detail 태그, React-Create Portal 등

작성해주셔야 하는 question 파일경로

q1

```
./question/q1_js/index.js  
./question/q1_js/style.css
```

q2

```
./question/q2_jquery/index.js  
./question/q2_jquery/style.css
```

q3

```
./question/q3_html-js/index.js  
./question/q3_html-js/style.css
```

q4

```
./question/q4_react.js/src/App.js
```

q5

```
./question/q5_react.js-createportal/src/App.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js`

index.html 열기

q2

경로 `./question/q2_jquery`

index.html 열기

q3

경로 `./question/q3_html-js`

index.html 열기

q4

경로 `./question/q4_react.js`

터미널

```
$ npm install  
$ npm start
```

q5

경로 `./question/q5_react.js-createportal`

터미널

```
$ npm install  
$ npm start
```

Case1 : ContextMenu - 출제자 해설

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요

이벤트 핸들러는 가급적 최소한으로만 활용할 것. 감시대상이 늘 수록 성능에 좋지 않다.

이벤트 캡쳐링/버블링을 이해하고 있는지

stopPropagation과 preventDefault를 구분하여 사용할 수 있는지

(추가로 생각해볼 문제) click 이벤트의 감시 대상을 더욱 줄일 수는 없을까?

A)

```
.item.open .context {  
    display: block;  
}
```

```
// 초보자의 접근법  
  
const items = document.querySelectorAll('.item');  
items.forEach(function(item) {  
    item.addEventListener('click', function(e) {  
        item.classList.toggle('open');  
        items.forEach(function(elem) {  
            if (elem !== item) elem.classList.remove('open');  
        });  
    });  
});
```

```
// 더 나은 방법 1  
const wrapper = document.querySelector('.wrapper');  
const items = document.querySelectorAll('.item');
```

```

wrapper.addEventListener('click', function(e){
  const targetElem = e.target;
  e.stopPropagation();
  if (!targetElem.classList.contains('item')) return;

  targetElem.classList.toggle('open');
  items.forEach(cont => {
    if(cont !== targetElem) cont.classList.remove('open');
  });
});

document.body.addEventListener('click', function(e) {
  if (e.target.classList.contains('context')) return;
  items.forEach(cont => {
    cont.classList.remove('open');
  });
});

```

```

// 더 나은 방법 (2)
const wrapper = document.querySelector('.wrapper');
const items = document.querySelectorAll('.item');

document.body.addEventListener('click', function(e) {
  const targetclassList = e.target.classList;
  if (targetclassList.contains('context')) return;
  if (targetclassList.contains('item')) {
    targetclassList.toggle('open');
    items.forEach(function(elem) {
      if (elem !== e.target) elem.classList.remove('open');
    });
    return;
  }
  items.forEach(function(elem) {
    elem.classList.remove('open');
  });
});

```

해설

초보자들은 DOM 제어와 관련하여 이벤트 리스너를 임의로 작성하곤 한다. 목록의 각 아이템 하나하나마다 등록하는 식이다. 본 문제의 경우를 예로 들면 다음과 같이.

```

items.forEach(function(item) {
  item.addEventListener('click', ...)

```

```
}
```

이렇게 작성하면 크게 두가지 문제가 있다.

(1) 이벤트 감시대상이 많은 만큼 메모리에 부담이 된다. (2) 어떤 변경에 의해 목록에 아이템이 추가될 경우 해당 아이템은 감시대상에 속하지 않아 팝오버 동작이 이뤄지지 않는다.

따라서 새로운 아이템이 추가될 때마다 그에 대한 리스너를 등록해주어야 한다.

반면 리스너를 상위 노드에 등록하면 위 두가지 문제가 모두 해결된다. 따라서 리스너 등록은 최소화하는 것이 바람직하다. 이벤트 핸들러를 최소화하기 위해서는 캡쳐링/버블링을 이해하는 것이 필요하다. 나아가 리스너 함수의 첫번째 인자인 event 객체의 내부에 어떤 정보가 들어있는지를 알 필요가 있다.

```
stopPropagation과 preventDefault를 구분하여 사용할 수 있는지도 중요한 요소
```

(추가로 생각해볼 문제)

```
click 이벤트의 감시 대상을 더욱 줄일 수는 없을까? 그럴 경우의 장단점은?
```

장점: 리스너가 줄어듦. 단점: 1) 함수 내부에 등장할수밖에 없는 조건문에 대한 최적화 필요. 2) 개별 등록/해제 가능한 리스너에 비해 관리가 어려움

q2. 문제 상황에 대하여 jquery로 동작을 구현시킬 수 있는 코드를 작성해보세요

```
javascript와 사실상 같은 구조.
```

```
jquery의 'delegate target'에 대한 이해가 필요하다.
```

```
상황에 꼭 맞는 메서드들을 알고 있는지가 관건.
```

A)

```
const $wrapper = $('.wrapper');
const $items = $wrapper.find('.item');
$wrapper.on('click', '.item', function(e) {
  e.stopPropagation();
  $(this).toggleClass('open').siblings().removeClass('open');
});
$('body').on('click', function(e) {
```

```
$items.removeClass('open');  
});
```

```
.item.open .context {  
  display: block;  
}
```

(추가로 생각해볼 문제)

마찬가지로 이벤트 감시 대상을 줄일 방법은 없을지?

```
const $items = $('.wrapper .item');  
$('body').on('click', e => {  
  const item = $(e.target);  
  if (item.is('.item')) {  
    item.toggleClass('open').siblings().removeClass('open');  
  } else {  
    $items.removeClass('open');  
  }  
});
```

q3. 문제 상황에 대하여 HTML(detail 태그)-JavaScript로 동작을 구현시킬 수 있는 코드를 작성해보세요

HTML5의 details 태그를 활용하면 팝오버 오픈을 위한 처리를 자바스크립트가 관여하지 않아도 되므로 코드가 훨씬 간결해진다.

A)

```
/* css */  
details[open] p {  
  display: block;  
}
```

```
const items = document.querySelectorAll('details');
document.body.addEventListener('click', function(e) {
  if (e.target.nodeName !== 'P' && e.target.nodeName !== 'SUMMARY') {
    items.forEach(function(item) {
      item.removeAttribute('open');
    });
    return;
  }
  items.forEach(function(item) {
    if (item !== e.target.parentElement) {
      item.removeAttribute('open');
    }
  });
});
```

q4. 문제 상황에 대하여 React로 동작을 구현시킬 수 있는 코드를 작성해보세요

A)

```
const togglePopover = index => e => {
  e.preventDefault();
  e.stopPropagation();
  setOpen(openedIndex === index ? null : index);
};

const closeAll = e => {
  if (e.target.nodeName !== 'P') setOpen(null);
};
```

q5. 문제 상황에 대하여 React-CreatePortal 기능으로 동작을 구현시킬 수 있는 코드를 작성해보세요

createPortal은 Modal, Popover 등 floating UI를 효과적으로 제어할 수 있는 강력한 도구다.

A)

```
<ContextPortal  
  target={detailRefs.current[openedIndex]}  
  children={<p>{dummyData[openedIndex]?.context}</p>}  
/>
```

결론 - 장단점 비교

html / css로 처리할 방법이 있다면 최대한 활용하는 것이 좋다.

1. 자바스크립트가 관여하지 못하는 경우에도 완결성 있는 화면을 보여줄 수 있다.
2. 개발자가 온갖 상황을 테스트하며 일일이 대응하여 작성한 코드보다 이미 브라우저에서 검증이 완료된 html / css의 기본 동작의 신뢰도가 높을 수밖에 없다.
3. 무엇보다 코드량이 줄어든다. simple is the best.

노출하지 않는 데이터를 html에 계속 보유하고 있는 것이 좋을까, 그렇지 않고 필요할 때만 그리는 편이 좋을까?

어느 경우에도 리페인트는 피할 수 없으니 논외. 리플로우는 position: absolute로 이미 최소화되어 있는 상태. 그렇다면 이미 만들어진 노드를 그대로 사용하는 것과 추가하는 것 중 어느 쪽이 빠를까가 관건인데, 모던브라우저에서 속도 차이는 거의 없다고 봐도 무방하다. 팝오버 안의 내용이 민감한 정보라거나, 매 번 오픈할 때마다 서버로부터 실시간 정보를 반영해야 하는 경우라면 선택의 여지가 없겠으나, 그렇지 않은 경우라면 개발의 편리성을 고려하는 것으로 충분하겠다.

Case1 : ContextMenu - 대기업 S사 프론트엔드 개발자님의 답안

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요

```
/**  
 * 각 아이템 항목의 클릭 이벤트를 위임받을 컨테이너  
 */  
  
const containerEl = document.querySelector(".container");  
  
/**  
 * 가장 마지막으로 선택된 엘리먼트를 저장  
 */  
  
let latestOpenedEl = null;  
  
/**  
 * 클릭 이벤트가 `Document` 까지 전파된 경우 마지막에 열린 창을 닫음  
 */  
  
document.addEventListener("click", e => {  
    if (latestOpenedEl) {  
        latestOpenedEl.classList.remove("open");  
        latestOpenedEl = null;  
    }  
});  
  
/**  
 * 컨테이너에 클릭 이벤트를 바인딩  
 */  
  
containerEl.addEventListener("click", e => {  
    /**  
     * 이벤트가 상위(`Document`) 객체로 전달되지 않도록 `stopPropagation` 호출  
     * @see https://developer.mozilla.org/ko/docs/Web/API/Event/stopPropagation  
     */  
    e.stopPropagation();  
    const targetEl = e.target;  
  
    /**
```

```

 * 가장 마지막에 선택된 엘리먼트의 `open` 클래스를 제거
 */
latestOpenedEl?.classList.remove("open");

/**
 * 현재 선택된 엘리먼트와 마지막에 선택된 엘리먼트가 동일하다면 아무것도 하지 않음
 */
if (latestOpenedEl === targetEl) {
    latestOpenedEl = targetEl;
    return;
}

targetEl?.classList.add("open");
latestOpenedEl = targetEl;
});

```

해설

1. 이벤트를 위임받기 적당한 컨테이너 셀렉터를 입력

아이템의 상위 요소라면 어떤걸 선택해도 문제되지는 않지만 가급적 가장 인접한 container 항목을 선택하는것이 좋음 (최상위 엘리먼트에 모든 이벤트가 집중된 경우, 매 이벤트 동작마다 등록된 모든 이벤트를 탐색해야하는 불필요한 연산이 발생하게되고, 더이상 사용되지 않는 자식요소 DOM 엘리먼트 제거시 바인딩된 이벤트를 직접 제거해 주지 않는경우에도 불필요한 연산이 발생)

2. 이벤트 객체에서 이벤트가 발생된 엘리먼트 객체를 선택해 주세요.

```

e.target
,e.currentTarget
두 값의 의미를 제대로 구분할 수 있어야함

```

[참고] <https://developer.mozilla.org/ko/docs/Web/API/Event/target>

[참고] <https://developer.mozilla.org/en-US/docs/Web/API/Event/currentTarget>

**각각의 항목에 이벤트를 바인딩 하지 않고 컨테이너에서 이벤트를 위임(Event Delegation) 받아 처리함으로써
이벤트 처리 성능 향상**

항목별 이벤트를 직접 등록하는 경우 이벤트 리스너를 등록하는 시간과 메모리 리소스 사용이 증가 합니다. 항목이 적거나 성능이 좋은 최신 브라우저에서는 차이를 느끼기 쉽지 않지만, IE8 이하 버전

과 같은 구형 브라우저 버전에서는 항목이 늘어나면 성능 차이가 눈에 보일 정도로 느껴질 수 있습니다.

```
containerEl
하위요소에서 발생하는 click이벤트는 버블링되어 containerEl
으로 전파되어 containerEl
에 등록된 클릭 이벤트 리스너에서 처리됨
[참고] https://developer.mozilla.org/ko/docs/Web/API/Event/eventPhase
[참고] https://developer.mozilla.org/ko/docs/Web/API/EventTarget/addEventListener
```

**DOM 탐색 시간을 줄이기 위해 latestOpenedEl
변수에 가장 마지막으로 열린 *HTMLElement* 항목을 저장**

매 번 *DOM*을 순회하며 *Attribute*에 저장된 상태를 확인하여 열려있는 *HTMLElement*를 확인하는 방법도 있지만, 불필요한 연산을 줄여 성능을 조금 더 빠르게 하기 위해 가장 마지막에 열린 *HTMLElement*를 latestOpenedEl 변수에 저장하도록 구현 하였습니다.

**latestOpenedEl
상태를 사용하는 이유**

[!] 위와 같이 성능을 더 높히는 대신 latestOpenedEl
변수의 상태를 관리해야하는 약간의 번거로움(?)이 있습니다. 이 부분에 대해서는 어플리케이션의 성능이나 복잡도를 고려하여 개발상황에 맞게 참고하면 좋을것 같습니다.

이벤트 위임을 사용함으로써 자동으로 이벤트 리스너를 등록되어 관리가 필요 없는 이점도 있지만, 가능하다면 성능적인 부분은 작은곳부터 관리하는게 좋습니다. 가령 항목의 갯수가 100개 이상이라고 가정 했을 때 항목을 클릭할때마다 100개의 리스트에서 열려있는 상태의 *HTMLElement* 찾아야 합니다.

q2. 문제 상황에 대하여 jquery로 동작을 구현시킬 수 있는 코드를 작성해보세요

```

// Write JQuery code here!
/***
 * 각 아이템 항목의 클릭 이벤트를 위임받을 컨테이너
 */
const $container = $(".container");

/***
 * 가장 마지막으로 선택된 엘리먼트를 저장
 */
let $latestOpened = null;

$(document).on("click", () => {
    if ($latestOpened) {
        $latestOpened.removeClass("open");
        $latestOpened = null;
    }
});

/***
 * Jquery `on` 함수를 이용하여 이벤트 위임하여 바인딩
 */
$container.on("click", ".item", e => {
    e.stopPropagation();

    // => 1) Jquery 라이브러리의 on 함수를 이용하여 이벤트 위임을 이용하여 이벤트를 추가해 주세요
    const $target = $(e.target);
    /***
     * 가장 마지막에 선택된 엘리먼트의 `open` 클래스를 제거
     */
    $latestOpened?.removeClass("open");

    /***
     * 현재 선택된 엘리먼트와 마지막에 선택된 엘리먼트가 동일하다면 아무것도 하지 않음
     */
    if ($latestOpened?.get(0) === $target?.get(0)) {
        $latestOpened = $target;
        return;
    }

    $target.addClass("open");
    $latestOpened = $target;
});

}

```

해설

1. Jquery라이브러리의 on
함수를 이용하여 이벤트 위임을 이용하여 이벤트를 추가

*Jquery*에서는 이벤트 위임이 이미 구현되어 있음

[참고] <https://api.jquery.com/on/>

2. 클릭 이벤트가 상위로 전파되지 않도록 함수를 호출

[참고] <https://developer.mozilla.org/ko/docs/Web/API/Event/stopPropagation>

기본적인 구현원리는 **Javascript** 문제풀이와 동일합니다.

이벤트 위임 구현 로직을 *Jquery* 이벤트 바인딩 함수(`on`)에서 제공하는 기능으로 대체

Jquery 라이브러리에서 제공하는 이벤트 바인딩 함수(`on`)

)의 두번째 인자에 *Selector*를 지정하는 경우 이벤트 위임을 사용하실 수 있습니다.

[참고] <https://api.jquery.com/on/>

q3. 문제 상황에 대하여 **jquery**로 동작을 구현시킬 수 있는 코드를 작성해보세요

```
/**  
 * 각 아이템 항목의 클릭 이벤트를 위임받을 컨테이너  
 */  
  
const containerEl = document.querySelector(".container");  
/**  
 * 가장 마지막으로 선택된 엘리먼트를 저장  
 */  
  
let latestOpenEl = null;  
  
document.addEventListener("click", () => {  
    latestOpenEl?.removeAttribute("open");  
    latestOpenEl = null;  
});  
  
containerEl.addEventListener("click", e => {  
    let detailsEl = e.target;
```

```

    /**
     * 가장 마지막에 선택된 `details` 엘리먼트의 `open` 속성을 제거
    */
latestOpenEl?.removeAttribute("open");
/***
 * 현재 클릭된 엘리먼트로 부터 부모 엘리먼트를 확인하여 `details` 엘리먼트를 찾음
 * 루트 엘리먼트에 도달한 경우 `target.parentElement` 값은 `null`
*/
do {
    /**
     * 현재 클릭된 엘리먼트의 부모 엘리먼트가 `details`인 경우
    */
    if (detailsEl?.tagName === "DETAILS") {
        /**
         * 태그의 기본동작을(이벤트) 막아 `open` 속성이 자동으로 추가되지 않도록 함
        */
        e.stopPropagation();
        e.preventDefault();

        if (detailsEl === latestOpenEl) {
            /**
             * 현재 선택된 `details` 엘리먼트가 마지막 선택된 `details`와 동일한 경우 아무것도 하지 않음
            */
            latestOpenEl = null;
            break;
        }

        /**
         * `open` 속성을 수동으로 추가
        */
        detailsEl.setAttribute("open", "");
        latestOpenEl = detailsEl;
        break;
    }
} while ((detailsEl = detailsEl?.parentElement) !== null);
});

```

해설

1. 기본 이벤트(태그 동작)이 실행되지 않도록 특정 함수를 호출

이벤트 위임 구현 원리, 이벤트 취소(`preventDefault`)에 대한 이해가 필요한 내용으로 보여짐
[참고] <https://developer.mozilla.org/ko/docs/Web/API/Event/preventDefault>

기본적인 구현원리는 Javascript 문제풀이와 동일합니다.

이벤트 발생시 현재 타겟(클릭된)으로부터 `details`

태그를 탐색하는 코드가 추가.

`details`

태그 하위에는 다른 태그 속성들이 중첩되어 포함될 수 있기 때문에, 클릭된 태그(`HTMLElement`)로 부터 한단계씩 상위 태그를 확인하며 `details`

태그가 나타날 때 까지 탐색하고 결과를 저장합니다.

`toggle`

이벤트 대신 `click`

이벤트를 사용하는 이유

`details`

태그는 `toggle`

이벤트를 지원하지만, 컨테이너 태그인 `div`

에서는 `toggle`

이벤트를 리스닝 할 수 없기 때문에, 컨테이너 `div`

에서는 `click`

이벤트를 활용하여 `toggle`

이벤트가 발생하기 전 `e.preventDefault()`

를 이용하여 `toggle` 동작을 실행되지 않도록하고 `open`

속성을 직접 추가하도록 구현 하였습니다.

[참고] <https://developer.mozilla.org/ko/docs/Web/HTML/Element/details>

q4. 문제 상황에 대하여 React로 동작을 구현시킬 수 있는 코드를 작성해보세요

```
import React, { useState, useRef, useLayoutEffect } from "react";
import "./style.css";
import Detail from "./Detail";
import dummyData from "./dummyData";

export default function App() {
  /**
   * 가장 최근에 열린 `detail` 저장
   */
  const latestOpenedItemElRef = useRef(null);
```

```
/***
 * 가장 최근에 열린 `detail` 닫음
 */
const closeLatestOpenedItem = () => {
  if (latestOpenedItemElRef.current) {
    latestOpenedItemElRef.current.removeAttribute("open");
  }
};

useLayoutEffect(() => {
  /**
   * `document`에 클릭 이벤트 발생시 가장 최근에 열린 `detail`을 닫음
   */
  document.addEventListener("click", closeLatestOpenedItem);
  return () =>
    document.removeEventListener("click", closeLatestOpenedItem);
}, []);

const handleClickDetail = e => {
  /**
   * `toggle` 이벤트가 발생하지 않도록 기본 이벤트 동작 취소
   */
  e.preventDefault();
  /**
   * `click` 이벤트가 `document` 까지 전파되지 않도록 방지
   */
  e.stopPropagation();
  const currentTarget = e.currentTarget;
  const latestOpenedItemEl = latestOpenedItemElRef.current;

  closeLatestOpenedItem();

  if (currentTarget === latestOpenedItemEl) {
    return;
  }

  currentTarget.setAttribute("open", "");
  latestOpenedItemElRef.current = currentTarget;
};

return (
  <div className="container">
    {dummyData.map(({ text, context }, index) => {
      return (
        <Detail
          key={`detail${index}`}
          popover={context}
          onClick={handleClickDetail}
        >
          {text}
        
```

```
        </Detail>
    );
}
</div>
);
}
```

Case2 : Scroll spy

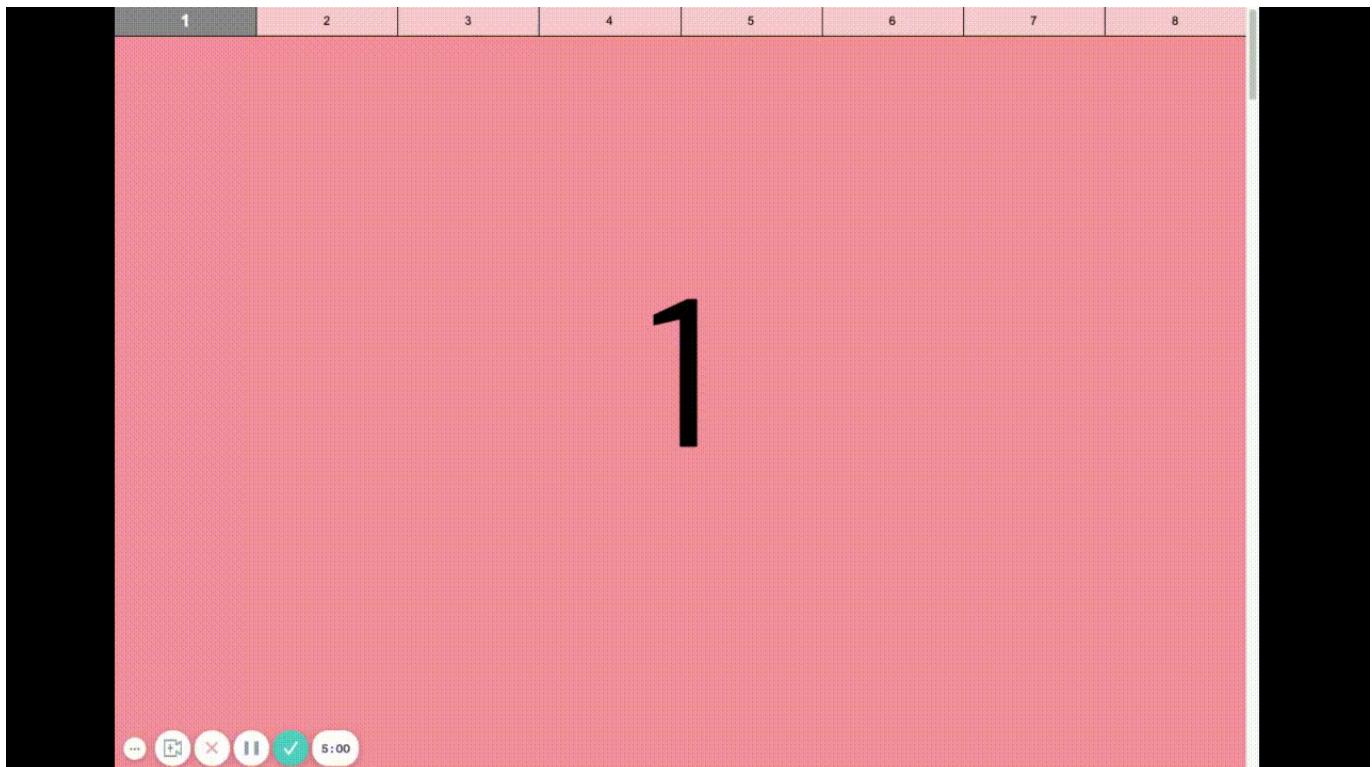
케이스 주제

Q. 스크롤스파이 구현하기

기능 요구사항

1. 최초 화면: 상단메뉴의 '1'이 활성화되어 있음.
2. 스크롤을 내리면서 나오는 숫자가 상단 메뉴에 활성화된 숫자와 일치하도록 합니다.
3. 상단 메뉴를 클릭하여, 해당 숫자가 적힌 스크롤위치로 이동할 수 있습니다.

기능 작동 이미지



실행 방법 / 풀이 방법 안내

문제 풀기 방식 :

1. 레포지토리를 clone
2. 터미널에서 각 문제 폴더 디렉토리로 이동하여 `npm install`로 의존성을 설치
3. `package.json`을 참고하여, 명시된 `scripts` 명령어로 개발서버 실행.
4. 코드 수정하면서 문제 해결하세요

기본 번들러로 `parcel`

을 사용했습니다. - `react`

문제의 경우, `react-scripts`

사용. 문제 디렉토리에서 `npm start`

또는 `npx parcel index.html watch`

로 개발서버를 실행하세요.

문제

q1. `offsetTop`, `scrollTop`, `clientHeight`의 상관관계를 통해 해당 기능을 구현하시오.

q2. `resize`에 무관하게 동작하게끔 기능을 구현하시오.

q3. `resize listener`를 적용하여 동작을 구현하시오.

q4. `throttle`과 `debounce`를 통해 해당 동작을 구현하시오.

q5. `Intersection Observer`를 활용하여 기능을 구현하시오.

q6. `Intersection Observer`를 활용하여 React로 해당 기능을 구현하시오.

주요 학습 키워드

스크롤 동작 감지

`offsetTop/scrollTop/clientHeight`의 상관관계

이벤트 리스너, `mousewheel`, `throttle`과 `debounce`, `IntersectionObserver`, `useEffect`

작성해주셔야 하는 question 파일경로

q1

```
./question/q1_js_1/index.js
```

q2

```
./question/q2_js_2/index.js
```

q3

```
./question/q3_js_resize_listener/index.js
```

q4

```
./question/q4_js_throttle/util.js
```

q5

```
./question/q5_js_Intersection_Observer/index.js
```

q6

```
./question/q6_react_Intersections_Observer/src/App.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js_1`

터미널

```
$ npm install  
$ npm start
```

q2

경로 `./question/q2_js_2`

터미널

```
$ npm install  
$ npm start
```

q3

경로 `./question/q3_js_resize_listener`

터미널

```
$ npm install  
$ npm start
```

q4

경로 `./question/q4_js_throttle`

터미널

```
$ npm install  
$ npm start
```

q5

경로 `./question/q5_js_Intersection_Observer`

터미널

```
$ npm install  
$ npm start
```

q6

경로 `./question/q6_react_Interaction_Observer`

터미널

```
$ npm install  
$ npm start
```

Case2 : ScrollSpy - 출제자 해설

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요

A)

```
// index.js

import "./style.css";

const navElem = document.querySelector("#nav");
const navItems = Array.from(navElem.children);
const contentsElem = document.querySelector("#contents");
const contentItems = Array.from(contentsElem.children);
const offsetTops = contentItems.map((elem) => {
  const [ofs, clh] = [elem.offsetTop, elem.clientHeight];
  return [ofs - clh / 2, ofs + clh / 2];
});

window.addEventListener("scroll", (e) => {
  const { scrollTop } = e.target.scrollingElement;
  // do something
  const targetIndex = Math.max(
    offsetTops.findIndex(([from, to]) => scrollTop >= from && scrollTop < to),
    0
  );
  Array.from(navElem.children).forEach((c, i) => {
    c.classList[i === targetIndex ? "add" : "remove"]("on");
  });
});

navElem.addEventListener("click", (e) => {
  const targetElem = e.target;
  if (targetElem.tagName === "BUTTON") {
    const targetIndex = navItems.indexOf(targetElem.parentElement);
    contentItems[targetIndex].scrollIntoView({
      block: "start",
      behavior: "smooth",
    });
  }
});
```

```
    }  
});
```

해설

offsetTop, scrollTop, clientHeight의 상관관계를 이해한다. scrollIntoView의 사용법을 익힌다.

q2. resize에 무관하게 동작하게끔 처리해보자

A)

```
// index.js  
  
import "./style.css";  
  
const navElem = document.querySelector("#nav");  
const navItems = Array.from(navElem.children);  
const contentsElem = document.querySelector("#contents");  
const contentItems = Array.from(contentsElem.children);  
const getOffsetTops = (() => {  
  let offsetTops = [];  
  let prevHeight = 0;  
  return () => {  
    if (window.innerHeight === prevHeight) {  
      return offsetTops;  
    }  
    offsetTops = contentItems.map(elem => {  
      const [ofs, clh] = [elem.offsetTop, elem.clientHeight];  
      return [ofs - clh / 2, ofs + clh / 2];  
    });  
    prevHeight = window.innerHeight;  
    return offsetTops;  
  };  
})();  
  
window.addEventListener("scroll", e => {  
  const { scrollTop } = e.target.scrollingElement;  
  const targetIndex = Math.max(  
    getOffsetTops().findIndex(  
      ([from, to]) => scrollTop >= from && scrollTop < to  
    ),  
    0  
  );
```

```

Array.from(navElem.children).forEach((c, i) => {
  i === targetIndex ? c.classList.add("on") : c.classList.remove("on");
});
});

navElem.addEventListener("click", (e) => {
  const targetElem = e.target;
  if (targetElem.tagName === "BUTTON") {
    const targetIndex = navItems.indexOf(targetElem.parentElement);
    contentItems[targetIndex].scrollIntoView({
      block: "start",
      behavior: "smooth",
    });
  }
});

```

해설

화면 높이가 바뀌면 변경내용을 반영할 필요가 있다.

q3. resize listener를 적용해서 구현해보자.

A)

```

// index.js

import "./style.css";

const navElem = document.querySelector("#nav");
const navItems = Array.from(navElem.children);
const contentsElem = document.querySelector("#contents");
const contentItems = Array.from(contentsElem.children);

let offsetTops = [];
const getOffsetTops = () => {
  // do something
  offsetTops = contentItems.map(elem => {
    const [ofs, clh] = [elem.offsetTop, elem.clientHeight];
    return [ofs - clh / 2, ofs + clh / 2];
  });
};
getOffsetTops();

```

```

window.addEventListener("scroll", e => {
  const { scrollTop } = e.target.scrollingElement;
  const targetIndex = Math.max(
    offsetTops.findIndex(([from, to]) => scrollTop >= from && scrollTop < to),
    0
  );
  Array.from(navElem.children).forEach((c, i) => {
    i === targetIndex ? c.classList.add("on") : c.classList.remove("on");
  });
});

window.addEventListener("resize", getOffsetTops);

navElem.addEventListener("click", e => {
  const targetElem = e.target;
  if (targetElem.tagName === "BUTTON") {
    const targetIndex = navItems.indexOf(targetElem.parentElement);
    contentItems[targetIndex].scrollIntoView({
      block: "start",
      behavior: "smooth"
    });
  }
});

```

해설

2의 함수는 리사이즈가 있을 때엔 의미가 있으나, 그렇지 않은 경우에도 스크롤할 때마다 함수가 실행되면서 각 엘리먼트의 높이를 구하게 된다. 이보다는 리사이즈시에만 계산을 하고, 평소에는 미리 계산된 값을 이용하는게 효율적일 것이다.

q4. throttle로 처리

A)

```

//util.js
export const debounce = (func, delay) => {
  let timeoutId = null;
  return (...arg) => {
    clearTimeout(timeoutId);
    timeoutId = setTimeout(func.bind(null, ...arg), delay);
  };
};

```

```

export const throttle = (func, ms) => {
  let throttled = false;
  // do something
  return (...args) => {
    if (!throttled) {
      throttled = true;
      setTimeout(() => {
        func(...args);
        throttled = false;
      }, ms);
    }
  };
};

```

해설

스크롤 이벤트는 스크롤 동작을 하는 동안 계속해서 발생하므로 모든 이벤트에 대해 콜백을 호출하는 것은 성능에 좋지 않다. 마지막 이벤트만을 감시하는 것으로 충분했던 무한스크롤과 달리 연속적인 이벤트에 대해 꾸준히 변경사항을 반영하는 것이 필요하므로, throttle이 적합하다.

q5. intersection Observer 활용

A)

```

//index.js

import "./style.css";

const navElem = document.querySelector("#nav");
const navItems = Array.from(navElem.children);
const contentsElem = document.querySelector("#contents");
const contentItems = Array.from(contentsElem.children);

const scrollSpyObserver = new IntersectionObserver(
  (entries) => {
    // do something
    const { target } = entries.find((entry) => entry.isIntersecting) || {};
    const targetIndex = contentItems.indexOf(target);
    Array.from(navElem.children).forEach((c, i) => {
      c.classList[i === targetIndex ? "add" : "remove"]("on");
    });
  },
{

```

```

        root: null,
        rootMargin: "0px",
        threshold: 0.5,
    }
);

contentItems.forEach((item) => scrollSpyObserver.observe(item));

navElem.addEventListener("click", (e) => {
    const targetElem = e.target;
    if (targetElem.tagName === "BUTTON") {
        const targetIndex = navItems.indexOf(targetElem.parentElement);
        contentItems[targetIndex].scrollIntoView({
            block: "start",
            behavior: "smooth",
        });
    }
});

```

q6. React + intersection Observer 활용

A)

```

//App.js

import React, { useState, useRef, useEffect } from "react";
import Nav from "./Nav";
import Content from "./Content";
import "./style.css";

const pages = Array.from({ length: 8 }).map((_, i) => i + 1);

const App = () => {
    const [viewIndex, setViewIndex] = useState(0);
    const contentRef = useRef([]);
    const moveToPage = (index) => () => {
        // do something
        contentRef.current[index].scrollIntoView({
            block: "start",
            behavior: "smooth",
        });
    };

    const scrollSpyObserver = new IntersectionObserver(
        (entries) => {
            // do something
        }
    );

```

```
const { target } = entries.find((entry) => entry.isIntersecting) || {};
setViewIndex(contentRef.current.indexOf(target));
},
{
  root: null,
  rootMargin: "0px",
  threshold: 0.5,
}
);

useEffect(() => {
  contentRef.current.forEach((item) => scrollSpyObserver.observe(item));
  return () => {
    contentRef.current.forEach((item) => scrollSpyObserver.unobserve(item));
  };
}, []);

return (
<div id="app">
  <Nav pages={pages} viewIndex={viewIndex} moveToPage={moveToPage} />
  <div id="contents">
    {pages.map((p, i) => (
      <Content key={p} ref={(r) => (contentRef.current[i] = r)} page={p} />
    ))}
  </div>
</div>
);
};

export default App
```

Case2 : ScrollSpy - 대기업 S사 프론트엔드 개발자 님의 답안

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요

A)

```
// index.js
/**
 * [!] 기존에 이미 답안이 작성되어 있어 다른 방법으로 구현
 * 보여지는 영역에 따라 버튼 활성
 */
const activeButtonEls = document.querySelectorAll("#nav > li")
let latestActiveButtonEl = activeButtonEls[0];

const intersectionObserver = new IntersectionObserver((entries) => {
  entries.forEach((entry) => {
    const {
      isIntersecting,
      boundingClientRect
    } = entry;
    /**
     * //이지의 현재 스크롤 위치
     */
    const scrollTop = window.pageYOffset;
    /**
     * Intersection 이벤트 발생한 요소의 높이 값
     */
    const { height } = boundingClientRect;

    if (isIntersecting) {
      /**
       * 스크롤 위치 / 요소의 높이 = 반올림 ⇒ 현재 요소의 위치
       */
      const index = Math.round(scrollTop / height);
      const activeButtonEl = activeButtonEls[index];
      /**
       */

    }
  })
})
```

```

        * 마지막 활성된 버튼에서 `on` 클래스를 제거
        */
    latestActiveButtonEl.classList.remove('on');

    /**
     * 현재 보여지는 요소와 동일한 순번에 있는 버튼에 `on` 클래스 추가
     */
    activeButtonEl.classList.add('on');

    /**
     * 현재 활성된 버튼 요소를 `latestActiveButtonEl`에다 저장
     */
    latestActiveButtonEl = activeButtonEl;
}

}

}, {
/**
 * 화면에 해당 요소가 50% 이상 보여지면 Intersection 이벤트 발생
 */
threshold: .5
});

/**
 * `content` 하위 `div` 요소들을 Intersection 감시 요소로 등록
*/
document.querySelectorAll('#contents > div').forEach((contentEl) => {
intersectionObserver.observe(contentEl);
});

```

해설

IntersectionObserver를 이용하여 화면에 해당 요소가 보여지는 타이밍에 이벤트를 실행하여 효율적으로 버튼활성 처리 할 수 있습니다.

참고자료

https://developer.mozilla.org/ko/docs/Web/API/Intersection_Observer_API

q2. resize에 무관하게 동작하게끔 처리해보자

A)

[문제 1 풀이와 동일합니다. :)]

해설

스크롤이 발생되는 시점에 요소의 높이, window 스크롤 위치를 구하기 때문에 화면 사이즈 변경과 무관하게 잘 동작합니다.

q3. resize listener를 적용해서 구현해보자.

A)

[문제 1, 2 풀이와 동일합니다. :)]

해설

스크롤이 발생되는 시점에 요소의 높이, window 스크롤 위치를 구하기 때문에 화면 사이즈 변경과 무관하게 잘 동작합니다.

q4. throttle로 처리

A)

```
// util.js
export const throttle = (func, ms) => {
  let latestExecuteTime = 0;
  let debounceFn = debounce(func, ms);
  return (...args) => {
    if (latestExecuteTime + ms < Date.now()) {
      func(...args);
    } else {
```

```
        debounceFn(...args);
    }
}
};
```

해설

이벤트가 연속하여 발생하는 경우 `throttle`, `debounce`를 이용하여 불필요한 처리를 제한하여 성능을 향상 시킬수 있습니다.

팁

`throttle`, `debounce` 동작원리와 차이를 잘 알아두었다가 적절하게 활용하면 좋을것 같습니다.

Throttle 와 Debounce 개념 정리하기

q5. intersection Observer 활용

A)

```
// App.js
...
const scrollSpyObserver = new IntersectionObserver(
  (entries) => {
    entries.forEach((entry) => {
      const {
        isIntersecting,
        boundingClientRect
      } = entry;
      /**
       * 페이지의 현재 스크롤 위치
      */
      const scrollTop = window.pageYOffset;
      /**
     */
```

```

        * Intersection 이벤트 발생한 요소의 높이 값
        */
    const { height } = boundingClientRect;

    if (isIntersecting) {
        /**
         * 스크롤 위치 / 요소의 높이 = 반올림 ⇒ 현재 요소의 위치
         */
        const index = Math.round(scrollTop / height);
        const activeButtonEl = activeButtonEls[index];
        /**
         * 마지막 활성된 버튼에서 `on` 클래스를 제거
         */
        latestActiveButtonEl.classList.remove('on');
        /**
         * 현재 보여지는 요소와 동일한 순번에 있는 버튼에 `on` 클래스 추가
         */
        activeButtonEl.classList.add('on');
        /**
         * 현재 활성된 버튼 요소를 `latestActiveButtonEl`에다 저장
         */
        latestActiveButtonEl = activeButtonEl;
    }
}
},
{
    root: null,
    rootMargin: "0px",
    threshold: 0.5,
}
);
...

```

해설

IntersectionObserver를 이용하여 화면에 해당 요소가 보여지는 타이밍에 이벤트를 실행하여 효율적으로 버튼활성 처리 할 수 있습니다.

결론

Intersection Observer를 이용하면 스크롤 이벤트를 이용하지 않고 간단히 해결이 가능하고, 스크롤 이벤트를 이용하는 것 보다 좋은 성능을 낼 수 있습니다.

Case3 : Infinite scroll

케이스 주제

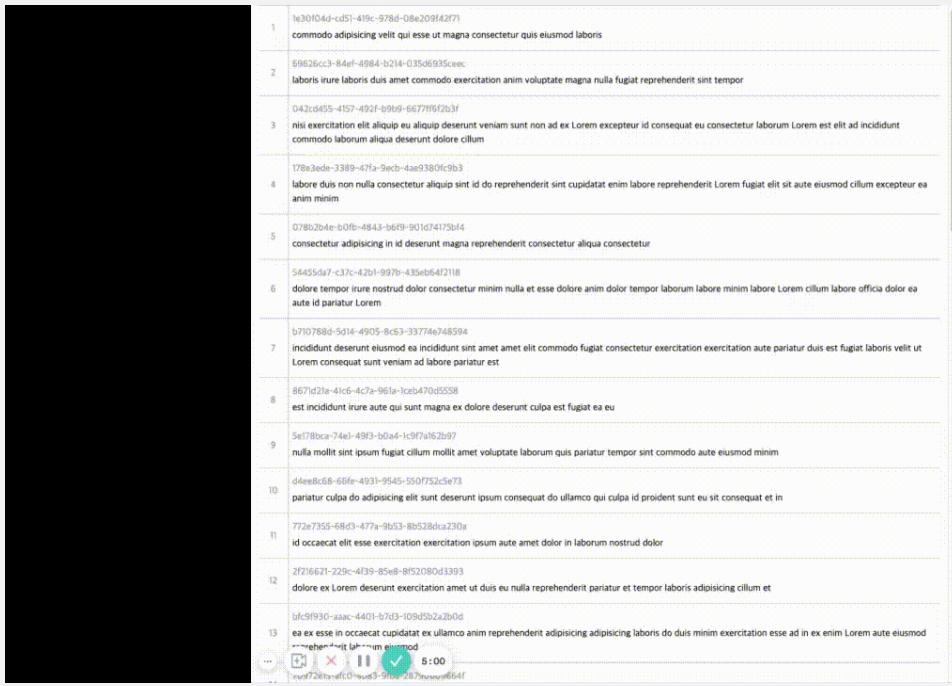
Q. 무한 스크롤 목록뷰를 구현하세요.

기능 요구사항

- 최초에는 20개의 목록을 불러옵니다.
- 스크롤을 최하단으로 이동시 ‘loading’ 상태표시가 나타나며, 이후의 20개 목록을 더 가져옵니다.
- 로딩 완료시 ‘loading’ 표시가 사라지며, 가져온 목록이 하단에 추가됩니다. (무한 반복)

기능 작동 이미지

1. 최초 접속시



실행 방법 / 풀이 방법 안내

문제 풀기 방식 :

1. 터미널에서 각 문제 폴더 디렉토리로 이동하여 npm install로 의존성을 설치
2. package.json을 참고하여, 명시된 scripts 명령어로 개발서버 실행.
3. 코드 수정하면서 문제 해결하세요

기본 번들러로 `parcel`

을 사용했습니다. - `react`

문제의 경우, `react-scripts`

사용. 문제 디렉토리에서 `npm start`

또는 `npx parcel index.html watch`

로 개발서버를 실행하세요.

문제

q1. Javascript - `scrollTop`

, `clientHeight`

, `scrollHeight`

등의 요소의 프로퍼티를 활용하여 해결하시오.

q2. Javascript - 앞서 작성한 코드에 debounce를 적용해 기능을 구현하시오.

q3. Javascript - 다음으로 IntersectionObserver를 활용해 기능을 구현하시오.

q4. React - 마지막으로, 리액트에서 IntersectionObserver를 활용해 기능을 구현하시오.

주요 학습 키워드

스크롤 동작 감지, 이벤트 리스너, scrollTop, mousewheel, IntersectionObserver, useEffect

작성해주셔야 하는 question 파일경로

q1

```
./question/q1_js/index.js
```

q2

```
./question/q2_js_debounce_trottle/index.js
```

```
./question/q2_js_debounce_trottle/util.js
```

q3

```
./question/q3_js_Intersection_Observer/index.js
```

q4

```
./question/q4_react_Interaction_Observer/src/FetchMore.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js`

터미널

```
$ npm install  
$ npm start
```

q2

경로 `./question/q2_js_debounce_trottle`

터미널

```
$ npm install  
$ npm start
```

q3

경로 `./question/q3_js_Intersection_Observer`

터미널

```
$ npm install  
$ npm start
```

q4

경로 ./question/q4_react_Interaction_Observer

터미널

```
$ npm install  
$ npm start
```

Case3 : Infinite Scroll - 출제자 해설

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요

A)

```
// index.js

import renderList from "./listRenderer";
import "./style.css";

const app = document.querySelector("#app");
const fetchMoreTrigger = document.querySelector("#fetchMore");
let page = 0;

const fetchMore = async () => {
  const target = page ? fetchMoreTrigger : app;
  target.classList.add("loading");
  await renderList(page++);
  target.classList.remove("loading");
};

const onScroll = e => {
  const { clientHeight, scrollTop, scrollHeight } = e.target.scrollingElement;
  if (scrollTop >= scrollHeight - clientHeight) {
    fetchMore();
  }
};

document.addEventListener("scroll", onScroll);
fetchMore();
```

해설

scrollTop, clientHeight, scrollHeight 등의 값이 무엇을 의미하는지를 개발자도구의 element 탭을 통해 파악하자.

q2. 앞서 작성한 코드에 debounce를 적용해보자.

A)

```
// index.js

import "./style.css";
import renderList from "./listRenderer";
import { debounce } from "./util";

// Write Javascript code!
const app = document.querySelector("#app");
const fetchMoreTrigger = document.querySelector("#fetchMore");
let page = 0;

const fetchMore = async () => {
    const target = page ? fetchMoreTrigger : app;
    target.classList.add("loading");
    await renderList(page++);
    target.classList.remove("loading");
};

const onScroll = e => {
    const { clientHeight, scrollTop, scrollHeight } = e.target.scrollingElement;
    if (scrollTop >= scrollHeight - clientHeight) {
        fetchMore();
    }
};

document.addEventListener("scroll", debounce(onScroll, 300));
fetchMore();
```

```
// util.js
const getRandomSeconds = () => (Math.round(Math.random() * 5) + 1) * 250;

export const randomTimer = (func, ...args) => resolve => {
    setTimeout(() => resolve(func(...args)), getRandomSeconds());
};

export const debounce = (func, delay) => {
    let timeoutId = null;
    // do something
    return (...arg) => {
        clearTimeout(timeoutId);
        timeoutId = setTimeout(func.bind(null, ...arg), delay);
    };
};
```

```
export const dummyFetcher = (method, args) =>
  new Promise(randomTimer(method, args));
```

해설

scroll 이벤트는 매우 많이 발생하므로 성능 저하가 우려된다. ‘fetchMore’ 동작은 스크롤의 최하단에서만 발생하므로 스크롤 이벤트 중 마지막 값에 대해서만 반응하는 것으로 충분하다. 이를 위한 기법인 throttle / debounce 소개.

q3. 이번에는 Intersection Observer를 활용해보자

A)

```
// index.js

import "./style.css";
import renderList from "./listRenderer";

const app = document.querySelector("#app");
const fetchMoreTrigger = document.querySelector("#fetchMore");
let page = 0;

const fetchMore = async () => {
  const target = page ? fetchMoreTrigger : app;
  target.classList.add("loading");
  await renderList(page++);
  target.classList.remove("loading");
};

const fetchMoreObserver = new IntersectionObserver(([ { isIntersecting } ]) => {
  // do something
  if (isIntersecting) fetchMore();
});
fetchMoreObserver.observe(fetchMoreTrigger);

fetchMore();
```

해설

IntersectionObserver는 자바스크립트 메인 쓰레드에서 실행되는 Event Listener가 아닌 브라우저에서 별도로 마련한 API이므로 성능 저하의 우려가 적다.

q4. React + Intersection Observer를 활용하여 구현해보세요.

A)

```
import React, { useRef, useEffect } from "react";

const FetchMore = ({ loading, setPage }) => {
  const fetchMoreTrigger = useRef(null);
  const fetchMoreObserver = new IntersectionObserver(([ { isIntersecting } ]) => {
    // do something
    if (isIntersecting) setPage(prev => prev + 1);
  });

  useEffect(() => {
    fetchMoreObserver.observe(fetchMoreTrigger.current);
    return () => {
      fetchMoreObserver.unobserve(fetchMoreTrigger.current);
    };
  }, []);

  return (
    <div
      id="fetchMore"
      className={loading ? "loading" : ""}
      ref={fetchMoreTrigger}
    />
  );
};

export default FetchMore;
```

해설

일반적으로 쓰지 않게 되는 시점에 unobserve / removeEventListener 등을 해줘야한다는 것을 잊어버리거나, 혹은 생각해냈더라도 코드 구현이 애매한 경우가 종종 있는데, React에서는 useEffect 내에서 observe - unobserve / addEventListener - removeEvenntListener를 쌍으로 매칭하여 구현하는 것이 자연스럽기 때문에 익숙해지면 잊어버릴 일이 없게 된다.

Case3 : Infinite Scroll - 대기업 S사 프론트엔드 개발자님의 답안

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해보세요

A)

```
// index.js

...

/**
 * 스크롤 임계치
 */
const scrollThreshold = .95;

const onScroll = (e) => {
    // do something (hint: e.target.scrollingElement)
    const {
        scrollTop,
        clientHeight,
        scrollHeight,
    } = e.currentTarget.scrollingElement

    /**
     * 현재 스크롤된 위치를 0-1 값으로 구함
     * `scrollTop` ⇒ 현재 스크롤된 위치, `clientHeight` ⇒ 현재 뷰포트의 높이, `scrollHeight` ⇒ 스크롤 영역 높이
     */
    const scrollRatio = (scrollTop + clientHeight) / scrollHeight;

    /**
     * [!] 브라우저 또는 단말마다 편차가 있을수 있어 1(100%) 값 보다는 `0.95 ~ 0.99` 값을 적용하는게 더 자연스러움
     */
    if (scrollRatio > scrollThreshold) {
        loadMore();
    }
};

document.addEventListener("scroll", onScroll);
loadMore();
```

해설

문제 힌트로 주어진 scrollTop

,clientHeight

,scrollHeight

속성 값을 이용하여 스크롤된 퍼센테이지 값을 구하고, 임계 값(95% 이상)에 초과하는 경우 추가적으로 데이터를 불러오도록 구현.

q2. 앞서 작성한 코드에 debounce를 적용해보자.

A)

```
// index.js

...

/***
 * 스크롤 임계치
 */
const scrollThreshold = .95;

const onScroll = (e) => {
  const {
    scrollTop,
    clientHeight,
    scrollHeight,
  } = e.target.scrollingElement

  /**
   * 현재 스크롤된 위치를 0-1 값으로 구함
   * `scrollTop` ⇒ 현재 스크롤된 위치, `clientHeight` ⇒ 현재 뷰포트의 높이, `scrollHeight` ⇒ 스크롤 영역 높이
   */
  const scrollRatio = (scrollTop + clientHeight) / scrollHeight;

  /**
   * [!] 브라우저 또는 단말마다 편차가 있을수 있어 1(100%) 값 보다는 `0.95 ~ 0.99` 값을 적용하는게 더 자연스러움
   */
  if (scrollRatio > scrollThreshold) {
    fetchMore();
  }
};

const debounceDelay = 100;
const onDebounceScroll = debounce(onScroll, debounceDelay);

document.addEventListener("scroll", onDebounceScroll);
fetchMore();
```

```
// debounce.js
```

```
/**
```

```
* 입력된 `func` 함수가 연속하여 호출되도 마지막으로 "함수가 호출된 시간 + `delay`" 시간이후에 1회만 실행
*
```

```

* @param {function} func
* @param {number} delay 단위는 milliseconds 입니다.
*/
const debounce = (func, delay) => {
    /**
     * `setTimeout` 아이디를 저장
     */
    let procId = null;
    return (...args) => {
        if (procId) {
            /**
             * `procId`가 존재하면 실행되지 않도록 제거
             */
            window.clearTimeout(procId);
        }
        /**
         * `delay` 이후 해당 함수가 실행되도록 `setTimeout`에 태스크를 등록
         */
        procId = setTimeout(() => func(...args), delay);
    }
};

```

```

// util.js

export const debounce = (func, delay) => {
    /**
     * `setTimeout` 실행시 태스크 아이디를 저장
     */
    let timeoutId = null;
    return (...args) => {
        /**
         * 이미 실행 대기중인 태스크가 존재하는 경우 해당 태스크를 제거
         */
        if (timeoutId) {
            clearTimeout(timeoutId);
        }
        /**
         * 입력받은 `delay` 후에 해당 함수가 실행되도록 `setTimeout` 실행
         */
        timeoutId = setTimeout(func, delay, ...args);
    }
};

export const dummyFetcher = (method, args) =>
    new Promise(randomTimer(method, args));

```

해설

q1

에서 구현한 `onScroll`

함수를 `debounce`

유tiles을 적용하여 스크롤이 연속적으로 동작하더라도 이벤트가 매번 실행되지 않고 가장 마지막 동작후 `delay` 가 초과된 시점에 한번만 실행되도록 하여 성능을 개선하였습니다.

팁

```
debounce  
, throttle  
유тиלצה수는 자주 사용되기 때문에 사용방법과 원리를 이해하고 계시는게 좋습니다.
```

<https://www.npmjs.com/package/lodash.debounce>
<https://www.npmjs.com/package/lodash.throttle>

```
debounce  
는 클로저(closure)의 원리를 이해하고, setTimeout  
를 알고 있으면 간단히 구현할 수 있습니다.
```

<https://developer.mozilla.org/ko/docs/Web/JavaScript/Guide/Closures>
[https://developer.mozilla.org/ko/docs/Web/API/WindowTimers.setTimeout](https://developer.mozilla.org/ko/docs/Web/API/WindowTimers	setTimeout)

q3. 이번에는 Intersection Observer를 활용해보자

A)

```
// index.js  
  
...  
  
const fetchMoreObserver = new IntersectionObserver(([ { isIntersecting } ]) => {  
    // do something  
    if(isIntersecting){  
        fetchMore();  
    }  
});  
  
fetchMoreObserver.observe(fetchMoreTrigger);  
  
fetchMore();
```

해설

IntersectionObserver 사용시 위와같은 케이스에서 성능 부하를 크게 줄일수 있습니다.
실제로 IntersectionObserver가 하는 역할을 유사하게 구현하기 위해서는, 스크롤 이벤트가 동작할때마다 각 객체의 offset(위치) 정보를 확인하고 계산하여야하는데, 반복되는 스크롤 이벤트에 따른 부하 + 객체의 offset를 알아내기 위한 속성값을 호출할때 발생하는 reflow 비용이 발생합니다.

관련자료

<https://developer.mozilla.org/ko/docs/Web/API/IntersectionObserver>
<https://developer.mozilla.org/en-US/docs/Web/API/Element/getBoundingClientRect>
<https://gist.github.com/paulirish/5d52fb081b3570c81e3a>

q4. React + Intersection Observer를 활용하여 구현해보세요.

[문제 3] 내용을 참고해 주세요!

결론

Intersection Observer를 이용하면 스크롤 이벤트를 이용하지 않고 간단히 해결이 가능하고, 스크롤 이벤트를 이용하는 것 보다 좋은 성능을 낼 수 있습니다.

Case4 : Dark mode

케이스 주제

Q. 다음과 같이 토글 버튼을 클릭하여 테마(다크 모드/라이트 모드)를 설정하면 테마가 뷰에 반영되도록 구현해보자. 테마는 로컬스토리지에 저장하여 웹페이지를 리로드하거나 다시 접근했을 때 저장된 테마를 적용하도록 한다.

기능 요구사항

1. 로컬스토리지에 저장되어 있는 테마(다크 모드/라이트 모드)를 기준으로 초기 렌더링한다.
2. 로컬스토리지에 저장된 테마가 없다면 라이트 모드로 초기 렌더링한다.
3. 테마를 적용하여 렌더링할 때 기존 테마가 변경되어 깜빡거리는 현상(flash of incorrect theme, FOIT)이 발생하지 않도록 한다.
4. 토글 버튼을 클릭하면 로컬스토리지에 테마를 저장하고 저장된 테마를 기준으로 다시 렌더링한다.

기능 작동 이미지

Light / Dark Mode - Toggle Button



 Lorem ipsum dolor, sit amet consectetur adipisicing elit. Laborum optio ab porro magni
 in sunt ipsam, doloremque minima, itaque sapiente consequatur, repellat velit
 voluptatum accusantium aperiam. Nostrum sunt reprehenderit nemo!

뷰의 기본 템플릿은 다음과 같다. body 요소에 dark 클래스를 추가하면 다크 모드가 적용되고 body 요소에서 dark 클래스를 제거하면 라이트 모드가 적용된다.

Light / Dark Mode - Toggle Button



 Lorem ipsum dolor, sit amet consectetur adipisicing elit. Laborum optio ab porro magni
 in sunt ipsam, doloremque minima, itaque sapiente consequatur, repellat velit
 voluptatum accusantium aperiam. Nostrum sunt reprehenderit nemo!

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree is visible, showing the structure of the HTML document. A green toggle switch button is shown at the top left. The body element has a class attribute set to 'dark'. The 'Properties' panel on the right shows the 'body' element's style rules, including 'font-family: "Open Sans"; font-weight: 300;' and 'display: block; margin: 0;'. There is also a note indicating 'template.html:11' for one of the rules.

문제

1. JS

로컬스토리지에 저장된 테마(다크 모드/라이트 모드)를 기준으로 초기 렌더링
로컬스토리지에 저장된 테마가 없으면 라이트 모드로 초기 렌더링

2. JS

로컬스토리지에 저장된 테마가 없을 때 window.matchMedia 메서드로 사용자 OS 테마를 감지해 이를 테마에 적용
로컬스토리지에 저장된 테마가 있으면 사용자 OS 테마보다 이를 우선하여 적용

3. React 바닐라 자바스크립트로 구현한 dark mode는 body 요소에 클래스를 추가/제거하는 방식으로 동작한다. React에서도 이 방식을 사용하면 컴포넌트에서 body 요소를 조작하는 부수 효과(side effect)에 의존하게 되므로 직관적이지 않고 컴포넌트의 재사용이 어려워지며 FOIT(flash of incorrect theme)을 방지하기도 번거롭다.

요구 사항은 다음과 같다.

함수 컴포넌트와 훙을 사용해 구현한다.

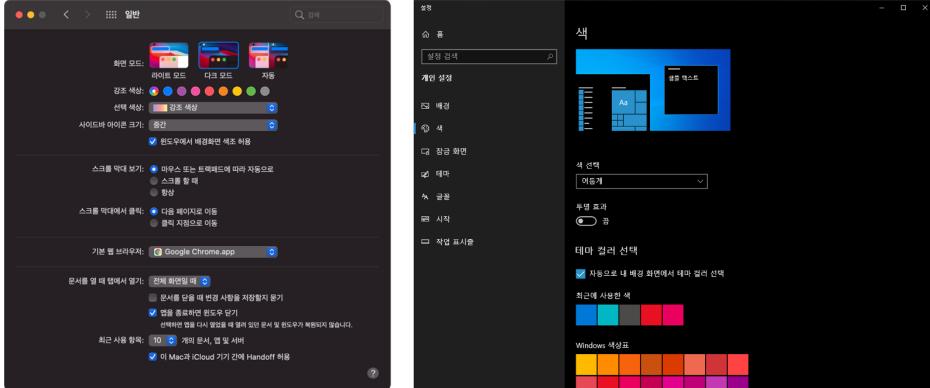
Styled-components의 ThemeProvider를 사용하면 간단하게 테마를 전역 관리할 수 있다. 테마(dark/light)를 객체로 정의하고 Styled-components의 ThemeProvider를 사용해 테마가 필요한 컴포넌트에게 전달한다.

주요 학습 키워드

localStorage
prefers-color-scheme
window.matchMedia
styled-components: theming
useState
useEffect
Context API
useContext

참고

Windows와 macOS 등은 운영 체제 레벨에서 사용자 테마(다크 모드/라이트 모드)를 설정할 수 있다.



CSS의 prefers-color-scheme media query나 자바스크립트의 window.matchMedia 메서드를 사용하면 운영 체제 레벨에서 설정한 사용자 테마를 감지할 수 있다.

```
prefers-color-scheme: Hello darkness, my old friend
```

prefers-color-scheme media query와 window.matchMedia 메서드의 간단한 예제는 다음과 같다.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>
      .themed {
        display: flex;
        justify-content: center;
        align-items: center;
        width: 200px;
        height: 100px;
        background-color: rgb(250, 250, 250);
      }
      .themed::after {
        content: 'Light mode(default)';
      }

      @media (prefers-color-scheme: dark) {
        .themed {
          background-color: #000;
          color: #fff;
        }
        .themed::after {
          content: 'Dark mode detected';
        }
      }
    </style>
  </head>
```

```
<body>
  <div class="themed"></div>
  <script>
    // https://web.dev/prefers-color-scheme
    // https://caniuse.com/?search=prefers-color-scheme
    const darkModeMediaQuery = window.matchMedia('(prefers-color-scheme: dark)');
    console.log(darkModeMediaQuery);
    // MediaQueryList {media: "(prefers-color-scheme: dark)", matches: true, onchange: null}

    darkModeMediaQuery.addListener(e => {
      const darkModeOn = e.matches;
      console.log(`Dark mode is ${darkModeOn ? '🌙 on' : '☀️ off'}`);
    });
  </script>
</body>
</html>
```

작성해주셔야 하는 question 파일경로

q1

./question/1.js-1/index.js

q2

./question/2.js-2/index.js

q3

./question/3.react/src/App.js

이외 필요한 디렉토리 / 파일구조는 각자 작성하시면 됩니다.

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/1.js-1

index.html 열기

q2

경로 ./question/2.js-2

index.html 열기

q3

경로 ./question/3.react

터미널

```
$ npm install  
$ npm start
```

Case4 : DarkMode - 출제자 해설

q1. Javascript

A)

```
document.addEventListener('DOMContentLoaded', () => {
    // 1. 로컬 스토리지에 저장되어 있는 테마(다크 모드/라이트 모드)를 기준으로 초기 렌더링한다.
    const theme = localStorage.getItem('theme');

    // 2. 로컬 스토리지에 저장된 테마가 없다면 라이트 모드로 초기 렌더링한다.
    if (!theme) localStorage.setItem('theme', 'light');

    // 로컬스토리지에 저장된 theme이 dark이면 body 요소에 dark 클래스를 추가하고 그렇지 않으면 제거한다.
    document.body.classList.toggle('dark', theme === 'dark');

    // 3. 테마를 적용하여 초기 렌더링할 때 기존 테마가 변경되어 깜빡거리는 현상(flash of incorrect theme, FOIT)이 발생하지 않도록 한다.
    // .toggle-button-switch 요소와 .toggle-button-text 요소는 0.3초(duration)에 걸쳐 transition이 일어나도록 되어 있다.
    setTimeout(() => {
        document.body.style.visibility = 'visible';
    }, 300);
});

// 4. 토큰 버튼을 클릭하면 로컬 스토리지에 테마를 저장하고 저장된 테마를 기준으로 다시 렌더링한다.
document.querySelector('.toggle-button').onclick = () => {
    // 로컬스토리지에 저장된 theme이 dark이면 light로 변경하고 light이면 dark로 변경한다.
    localStorage.setItem('theme', `${localStorage.getItem('theme') === 'dark' ? 'light' : 'dark'}`);

    // body 요소에 dark 클래스를 추가되어 있으면 제거하고 그렇지 않으면 추가한다.
    document.body.classList.toggle('dark');
};
```

q1. Javascript-1

A)

```
document.addEventListener('DOMContentLoaded', () => {
    // 1. 로컬 스토리지에 저장되어 있는 테마(다크 모드/라이트 모드)를 기준으로 초기 렌더링한다.
    const theme = localStorage.getItem('theme');

    // 2. 로컬 스토리지에 저장된 테마가 없다면 라이트 모드로 초기 렌더링한다.
    if (!theme) localStorage.setItem('theme', 'light');

    // 로컬스토리지에 저장된 theme이 dark이면 body 요소에 dark 클래스를 추가하고 그렇지 않으면 제거한다.
    document.body.classList.toggle('dark', theme === 'dark');

    // 3. 테마를 적용하여 초기 렌더링할 때 기존 테마가 변경되어 깜빡거리는 현상(flash of incorrect theme, FOIT)이 발생하지 않도록 한다.
    // .toggle-button-switch 요소와 .toggle-button-text 요소는 0.3초(duration)에 걸쳐 transition이 일어나도록 되어 있다.
    setTimeout(() => {
        document.body.style.visibility = 'visible';
    }, 300);
});
```

```

    }, 300);
});

// 4. 토큰 버튼을 클릭하면 로컬 스토리지에 테마를 저장하고 저장된 테마를 기준으로 다시 렌더링한다.
document.querySelector('.toggle-button').onclick = () => {
    // 로컬스토리지에 저장된 theme이면 dark로 변경하고 light이면 dark로 변경한다.
    localStorage.setItem('theme', `${localStorage.getItem('theme') === 'dark' ? 'light' : 'dark'}`);

    // body 요소에 dark 클래스를 추가되어 있으면 제거하고 그렇지 않으면 추가한다.
    document.body.classList.toggle('dark');
};


```

q2. Javascript-2

```

document.addEventListener('DOMContentLoaded', () => {
    let theme = localStorage.getItem('theme');

    // 1. 로컬 스토리지에 저장된 테마가 없으면 window.matchMedia 메서드로 사용자 OS 테마를 감지해 이를 테마에 적용한다.
    // 2. 로컬 스토리지에 저장된 테마가 있다면 사용자 OS 테마보다 이를 우선 적용한다.
    if (!theme) {
        // 사용자 OS 테마가 다크 모드이면 matches는 true다.
        const {
            matches
        } = window.matchMedia('(prefers-color-scheme: dark)');
        theme = matches ? 'dark' : 'light';
        localStorage.setItem('theme', theme);
    }

    // 로컬스토리지에 저장된 theme이면 body 요소에 dark 클래스를 추가하고 그렇지 않으면 제거한다.
    document.body.classList.toggle('dark', theme === 'dark');
    // FOIT 방지
    setTimeout(() => {
        document.body.style.visibility = 'visible';
    }, 300);
};

document.querySelector('.toggle-button').onclick = () => {
    // 로컬스토리지에 저장된 theme이면 light로 변경하고 light이면 dark로 변경한다.
    localStorage.setItem('theme', `${localStorage.getItem('theme') === 'dark' ? 'light' : 'dark'}`);

    // body 요소에 dark 클래스를 추가되어 있으면 제거하고 그렇지 않으면 추가한다.
    document.body.classList.toggle('dark');
};

```

q3. React

A)

해설 영상 참조

Case5 : Stop watch

케이스 주제

Q. 아래와 같이 작동하는 스톱워치를 구현하시오.

기능 요구사항

1. 스톱워치의 시간은 mm:ss:ms 형식(예시 '01:59:89')으로 표시한다.

구분 의미 범위 비고

mm 분 0 ~

ss 초 0 ~ 59

ms 미리초 0 ~ 99 미리초는 100분의 1초를 나타내지만 10ms 단위로 표시한다.

2. 컨트롤 버튼

스톱워치는 2개의 컨트롤 버튼을 가진다.

왼쪽 버튼: 클릭할 때마다 Start/Stop으로 토글된다.

오른쪽 버튼: 오른쪽 버튼은 아래와 같이 왼쪽 버튼에 종속적이다. 왼쪽 버튼이 Start이면 오른쪽 버튼은 Reset이고 왼쪽 버튼이 Stop이면 오른쪽 버튼은 Lap이다.

왼쪽 버튼 오른쪽 버튼

Start Reset

Stop Lap

각 버튼의 기능은 다음과 같다.

버튼 기능

Start 스톱워치를 시작한다.

Stop 스톱워치를 일시정지시킨다.

Reset 스톱워치와 랩 타임을 초기화한다. 스톱워치의 현재 시간이 '00:00:00'이면 disabled 상태이어야 한다.

Lap 랩 타임을 기록한다.

기능 작동 이미지

Stopwatch

00:00:00

Start

Reset

문제

스톱워치 경과 시간을 '00:00:00' 형식의 문자열로 변환

스톱워치 경과 시간을 렌더링

랩 타임 렌더링

Start/Stop 버튼 클릭 이벤트 핸들러

Reset/Lap 버튼 클릭 이벤트 핸들러

2. React

함수 컴포넌트와 흑을 사용해 구현한다.

스타일은 CSS, Sass, CSS Module, styled-components 중 어느 것을 사용해도 좋으나 가급적 styled-components 사용을 권장한다.

주요 학습 키워드

CSS 그리드 레이아웃

setInterval

Document.createDocumentFragment

useState

useEffect

작성해주셔야 하는 question 파일경로

q1

`./question/1.js/index.js`

q2

`./question/2.react/src/App.js`

이외 필요한 디렉토리 / 파일구조는 각자 작성하시면 됩니다.

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/1.js`

[index.html 열기](#)

q6

경로 [./question/2.react](#)

터미널

```
$ npm install  
$ npm start
```

Case5 : StopWatch 1 - 출제자 해설

q1. Javascript

A)

```
document.querySelector('.stopwatch').onclick = (() => {
  let isRunning = false;
  let elapsedTime = { mm: 0, ss: 0, ms: 0 };
  let laps = [];

  const [$btnStartOrStop, $btnResetOrLap] = document.querySelectorAll('.stopwatch > .control');

  // 스톱워치의 경과 시간을 '00:00:00' 형식의 문자열로 변환한다.
  const formatElapsedTime = () => {
    // 1 => '01', 10 => '10'
    const format = n => (n < 10 ? '0' + n : n + '');
    return ({ mm, ss, ms }) => `${format(mm)}:${format(ss)}:${format(ms)}`;
  }();

  // 스톱워치의 경과 시간을 렌더링한다.
  const renderElapsedTime = () => {
    const $display = document.querySelector('.stopwatch > .display');
    return () => {
      $display.textContent = formatElapsedTime(elapsedTime);
    };
  }();

  // 랩 타임을 렌더링한다.
  const renderLaps = () => {
    const $laps = document.querySelector('.stopwatch > .laps');

    // 랩 타임을 생성하고 DOM에 반영한다.
    const createLapElement = (newLap, index) => {
      const $fragment = document.createDocumentFragment();

      const $index = document.createElement('div');
      $index.textContent = index;
      $fragment.appendChild($index);

      const $newLab = document.createElement('div');
      $newLab.textContent = formatElapsedTime(newLap);
      $fragment.appendChild($newLab);

      $laps.appendChild($fragment);
    };
  };
});
```

```

};

// 랩 타임을 초기화(DOM에서 모두 제거)한다.
const removeAllLapElement = () => {
  document.querySelectorAll('.laps > div:not(.lap-title)').forEach($lap => $lap.remove());
  $laps.style.display = 'none';
};

return () => {
  const { length } = laps;

  if (length) {
    const newLap = laps[length - 1]; // 마지막 lap을 DOM에 append한다.
    createLapElement(newLap, length);
  } else {
    removeAllLapElement();
  }
};

})();

// Start/Stop 버튼 클릭 이벤트 핸들러
const handleBtnStartOrStop = () => {
  let timerId = null;

  // Stop => Start
  const start = () => {
    let { mm, ss, ms } = elapsedTime;

    timerId = setInterval(() => {
      ms += 1;
      if (ms >= 100) {
        ss += 1;
        ms = 0;
      }
      if (ss >= 60) {
        mm += 1;
        ss = 0;
      }
    }, 10); // 10ms 단위로 증가
  };

  // $btnResetOrLap의 disabled 상태 변경
  $btnResetOrLap.disabled = !(mm + ss + ms);

  elapsedTime = { mm, ss, ms };
  renderElapsedTime();
}, 10); // 10ms 단위로 증가
};

// Start => Stop
const stop = () => clearInterval(timerId);

return () => {
  isRunning ? stop() : start();
  isRunning = !isRunning;

  // isRunning이 변경되면 버튼 텍스트를 변경한다.
  $btnStartOrStop.textContent = isRunning ? 'Stop' : 'Start';
  $btnResetOrLap.textContent = isRunning ? 'Lap' : 'Reset';
};
}

```

```
})();

// Reset/Lap 버튼 클릭 이벤트 핸들러
const handleBtnResetOrLap = () => {
    // elapsedTime과 laps를 초기화한다.
    const reset = () => {
        // $btnResetOrLap의 disabled 상태 변경
        $btnResetOrLap.disabled = true;

        elapsedTime = { mm: 0, ss: 0, ms: 0 };
        renderElapsedTime();

        laps = [];
        renderLaps();
    };

    // elapsedTime을 laps에 추가한다.
    const addLap = () => {
        laps = [...laps, elapsedTime];
        renderLaps();
    };

    return () => {
        isRunning ? addLap() : reset();
    };
})();

return ({ target }) => {
    if (!target.classList.contains('control')) return;
    target === $btnStartOrStop ? handleBtnStartOrStop() : handleBtnResetOrLap();
};

})();
```

q2. React

A)

해설 영상 참조

Case6 : Tabs

케이스 주제

Q. 아래와 같이 작동하는 탭 메뉴를 구현하시오.

기능 요구사항

요구 사항은 아래와 같다.

1. 비동기 함수인 fetchTabsData 함수를 사용해 탭 메뉴 정보를 담고 있는 배열을 전달받아 탭 메뉴를 생성한다.
2. fetchTabsData 함수는 프로미스를 반환하며 이 프로미스가 fulfilled 상태가 될 때까지는 1초 소요된다. 프로미스가 fulfilled 상태가 될 때까지 스피너(.spinner 요소)를 표시한다.
3. 탭 정보를 담고 있는 배열의 length는 가변적이다.

React

1. 함수 컴포넌트와 흑을 사용해 구현한다.
2. 스타일은 CSS, Sass, CSS Module, styled-components 중 어느 것을 사용해도 좋으나 가급적 styled-components 사용을 권장한다.

실행 방법 / 문제 풀이 방법

q1. Javascript - index.html 실행 q2. React

```
$ yarn install $ yarn start
```

주요 학습 키워드

```
비동기 처리와 Promise  
React hook을 사용해 데이터 패칭하기
```

작성해주셔야 하는 question 파일경로

q1

```
./question/q1_js/index.js
```

q2

```
./question/q2_react/src/App.js
```

이외 필요한 디렉토리 / 파일들은 각자 작성하시면 됩니다.

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js`

index.html 열기

q2

경로 `./question/q2_react`

터미널

```
$ yarn install  
$ yarn start
```

Case7 : Analog clock

케이스 주제

Q. 아래와 같이 동작하는 아날로그 시계를 구현해보십시오.

기능 요구사항

JavaScript

1. 시계의 시침(.hand.hour 요소), 분침(.hand.minute 요소), 초침(.hand.second 요소)을 1초 간격으로 회전시켜 현재 시간을 표시하시오.
2. 뷰의 기본 템플릿을 그대로 사용해도 좋으나, 더 나은 방법이 있다면 변경해도 좋습니다.

React

3. 함수 컴포넌트와 흑을 사용해 구현하시오.
4. 스타일은 CSS, Sass, CSS Module, styled-components 중 어느 것을 사용해도 좋으나 가급적 styled-components 사용을 권장한다.

기능 작동 이미지

Analog clock



문제

1. JS : Javascript로 해당 기능을 구현하시오.
2. React : React로 해당 기능을 구현하시오.

styled-components를 사용하여 뷰를 구현하시오.
useRef와 휙을 사용하여 1번과 동일한 동작을 구현하시오.

주요 학습 키워드

CSS 변수(CSS 커스텀 프로퍼티)
styled-components
useRef
useEffect

작성해주셔야 하는 question 파일경로

q1

```
./question/1.js/index.js
```

q2

```
./question/2.react-styled-component/src/App.js
```

이외 필요한 디렉토리/파일구조는 각자 작성하시면 됩니다.

q3

```
./question/3.react-useRef/src/App.js
```

출제자 강사님 코드 기반으로, 해당 경로에서 요구한 문제사항을 해결해주세요

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/1.js`

index.html 열기

q2

경로 `./question/2.react-styled-component`

터미널

```
$ npm install  
$ npm start
```

q3

경로 `./question/3.react-useRef`

터미널

```
$ npm install  
$ npm start
```

Case7 : Analog 1 - 출제자 해설

q1. Javascript

A)

```
const renderTime = (() => {
    const $hourHand = document.querySelector('.hand.hour');
    const $minuteHand = document.querySelector('.hand.minute');
    const $secondHand = document.querySelector('.hand.second');

    return () => {
        const date = new Date();
        const seconds = date.getSeconds();
        const minutes = date.getMinutes();
        const hours = date.getHours();

        // Updating a CSS variable(CSS custom property)
        // @see: https://css-tricks.com/updating-a-css-variable-with-javascript
        // 초침: 1초당 6도(360deg/60s) 회전
        $secondHand.style.setProperty('--deg', seconds * 6);
        // 분침: 1시간당 360도, 1분당 6도(360deg/60m), 1초당 0.1도(6deg/60s) 회전
        $minuteHand.style.setProperty('--deg', minutes * 6 + seconds * 0.1);
        // 시침: 1시간당 30도(360deg/12h), 1분당 0.5도(30deg/60m), 1초당 약 0.0083도(0.5deg/60s) 회전
        $hourHand.style.setProperty('--deg', hours * 30 + minutes * 0.5 + seconds * (0.5 / 60));
    };
})();

document.addEventListener('DOMContentLoaded', () => {
    setInterval(renderTime, 1000);
});
```

q2. React

A)

해설 영상 참조

Case8 : Carousel

케이스 주제

Q. 아래와 같이 작동하는 캐러셀을 구현하시오.

캐러셀(Carousel)은 컨텐츠를 슬라이드 형태로 순환하며 표시하는 UI를 말한다. 캐러셀은 비생산적인 디자인 패턴이라는 주장이 있기도 하지만 사용자가 스크롤을 내리지 않은 상태에서도 많은 정보를 노출할 수 있는 장점이 있어 많은 웹사이트에서 사용하고 있다.

기능 요구사항

요구 사항은 아래와 같다.

1. 무한 루핑 기능을 지원한다.
2. 슬라이딩 애니메이션을 지원한다.
3. 각 슬라이드의 width/height는 가변적이다. 단, 모든 슬라이드의 width/height는 동일하다.
4. 슬라이드 이동 버튼을 연타해도 이상없이 동작해야 한다.
5. 캐러셀 슬라이더를 표시할 HTML 요소와 슬라이드 이미지의 url로 구성된 배열을 전달하면 동적으로 캐러셀 슬라이더를 생성한다.

캐러셀 슬라이더 알고리즘

정상적으로 한칸씩 이동하는 경우

.carousel 요소의 위치를 고정시키고 넘치는 자식 요소를 감추고
.carousel-slides 요소 .carousel 요소의 width만큼 좌우로 이동시킨다.

마지막 슬라이더를 클론하여 추가

.carousel-slides {
 --currentSlide: 1;
 --duration: 0;
 display: flex;
 transition: transform calc(var(--duration) * 1ms) ease-out;
 transform: translate3D(calc(var(--currentSlide) * -100%), 0, 0);
}

첫번째 슬라이더를 클론하여 추가

.carousel-slides {
 --currentSlide: 2;
 --duration: 500;
 display: flex;
 transition: transform calc(var(--duration) * 1ms) ease-out;
 transform: translate3D(calc(var(--currentSlide) * -100%), 0, 0);
}

슬라이더의 선두 또는 가장 뒤에서 이동하는 경우

.carousel 요소의 위치를 고정시키고 넘치는 자식 요소를 감추고
.carousel-slides 요소 .carousel 요소의 width만큼 좌우로 이동시킨다.

마지막 슬라이더를 클론하여 추가

.carousel-slides {
 --currentSlide: 1;
 --duration: 0;
 display: flex;
 transition: transform calc(var(--duration) * 1ms) ease-out;
 transform: translate3D(calc(var(--currentSlide) * -100%), 0, 0);
}

첫번째 슬라이더를 클론하여 추가

.carousel-slides {
 --currentSlide: 3;
 --duration: 0;
 display: flex;
 transition: transform calc(var(--duration) * 1ms) ease-out;
 transform: translate3D(calc(var(--currentSlide) * -100%), 0, 0);
}

React

- 함수 컴포넌트와 흑을 사용해 구현한다.
- 스타일은 CSS, Sass, CSS Module, styled-components 중 어느 것을 사용해도 좋으나 가급적 styled-components 사용을 권장한다.

주요 학습 키워드

CSS 변수(CSS 커스텀 프로퍼티)

HTMLElement.offsetWidth

HTMLElement:transitionend event

styled-components

useState

작성해주셔야 하는 question 파일경로

q1_js

./question/q1_js/src/app.js

q2_react

./question/q2_react/src/App.js

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/q1_js

index.html 열기

q2

경로 ./question/q2_react

터미널

```
$ yarn install  
$ yarn start
```

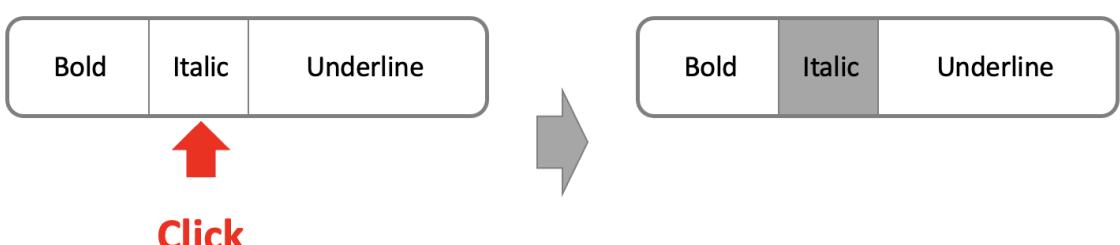
Case9 : Toggle button

케이스 주제

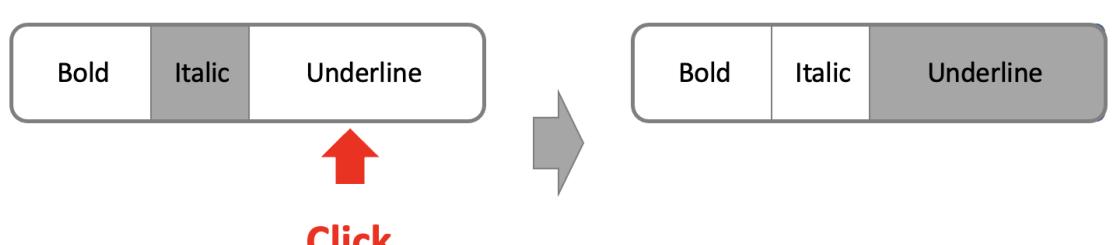
Q. on, off 기능 (toggle 기능)을 가지는 버튼 리스트 컴포넌트를 구현하십시오.

기능 요구사항

- 각 버튼이 선택이 되었는지 안되었는지를 확인할 수 있다.



- 여러개의 버튼 중 하나의 버튼만 toggle 할 수 있다.



기능 작동 이미지



문제

- q1. 데이터에 따른 버튼리스트를 가로로 출력하시오.
- q2. 한개의 버튼만이 on이 될 수 있도록 하시오.
- q3. 선택된 버튼의 index를 application으로 전달 하시오.

주요 학습 키워드

버튼 스타일 적용
컴포넌트와 어플리케이션과의 통신

작성해주셔야 하는 question 파일경로

q1

.src/question/toggle-button/index.js

line : 33

q2

.src/question/toggle-button/index.js

line : 44

q2

.src/question/toggle-button/index.js

line : 47

실행 방법 및 의존성 모듈 설치

경로 ./

(root directory)

터미널

```
$ npm install
$ npm run dev
```



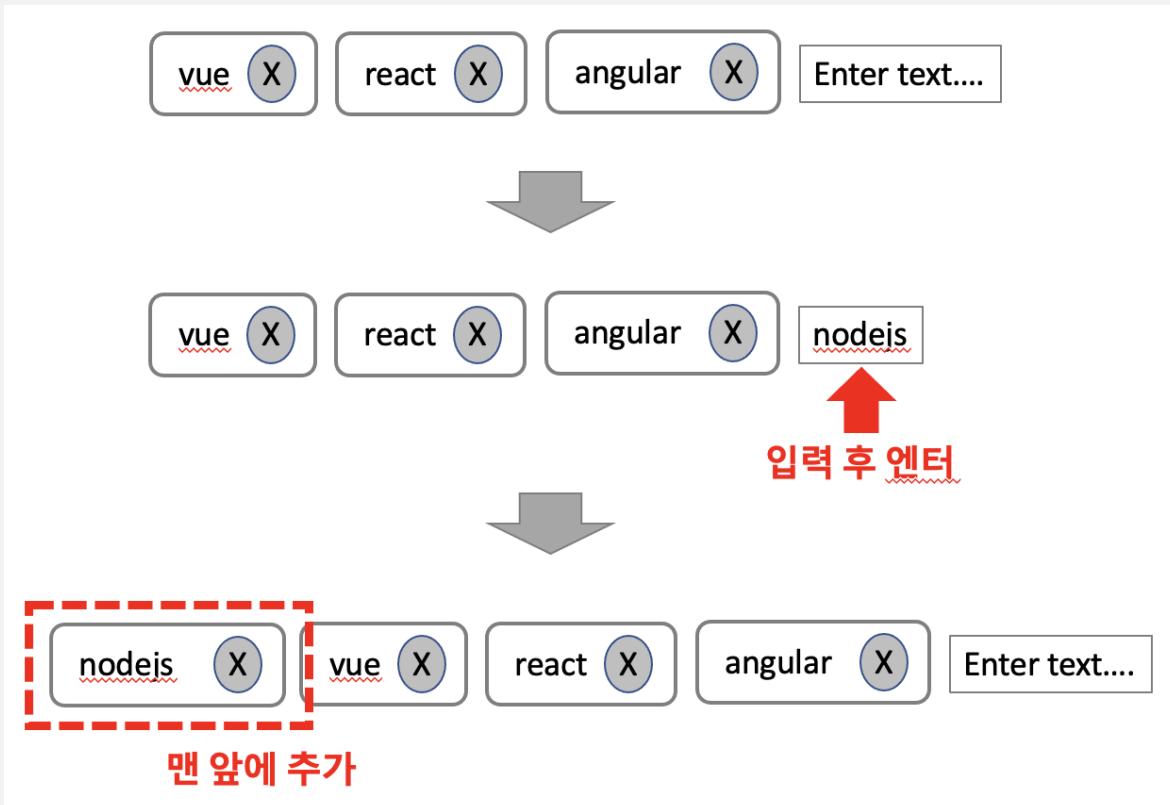
Case10 : Chips UI

케이스 주제

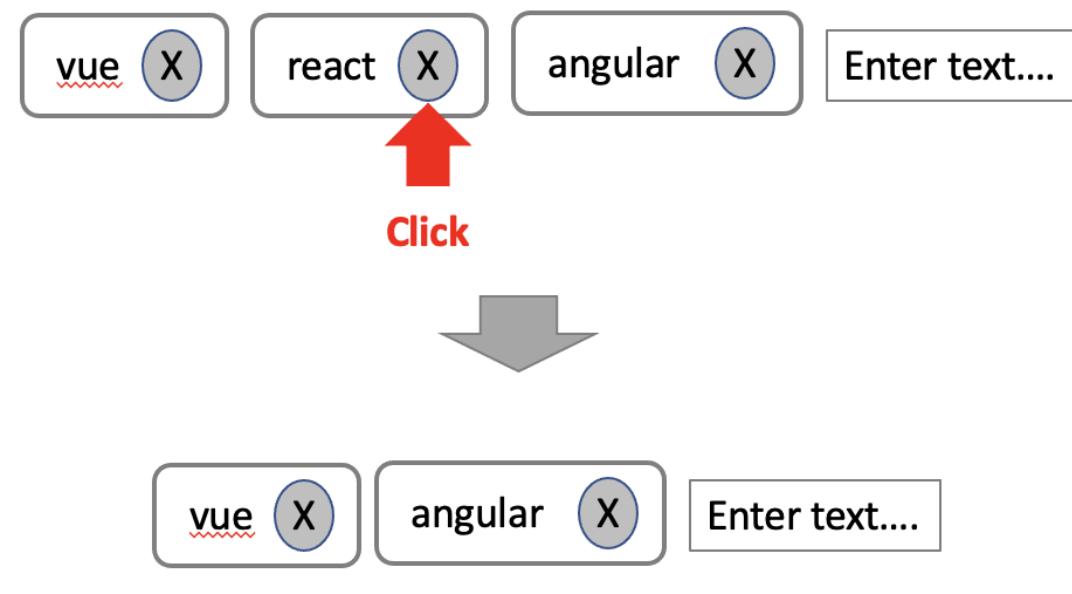
Q. 임의 문자열을 추가 삭제할 수 있는 (SNS 태그 입력) 컴포넌트를 구현하십시오.

기능 요구사항

1. 텍스트 입력 후 엔터를 치면 리스트의 맨 앞에 추가가 된다.



2. 리스트 중에 아이템을 삭제 할 수 있다.



기능 작동 이미지



문제

- q1. 데이터에 따른 문자열 리스트와 입력 폼을 함께 출력하시오.
- q2. 입력된 문자열은 키보드 이벤트 엔터를 통해 리스트의 앞단에 추가 하시오.
- q3. 입력된 문자열을 삭제할 수 있도록 하시오.
- q4. 입력된 문자열에 대한 데이터를 가져올 수 있도록 하시오.

주요 학습 키워드

동적으로 추가되는 element

template 분리
element.insertAdjacentElement 함수 활용

작성해주셔야 하는 question 파일경로

q1

.src/question/index.js

line : 43

q2

.src/question/index.js

line : 54

q3

.src/question/index.js

line : 72

q4

.src/question/index.js

line : 76

실행 방법 및 의존성 모듈 설치

경로 ./

(root directory)

터미널

```
$ npm install  
$ npm run dev
```

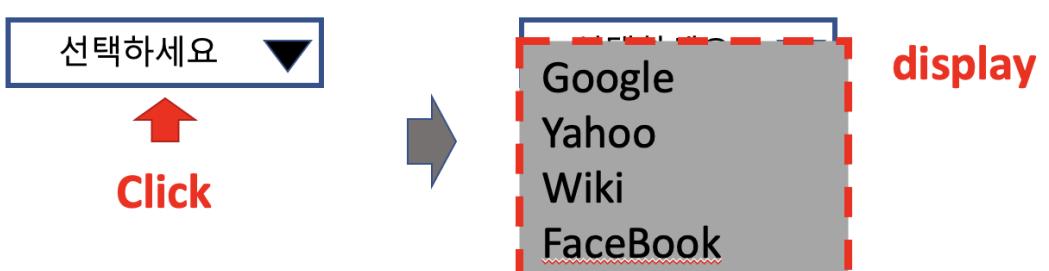
Case11 : Dropdown menu

케이스 주제

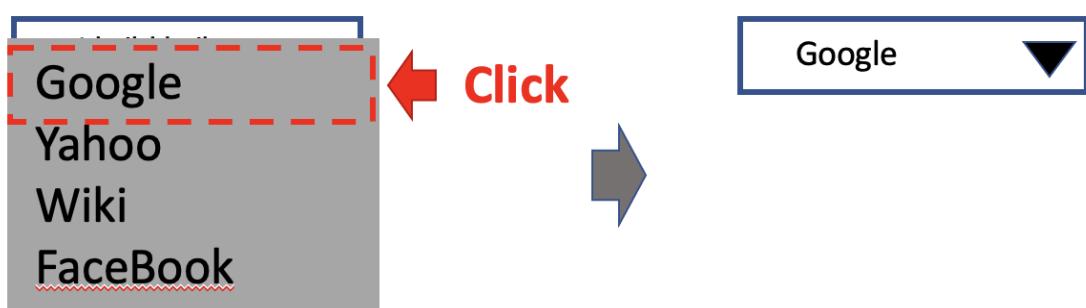
Q. 라벨 클릭 시 선택 가능한 list가 출력되고, dropdown item 선택 시 선택된 데이터의 라벨을 출력해주는 dropdown item을 만드시오.

기능 요구사항

- 라벨 클릭 시 선택 가능한 list가 출력된다.

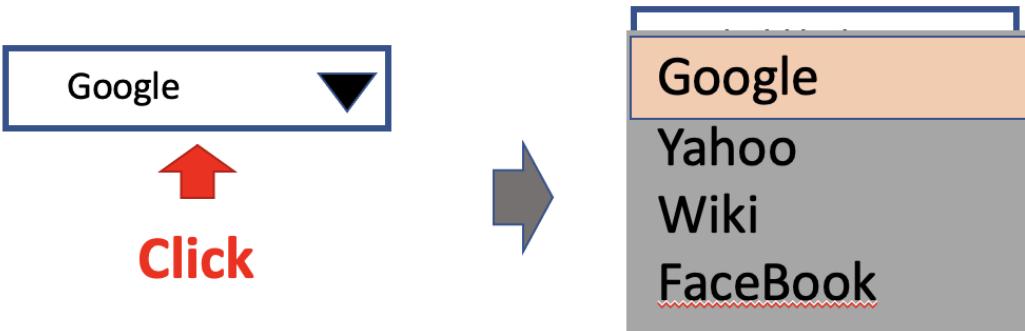


- dropdown item을 선택 시 선택된 데이터를 라벨에 출력한다.



- dropdown item을 선택 시 선택된 데이터를 외부로 전달할 수 있도록 한다.

4. dropdown item을 선택 시 선택된 데이터에 선택 여부를 표시한다.



5. dropdown 외에 영역 클릭 시 아무런 이벤트 없이 리스트를 닫는다.



6. 디자인 템플릿을 변경할 수 있도록 템플릿을 분리한다.

기능 작동 이미지

A screenshot of a dropdown menu interface. The menu is titled "선택하세요." with a dropdown arrow. The menu lists user preferences: "User: Kenneth, favorites: Google", "User: John, favorites: Google", "User: Daniel, favorites: Yahoo", "User: Peter, favorites: FaceBook", and "User: Brian, favorites: Wiki".

문제

- q1. label position에 dropdown list 영역을 출력하시오.
q2. backdrop 영역 클릭 시의 이벤트를 처리하시오.
q3. label 영역 클릭 시의 이벤트를 처리하시오.

q4. 함수(dispatchEvent)를 참조하여 id, label 값을 인자로 넘겨 이벤트를 발생시키시오.

주요 학습 키워드

click event target element 설정에 대한 고민
selector.getBoundingClientRect() 함수
arrow function의 활용
Template literals 을 활용한 html template 관리

작성해주셔야 하는 question 파일경로

q1

`./src/question/dropdown/index.js`

line : 80

q2

`./src/question/dropdown/index.js`

line : 95

q3

`./src/question/dropdown/index.js`

line : 101

q4

`./src/question/dropdown/index.js`

line : 118

실행 방법 및 의존성 모듈 설치

경로 ./

(root directory)

터미널

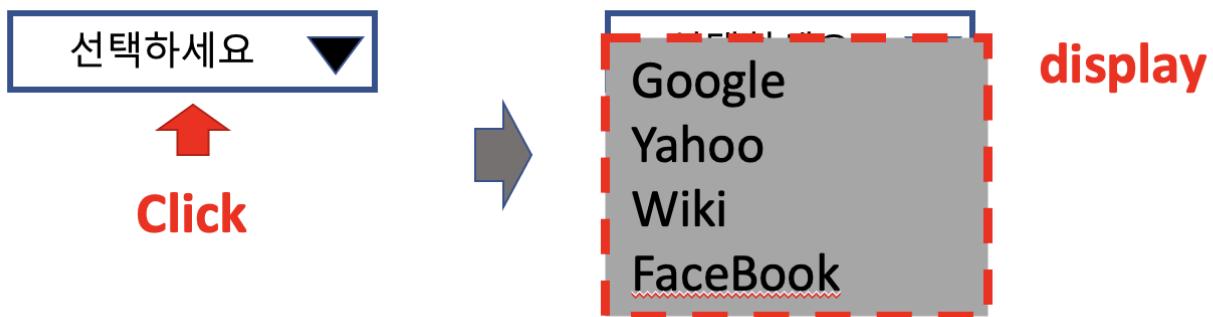
```
$ npm install
$ npm run dev
```



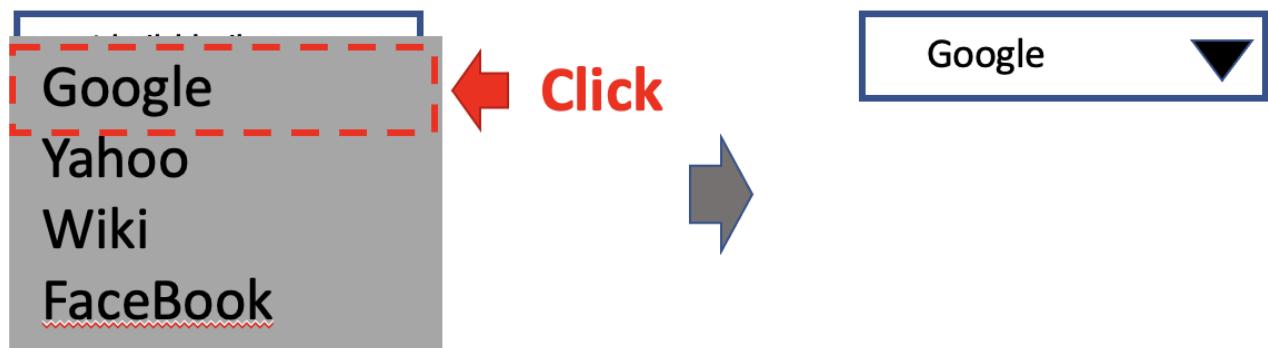
DropDown List Component

요구사항

1. 라벨 클릭 시 선택 가능한 list가 출력된다.

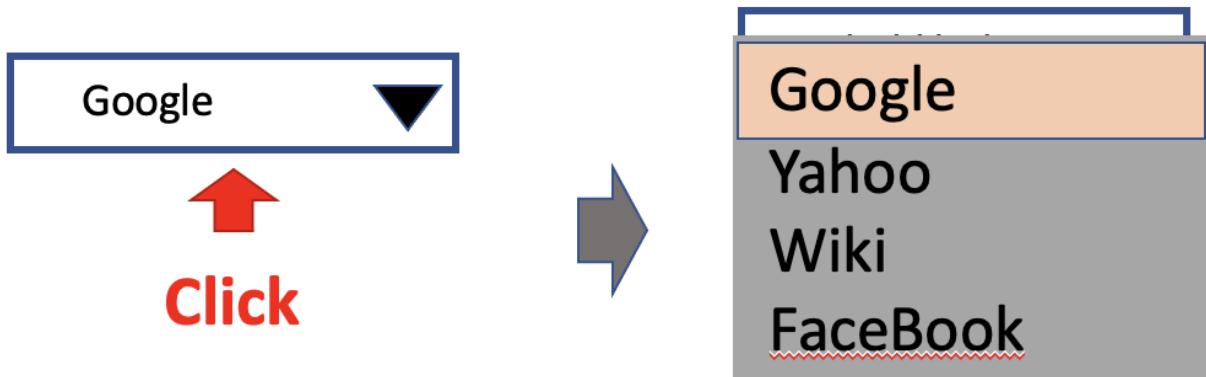


2. dropdown item을 선택 시 선택된 데이터를 라벨에 출력한다.



3. dropdown item을 선택 시 선택된 데이터를 외부로 전달할 수 있도록 한다.

4. dropdown item을 선택 시 선택된 데이터에 선택 여부를 표시한다.



5. dropdown 외에 영역 클릭 시 아무런 이벤트 없이 리스트를 닫는다.



6. 디자인 템플릿을 변경할 수 있도록 템플릿을 분리한다.

Case17 : DropdownMenu - 타 시니어 개발자의 해설지

A)

```
// solution/2.others_1/index.js
...
/**
 * 유저 데이터
 */
const userDataItems = [
  {
    id: "01",
    userName: "Kenneth",
    favorites: "0001"
  },
  ...
];
//*
 * 항목 목록
 */
const selectorDataItems = [
  {
    id: '',
    label: '선택하세요.'
  },
  {
    id: '0001',
    label: 'Google'
  },
  ...
];
//*
 * 아이디를 이용해 빠르게 찾기 위해 맵 구조의 데이터를 생성
 */
const websiteIndex = selectorDataItems.reduce((map, { id, label }) => {
  map[id] = label;
  return map;
}, {});

class DropDownList {
  constructor(selectorEl, selectorOptionsEl, dataItems = [], updateSelectorIndexCallbackFn = () => null) {
    /**
     * 선택器 엘리먼트
     */
    this.selectorEl = selectorEl;
    /**
     * 옵션 엘리먼트
     */
    this.selectorOptionsEl = selectorOptionsEl;
    /**
     * 초기 선택된 아이템
     */
    this.currentSelectedIndex = 0;
    /**
     * 항목 아이템
     */
  }
}
```

```

        */
        this.dataItems = dataItems;
    /**
     * 선택 항목 업데이트시 호출되는 콜백 함수
     */
    this.updateSelectorIndexCallbackFn = updateSelectorIndexCallbackFn;
    this.renderSelector();
    this.renderSelectorOptions();
    this.bindSelectorEvents();
    this.bindSelectorOptionsEvents();
}

/**
 * 현재 선택된 인덱스를 반환
 */
getCurrentSelectedIndex() {
    return this.currentSelectedIndex;
}

/**
 * 현재 선택된 인덱스의 항목 데이터를 반환
 */
getCurrentSelectedItem() {
    const { dataItems, currentSelectedIndex } = this;
    return dataItems[currentSelectedIndex];
}

/**
 * 셀렉터 랜더링
 */
renderSelector() {
    const { currentSelectedIndex, dataItems, selectorEl } = this;
    const selectedItem = dataItems[currentSelectedIndex];
    selectorEl.innerHTML = `

<div class="dropdown-select-label-container">
    <span class="dropdown-select-label">${selectedItem.label}</span>
    <div class="dropdown-select-arrow-container">
        <div class="dropdown-select-arrow"></div>
    </div>
`;
}

/**
 * 셀렉터 항목 랜더링
 */
renderSelectorOptions() {
    /**
     * 선택 항목
     */
    const { currentSelectedIndex, selectorOptionsEl, selectorEl } = this;
    const items = this.dataItems.map((item, index) => {
        const { label } = item;
        return `

<div class="dropdown-item-box ${currentSelectedIndex === index ? 'selected' : ''}" data-index=${index}>
    <span>${label}</span>
</div>
`;
    });
    const { height } = selectorEl.getBoundingClientRect();
    selectorOptionsEl.style.cssText = `display: none; position: absolute; top: ${height}px; `;
    selectorOptionsEl.innerHTML = `

<div class="dropdown-item-list-box">
    ${items.join('')}
</div>
`;
}
}

```

```

    /**
     * 셀렉터 이벤트 바인딩
     */
    bindSelectorEvents() {
        const { selectorEl, selectorOptionsEl } = this;
        selectorEl.addEventListener('click', (e) => {
            e.preventDefault();
            selectorOptionsEl.style.display = 'block';
        });
    }

    /**
     * 셀렉터 항목 이벤트 바인딩
     */
    bindSelectorOptionsEvents() {
        const {
            dataItems,
            selectorOptionsEl,
        } = this;

        selectorOptionsEl.addEventListener('click', ({ target }) => {
            let el = target;
            if (el.tagName === 'SPAN') {
                /**
                 * 기존 마크업을 수정하지 않고 `data-id`가 있는 엘리먼트를 탐색
                 */
                el = el.parentElement;
            }
            const index = Number(el.dataset.index);
            this.currentSelectedIndex = index;
            /**
             * 셀렉터 렌더링
             */
            this.renderSelector();
            /**
             * 콜백으로 데이터 전달
             */
            this.updateSelectorIndexCallbackFn(dataItems[index], index);
        });
    }

    document.addEventListener('click', () => {
        this.selectorOptionsEl.style.display = 'none';
    }, 'click')
}

}

/**
 * DOM 파싱이 완료되면 스크립트 실행
 */
document.addEventListener('DOMContentLoaded', () => {
    const selectorEl = document.querySelector('#dropdown');
    const selectorOptionsEl = document.querySelector('.back-drop');
    const userListEl = document.querySelector('#userlist');
    /**
     * 유저 목록 페이지를 그림
     * @param {*} item
     */
    const renderUserList = (item) => {
        const { id } = item;
        const userItems = userDataItems.filter(({ favorites }) => {
            /**
             * 아이디가 빈 값인경우 모든 아이템 노출,
             * 그렇지 않은경우 해당 아이디만 노출
             */
            return id === '' ? true : id === favorites;
        });
        userListEl.innerHTML = userItems.map((user) => {
            const { favorites, userName } = user;

```

```
const favoriteLabel = websiteIndex[favorites];
return (`
<div>
    <span>User: ${userName}</span>,
    <span>favorites: ${favoriteLabel}</span>
</div>`);
}).join('');
};

const dropDownList = new DropDownList(selectorEl, selectorOptionsEl, selectorDataItems, renderUserList);
const currentSelectedItem = dropDownList.getCurrentSelectedItem();
/**
 * 최초 유저목록 강제로 랜더링
 */
renderUserList(currentSelectedItem);
});
```

해설

[!] 기존 문제 HTML + CSS + 더미데이터 내용을 가지고 다른 방법으로 접근하여 풀이하였습니다.

Dropdown 기능을 셀렉터 기능으로 제한하고, 유저 목록이 랜더링 되는 부분은 분리하여 결합도와 복잡도를 낮추도록 해 보았습니다.

Dropdown 메뉴에서 항목을 선택하면 Callback 함수로 선택된 항목 데이터가 전달됩니다.
선택된 항목 데이터를 이용하여 유저 목록을 랜더링합니다.

위 코드에 사용된 기술 원리는 아래 참고자료를 확인해 주세요.

참고자료

https://developer.mozilla.org/en-US/docs/Glossary/Callback_function
<https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes>
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/Reduce
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/filter
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Array/map
<https://medium.com/@uriyang/data-structure-2-map-8ca759c74445>
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Object

결론

위와같은 구현은 어떤 방법으로 풀든 정답은 없는것 같습니다. 저의 경우는 코드의 복잡도를 낮추는것을 선호하기 때문에 가능한 기능단위로 묶어서 구현하고자 하였습니다.

위 내용의 풀이가 꼭 정답은 아니며 이런 방법의 접근도 있다는것 정도만 참고해 주셔도 좋을것 같습니다.

Case12 : Auto Complete 2

케이스 주제

Q. 검색어를 입력하는 동시에 DB에 저장되어 있는 유관 검색어를 실시간으로 출력하는 기능을 구현하십시오.

기능 요구사항

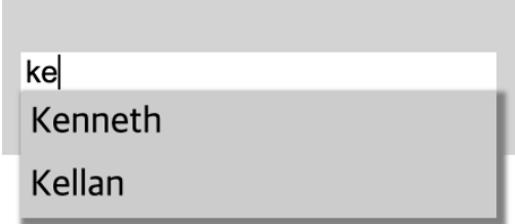
1. 키보드 이벤트를 자연시간(debounce 기능)을 통해 request 횟수를 줄인다.



2. 검색 필드를 벗어나면 출력된 리스트는 사라진다.



기능 작동 이미지



<텍스트 입력 중>

<텍스트 입력 후>

문제

- q1. 검색결과 리스트를 text input 하단에 출력하시오.
- q2. debounce 기능을 구현하시오.
- q3. debounce 기능을 통해 request 호출을 최소화 하시오.
- q4. 검색 필드 외에 다른 곳을 클릭할 때 출력된 리스트를 보이지 않도록 하시오.

주요 학습 키워드

- input event의 활용도
- selector.getBoundingClientRect() 함수
- 지연시간을 적용하여 마지막 이벤트만 발생시키는 기능 (debounce)

작성해주셔야 하는 question 파일경로

q1

./src/question/auto-complete/index.js

line : 90

q2

./src/question/auto-complete/util/index.js

q3

```
./src/question/auto-complete/index.js
```

line : 104

q4

```
./src/question/auto-complete/index.js
```

line : 115

실행 방법 및 의존성 모듈 설치

경로 ./

(root directory)

터미널

```
$ npm install  
$ npm run dev
```

Case12 : AutoComplete 2 - 출제자 해설

q1. 검색결과 리스트를 text input 하단에 출력하시오.

A)

```
displayWordList(selector, data) {
    if (!selector || !data.length) return;

    selector.style.cssText = 'display: ""';
    let render = '';
    for (let i = 0; i < data.length; i++) {
        render += `
            <div class="auto-complete-item-box">
                <span>${data[i]['text']}</span>
            </div>
        `;
    }
    // 리스트를 갱신해야하므로 innerHTML 사용함.
    selector.innerHTML = render;

    // input element position에 list 영역을 출력하기 위해 좌표를 가져오기 위한 getBoundingClientRect 함수 호출
    const inputRect = this.textInputElement.getBoundingClientRect();
    selector.style.cssText = `
        position: absolute;
        width: ${inputRect.width}px;
        top: ${inputRect.top + inputRect.height}px;
        left: ${inputRect.left}px;
    `;

    return document.querySelectorAll('.auto-complete-item-box');
}
```

q2. debounce 기능을 구현하시오.

A)

```
/*
 * title: debounceTime 함수
 * input: callback 실행될 함수, delayTime 지연시간
 * output: setTimeout이 적용된 함수
 * description: 해당 함수의 커플링과 독립성을 위해 delayTime 의 디폴트 값을 500ms로 함.
 */
export const debounce = (callback, delayTime = 500) => {
    let timeout = null;
```

```

        return (...args) => {
            // 실행되지 않은 setTimeout은 clear
            if (timeout) clearTimeout(timeout);

            // setTimeout clear를 위해 저장.
            timeout = setTimeout(() => {
                // 지정된 함수 실행
                callback(...args);
                // 함수 실행 후 setTimeout clear
                clearTimeout(timeout);
            }, delayTime);
        }
    }
}

```

q3. debounce 기능을 통해 request 호출을 최소화 하시오.

q4. 검색 필드 외에 다른 곳을 클릭할 때 출력된 리스트를 보이지 않도록 하시오.

```

/*
 * title: event binding method
 * description: 모든 이벤트를 처리한다.
 */
eventBinding() {
    let isListBoxFocus = false;
    const requestAdapter = new RequestMockAdapter();
    const dispatchEvent = debounce((targetText) => {
        // 공백 제거 후 단어가 있다면 호출한다.
        if (targetText.replace(/\ /g, '')) {
            // 해당 text를 parameter로 api 호출
            requestAdapter.get(this.requestUrl, targetText)
                .then((result) => {
                    this.displayWordList(
                        this.searchListElement,
                        result
                    ).forEach((element, index) => {
                        element.addEventListener('click', (event) => {
                            this.textInputElement.value = result[index].text;
                            this.hiddenElement(this.searchListElement);
                        });
                    });
                });
        } else {
            this.hiddenElement(this.searchListElement);
        }
    }, this.delayTime);

    // input에서 focusout 이벤트 발생 시 출력된 리스트를 숨긴다.
    this.textInputElement.addEventListener('focusout', () => {
        if (isListBoxFocus) return;
        this.hiddenElement(this.searchListElement);
    });
}

// 필드에 마우스 포인터가 오면 리스트를 불러온다.

```

```
this.textInputElement.addEventListener('focusin', (event) => {
    dispatchEvent(event.target.value);
});

// 키보드 이벤트
this.textInputElement.addEventListener('keyup', (event) => {
    dispatchEvent(event.target.value);
});

// focusout 이 되면 리스트가 사라지므로 리스트에 포인터가 머물러 있을 시에는 focusout 이벤트가 일어나도 리스트를 유지하기 위함
this.searchListElement.addEventListener('mouseover', () => {
    isListBoxFocus = true;
});

// 포인터가 리스트에서 떠나면 focusout 이벤트 시 리스트 시리짐
this.searchListElement.addEventListener('mouseout', () => {
    isListBoxFocus = false;
});

}

/*
 * title: element hidden method
 * description: 특정 element에 대해 display: none 옵션을 적용한다.
 */
hiddenElement(selector) {
    selector.style.cssText = 'display: none;';
}
```

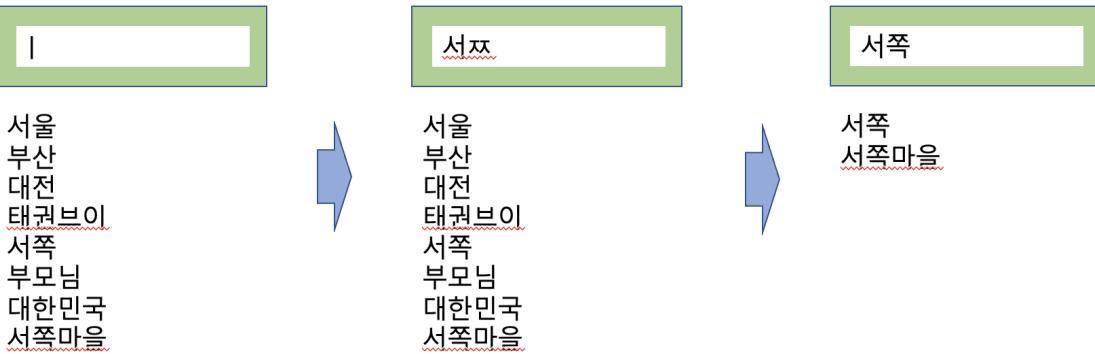
Case13 : Instant Search

케이스 주제

Q. 검색어를 입력하는 동시에 엔터키, 검색 버튼 누를 필요 없이 결과값을 즉시 반영할 수 있는 순간 검색 기능을 구현하십시오.

기능 요구사항

1. 키보드 이벤트를 지연시간(debounce 기능)을 통해 request 횟수를 줄인다.



2. 디자인 템플릿을 변경할 수 있도록 템플릿을 분리한다.

기능 작동 이미지

please enter keyword

서울
부산
대전
태권브이
서쪽
부모님
대한민국
서쪽마을

<텍스트 입력 전>

서

서울
서쪽
서쪽마을

<텍스트 입력 후>

문제

- q1. configuration을 참고하여 input element를 생성하시오. (기능과 디자인을 분리하기 위한 방법, src->question->instant-search->index.js)
- q2. debounce 기능을 구현하시오. (src->question->instant-search->util->index.js)
- q3. debounce 기능을 통해 가져온 데이터를 외부로 전달한다. (src->question->instant-search->index.js)
- q4. Promise를 사용하여 검색 키워드에 맞는 데이터를 가져와 리스트를 출력하시오. (src->question->instant-search->mock->word->index.js / src->index.js)

주요 학습 키워드

지연시간을 적용하여 마지막 이벤트만 발생시키는 기능 (debounce)
request api를 대신하기 위한 Promise 활용 (test를 위한 mock data 활용)
디자인 부분과 기능을 분리하기 위한 방법

작성해주셔야 하는 question 파일경로

q1

./src/question/instant-search/index.js

line : 43

q2

```
./src/question/instant-search/util/index.js
```

q3

```
./src/question/instant-search/index.js
```

line : 53

q4

```
./src/question/instant-search/mock/word/index.js
```

line : 30

```
./src/index.js
```

line : 35

실행 방법 및 의존성 모듈 설치

경로 ./

(root directory)

터미널

```
$ npm install  
$ npm run dev
```

Case13 : Instant Search - 출제자 해설

**q1. configuration을 참고하여 input element를 생성하시오.
(기능과 디자인을 분리하기 위한 방법)**

A)

```
/*
 * title: instant search display method
 * input: display 되는 element, configuration placeholder, css 정보
 * output: text input element
 * description: 최초 생성할 때 input element를 생성한다. 템플릿을 관리한다.
 */
initialize(selector, configuration) {
    const textinput = document.createElement('input');
    textinput.setAttribute('type', 'text');
    textinput.setAttribute('placeholder', configuration.placeholder ?? 'Please enter');
    textinput.classList.add(configuration.css);
    selector.appendChild(textinput);
    // 생성된 input element를 리턴해준다.
    return textinput;
}
```

q2. debounce 기능을 구현하시오.

A)

```
/*
 * title: debounceTime 함수
 * input: callback 실행될 함수, delayTime 지연시간
 * output: setTimeout이 적용된 함수
 * description: 해당 함수의 커플링과 독립성을 위해 delayTime 의 디폴트 값을 500ms로 함.
 */
export const debounce = (callback, delayTime = 500) => {
    let timeout = null;
```

```

return (...args) => {
    // 실행되지 않은 setTimeout은 clear
    if (timeout) clearTimeout(timeout);

    // setTimeout clear를 위해 저장.
    timeout = setTimeout(() => {
        // 지정된 함수 실행
        callback(...args);
        // 함수 실행 후 setTimeout clear
        clearTimeout(timeout);
    }, delayTime);
}
}

```

q3. debounce 기능을 통해 가져온 데이터를 외부로 전달한다.

```

/*
 * title: event binding method
 * description: 모든 이벤트를 처리한다.
 */
eventBinding() {
    // debounce util case
    const dispatchEvent = debounce((targetText) => {
        // 입력된 단어를 callback 함수를 통해 전달.
        this.callback(targetText);
    }, this.delayTime);

    this.textInputElement.addEventListener('keyup', (event) => {
        dispatchEvent(event.target.value);
    });
}

// rxjs debounceTime operator case
// fromEvent 는 지정된 이벤트 대상에서 오는 특정 유형의 이벤트를 내보내는 Observable 만들어 반환합니다.
// pipe라는 함수를 통해 이벤트 흐름을 만들 수 있는데, 여기에 debounceTime operator를 넣습니다.
const eventSource = fromEvent(this.textInputElement, 'keyup')
    .pipe(
        debounceTime(this.delayTime)
    );

// 이벤트 소스에서 구독을 하게되면 이벤트가 발생할 때마다 값을 전달 받게 됩니다.
this.subscription = eventSource.subscribe((event) => {
    dispatchEvent(event.target.value);
})
}

```

q4. Promise를 사용하여 검색 키워드에 맞는 데이터를 가져와 리 스트를 출력하시오.

```
const getData = (targetWord = '') => {
    // q4. Promise를 사용하여 검색 키워드에 맞는 데이터를 가져와 리스트를 출력하시오.
    // TODO: Write JS code here!
    // 정답 본 파일 60행에 주석으로 처리되어 있습니다.

    retrieveWordList(targetWord)
        .then((result) => {
            displayWordList(document.querySelector('#wordlist'), result);
        });
}
```

Case13 : Instant Search - 타 시니어 개발자 해설

[!] 기존 문제 내용을 조금 다른 관점에서 풀어보았습니다(코드구조 변경 재작성). 사용된 라이브러리(RxJS), 학습 키워드 및 개발환경은 동일합니다.

풀이

```
// src/index_other_1.js
import './style.css';
import { InstantSearch } from './solution/other_1/instant-search';
/**
 * 더미 데이터
 */
const worldItems = [
  {
    word: '서울'
  },
  ...
];
/***
 * 단어 목록을 노출할 엘리먼트
 */
const wordListEl = document.querySelector('#wordlist');

/**
 * 데이터 입력으로 템플릿(HTML) 문자열을 생성하는 함수
 */
const makeWordListHtml = (data) => {
  let itemList = '';
  for (let i = 0; i < data.length; i++) {
    itemList += `
      <div>
        <span>${data[i].word}</span>
      </div>`;
  }
  return itemList;
}

/***
 * 내부 구현이 서버 데이터를 불러오는 로직으로 변경될 경우 응답을 동일하게 구현할 수 있도록 비동기 응답 구현
 */
const selectDataItems = (query) => {
  const selectedDataItems = query ? worldItems.filter(({ word }) => word.startsWith(query)) : worldItems;
  return Promise.resolve(selectedDataItems);
}

const renderWordList = (query = '') => {
  /**
   * async/await 사용 가능한 환경에서는 더 간단하게 코드를 구현할 수 있습니다.
   */
  return new Promise((resolve) => {
    selectDataItems(query).then((wordItems) => {
```

```

        const wordListHtml = makeWordListHtml(wordItems);
        wordListEl.innerHTML = wordListHtml;
        resolve(wordListEl);
    });
});
};

const bootstrap = () => {
    /**
     * 단어 목록이 노출될 엘리먼트
     */
    const instantSearch = new InstantSearch({
        wrapperSelector: '#instant-search',
        classNames: 'instant-search-input',
        placeholder: 'please enter keyword'
    });

    instantSearch.subscribe(({ target }) => {
        const { value: query } = target
        renderWordList(query);
    });
}

// 최초에 전체 리스트를 출력.
renderWordList();
};

// src/solution/other_1/instant-search/index.js
...
export class InstantSearch {
    constructor({
        wrapperSelector,
        placeholder,
        classNames,
        debounceDelay = 500
    }) {
        /**
         * 멤버 변수로 등록
         */
        this._debounceDelay = debounceDelay;
        /**
         * <해설>
         * `initialize` 함수를 분리하는것도 좋지만 `constructor` 생성자 자체가 그 역할을 하기 때문에 따로 분리하지 않았습니다.
         */
        const eventStream = new Subject();
        const inputEl = this._makeSearchInputEl(classNames, placeholder);
        document.querySelector(wrapperSelector).append(inputEl);
        /**
         * 입력된 키 이벤트를 `eventStream` 스트림으로 전송
         */
        inputEl.addEventListener('keyup', eventStream.next.bind(eventStream));
        /**
         * `destroy` 메서드에서 참조 가능하도록 멤버변수로 등록
         */
        this._eventStream = eventStream;
        this._inputEl = inputEl;
    }
    /**
     * <해설>
     * 함수의 이름만 보고 함수가 하는 역할을 알 수 있도록 하는것이 좋습니다.
     * [>] 하나의 함수에서 너무 많은일을 하지 않도록 합니다.
     * [>] 함수의 입출력 값이 명확하다면 나중에 테스트 코드를 쉽게 만드실 수 있습니다.
     */
    _makeSearchInputEl(classNames, placeholder = 'Please enter') {

```

```

        const inputEl = document.createElement('input');
        inputEl.setAttribute('type', 'text');
        inputEl.setAttribute('placeholder', placeholder);
        inputEl.classList.add(classNames);
        return inputEl;
    }
    /**
     * `subscribe` 메서드를 이용하여 이벤트 쿠독 콜백 등록
     */
    subscribe(callbackFn) {
        const { _eventStream: eventStream, _debounceDelay: debounceDelay } = this;
        return eventStream.pipe(debounceTime(debounceDelay)).subscribe(callbackFn);
    }
    /**
     * 검색 엘리먼트를 제거하고, 이벤트 스트림을 종료
     */
    destroy() {
        this._eventStream.complete();
        this._inputEl.remove();
    }
}
...

```

해설

용어: 클래스 내부에 사용되는 함수는 '메소드', 클래스 내부에서 선언되어 여러 메소드에서 참조되어 사용되는 변수들을 멤버변수라고 합니다.

기존 문제 코드에서 변수 및 함수 네이밍을 다르게 작성해 보았습니다.

[!] 함수의 이름이나 구현 범위는 꼭 정답이 있는 것은 아닙니다.

함수의 경우 이름만 보고 함수가 하는 역할을 알 수 있도록 개선해 보았습니다.

하나의 함수에서 너무 많은 일을 하지 않도록 하였습니다.

함수의 입력력 값이 명확하다면 테스트가 쉬워집니다.

그렇다고 함수가 너무 작은 단위로 패턴화 되지 않게 주의하세요, 함수가 실행될 때마다 발생하는 스코프를 생성하는 시간이 쌓이면 성능에 영향을 줄 수 있습니다.

내부적으로 사용되는 private 멤버변수, 메소드의 이름은 _(언더바)로 네이밍 하였습니다.

일반적으로 오픈소스 라이브러리에서 자주 보이는 패턴이기도 한데 언더바로 시작하는 필드는 찾아보시면, 대부분 내부 처리를 위해 만든 필드이거나 변경/삭제 될 수 있기 때문에 꼭 확인해 보시는 게 좋습니다.

public과 private 필드 선언은 자바스크립트 표준화 위원회에 실험적 기능 (stage 3) TC39로 제안되어 있습니다. 현재 이를 지원하는 브라우저는 제한적인 상태입니다만, Babel과 같은 build 시스템을 사용한다면 이 기능을 사용해볼 수 있습니다.

InstantSearch

클래스로 생성된 인스턴스의 `subscribe` 기능을 이용하여 키 입력 이벤트를 구독할 수 있습니다.

`InstantSearch` 클래스는 내부적으로 이벤트 스트림(Rx.Subject)를 가지고 있고, 구독하는 모든 대상에게 이벤트 스트림을 전달합니다.

참고자료

<http://reactivex.io/documentation/ko/subject.html>
<https://rxjs-dev.firebaseio.com/api/operators/debounceTime>
<https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Classes>

결론

RxJS를 잘 모른다면 조금 어려울 수 있는 문제 같습니다. 다양한 라이브러리를 사용해 보는 것도 좋은 것 같습니다. 대부분의 라이브러리나 프레임워크에는 만들게 된 동기(Motivation)나, 지향점, 철학 또는 해결하고자 하는 목적이 있는데 라이브러리를 사용하기 전에 한번 찾아보고 공감이 되신다면 금방 익숙해질 것 같습니다.

Case14 : Stop watch

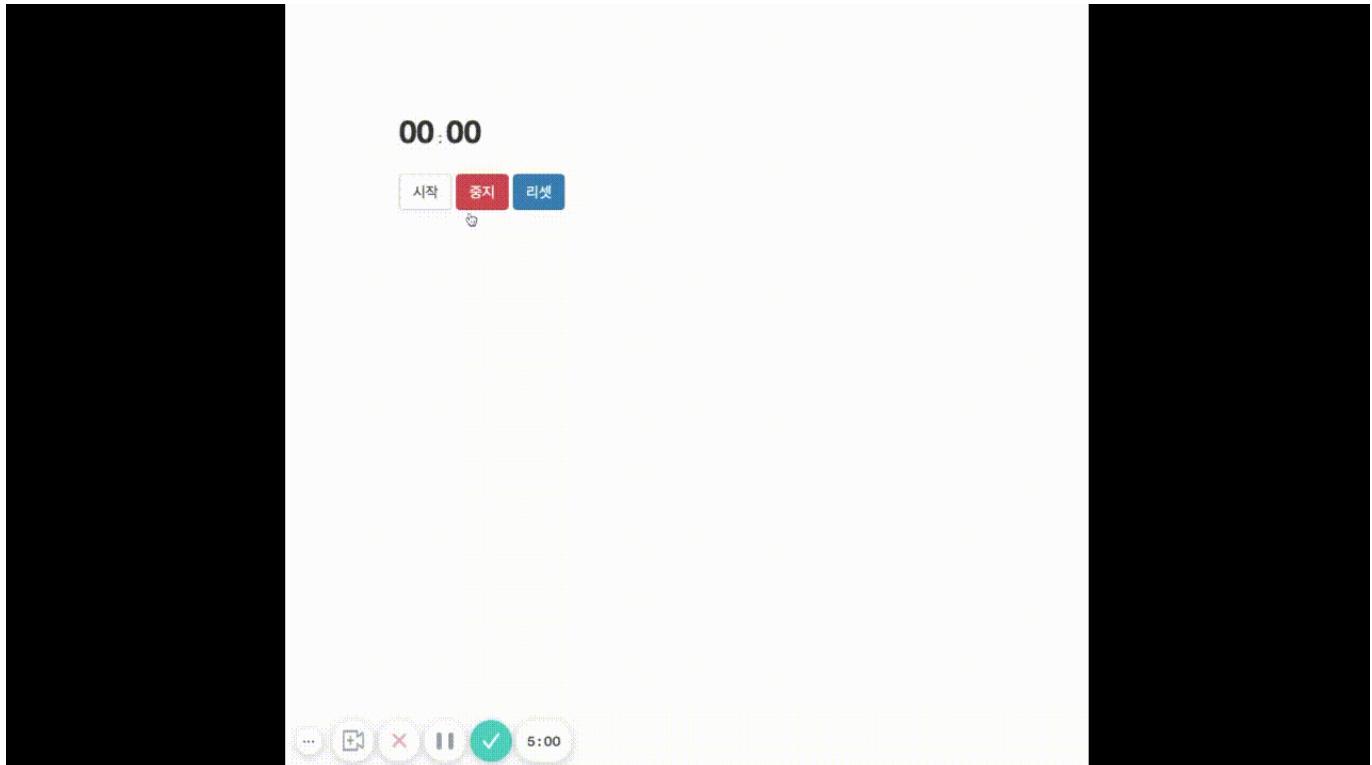
케이스 주제

Q. 타이머(스톱워치) 구현

기능 요구사항

1. 타이머 뷰를 구현하고 시작 / 중지 / 리셋에 따라 타이머가 동작하도록 한다.

기능 작동 이미지



문제

q1. Javascript - 주어진 템플릿(index.html)
)의 요소에 페이지에 맞는 요소를 표현하는 문제

1. isPending이 false 일때만 setTimeout을 재귀 호출하는 것을 만드세요.
2. reset 버튼을 누를 시 화면과 시간을 00:00:00으로 초기화되게 만드세요.
3. 타이머를 시작하는 부분을 만드세요.

q2. Jquery - Javascript로 구현한 기능을 동일하게 Jquery로도 구현해보는 문제
q3. React - 주어진 코드를 이해하고, 상태를 조작하는 이벤트핸들러와 라이프사이클 흐름을 활용하는 문제

1. setInterval 또는 setTimeout으로 반복시키는 부분을 완성해주세요.
2. 60초가 될때를 감지하는 부분을 완성한다.

주요 학습 키워드

데이터 모델 -> 뷰 방향의 단방향 데이터 바인딩 구현

작성해주셔야 하는 question 파일경로

q1

`./question/q1_js/index.js`

q2

`./question/q2_jquery/index.js`

q3

`./question/q3_react/src/App.js`

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js`

index.html 열기

q2

경로 `./question/q2_jquery`

index.html 열기

q3

경로 `./question/q3_react`

터미널

```
$ npm install  
$ npm start
```

Case14 : Stopwatch - 출제자 해설

q1. vanilla.js 로 코드를 작성해보세요.

중점포인트

setInterval, setTimeout 의 이해도를 체크한다.
setTimeout 을 사용할시

재귀호출로 처리한다
첫번째 함수를 전역으로 함에 유의한다.

A)

Timer 객체를 만든다.

Timer 객체를 만드는 이유는 시작 버튼 클릭시 아래와 같이 한줄로 가독성 좋게 이벤트를 만들기 위함이다.

```
Timer.startTimer();
```

기본 변수는 아래와 같다.

```
isPending : true, // true 면 멈춤 false 면 진행
currentSec : 0, // 현재 초
currentMin : 0, // 현재 분
```

시작 버튼을 누를시

```
startTimer : function(){
    if(!this.isPending){
        this.currentSec+=1;
        if(this.currentSec==60){
            this.currentSec=0;
            this.currentMin+=1;
        }
        document.querySelector('#min').innerHTML = ('0' + this.currentMin).slice(-2);
        document.querySelector('#sec').innerHTML = ('0' + this.currentSec).slice(-2);
        setTimeout("Timer.startTimer()", 1000);
    }
},
```

재귀 호출을 진행하면서, pending이 false 면 계속 진행 true 인 경우, 재귀 호출을 멈춘다.

```
("0" + this.currentMin).slice(-2);
```

1초인경우 01 을 붙여주기 위함인데 12초면 012 이다. 이를 뒤에서부터 두자리 끊어서 12로 나타낸다.

```
pauseTimer : function(){
    this.isPending = true;
},
resetTimer : function(){
    this.isPending = true;
    this.currentSec = 0;
    this.currentMin = 0;
```

```
document.querySelector('#min').innerHTML = "00";  
  
document.querySelector('#sec').innerHTML = "00";  
  
}  
  
}
```

pauseTimer(멈춤) => isPending 을 false resetTimer(리셋) => isPending 을 false 하고, 시계를 00:00 으로 초기화

```
document.addEventListener("DOMContentLoaded", function () {  
    document.querySelector("#start").addEventListener("click", function () {  
        if (Timer.isPending) {  
            Timer.isPending = false;  
  
            Timer.startTimer();  
        }  
    });  
  
    document.querySelector("#pause").addEventListener("click", function () {  
        Timer.pauseTimer();  
    });  
  
    document.querySelector("#reset").addEventListener("click", function () {  
        Timer.resetTimer();  
    });  
});
```

DOMContentLoaded 문서가 로드되고 각 버튼을 클릭시 타이머의 객체의 함수를 콜한다.

시작을 누를시 if 로 체크하는 것은 두번 누르더라도 반복해서 증가함을 방지한다. 궁금하면 if 를 없애고 돌려본다.

```
if (Timer.isPending) {  
    Timer.isPending = false;  
  
    Timer.startTimer();  
}
```

(추가로 생각해볼 문제)

setInterval 으로 처리 한다면 어떻게 할수 있을까 고민해본다.

q2. jQuery 로 코드를 작성해보세요.

중점포인트

vanilia.js 와의 차이점 jQuery로 문서 로드 시점, jQuery 이벤트 할당으로 처리 했는지

A)

html 안에 텍스트 넣기

```
$( "#min" ).html( "00" );
```

```
$(document).ready(function () {
  $("#start").click(function () {
    if (Timer.isPending) {
      Timer.isPending = false;

      Timer.startTimer();
    }
  });

  $("#pause").click(function () {
    Timer.pauseTimer();
  });

  $("#reset").click(function () {
    Timer.resetTimer();
  });
});
```

문서를 로드후에 각 click 이벤트를 할당한다.

q3. React 로 코드를 작성해보세요.

중점포인트

state의 값이 변화해서 랜더링이 되더라도 변수의 변화값을 유지하고 있는지
useEffect 의 두번째 인자를 잘 이해하고 있는지
useState 의 콜백 이해
setInterval 을 React로 제어하는 방법

A)

<순서> 시작, 중지를 누르면 pending 을 변화 시키고 -> useEffect 로 감지해서 타이머 진행 -> sec가 1씩 증가 -> useEffect 로 감지해서 화면 변화

```
const intervalId = useRef(null)
```

컴포넌트가 랜더링 되더라도 변화하는 값을 저장하기 위해서 useRef 를 처리한다.
useRef 는 현재 상태 값을 intervalId.current 처럼 current 값으로 처리한다.
해당 useRef 가 궁금하면 console.log(intervalId) 로 찍어본다.

아래와 같이 intervalId.current 로 setInterval 의 값을 저장하고, clearInterval 로 해제한다.

```
useEffect(() => {
  if (!pending) {
    intervalId.current = setInterval(() => setSec((sec) => sec + 1), 1000);
  } else {
    clearInterval(intervalId.current);
  }
}, [pending]);
```

타이머의 제어는 pending의 상태로 분기해서 처리한다.
pending : false => 타이머 진행
pending : true => 타이머 멈춤
pending 의 상태가 변화 하면 useEffect 로 감지하는데 두번째 인자로 체크한다.
하나의 상태가 변화하는 것을 감지해서 위와 같이 [pending]으로 처리한다.

아래와 같이 여러개의 상태를 useEffect 로 감지해서 처리하지 않는다.

```
useEffect(() => {
  setSec(0);
  setMin(0);
});
```

```
useEffect(() => {
  if (sec === 60) {
    setSec(0);
    setMin((min) => min + 1);
  }
}, [sec]);
```

sec의 변화를 감지하는데, 60초가 되면 분1증가 초는 0으로 초기화

(추가로 생각해볼 문제)

setTimeout으로 처리 한다면 어떻게 할수 있을까 고민해본다.

Case14 : Stopwatch - 대기업 S사 프론트엔드 개발자님의 답안

q1. vanilla.js 로 코드를 작성해보세요.

A)

```
// solution/2.others_1/q1_js/index.js
function Timer() {
    /**
     * 타이머 진행 시간(초)
     */
    this.seconds = 0;
    /**
     * Pause 여부
     */
    this.isPause = false;
    /**
     * `intervalId` 0이면
     */
    this.intervalId;
    /**
     * 타이머가 업데이트 됐을때 콜백
     */
    this.updateCallbackFn;
}

Timer.prototype.start = function () {
    const { intervalId } = this;
    if (intervalId) {
        clearInterval(intervalId);
    }
    /**
     * @see https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval
     */
    this.intervalId = setInterval(() => {
        const { seconds } = this;
        this.setTimer(seconds + 1);
    }, 1000);
};

Timer.prototype.pause = function () {
    if (this.intervalId) {
        clearInterval(this.intervalId);
    }
};
```

```

Timer.prototype.reset = function () {
    this.pause();
    this.setTimer(0);
};

Timer.prototype.setUpdateCallback = function (callbackFn) {
    this.updateCallbackFn = callbackFn;
}

Timer.prototype.setTimer = function (seconds) {
    const { updateCallbackFn } = this;
    this.seconds = seconds;
    if (typeof updateCallbackFn === 'function') {
        updateCallbackFn(this.seconds);
    }
}

document.addEventListener("DOMContentLoaded", function () {
    const minEl = document.querySelector('#min');
    const secEl = document.querySelector('#sec');

    const seconds = new Timer();
    seconds.setUpdateCallback((seconds) => {
        let min = Math.floor(seconds / 60);
        min = 10 > min ? '0' + min : min;
        let sec = seconds % 60;
        sec = 10 > sec ? '0' + sec : sec;
        /**
         * Elapsed time update
         */
        minEl.textContent = min;
        secEl.textContent = sec;
    })
}

document.querySelector('#start').addEventListener('click', function () {
    seconds.start();
});

document.querySelector('#pause').addEventListener('click', function () {
    seconds.pause();
});

document.querySelector('#reset').addEventListener('click', function () {
    seconds.reset();
});
});

```

해설

기존 Timer를 인스턴스 생성 가능한 함수로 구현하여 개발
여러개의 인스턴스를 생성하여 동시에 여러 타이머를 만들수 있음
타이머 메소드 네이밍 변경, 이미 객체(인스턴스)는 타이머를 가르키기 때문에 메소드에 `Timer`가 포함되지 않도록 변경
데이터와 랜더링 처리 로직을 완전히 분리하여 처리

<https://developer.mozilla.org/en-US/docs/Web/API/WindowOrWorkerGlobalScope/setInterval>
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype

q2. jQuery로 코드를 작성해보세요.

A)

```
// solution/2.others_1/q2_jquery/index.js
...

$(() => {
    const minEl = $('#min');
    const secEl = $('#sec');

    const seconds = new Timer();
    seconds.setUpdateCallback((seconds) => {
        let min = Math.floor(seconds / 60);
        min = 10 > min ? '0' + min : min;
        let sec = seconds % 60;
        sec = 10 > sec ? '0' + sec : sec;
        /**
         * 이곳에 업데이트
         */
        minEl.text(min);
        secEl.text(sec);
    })
}

$(document.body)
/**
 * Jquery Event Delegation을 이용하여 `body`에서 이벤트를 위임받아 모든 버튼이벤트를 처리
 */
.on('click', '#start, #pause, #reset', ({ target }) => {
    const id = target.getAttribute('id');
    switch (id) {
        case 'start':
            seconds.start();
            break;
        case 'pause':
            seconds.pause();
            break;
        case 'reset':
            seconds.reset();
            break;
    }
});
})
```

1번 문제와 내용은 거의 동일하지만 이벤트 바인딩, DOM을 제어하기 위해 Jquery 라이브러리를 사용하였습니다.

참고자료

<https://api.jquery.com/ready/>
<https://api.jquery.com/on/>
<https://api.jquery.com/text/>

q3. React 로 코드를 작성해보세요.

A)

```
// solution/2.others_1/q3_react/src/App.js
export default function App() {
    /**
     * 타이머 시간(초)
     */
    const [seconds, setSeconds] = useState(0)
    /**
     * `intervalId`는 랜더링과 관련이 없기 때문에 `useRef`를 이용하여 처리합니다.
     * [>] 값 변화시 화면에 즉각적으로 업데이트 되어야하는 데이터라면 `useState`를 이용해 주세요
     */
    const intervalIdRef = useRef(null);

    /**
     * [!] 리액트에서 이벤트 핸들러는 `handle~`으로 네이밍하는 관례(Convention)가 있습니다.
     * @see https://blog.sonim1.com/220
     */
    const handleClickStartTimerButton = useCallback(() => {
        intervalIdRef.current = setInterval(() => {
            setSeconds((seconds) => {
                return seconds += 1;
            })
        }, 1000);
    }, []);

    const handleClickPauseTimerButton = useCallback(() => {
        const intervalId = intervalIdRef.current;
        if (intervalId) {
            clearInterval(intervalId)
        }
    }, []);

    const handleClickResetTimerButton = useCallback(() => {
        const intervalId = intervalIdRef.current;
        if (intervalId) {
            clearInterval(intervalId)
        }
        setSeconds(0);
    }, []);
}
```

```
let min = Math.floor(seconds / 60);
min = 10 > min ? '0' + min : min;
let sec = seconds % 60;
sec = 10 > sec ? '0' + sec : sec;

return (
  <div className="container">
    <Clock>
      <span>{min}</span> : <span>{sec}</span>
    </Clock>
    <div>
      <Button text={'시작'} onClick={handleClickStartTimerButton} className="btn-default" />
      <Button text={'중지'} onClick={handleClickPauseTimerButton} className="btn-danger" />
      <Button text={'리셋'} onClick={handleClickResetTimerButton} className="btn-primary" />
    </div>
  </div>
);
}
```

해설

화면 랜더링과 관련없는 `intervalId` 값은 `useRef`으로 처리하였습니다.

타이머에 의해 화면이 매번 랜더링 될때마다 `handle**` 함수가 재정의되는 불필요한 연산을 줄이기 위해 `useCallback`을 이용하여 캐싱하였습니다.

`handle**` 함수내에서 상태를 업데이트 하기 위해서는 `seconds` 상태값을 직접 참조하는 대신 `setSeconds` 함수의 인자로 콜백을 실행하여 처리하였습니다(자세한 내용은 참고자료를 확인해 주세요).

참고자료

<https://blog.sonim1.com/220>

<https://ko.reactjs.org/docs/hooks-reference.html#usestate>

결론

`setInterval`
그리고 `clearInterval`
를 잘 이해하고 있다면 아주 쉽게 구현이 가능한 문제입니다!

Case15 : Modal window

케이스 주제

Q. 모달 팝업 구현

기능 요구사항

1. 팝업을 엽니다를 누르면 팝업이 뜬다.
2. x 버튼을 누르면 팝업이 닫힘
3. 팝업 이외 부분을 누르면 닫힘
4. 팝업 안을 눌렀을 때 닫히면 안됨

기능 작동 이미지

팝업을 엽니다.



문제

q1. JavaScript - 주어진 템플릿(q_index.html)
)의 요소에 페이지에 맞는 요소를 표현하는 문제 q2. jQuery

1. JavaScript로 구현한 기능을 동일하게 jQuery로도 구현해보는 문제
2. <https://jquerymodal.com/> 문서를 참조해서
3. 자바스크립트 소스 없이 완성하세요

q3. React - 주어진 코드를 이해하고, 상태를 조작하는 툴을 이용해서 완성

풀이 참고

q_index.html안에 body-blackout이 모달 바탕화면이고, Modal 보다 z-index 가 낮다. body-blackout 위로 Modal 이 뛴다.

팝업 띄우기 클릭시 is--visible 클래스를 활성화

주요 학습 키워드

CSS의 z-index를 사용하여 팝업 / 배경 순서 위치 지정하기

CSS의 pointer-events를 사용하여 popup이 없을 때, 이벤트가 발생하지 않도록 설정하기

classList를 사용하여 버튼에 click event 발생 시, 팝업 보여주고 닫아주기

작성해주셔야 하는 question 파일경로

q1

`./question/q1_js/q1_index.js`

q2

`./question/q2_jquery/q2_index.html`

<https://jquerymodal.com/> 문서를 참조해서 자바스크립트 소스 없이 CDN을 html파일에 추가하여 완성하세요

q3

`./question/q3_react/src/App.js`

`./question/q3_react/src/BodyBlackout.js`

`./question/q3_react/src/Modal.js`

출제자 강사님 코드 기반으로, 해당 경로에서 요구한 문제사항을 해결해주세요

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js`

index.html 열기

q2

경로 `./question/q2_jquery`

index.html 열기

q3

경로 `./question/q3_react`

터미널

```
$ npm install  
$ npm start
```

Case15 : ModalWindow - 출제자 해설

q1. vanilla.js로 코드를 작성해보세요.

A) 중점포인트

모달이 떳을 때, 모달안을 클릭해도 안닫힌다.
모달 바탕을 클릭 했을 때, 모달도 닫혀야 된다.

```
document.querySelector('.popup-trigger').addEventListener('click' , function(){
    popupModal.classList.add('is--visible')
    bodyBlackout.classList.add('is-blacked-out')
});
```

팝업 띄우기 클릭시,

popupModal class명에 is-visible 추가
bodyBlackout class명에 is-blacked-out 추가

```
popupModal.querySelector('.popup-modal__close').addEventListener('click' , () => {
    popupModal.classList.remove('is--visible')
    bodyBlackout.classList.remove('is-blacked-out')
})
```

x 버튼 클릭시

popupModal class명에 is-visible 제거
bodyBlackout class명에 is-blacked-out 제거

```
bodyBlackout.addEventListener('click' , () => {
    popupModal.classList.remove('is--visible')
    bodyBlackout.classList.remove('is-blacked-out')
})
```

```
})
```

바탕을 클릭 시 클래스 리스트에서 제거한다. 바탕을 클릭했을 때 모달이 닫히는 것은 z-index 순서를 조절해서 모달이 바탕화면보다 위에 있기 때문이다.

q2. jQuery 로 코드를 작성해보세요.

A) 중점포인트

q_index.html 만을 건드린다.

js 를 작성하지 않는다.

문서를 보고 완성

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<!-- jQuery Modal -->

<script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-modal/0.9.1/jquery.modal.min.js"></script>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jquery-modal/0.9.1/jquery.modal.min.css" />

<script src='./s2_index.js'></script>
```

body 위로 모듈을 불러온다.

```
<a href="#" rel="modal:close"><i class="fa fa-window-close popup-modal__close"></i>

</a>
```

modal:close 닫기버튼 modal:open 열기버튼 기타 css 는 따로 작성할 필요없이 script 에 불러온 내용을 넣습니다.

q3. React 로 코드를 작성해보세요.

A) 중점포인트

useState 흑으로 잘 처리 합니다. <해설> css 로 props 를 받는 부분을 봅니다

```
display: ${props => props.isVisible ? "block" : "none"};
```

app.js

```
const [ isVisible , setIsVisible ] = useState(false);

const onSetIsVisible = (active) => {
  setIsVisible(active);
}

}
```

useState boolean 으로 모달의 상태를 컨트롤 합니다.

```
<BodyBlackout  isVisible={isVisible}  onSetIsVisible={onSetIsVisible}  />

<Modal  isVisible={isVisible}  onSetIsVisible={onSetIsVisible}  />
```

추가로 생각해볼 점

ModalContainer.js 로 BodyBlackout , Modal 컴포넌트를 포함해서 한번에 넘겨서 처리하도록 만들어 봅니다.

Case16 : Progress Bar

케이스 주제

Q. Progressbar 구현

기능 요구사항

- 총 4번 스텝 이동을 할수 있고 숫자는 조절 가능하다.
- 이전으로 하면 왼쪽으로(25%), 다음으로 오른쪽(25%)
- 다 채우거나 다 사라지면 클릭해도 작동안한다.

기능 작동 이미지



[이전](#) [다음](#)

실행 방법 / 풀이 방법 안내

jQuery 와 Vanilia.js 는 span 안에 width 값을 조절한다.
css 로는 애니메이션 효과를 주지않고, setInterval 함수를 사용한다.

q1. Javascript - q1_index.html 실행

q2. JQuery - q2_index.html 실행

q3. React

css를 활용한다. (src/ProgressBar.js 에 transition 참고)

다음이나 이전을 연속클릭해도 함께 동작하지 않게 한다.

문제

q1. Javascript - 주어진 템플릿(q_index.html

)으로 캐러셀 구현

q2. Jquery - Javascript로 구현한 기능을 동일하게 Jquery로도 구현해보는 문제

q3. React - 주어진 코드를 이해하고, 상태를 조작하는 툥을 이용해서 완성

풀이 참고

JQuery animate 를 활용한다.

작성해주셔야 하는 question 파일경로

q1

./question/q1_js/q1_index.js

q2

./question/q2_jquery/q2_index.js

q3

./question/q3_react/src/App.js

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/s1_js

index.html 열기

q2

경로 ./question/s2_jquery

index.html 열기

q3

경로 ./question/s3_react

터미널

```
$ yarn install  
$ yarn start
```

Case16 : ProgressBar - 출제자 해설

q1. vanilla.js 로 코드를 작성해보세요.

A)

css 사용없이 vanilla.js animation 처리하기
setInterval 함수의 이해

해설

이전으로 상태바이동

```
ProgressBar.prototype.movePrev = function(){
    // ... 이전 생략

    var intervalId = setInterval(frame, this.intervalSpeed );

    var elem = this.targetElement;

    function frame() {
        if ( start <= end ) {
            clearInterval(intervalId);
        } else {
            start--;
            elem.style.width = start + "%";
        }
    }
}
```

frame 함수를 호출한다
시작지점의 길이가 끝지점의 길이보다 작으면 setInterval 해제
그밖에 지속적으로 감소

다음으로 이동

```
function frame() {
    if ( start >= end ) {
        clearInterval(intervalId);
    } else {
        start++;
        elem.style.width = start + "%";
    }
}
```

도착지점보다 길면 인터벌 해제
그밖에는 지속적으로 길이 감소

q2. jQuery로 코드를 작성해보세요.

A) 중점포인트

jQuery animate의 이해

해설

이전으로 가기 클릭시

```
$( '#prev' ).click(function(){
    if( current == 0 ) return;

    current--;

    $(".progress-bar > span").animate({
        width: 25 * current + '%'
    }, 500);
})
```

```
});
```

25%씩 줄어든 길이를 입력해 놓으면 해당 지점까지 애니메이션을 보여주며 이동한다.
500은 애니메이션 스피드
다음으로 이동하기도 늘어난 길이를 입력한다.

q3. React로 코드를 작성해보세요.

A)

애니메이션이 진행되는 동안 이벤트 작동 안되게 처리
useRef 활용

해설

src/App.js

```
const isLoading = useRef(false);
```

애니메이션이 진행되는지 체크
컴포넌트가 리랜더링되더라도 유지

애니메이션 딜레이 체크 하는 함수

```
const delay = ( delay ) => {
  isLoading.current = true;
  return new Promise( () =>
    setTimeout(
      () => isLoading.current = false
    )
  );
};
```

```
        , delay )
    );
}
```

true로 설정
params로 받은 delay 시간뒤에 false로 바꾼다.
await로 처리하기 위해 Promise를 리턴한다.

다음으로 이동

```
const handleNext = async() => {

    if( isLoading.current ) return;
    if( current === limit ) return;
    setCurrent( current + 1 );
    await delay(animationSpeed);

}
```

애니메이션이 진행중이면 함수 종료
마지막 까지 다차면 종료
그게 아니면 스텝 증가
delay를 실행한다.

이전으로 이동

```
if( isLoading.current ) return;
if( current === 0 ) return;
```

함수 종료 조건 : 애니메이션 중인가 or progressbar 가 처음
이하 다음으로 이동과 동일

Case17 : Carousel

케이스 주제

Q. 캐러셀(이미지슬라이더) 구현

기능 요구사항

1. 오른쪽, 왼쪽 화살표로 슬라이더 컨트롤
2. 슬라이더 갯수는 3개 이상
3. 슬라이더가 이동하는 동안 화살표 작동안됨
4. 슬라이더 자연시간은 0.5초
5. 첫 로딩시 현재위치는 0번째 슬라이더

기능 작동 이미지



The screenshot shows the browser's developer tools with the 'Elements' tab selected. The DOM tree displays a structure for a carousel, including a wrapper and multiple items. The 'Styles' panel on the right shows the computed styles for the active item, which includes a background image and some margin settings.

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body> == $0
    <div class="carousel-wrapper">
      <div class="carousel">
        
        
        
        
        
      </div>
    </div>
  </body>
</html>
```

Styles Computed >
element.style {
}
body user agent stylesheet
y {
 display: block;
 margin: 8px;
}

문제

q1. javaScript - 주어진 템플릿(`q_index.html`

)으로 캐러셀 구현 q2. jQuery - Javascript로 구현한 기능을 동일하게 Jquery로도 구현해보는 문제 q3.

React - 주어진 코드를 이해하고, 상태를 조작하는 툥을 이용해서 완성

풀이 참고

1. 슬라이더 갯수가 5개라면 첫로딩시 아래 클래스 활성화

0번째에 active

1번째에 next

4번째 prev

2. css 설명

.carousel_item 의 transition 딜레이 조절

transform: translateX(-100%) x축 왼쪽이동

.carousel 의 preserve-3d 효과

주요 학습 키워드

prototype을 사용하여 메소드와 속성을 정의한 뒤, new 연산자를 사용해 함수로 호출하여 기능을 구현

각각의 슬라이드 이동은 translateX의 값으로 슬라이드의 위치를 조절

반복문을 사용하여 슬라이드 위치에 맞게 className을 지정

useState, useRef, useEffect hook 사용

작성해주셔야 하는 question 파일경로

q1

`./question/q1_js/q1_index.js`

q2

```
./question/q2_jquery/q2_index.js
```

q3

```
./question/q3_react/src/Carousel.js
```

출제자 강사님 코드 기반으로, 해당 경로에서 요구한 문제사항을 해결해주세요

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/q1_js

index.html 열기

q2

경로 ./question/q2_jquery

index.html 열기

q3

경로 ./question/q3_react

터미널

```
$ npm install  
$ npm start
```

Case17 : Carousel 1 - 출제자 해설

q1. vanilla.js 로 코드를 작성해보세요.

A)

다음 슬라이드로 이동하는 시간(0.5초)동안 다음 슬라이드로 이동하면 안된다. 이전으로 가기 클릭시에도 동일하게 작동 한다.

마지막 슬라이드(5)에서 다음 슬라이드로 클릭하면 첫번째로, 첫번째 슬라이드에서 이전 슬라이드로 이동 클릭하면 마지막 슬라이드로 간다.

슬라이드의 갯수를 늘리더라도 슬라이드 갯수만큼 작동해야 한다.

```
function Carousel( carouselElement ){

    this.carouselElement = carouselElement;

    this.itemClassName = "carousel_item";

    this.items = this.carouselElement.querySelectorAll('.carousel_item');

    //.... 이하 생략
```

.carousel_item 슬라이드 각 아이템을 설정한다. querySelectorAll

```
Carousel.prototype.disableInteraction = function(){

    this.isMoving = true;

    setTimeout( () => {

        this.isMoving = false
```

```
}, 500);  
}
```

슬라이드 클릭 시 isMoving 을 true 기본상태로, 0.5 초뒤 isMoving 을 false 로 바꾼다.

```
Carousel.prototype.movePrev = function() {  
  
    if (!this.isMoving) {  
  
        if (this.current === 0) {  
  
            this.current = (this.totalItems - 1);  
  
        } else {  
  
            this.current--;  
  
        }  
  
        this.moveCarouselTo();  
  
    }  
}
```

이전으로 가기 클릭시, isMoving 의 상태를 확인한다.
0 번째는 현재 위치가 첫번째 이므로, 마지막 슬라이드로 현재 위치를 변경한다,
그게 아니면 현재 위치를 감소 시킨다.

```
Carousel.prototype.moveNext = function() {  
  
    if (!this.isMoving) {  
  
        if (this.current === (this.totalItems - 1)) {  
            this.current = 0;  
  
        } else {  
  
        }  
    }  
}
```

```
        this.current++;

    }

    this.moveCarouselTo();

}

}
```

다음으로 가기는 현재위치가 마지막 위치인지만 확인하면 된다(ex - this.totalItems - 1)
마지막 위치라면 다음 슬라이드는 처음(0) 으로 이동한다.
그밖의 상황은 슬라이드 위치를 증가시킨다.

```
Carousel.prototype.moveCarouselTo = function() {

    // ... 이전 생략
    this.disableInteraction();

    var prev = this.current - 1,
        next = this.current + 1;

    if (this.current === 0) {
        prev = (this.totalItems - 1);
    } else if (this.current === (this.totalItems - 1)) {
        next = 0;
    }

    //... 다음 설명 참조

}
```

이 함수는 슬라이드 이동을 완료하는 함수다.
현재 위치를 가지고, 이전과 이후 슬라이드를 셋팅한다.
현재위치가 0 번째인지와 마지막 인지를 체크한다.
현재위치가 0 이면 next는 현재위치 + 1
현재위치가 마지막 이면 next는 처음으로 간다.

인덱스를 반복문으로 돌리면서 현재위치면 active 이전이면 prev 다음이면 next 를 붙인다. 그 외에는 기본 클래스르 명만 붙인다.

```
if ((this.totalItems - 1) > 3) {  
  
    for(var i=0 ; i<this.totalItems ; i++ ){  
  
        if(i==this.current){  
            this.items[i].className = this.itemClassName + " active";  
        }else if(i==prev){  
            this.items[i].className = this.itemClassName + " prev";  
        }else if(i==next){  
            this.items[i].className = this.itemClassName + " next";  
        }else{  
            this.items[i].className = this.itemClassName;  
        }  
    }  
}
```

q2. jQuery 로 코드를 작성해보세요.

A)

querySelectorAll 을 jQuery로 셀렉터로 지정하기
클릭이벤트 jQuery 로 할당

해설

하위 엘리먼트 설정

```
$(셀렉터).children()
```

아래와 같이 클릭이벤트 설정

```
this.prevButton = this.carouselElement.children('.carousel_button--prev');
this.prevButton.click(() => { ... })
```

q3. React로 코드를 작성해보세요.

A) 중점포인트

0.5초 뒤에 isMoving을 false로 처리하기
useState 흑으로 현재 슬라이드의 위치

해설

useRef로 컴포넌트가 리랜더링 되더라도 변수를 유지

src/Carousel.js

```
const isMoving = useRef(false);

useEffect(() => {

    isMoving.current = true;
    setTimeout(() => {
        isMoving.current = false;
    }, 500);

}, [current]);
```

isMoving.current를 시작전에 true
0.5초뒤에 false로 바꾼다.

다음 슬라이드로 가기

```
const moveNext = () => {
  if(!isMoving.current){
    if (current === (totalItems-1)) {
      setCurrent(0);
    } else {
      setCurrent(current+1);
    }
  }
}
```

isMoving.current 현재 슬라이드가 작동중인지 체크한다.

if: 마지막 위치라면 첫번째 (0)로 이동

else: 다음 슬라이드로 이동

이전 슬라이드로 가기

```
const movePrev = () => {

  if(!isMoving.current){
    if (current === 0) {
      setCurrent(totalItems - 1);
    } else {
      setCurrent(current-1);
    }
  }
}
```

isMoving.current 현재 슬라이드가 작동중인지 체크한다.

if: 첫번째 슬라이드라면 다음슬라이드는 마지막 슬라이드로

else: 현재 슬라이드 위치 감소

버튼의 이전위치와 다음 위치를 설정해주어서 Button에 prev와 next boolean 값을 넘긴다.

```
const prev = ((current) === 0 ? totalItems - 1 : current-1);
const next = (current === (totalItems-1) ) ? 0 : current+1;
```

prev: 현재위치가 0 이면 이전위치는 마지막 슬라이드, 그밖은 -1

next: 현재위치가 마지막이면 다음 위치는 0, 그밖은 +1

Case18 : Pagination

케이스 주제

Q. 페이지네이션 구현

기능 요구사항

1. 주어진 댓글 데이터 22개에 대해 페이지를 나눠 댓글 요소를 뷰에 표현한다.
2. 페이지 당 표현할 수 있는 댓글 요소 수를 설정할 수 있다.
3. 현재 페이지를 알 수 있고, 다음 페이지로 넘어갈 수 있는 뷰에 기능을 구현한다.

기능 작동 이미지



abc_1

2020-05-01

UI 테스트는 어떻게 진행하나요



abc_2

2020-05-02

막히면 대답은 빨리 해주나요



abc_3

2020-05-03

코드에러가 발생했는데 어딘지 모르겠어요



abc_4

2020-05-04

Javascript 기초가 부족한데 좋은 가이드 있나요



abc_5

2020-05-05



TypeScript는 어떤점에서 좋나요

- 1
- 2
- 3
- 4
- 5

실행 방법 / 풀이 방법 안내

문제 풀기 방식 :

1. 레포지토리를 clone
2. question 디렉토리에 package.json이 없는 경우, index.html을 열거나, live-server extension으로 열기. 이외 경우 아래 안내 참고.
3. 터미널에서 각 문제 폴더 디렉토리로 이동하여 npm install로 의존성을 설치
4. package.json을 참고하여, 명시된 scripts 명령어로 개발서버 실행.
5. 코드 수정하면서 문제 해결하세요

기본 번들러로 `parcel`을 사용했습니다. - `react` 문제의 경우, `react-scripts` 사용. 문제 디렉토리에서 `npm start` 또는 `npx parcel index.html watch`로 개발서버를 실행하세요.

문제

q1. JavaScript로 해당 기능을 구현하세요. (* `index.js` 파일 확인)

페이지 번호를 동적으로 생성했을 때, 이벤트를 할당하기
페이지네이션 페이지번호를 주었을 때, 나오는 게시물수 컨트롤하기
활성화 된 버튼 처리하기

q2. jQuery 해당 기능을 구현하세요. (* `index.js` 파일 확인)

jQuery 문서로드 처리하기
jQuery로 동적으로 엘리먼트 생성시, 이벤트를 할당하는 방법

q3. React로 해당 기능을 구현하세요. (* `comment.js` 파일 확인)

각 클릭 시 react로 상태를 어떻게 처리하는지
redux toolkit을 사용

주요 학습 키워드

페이지 설정에 따른 댓글 데이터 가공 표현

작성해주셔야 하는 question 파일경로

q1

`./question/q1_js/index.js`

q2

`./question/q2_jquery/index.js`

q3

```
./question/q3_react_redux/src/containers/CommentListContainer.js
```

```
./question/q3_react_redux/src/containers/PageListContainer.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js`

index.html 열기

q2

경로 `./question/q2_jquery`

index.html 열기

q3

경로 `./question/q3_react_redux`

터미널

```
$ npm install  
$ npm start
```

Case18 : Pagination - 출제자 해설

q1. vanilla.js 로 코드를 작성해보세요.

중점포인트

페이지 번호를 동적으로 생성했을 때, 이벤트를 할당하기
페이지네이션 페이지번호를 주었을 때, 나오는 게시물수 컨트롤하기
활성화 된 버튼 처리하기

A)

```
function Pagination(){

    // 초기값 설정
    this.current_page = 1;
    this.limit = 5;
    this.total_page = Math.ceil( comments.length / this.limit );
}
```

총 페이지 수는 commets 배열의 길이를 한페이지에 보여질 게시물수로 나눈다. ex) 한페이지에 5개씩 보이고 총 게시물 수가 23 개라면 23/5 를 올림 계산해서 5페이지가 나온다. 올림을 하는 이유는 마지막 페이지에 게시물 수가 3개라도 1페이지로 치기 때문이다.

```
Pagination.prototype.getComments = function(page){

    var start = ( page - 1 ) * this.limit;
    this.current_page = page;
    var limit = start + this.limit;
    return comments.filter( function( _ , index ){

        if( index >= start && index < limit ) return _;
```

```
        if(index >= start && index < limit ) return true;
        else return false;
    );
}
```

Parameter 각 페이지를 인자로 주었을때 Return 해당하는 게시물 수 시작지점부터 한페이지에 보여질 게시물 수를 보여준다

```
1 page ) 0번부터 5(한페이지에 보여줄숫자)까지의 게시물을 보여주면된다. start : 23*(1페이지-1) * limit
2 page ) (2-1) * 5
3 page ) (3-1) * 5 각 시작지점은 위와 같고 5개의 게시물씩 보여주면 된다.
```

아래와 같이 처리 하는 것은 배열의 인덱스가 시작지점부터 한페이지에 보여질 게시물 수의 인덱스로 처리하기 위함이다.

```
return comments.filter( function( _ , index ){
    if(index >= start && index < limit ) return true;
    else return false;
});
```

게시물 배열을 넘겼을 때 html 태그에 맞춰서 출력해주는 함수를 구현한다.

```
Pagination.prototype.getCommentFormat = function (comments) {
    return comments.reduce(function (acc, comment) {
        return (
            acc +
            `<div class="commentWrap">
                
                ${comment.author} +
                <div class="createdAt">
                    ${comment.created_at}
```

```
comment.createdAt +  
`  
  </div>  
  <div class="content">  
    ` +  
    comment.content +  
`  
  </div>  
  <hr />  
`</div>`  
);  
, "");  
};
```

아래와 같이 처리 하는것은 html 양식에 맞춰 순회하면서 acc에 누적해서 리턴해주기 위함이다.

```
return comments.reduce( function(acc, comment ){
```

총 페이지의 숫자를 입력시 페이지를 화면에 뿌려주는 함수를 구현한다.

```
Pagination.prototype.getPageListFormat = function () {  
  var result = "";  
  
  for (var num = 1; num <= this.total_page; num++) {  
    var className = num == this.current_page ? "active" : "";  
    result += '<button class="' + className + '">' + num + "</button>";  
  }  
  
  return result;  
};
```

```
document.addEventListener("DOMContentLoaded", function () {  
  var pagination = new Pagination();  
  
  var commentContainer = document.querySelector("#commentContainer");  
  var pageContainer = document.querySelector("#pageContainer");
```

```
commentContainer.innerHTML = pagination.getCommentFormat(
    pagination.getComments(1)
);
pageContainer.innerHTML = pagination.getPageListFormat();
```

1. 문서가 로드 된 뒤 1페이지에 해당하는 게시물을 보여준다
2. 페이지 리스트들을 보여주고, 1페이지를 활성화한다.

```
document.addEventListener(
    "click",
    function (event) {
        // If the clicked element doesn't have the right selector, bail
        if (event.target.matches("#pageContainer button")) {
            var page = event.target.innerHTML;
            commentContainer.innerHTML = pagination.getCommentFormat(
                pagination.getComments(page)
            );
            pageContainer.innerHTML = pagination.getPageListFormat();
            return;
        }
    },
    false
);
});
```

1. 각 페이지 버튼을 클릭했을 때, 페이지에 해당하는 게시물을 뿌려주고
2. 하단 페이지리스트를 뿌려준다.
3. event.target.matches로 처리한것은 각 클릭시 하단도 새로 리프레시 되므로 문서가 로드하고나서 이벤트를 할당하지 않고 클릭하고나서 해당 셀렉터 인지 확인한 후에 작동하게 한다.

생각해볼 점

pagination 객체로 생성했는데, html 안에 데이터를 넣어주는 부분도 pagination 함수안에 넣어줄지 말지 어느부분이 설계상 더 좋을지 고민해보기

아래부분

```
pagination.getCommentFormat(pagination.getComments(1));
```

q2. jQuery로 코드를 작성해보세요.

중점포인트

1. jQuery 문서로드 처리하기
2. jQuery로 동적으로 엘리멘트 생성시, 이벤트를 할당하는 방법

A)

vanilia.js와 구현은 비슷하다
문서 로드시 \$(document).ready로 처리하고

아래와 같이 on click으로 동적 처리를 실행한다.

```
$(document).on("click", "#pageContainer button", function () {  
    var page = $(this).html();  
    commentContainer.html(  
        pagination.getCommentFormat(pagination.getComments(page))  
    );  
    pageContainer.html(pagination.getPageListFormat());  
});
```

q3. React로 코드를 작성해보세요.

중점포인트

각 클릭시 React로 상태를 어떻게 처리하는지

A)

Redux toolkit을 사용했다.
actions와 reducers는 아래와 같이 작성한다.

```
export const { getComments } = slice.actions;
export const commentsReducer = slice.reducer;
```

```
getComments( state, action ) {

  const start = ( action.payload.page - 1 ) * state.limit;
  const limit = start + state.limit;

  state.data = state.comments.filter(
    ( _, index ) => {
      if(index >= start && index < limit ) return true;
      else return false;
    }
  );

  state.current_page = action.payload.page;
}
```

위에 vanila.js와 같이 페이지 네이션을 처리한다.

getComments라는 액션을 생성한다.

initialState에 data라는 변수를 생성한 이유는 comments 임시 데이터는 유지하고 보여질 부분만 그때그때 data 변수를 생성한다. - 게시물을 보여주는 부분은 data를 참조해서 본다.

추가로 생각해볼 점

useState로 redux 없이 구현해보기

Case18 : Pagination - 대기업 S사 프론트엔드 개발자님의 답안

q1. 문제 상황에 대하여 Java Script로 동작을 구현시킬 수 있는 코드를 작성해 보세요

A)

```
/*
 * 페이지 객체
 * @param {*} commentContainerEl
 * @param {*} paginationContainerEl
 * @param {*} initialItems
 */
const Pagination = function (
    commentContainerEl,
    paginationContainerEl,
    initialItems = []
) {
    if (this.constructor !== Pagination) {
        throw Error('객체를 생성후 사용해 주세요!');
    }
    /**
     * 현재 페이지 인덱스(0부터 시작)
     */
    this.currentPageIndex = 0;
    /**
     * 페이지당 최대 노출 목록 갯수
     */
    this.maxPageItemCount = 5;
    /**
     * 초기 데이터 세팅
     */
    this.commentItems = initialItems;
    /**
     * Comment 컨테이너
     */
    this.commentContainerEl = commentContainerEl;
    /**
     * Pagination 컨테이너
     */
    this.paginationContainerEl = paginationContainerEl;

    /**
     * 초기 화면을 랜더링
     */
    this.render();
}
```

```

const onClickButtons = (event) => {
    /**
     * 버튼 컨테이너에서 이벤트를 위임받아 처리
     */
    const { target } = event
    if (target.tagName === 'BUTTON') {
        const index = Number(target.dataset.index);
        this.setCurrentPageIndex(index);
    }
};

this.paginationContainerEl
    .addEventListener('click', onClickButtons);

/**
 * 렌더링된 페이지를 지우고 할당된 데이터를 모두 해제
 */
this.destroy = () => {
    this.currentPageIndex = 0;
    this.commentItems = null;

    this.commentContainerEl.innerHTML = '';
    this.paginationContainerEl.innerHTML = '';

    paginationContainerEl.removeEventListener('click', onClickButtons);

    this.commentContainerEl = undefined;
    this.paginationContainerEl = undefined;
}

};

/***
 * 데이터 변경
 * @param {*} commentItems
 */
Pagination.prototype.setPageItems = function (commentItems) {
    if (this.constructor !== Pagination) {
        throw Error('객체를 생성후 사용해 주세요!');
    }
    this.commentItems = commentItems;
    this.render();
};

/***
 * 컨테이너 엘리먼트를 변경
 * @param {*} commentContainerEl
 */
Pagination.prototype.setCommentContainerEl = function (commentContainerEl) {
    this.commentContainerEl = commentContainerEl;
};

/***
 * 페이징 컨테이너 변경
 * @param {*} paginationContainerEl
 */
Pagination.prototype.setPaginationContainerEl = function (paginationContainerEl) {
    this.paginationContainerEl = paginationContainerEl;
};

/***
 * 게시글 목록 생성
 */

```

```

Pagination.prototype.makeCommentsHtml = function () {
    if (this.constructor !== Pagination) {
        throw Error('객체를 생성후 사용해 주세요!');
    }
    const {
        commentItems,
        currentPageIndex,
        maxPageItemCount
    } = this;
    const startIndex = currentPageIndex * maxPageItemCount;
    return commentItems
        .slice(startIndex, startIndex + maxPageItemCount)
        .map((item) => {
            const {
                id,
                profile_url,
                author,
                content,
                createdAt
            } = item;
            return (
                `<div class="commentWrap" data-id="${id}">
                    
                    ${author}
                    <div class="createdAt">
                        ${createdAt}
                    </div>
                    <div class="content">
                        ${content}
                    </div>
                </div>
                <hr />
                `
            );
        })
        .join(' ');
};

/**
 * 페이지 버튼 생성
 */
Pagination.prototype.makePaginationHtml = function () {
    if (this.constructor !== Pagination) {
        throw Error('객체를 생성후 사용해 주세요!');
    }
    const {
        commentItems,
        currentPageIndex,
        maxPageItemCount,
    } = this;
    const count = Math.ceil(commentItems.length / maxPageItemCount);
    const buttons = [];
    for (let i = 0; i < count; i++) {
        buttons.push(
            i === currentPageIndex
                ? `<button class="active" data-index=${i}>${(i + 1)}</button>`
                : `<button data-index=${i}>${(i + 1)}</button>`
        )
    }
    return buttons.join(' ');
};

```

```

/**
 * 템플릿 랜더링
 */
Pagination.prototype.render = function () {
    if (this.constructor !== Pagination) {
        throw Error('객체를 생성후 사용해 주세요!');
    }
    const {
        commentContainerEl,
        paginationContainerEl
    } = this;
    if (commentContainerEl && paginationContainerEl) {
        commentContainerEl.innerHTML = this.makeCommentsHtml();
        paginationContainerEl.innerHTML = this.makePaginationHtml();
        return true;
    }
    return false;
};

Pagination.prototype.setCurrentPageIndex = function (pageIndex) {
    if (this.constructor !== Pagination) {
        throw Error('객체를 생성후 사용해 주세요!');
    }
    this.currentPageIndex = pageIndex;
    this.render();
}

document.addEventListener('DOMContentLoaded', () => {
    const commentContainerEl = document.querySelector('#commentContainer');
    const paginationContainerEl = document.querySelector('#pageContainer');
    /**
     * 페이지네이션 객체 인스턴스 생성
     */
    const pagination = new Pagination(
        commentContainerEl, paginationContainerEl, comments);
})

```

해설

[!] 제공된 Question 템플릿과 다른 방법으로 문제를 풀이하였습니다. 참고 부탁드립니다.

Pagination 객체를 만들고, 모든 인스턴스에서 공용으로 사용되는 메서드는 `prototype` 속성을 이용하여 할당하였습니다.

`prototype`
 을 이용한 속성은 해당 객체에서 생성되는 모든 인스턴스에서 참조하여 사용되기 때문에 인스턴스가 생성될 때마다 다시 생성되지 않고 모두 공유됩니다.
`prototype`
 내부에서 `this`
 객체를 참조하기 위해서 모두 `function`
 표현식을 사용하였고 인스턴스를 통한 접근이 아닌, 직접 `prototype`
 을 접근하는 경우 에러를 발생하도록 만들어 두었습니다(Pagination으로 생성된 객체를 bind
 할 경우 직접 접근도 가능합니다.)

Pagination 객체는 다음과 같은 구성입니다.

```
setPageItems: 페이지 데이터를 변경  
setCommentContainerEl: 코멘트 컨테이너 객체 엘리먼트 변경  
setPaginationContainerEl: 페이지네이션 컨테이너 엘리먼트 변경  
makeCommentsHtml: 코멘트 마크업 생성  
makePaginationHtml: 페이지네이션 마크업 생성  
render: 페이지 랜더링  
setCurrentPageIndex: 현재 페이지 인덱스 변경
```

Pagination 객체를 이용하여 인스턴스를 생성하면 자동으로 지정된 엘리먼트 영역에 랜더링이 됩니다.

DOMContentLoaded
이벤트를 이용하여 HTML이 파싱되는 즉시 화면을 그리도록 구현하였습니다(참고자료에 있는 load 이벤트와 차이점을 비교해 보세요!).

참고자료

```
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/this  
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/function  
https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Global\_Objects/Object/prototype  
https://developer.mozilla.org/ko/docs/Web/Events/DOMContentLoaded  
https://developer.mozilla.org/ko/docs/Web/Events/load
```

q2. jQuery로 코드를 작성해보세요.

A)

```
/**  
 * 페이지 객체  
 * @param {*} $commentContainerEl  
 * @param {*} $paginationContainerEl  
 * @param {*} initialItems  
 */  
const Pagination = function (  
    $commentContainerEl,  
    $paginationContainerEl,  
    initialItems = []  
) {  
    if (this.constructor !== Pagination) {  
        throw Error('객체를 생성후 사용해 주세요!');  
    }  
    ...  
  
    const onClickButtons = (event) => {  
        const { target } = event  
        /**  
         * @see https://api.jquery.com/data/  
         */  
        const index = $(target).data('index');
```

```

        this.setCurrentPageIndex(index);
    };

    this.$paginationContainerEl
        .on('click', onClickButtons);

    /**
     * 랜더링된 페이지를 지우고 할당된 데이터를 모두 해제
     */
    this.destroy = () => {
        this.currentPageIndex = 0;
        this.commentItems = null;

        this.$commentContainerEl.html('');
        this.$paginationContainerEl.html('');

        /**
         * Jquery Event Delegation 기능을 활용
         * @see https://api.jquery.com/on/
         */
        $paginationContainerEl.off('click', 'button', onClickButtons);

        this.$commentContainerEl = undefined;
        this.$paginationContainerEl = undefined;
    }
};

...
$(() => {
    /**
     * DOMContentLoaded 동일한 시점에 호출
     * @see https://api.jquery.com/ready/
     */
    const $commentContainerEl = $('#commentContainer');
    const $paginationContainerEl = $('#pageContainer');
    /**
     * 페이지네이션 객체 인스턴스 생성
     */
    const pagination = new Pagination(
        $commentContainerEl, $paginationContainerEl, comments);
})

```

해설

1번 문제와 내용은 거의 동일하지만 이벤트 바인딩, DOM을 제어하기 위해 Jquery 라이브러리를 사용하였습니다.

참고자료

<https://api.jquery.com/ready/>
<https://api.jquery.com/on/>
<https://api.jquery.com/data/>

q3. React 로 코드를 작성해보세요.

A)

```
// solution/2.others_1/q3_react_redux/src/store/modules/comments.js
import { createSlice } from '@reduxjs/toolkit';

export const initialState = {
  commentItems: [...],
  currentPageIndex: 0,
  maxPageItemCount: 5,
};

const name = 'comments';

const slice = createSlice({
  name,
  initialState,
  reducers: {
    setCurrentPageIndex(state, action) {
      state.currentPageIndex = action.payload;
    }
  }
});

export const { getComments, setCurrentPageIndex } = slice.actions;
export const commentsReducer = slice.reducer;
```

```
// solution/2.others_1/q3_react_redux/src/containers/CommentListContainer.js
...

function CommentListContainer() {
  /**
   * [!] 불필요한 상태값을 모두 불러들이는 경우 상태값 변화에 따라 불필요하게 컴포넌트 랜더링 로직이 실행될 수 있습니다.
   * 필요한 속성만 `useSelector`를 이용하여 불러 사용하는 습관을 만들면 좋습니다.
   */
  const { commentItems, currentPageIndex, maxPageItemCount } = useSelector(({ comments }) => {
    const { commentItems, currentPageIndex, maxPageItemCount } = comments;
    return {
      commentItems, currentPageIndex, maxPageItemCount
    };
  });

  const data = useMemo(() => {
    const startIndex = currentPageIndex * maxPageItemCount;
    return commentItems.slice(startIndex, startIndex + maxPageItemCount);
  }, [commentItems, currentPageIndex, maxPageItemCount]);

  return <CommentList data={data} />;
}

export default CommentListContainer;

// solution/2.others_1/q3_react_redux/src/containers/PageListContainer.js
...
function PaginationContainer() {
```

```

const {
  commentItems,
  maxPageItemCount,
  currentPageIndex } = useSelector(
  (state) => state.comments
);
const dispatch = useDispatch();
const boundActionCreators = bindActionCreators({ setCurrentPageIndex }, dispatch);

/**
 * 페이지 갯수
 */
const buttonCount = useMemo(() => {
  return Math.ceil(commentItems.length / maxPageItemCount);
}, [maxPageItemCount, commentItems]);

/**
 * 현재 활성된 페이지
 */
const activeIndex = useMemo(() => currentPageIndex, [currentPageIndex]);

const handleClickPaginationButton = (_, index) => {
  boundActionCreators.setCurrentPageIndex(index);
}

return (
  <PaginationButtons
    buttonCount={buttonCount}
    activeIndex={activeIndex}
    onClick={handleClickPaginationButton}
  />
);
}

export default PaginationContainer;

```

```

// solution/2.others_1/q3_react_redux/src/components/CommentList.js
...
function CommentList({ data }) {
  return data.map((comment, index) => {
    /**
     * [!] React Component에서 `key` 값은 `index`로 바로 적용하지 않고, 유니크한 값으로 지정하는게 좋습니다.
     * 컴포넌트가 랜더링될때 같은 레벨의 형제(siblings) 컴포넌트와 중복된 `key` 값을 갖게되면 불필요한 랜더링이 발생할 수 있습니다.
     */
    const key = `comment_` + index;
    const {
      author,
      profile_url,
      createdAt,
      content
    } = comment;
    return (
      <Comment key={key}>
        <img src={profile_url} alt="" />
        {author}
        <CreatedAt>{createdAt}</CreatedAt>
        <Content>{content}</Content>
        <hr />
    );
  });
}

export default CommentList;

```

```

        </Comment>
    )
});
}

export default CommentList;

// solution/2.others_1/q3_redux_src/components/PageList.js
...
/***
 * 버튼 컴포넌트를 재사용 가능할 수 있도록 입력되는 데이터 값의 연산을 제거하여 결합도, 복잡도를 낮춤
 */
function PaginationButtons({ buttonCount = 0, onClick, activeIndex = 0 }) {
    if (buttonCount === 0) {
        /**
         * 버튼이 존재하지 않는 경우
         */
        return null;
    }
    const $buttons = Array(buttonCount).fill().map(_, index) => {
        const key = `button_${index}`;
        return (
            <PageButton
                active={index === activeIndex}
                onClick={(e) => onClick(e, index)} key={key}>
                {index + 1}
            </PageButton>
        );
    }
    return <PageListStyle>{$buttons}</PageListStyle>;
}

export default PaginationButtons;

```

해설

[!] 제공된 Question 템플릿과 다른 방법으로 문제를 풀이하였습니다. 참고 부탁드립니다.

상태값 및 일부 컴포넌트 네이밍 변경하였습니다.
 기존 제공된 컨테이너와 Redux를 최대한 활용하였고 컴포넌트와 데이터 결합도를 최대한 낮췄습니다.
 Redux 액션 형식은 FSA(Flux Standard Action)를 사용하였습니다.

참고자료

<https://github.com/redux-utilities/flux-standard-action>
<https://redux-toolkit.js.org/api/createSlice>
<https://ko.reactjs.org/docs/hooks-reference.html#usememo>
<https://ko.reactjs.org/docs/hooks-reference.html#usecallback>
<https://ko.reactjs.org/docs/lists-and-keys.html#keys-must-only-be-unique-among-siblings>

Case19 : Scroll top

케이스 주제

Q. 스크롤을 내렸을 경우 상단에 고정된 내비게이션바의 배경과 폰트 색상이 변경되는 기능을 구현하세요.

기능 요구사항

- 스크롤을 내릴 경우 내비게이션 배경 색상과 폰트 색상이 변경됩니다.
- 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 이전 상태로 돌아가게 됩니다.
- 스크롤을 다시 올릴 경우 곧바로 배경/폰트 색상을 이전 상태로 변경하는 방식으로도 구현해봅니다.

기능 작동 이미지

<스크롤 전>



<스크롤 후>



문제

q1. javaScript - 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 이전 상태로 변경

1. 현재 스크롤 위치를 가져온다
2. 스크롤 위치를 바탕으로 active 클래스를 추가하거나 제거한다

q2. javaScript - 스크롤을 다시 올릴 경우 곧바로 배경/폰트 색상을 이전 상태로 변경

1. 현재 스크롤 위치를 가져온다.
2. oldvalue, 스크롤의 위치와 연산 작업을 하여 active 클래스를 추가하거나 제거한다.
3. oldvalue를 스크롤 위치로 변경한다.

q3. javaScript - 자바스크립트에서 제공하는 마우스 휠 이벤트 동작 감지 기능을 사용해서 구현

1. 마우스 휠 이벤트 동작을 감지하여 동작시킨다
2. *참고 : 자바스크립트에서는 마우스 휠 방향을 알 수 있는 mousewheel, wheel, DOMMouseScroll 이벤트를 제공합니다.

q4. jQuery - 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 이전 상태로 변경

1. 현재 스크롤 위치를 가져온다.
2. 스크롤 위치를 바탕으로 active 클래스를 추가하거나 제거한다.

q5. jQuery - 스크롤을 다시 올릴 경우 곧바로 배경/폰트 색상을 이전 상태로 변경

주요 학습 키워드

크로스 브라우징을 고려하여 스크롤 위치를 가져오는 방법을 학습하게 됩니다.

작성해주셔야 하는 question 파일경로

q1

`./Question/q1_js_1/main.js`

q2

`./Question/q2_js_2/main.js`

q3

`./Question/q3_js_3/main.js`

q4

`./Question/q4_jq_1/main.js`

q5

`./Question/q5_jq_2/main.js`

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/q1_js_1`

index.html 열기

q2

경로 `./question/q2_js_2`

index.html 열기

q3

경로 `./question/q3_js_3`

index.html 열기

q4

경로 `./question/q4_jq_1`

index.html 열기

q5

경로 `./question/q5_jq_2`

index.html 열기

Case19 : ScrollTop - 출제자 해설

q1. Javascript - 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 이전 상태로 변경

A)

스크롤 다운은 배경과 폰트 색상 변경 / 스크롤 업은 변경 상태를 유지하던 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 최초 상태로 변경
스크롤 동작을 감지하기 위해서는 window 객체 또는 document 객체에 addEventListener를 사용하여 스크롤 이벤트를 추가합니다. 스크롤 이벤트는 지금 스크롤 중인지 아닌지를 감지하게 됩니다.

해설

```
// window 객체
window.addEventListener('scroll', function() {
  console.log('scrolling');
})

// or

window.onscroll = function() {.....}

// document 객체
document.addEventListener('scroll', function() {.....})
```

만약 특정 영역 안에서 스크롤 이벤트를 적용하고 싶다면 아래와 같이 변경합니다.

```
// 선택자로 특정 영역을 가리킨 후 스크롤 이벤트 추가
const section1 = document.querySelector('#section-1');

section1.addEventListener('scroll', function() {.....})
```

다음 스크롤 위치를 가져오는 코드를 추가합니다. 이 때 크로스 브라우징을 고려해서 복수의 코드를 입력해야 합니다. 각 코드가 지원하는 브라우저는 아래 표에서 확인할 수 있습니다.

```
// window 객체
window.addEventListener('scroll', function() {
  const top = window.scrollY || window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop;
})
```



스크롤 특정 위치를 가져올 수 있게 되었다면 조건문 또는 삼항 연산자를 사용하여 참일 경우에는 active 클래스를 추가하고 거짓을 경우에는 제거해주는 코드를 작성합니다.

```
// window 객체
window.addEventListener('scroll', function() {
  const top = window.scrollY || window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop;
  (top >= 50 )
  ? nav.classList.add('active')
  : nav.classList.remove('active');
})
```

q2. Javascript - 스크롤을 다시 올릴 경우 곧바로 배경/폰트 색상을 이전 상태로 변경

A)

```
let oldValue = 0;
window.addEventListener('scroll', function(e){
  const newValue = window.scrollY || window.pageYOffset || document.documentElement.scrollTop || document.body.scrollTop;
  // 음수는 스크롤 다운, 양수는 스크롤 업
  if(oldValue - newValue < 0) nav.classList.add('active');
  if(oldValue - newValue > 0) nav.classList.remove('active');
  // 기준 값을 변경 값으로 치환
  oldValue = newValue;
});
```

해설

스크롤 다운은 배경과 폰트 색상 변경 / 스크롤 업은 이전 상태로 변경
최초 기준 값을 설정한 후 기준 값 - 변경 값을 연산하여 스크롤 다운 / 스크롤 업 상태를 판단할 수 있습니다.
기준 값 - 변경 값 연산이 음수면 스크롤 다운, 양수면 스크롤 업입니다.
기준 값은 항상 변경 값으로 치환하여 새롭게 갱신을 해야 합니다.

q3. Javascript - 자바스크립트에서 제공하는 마우스 휠 이벤트 동작 감지 기능을 사용해서 구현

A)

```
window.addEventListener('wheel', mouseWheelEvent);
window.addEventListener('DOMMouseScroll', mouseWheelEvent);

function mouseWheelEvent(e) {
  const delta = e.wheelDelta ? e.wheelDelta : -e.detail;
  (delta < 0)
  ? nav.classList.add('active')
  : nav.classList.remove('active');
}

// or

const isFireFox = (navigator.userAgent.indexOf('Firefox') !== -1);
const wheelEvt = isFireFox ? 'DOMMouseScroll' : 'wheel';

window.addEventListener(wheelEvt, mouseWheelEvent);
```

```
function mouseWheelEvent(e) {
  const delta = e.wheelDelta ? e.wheelDelta : -e.detail;
  (delta < 0)
  ? nav.classList.add('active')
  : nav.classList.remove('active');
}
```

해설

자바스크립트에서는 마우스 휠 방향을 알 수 있는 mousewheel, wheel, DOMMouseScroll 이벤트를 제공합니다.
mousewheel은 비표준으로, wheel을 사용해야 합니다.
파이어폭스에서는 DOMMouseScroll을 사용해야 합니다.

q4. Jquery - 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때 (최상단에 스크롤이 위치할 때) 이전 상태로 변경

A)

```
$(window).scroll(function () {
  const $top = $(this).scrollTop();

  ($top >= 50 )
  ? $nav.addClass('active')
  : $nav.removeClass('active');
});
```

해설

스크롤 다운은 배경과 폰트 색상 변경 / 스크롤 업은 변경 상태를 유지하되 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 최초 상태로 변경
scrollTop()는 제이쿼리에서 스크롤의 위치를 가져올 때 사용되는 메서드입니다. 이를 사용하면 자바스크립트 구현 방법 1과 동일한 방식으로 구현할 수 있습니다.

q5. Jquery - 스크롤을 다시 올릴 경우 곧바로 배경/폰트 색상을 이전 상태로 변경

A)

```
$(window).on('mousewheel DOMMouseScroll', function(e) {
  const delta = e.originalEvent.wheelDelta
  ? e.originalEvent.wheelDelta
  : -e.originalEvent.detail;

  (delta < 0)
  ? $nav.addClass('active')
  : $nav.removeClass('active');
});
```

해설

스크롤 다운은 배경과 폰트 색상 변경 / 스크롤 업은 이전 상태로 변경
제이쿼리에서도 마우스 휠 이벤트를 적용할 수 있습니다.

originalEvent : 제이쿼리의 이벤트 객체에서 지원하지 않는 브라우저 기능을 활용하고자 할 때 사용되는 이벤트 객체

Case19 : ScrollSpy - 대기업 S사 프론트엔드 개발자님의 답안

q1. Javascript - 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 이전 상태로 변경

A)

```
// main.js
const navEl = document.querySelector('nav');
const scrollThreshold = 50;
const debounceDelay = 10;

/**
 * 스크롤 이벤트가 연속적으로 발생하는 경우 성능저하를 방지하기 위해 `debounce` 유ти을 이용하여,
 * 마지막 스크롤 이벤트가 발생하고 일정시간(debounceDelay=10ms) 이상 시간이 지연됐을때 이벤트 핸들러를 1회만 실행
 */
const handleDebounceScrollEvent = debounce(() => {
    const windowScrollY = window.scrollY;
    if (windowScrollY > scrollThreshold) {
        /**
         * 스크롤된 값이 임계값 보다 큰 경우 `nav` 엘리먼트에 `active` 클래스 추가
         */
        navEl.classList.add('active');
    } else {
        /**
         * 스크롤된 값이 임계값 보다 큰 경우 `nav` 엘리먼트에 `active` 클래스 제거
         */
        navEl.classList.remove('active');
    }
}, debounceDelay);

window.addEventListener('scroll', handleDebounceScrollEvent);
```

```
// debounce.js
/**
 * 입력된 `func` 함수가 연속하여 호출되도 마지막으로 "함수가 호출된 시간 + `delay`" 시간이후에 1회만 실행
 *
 * @param {function} func
 * @param {number} delay 단위는 milliseconds 입니다.
 */
const debounce = (func, delay) => {
    /*
```

```

    * `setTimeout` 아이디를 저장
  */
let procId = null;
return (...args) => {
  if (procId) {
    /**
     * `procId`가 존재하면 실행되지 않도록 제거
    */
    window.clearTimeout(procId);
  }
  /**
   * `delay` 이후 해당 함수가 실행되도록 `setTimeout`에 태스크를 등록
  */
  procId = setTimeout(() => func(...args), delay);
}
};

}

```

해설

기본적인 접근 방법은 `window`

객체에 `scroll`

이벤트를 바인딩하고, 스크롤 이벤트가 발생할 때 `window.scrollY`

값을 확인하여 클래스를 적용하는 것입니다.

스크롤 이벤트는 스크롤이 진행되는 동안 짧은 시간 내 수십여 번 실행될 수 있기 때문에 성능 문제를 고려해야 하는데, 이러한 부수는 `debounce` 유ти을 구현하여 해결하였습니다.

 `debounce`

유티은 `throttle`

과 함께 개발 시 자주 사용되는 라이브러리입니다. 사용하는 방법과 원리를 익혀두시면 좋습니다.

@see <https://www.npmjs.com/package/debounce>

q2. Javascript - 스크롤을 다시 올릴 경우 곧바로 배경/폰트 색상을 이전 상태로 변경

A)

```

// main.js
const navEl = document.querySelector('nav');
const scrollThreshold = 50;
const throttleDelay = 100;

/**
 * 스크롤 이벤트가 연속적으로 발생하는 경우 성능 저하를 방지하기 위해 `debounce` 유ти을 이용하여,
 * 마지막 스크롤 이벤트가 발생하고 일정 시간(throttle=100ms) 이상 시간이 지연됐을 때 이벤트 핸들러를 1회만 실행
 */
let latestWindowScrollY = 0;
const handleThrottleScrollEvent = throttle(() => {
  const windowScrollY = window.scrollY;
  if (windowScrollY > latestWindowScrollY + scrollThreshold) {
    // 배경 색상 및 폰트 색상 변경 로직
  }
  latestWindowScrollY = windowScrollY;
});
window.addEventListener('scroll', handleThrottleScrollEvent);

```

```

if (latestWindowScrollY - windowScrollY > 0) {
    navEl.classList.remove('active');
} else {
    navEl.classList.add('active');
}
latestWindowScrollY = windowScrollY;
}, throttleDelay);

window.addEventListener('scroll', handleThrottleScrollEvent);

```

```

// debounce.js
/**
 * 입력된 `func` 함수가 연속하여 호출되도 마지막으로 "함수가 호출된 시간 + `delay`" 시간이후에 1회만 실행
 *
 * @param {function} func
 * @param {number} delay 단위는 milliseconds 입니다.
 */
const debounce = (func, delay) => {
    /**
     * `setTimeout` 아이디를 저장
     */
    let procId = null;
    return (...args) => {
        if (procId) {
            /**
             * `procId`가 존재하면 실행되지 않도록 제거
             */
            window.clearTimeout(procId);
        }
        /**
         * `delay` 0이후 해당 함수가 실행되도록 `setTimeout`에 태스크를 등록
         */
        procId = setTimeout(() => func(...args), delay);
    }
};

```

해설

기본적인 접근 방법은 `window`

객체에 `scroll`

이벤트를 바인딩하고, 스크롤 이벤트가 발생할 때 가장 마지막으로 실행된 스크롤 이벤트에서 `window.scrollY`

값을 `latestWindowScrollY`

변수에 저장하고

`latestWindowScrollY`

변수에서 `window.scrollY`

값을 뺀 결과 값이 0보다 큰 경우 `active`

클래스를 부여, 그렇지 않은 경우 제거하는 원리입니다.

스크롤 이벤트는 스크롤이 진행되는 동안 짧은 시간 내 수십 번 실행될 수 있기 때문에 성능 문제를 고려해야 하는데, 이러한 부수는 `throttle` 유틸을 구현하여 해결하였습니다.

이전에 구현했던 `debounce`

와 다른 점은, `throttle`

이벤트는 호출되는 즉시 실행이 된다는 것 입니다. `debounce`

와 `throttle`

의 차이를 잘 알아두었다가 활용하면 좋습니다.

✨ throttle

유틸은 debounce

과 함께 개발시 자주 사용되는 라이브러리입니다. 사용하는 방법과 원리를 익혀두시면 좋습니다.

@see <https://www.npmjs.com/package/lodash.throttle>

q3. Javascript - 자바스크립트에서 제공하는 마우스 휠 이벤트 동작 감지 기능을 사용해서 구현

A)

```
// main.js
const scrollThreshold = 50;
const debounceDelay = 10;

/**
 * q1 문제와 동일한 방법으로 구현, Jquery에서 지원하는 함수를 응용하여 DOM을 제어
 */
const handleThrottleScrollEvent = debounce(() => {
    const windowScrollY = window.scrollY;
    if (scrollThreshold < windowScrollY) {
        $nav.addClass('active');
    } else {
        $nav.removeClass('active');
    }
    latestWindowScrollY = windowScrollY;
}, debounceDelay);

/**
 * [!] `mousewheel` 이벤트를 사용하는 경우 (마우스가 아닌) 키보드 또는 스크린리더등으로
 * 스크롤을 제어하는 경우 이벤트를 트리거 할 수 없음
 */
$(window).on('scroll', handleThrottleScrollEvent);
```

```
// debounce.js
/**
 * 입력된 `func` 함수가 연속하여 호출되도 마지막으로 "함수가 호출된 시간 + `delay`" 시간이후에 1회만 실행
 *
 * @param {function} func
 * @param {number} delay 단위는 milliseconds 입니다.
 */
const debounce = (func, delay) => {
    /**
     * `setTimeout` 아이디를 저장
     */
    let procId = null;
    return (...args) => {
        if (procId) {
```

```

    /**
     * `procId`가 존재하면 실행되지 않도록 제거
     */
    window.clearTimeout(procId);
}
/**
 * `delay` 이후 해당 함수가 실행되도록 `setTimeout`에 태스크를 등록
 */
procId = setTimeout(() => func(...args), delay);
}
};


```

해설

기본적인 접근 방법은 q1
 문제와 동일합니다. scroll
 이벤트와 mousewheel
 이벤트 차이에 대해 알아두시면 도움이 될 것 같습니다.
 [!] mousewheel
 이벤트를 사용하는 경우 (마우스가 아닌) 키보드 또는 스크린리더등으로 스크롤을 제어하는 경우 이벤트를 트리거 할 수 없습니다. 주의하세요!

q4. Jquery - 스크롤을 다시 올릴 경우 변경된 상태를 유지하다가 더 이상 올릴 수 없을 때(최상단에 스크롤이 위치할 때) 이전 상태로 변경

A)

```

//main.js
const scrollThreshold = 50;
const throttleDelay = 250;
const debounceDelay = 100;

/**
 * q2 문제와 동일한 방법으로 구현, Jquery에서 지원하는 함수를 응용하여 DOM을 제어
 */
let latestWindowScrollY = 0;

const handleScrollEvent = () => {
    const windowScrollY = window.scrollY;
    if (latestWindowScrollY - windowScrollY > 0) {
        $nav.addClass('active');
    } else {
        $nav.removeClass('active');
    }
    latestWindowScrollY = windowScrollY;
};

const handleThrottleScrollEvent = throttle(handleScrollEvent, throttleDelay);

/**
 * `throttle`, `debounce` 모두 이용하여 이벤트를 바인딩 하였는데,
 * - 유저가 너무 빠른속도로 스크롤 방향을 바꾸는 경우 `throttleDelay`에 의해 중간과정은 생략될 수 있으나 성능은 유지할 수 있음.

```

```
 */
$(window).on('scroll', handleThrottleScrollEvent)
```

해설

기본적인 접근방법은 `q2` 와 동일하나, `throttle` 의 딜레이를 늘려 함수가 실행되는 횟수를 줄여 성능을 향상시켰습니다.

결론

스크롤 이벤트에 대한 기본적인 원리를 알면 구현이 가능하나, `debounce`, `throttle` 유틸을 이해하고 활용하면 좋은 성능의 코드를 만들 수 있습니다.

Case20 : List

케이스 주제

Q. 서버 통신으로 가져온 Post 정보를 화면에 리스트 방식으로 적용하세요.

기능 요구사항

GET 방식으로 Post 정보를 가져옵니다.

가져온 정보를 순차적으로 화면에 적용시킵니다.

기능 작동 이미지

<적용 전>

(빈 화면)

<적용 후>



문제

q1. javaScript

1. fetch 함수를 사용하여 해당 기능을 구현하세요.
2. Fetch Refactoring : 콜백함수를 사용하여 통신 영역, 템플릿 영역을 분할하여 작성하시오

q2. javaScript

1. axios 라이브러리를 사용하여 해당 기능을 구현하세요.
2. Axios Refactoring

q3. javaScript - async/await와 콜백 함수를 사용하여 해당 기능을 구현하세요.
q4. jQuery로 해당 기능을 구현하세요.

주요 학습 키워드

Fetch, Axios, Ajax 기본 사용법과 함께 async / await를 사용한 비동기 방식과 함께 역할별로 코드를 리팩토링 하는 방법을 익하게 됩니다.

작성해주셔야 하는 question 파일경로

q1

```
./question/1. js-fetch/main.js
```

q2

```
./question/2. js-axios/main.js
```

q3

```
./question/3. js-async-await/main.js
```

q4

```
./question/4. jq/main.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/1. js-fetch`

index.html 열기

q2

경로 `./question/2. js-axios`

index.html 열기

q3

경로 ./question/3. js-async-await

index.html 열기

q4

경로 ./question/4. jq

index.html 열기

Case5 : List - 출제자 해설

q1. Javascript - fetch

A)

Fetch : <https://mzl.la/3qHZmlg>

브라우저에 내장된 함수로 외부 라이브러리 의존 없이 HTTP 통신이 가능합니다. 단, 익스플로러는 지원하지 않습니다.

첫 번째 인수는 API URL, 두 번째 인수는 옵션 객체를 받습니다.

Promise 타입의 객체를 반환합니다. API 호출에 성공하면 resolve, 실패하면 reject 합니다.

```
const ul = document.querySelector('ul');
const API = 'https://jsonplaceholder.typicode.com/posts';
// Fetch
fetch(API)
  .then(res => res.json())
  .then(data => {
    data.forEach(item => {
      const li = `<li>
                    <h2>${item.title}</h2>
                    <p>${item.body}</p>
                  </li>`;
      ul.insertAdjacentHTML("beforeend", li);
    })
  })
  .catch(e => console.error(e))
```

해설

보통 REST API는 JSON 형태로 응답하며, response 객체는 json()를 제공합니다.

json()를 사용하면 JSON 포맷의 내용물을 자바스크립트 객체로 변환할 수 있습니다.

insertAdjacentHTML() : <https://developer.mozilla.org/ko/docs/Web/API/Element/insertAdjacentHTML>

특정 영역의 DOM 트리 안에, 원하는 위치에, 원하는 내용물을 삽입할 때 사용합니다.

beforeend는 특정 요소 내부에서 뒤에서부터 삽입하겠다는 의미입니다.

옵션 객체 예시

```
fetch(API, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  data: JSON.stringify({
    title: "Hello",
    body: "Nice to meet you",
    userId: 10,
  }),
}).then((res) => console.log(res))
```

Fetch Refactoring

콜백 함수를 사용하여 통신 영역, 템플릿 영역을 분할하여 작성하면, 코드 관리가 용이해집니다.

```
// 응답 상태 체크
const checkStatusAndParse = res => {
  if(!res.ok) throw new Error(`Status Code Error: ${res.status}`);
  return res.json();
}

// 화면에 정보 출력
const printPosts = data => {
  data.forEach(item => {
    const li = `<li>
      <h2>${item.title}</h2>
      <p>${item.body}</p>
    </li>`;
    ul.insertAdjacentHTML("beforeend", li);
  })
}

// API
const fetchPosts = (url) => {
  return fetch(url);
}

fetchPosts(API)
```

```
.then(checkStatusAndParse)
.then(printPosts)
.catch(e => console.log(e))
```

q2. Javascript - axios

A)

Axios : <https://github.com/axios/axios>

HTTP 통신을 위한 자바스크립트 라이브러리로 대부분의 브라우저에서 사용이 가능합니다.

```
const ul = document.querySelector('ul');
const API = 'https://jsonplaceholder.typicode.com/posts';

// Axios
axios.get(API)
.then(res => {
  const {data} = res;
  data.forEach(item => {
    const li = `<li>
      <h2>${item.title}</h2>
      <p>${item.body}</p>
    </li>`;
    ul.insertAdjacentHTML("beforeend", li);
  })
})
.catch(e => {
  console.error(e);
})
```

Axios Refactoring

```
const printPosts = res => {
  const {data} = res;
  data.forEach(item => {
    const li = `<li>
      <h2>${item.title}</h2>
      <p>${item.body}</p>
    </li>`;
  })
}
```

```

        ul.insertAdjacentHTML("beforeend", li);
    }

const fetchPosts = (url) => {
    return axios.get(url);
}

fetchPosts(API)
    .then(printPosts)
    .catch(e => console.log(e))

```

q3. Javascript - async/await와 콜백 함수

A)

async/await와 콜백 함수를 사용하면 비동기 처리를 수월하게 진행할 수 있습니다.
try ~ catch 문으로 오류 상황을 효율적으로 관리할 수 있습니다.

```

const printPosts = res => {
    const {data} = res;
    data.forEach(item => {
        const li = `<li>
                    <h2>${item.title}</h2>
                    <p>${item.body}</p>
                </li>`;
        ul.insertAdjacentHTML("beforeend", li);
    })
}

const fetchPosts = async () => {
    return await axios.get(API);
}

const render = async (callApi, callTemplate) => {
    const res = await callApi();
    callTemplate(res);
}

(async () => {
    try {
        await render(fetchPosts, printPosts);
    } catch (e) {
        console.log(e);
    }
})

```

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

4. Jquery로 기능 구현

A)

```
beforeSend : 요청 전에 호출  
Success : 요청이 성공했을 경우 호출  
Error : 요청이 실패했을 경우 호출  
Complete : 성공, 실패 여부에 상관없이 무조건 호출되는 함수
```

```
const $ul = $('ul');  
const API = 'https://jsonplaceholder.typicode.com/posts';  
  
// Ajax  
$.ajax({  
    url: API,  
    type: 'GET',  
    timeout : 1000,  
    beforeSend : function() {  
        console.log('요청 준비 중입니다. ');  
    },  
    success: function (data) {  
        console.log("요청이 성공했습니다.");  
        data.forEach(item => {  
            const li = `<li>  
                <h2>${item.title}</h2>  
                <p>${item.body}</p>  
            </li>`;  
            $ul.append(li);  
        })  
    },  
    error: function (error) {  
        console.log("요청이 실패했습니다.");  
        console.error(error);  
    },  
    complete : function() {  
        console.log('요청이 완료되었습니다. ')  
    }  
});
```

beforeSend, complete의 경우 Loading 상태를 처리할 때 주로 활용됩니다.

Case20 : List - 대기업 S사 프론트엔드 개발장님의 답안

문제 1 ~ 4 공통 코드 조각

```
...
<body>
  <ul id="list">
    <script type="text/script" id="template">
      <li>
        <h2><%=title%></h2>
        <p><%=description%></p>
      </li>
    </script>
  </ul>
  ...
</body>
...
```

해설

```
<script type="text/script">
태그내 포함된 태그 내용은 브라우저가 화면에 랜더링 하지 않습니다.
위 script
내 태그정보는 JS에서 템플릿 정보로 활용합니다(아래 JS 코드를 확인해 주세요)
```

```
/**
 * 템플릿 유틀
 * @param {*} htmlString
 */
const makeTemplate = (htmlString) => {
  return (obj) => {
    let template = htmlString;
    Object.keys(obj).forEach((key) => {
      template = template.replace(RegExp(`<%= ${key} %>`, "g"), obj[key]);
    })
    return template.replace(/<=%=\w+?%>/g, '');
  }
}
const template = makeTemplate(document.body.querySelector('#template').textContent);
```

해설

```
lodash.template
코드와 유사한 HTMLString을 이용하여 템플릿을 생성하는 코드를 구현해 보았습니다.
입력받은 HTMLString, Object 정보를 이용하여 템플릿을 생성합니다.
```

참고자료

q1. Javascript - fetch

A)

```
// solution/2.other_1/1. js-fetch/main.js
/**
 * 가급적 셀렉터 대상에게는 class or id 값을 명시적으로 지정해 주시는것이 좋습니다.
 * [>] 태그 이름을 이용해 셀렉트 하는 경우 의도하지 않는 대상이 선택될 수 있습니다.
 */
const listEl = document.querySelector('#list');
const API = 'https://jsonplaceholder.typicode.com/posts';

...
// 통신 상태 처리
const checkStatusAndParse = res => {
    /**
     * [!] `res.ok`는 서버 에러를 탐지하기 위한 값이 아닙니다.
     * HTTP status `200~299` 응답이 오는경우 `true` 값이 전달되며, `30x` redirection 응답이 오는 경우 `false` 값이 전달됩니다.
     * 조회 실패, 서버에러등 `40x, 50x` 처리를 위해서는 `catch` 블록을 사용해 주세요
     * @see https://ko.javascript.info/fetch
     */
    if (res.ok) { // `res.ok` 대신 `res.status < 300` 값으로 동일하게 확인할 수 있습니다.
        return res.json();
    } else {
        throw Error(`예상하지 못한 HTTP Status(${res.status}) 응답입니다.`);
    }
}

// 템플릿
const printPosts = data => {
    const listHtml = data.map(({ title, body: description }) => {
        return template({ title, description });
    })
    listEl.innerHTML = listHtml;
}

// HTTP 통신
const fetchPosts = (url) => {
    return fetch(url);
}

fetchPosts(API)
    .then(checkStatusAndParse)
    .then(printPosts)
    .catch(error => {
        alert(error.message);
    })
}
```

해설

```
fetch
함수의 res.ok
의미에 대해 잘 알아두면 좋을것 같습니다.
```

```
HTTP status 200~299  
응답이 오는 경우 true  
값이 전달되며, 30x  
 redirection 응답이 오는 경우 false  
값이 전달됩니다.  
조회 실패, 서버에러등 40x, 50x  
처리를 위해서는 catch  
블록을 사용해 주세요
```

참고자료

<https://ko.javascript.info/fetch>

q2. Javascript - axios

A)

```
/**  
 * 가급적 셀렉터 대상에게는 class or id 값을 명시적으로 지정해 주시는것이 좋습니다.  
 * [>] 태그 이름을 이용해 셀렉트 하는 경우 의도하지 않는 대상이 선택될 수 있습니다.  
 */  
const listEl = document.querySelector('#list');  
const API = 'https://jsonplaceholder.typicode.com/posts';  
  
...  
// solution/2.other_1/2. js-axios/main.js  
// 통신 상태 처리  
const checkStatusAndParse = res => {  
    /**  
     * `Fetch`에서 사용된 `res.ok` 값과 동일한 기능을 아래와 같이 구현하실 수 있습니다.  
     */  
    if (res.status < 300) {  
        return res.data;  
    } else {  
        throw Error(`예상하지 못한 HTTP Status(${res.status}) 응답입니다.`);  
    }  
}  
  
// 템플릿  
const printPosts = data => {  
    const listHtml = data.map(({ title, body: description }) => {  
        return template({ title, description });  
    })  
    listEl.innerHTML = listHtml;  
}  
  
// HTTP 통신  
const fetchPosts = (url) => {  
    return axios.get(url);  
}  
  
fetchPosts(API)  
    .then(checkStatusAndParse)  
    .then(printPosts)  
    .catch(error => {
```

```
        alert(error.message);
    })
}
```

해설

문제 1번과 다른점은 fetch 대신 axios 라이브러리를 사용한 것입니다. 다른 라이브러리를 사용했더라도 동작은 완전히 같습니다!

참고자료

<https://ko.javascript.info/fetch>
<https://github.com/axios/axios#response-schema>

q3. Javascript - async/await와 콜백 함수

A)

```
// solution/2.other_1/3. js-async-await/main.js
/**
 * 가급적 셀렉터 대상에게는 class or id 값을 명시적으로 지정해 주시는것이 좋습니다.
 * [>] 태그 이름을 이용해 셀렉트 하는 경우 의도하지 않는 대상이 선택될 수 있습니다.
 */
const listEl = document.querySelector('#list');
const API = 'https://jsonplaceholder.typicode.com/posts';

...
// 템플릿
const printPosts = data => {
    const listHtml = data.map(({ title, body: description }) => {
        return template({ title, description });
    })
    listEl.innerHTML = listHtml;
}

// HTTP 통신
const fetchPosts = () => {
    return axios.get(API);
}

/**
 * [!] 인자로 전달되는 함수 이름을 헷갈리지 않도록 임의로 변경해 주었습니다. 😊
 */
const render = async (asyncFetchPostsFn, printPostsFn) => {
    try {
        const res = await asyncFetchPostsFn();
        /**
         * checkStatusAndParse 함수에서 체크하던 로직
         */
        const { status, data } = res;
        if (status > 300) {
            throw Error(`예상하지 못한 HTTP Status(${res.status}) 응답입니다.`);
        }
        /**
         */
    }
}
```

```

        * 데이터를 화면에 그림
    */
    printPostsFn(data);
} catch (e) {
    alert(e.message);
}
}

(async () => {
    await render(fetchPosts, printPosts);
})();

```

해설

문제 1,2번과 동작은 완전히 같으며 `async/await` 을 이용하여 처리하였습니다.(코드내 변수 네이밍을 약간 변경하였는데 이부분은 꼭 정답이 있는것은 아닙니다.)

참고자료

<https://ko.javascript.info/fetch>
<https://github.com/axios/axios#response-schema>

4. Jquery로 기능 구현

A)

```

/**
 * 가급적 셀렉터 대상에게는 class or id 값을 명시적으로 지정해 주시는것이 좋습니다.
 * [>] 태그 이름을 이용해 셀렉트 하는 경우 의도하지 않는 대상이 선택될 수 있습니다.
 */
const $list = $('#list');
const API = 'https://jsonplaceholder.typicode.com/posts';

...

// Ajax
$.ajax(API, {
    method: 'GET',
    beforeSend: (_xhr, _opts) => {
        alert('요청 보내기 전입니다!');
    },
    success: function (data, _, res) {
        const { status } = res;
        if (status > 300) {
            /**
             * 기존 `checkStatusAndParse` 기능을 구현
             * [>] 여러 콜백을 강제로 호출(더 좋은 방법이 있다면 알려주세요!)
             */
            this.error(Error(`예상하지 못한 HTTP Status(${res.status}) 응답입니다.`));
        }
        /**
         * 템플릿을 이용한 HTML 생성
         */
        const listHtml = data.map(({ title, body: description }) => {
            return template({ title, description });
        })
        $list.html(listHtml);
    }
});

```

```
        }
        /**
         * 화면에 목록을 그림
         */
        $list.html(listHtml);
    },
    error: (e) => {
        alert(e.message);
},
complete: () => {
    alert('요청에 대한 응답 처리가 완료 되었습니다!');
},
})
})
```

해설

문제 1, 2, 3번과 동작은 완전히 같으며 jQuery에서 지원하는 `$.ajax` 유ти를 사용하였습니다.

참고자료

<https://api.jquery.com/jquery.ajax/>

결론

다른 라이브러리라 하더라도 구현 원리만 잘 이해하고 있다면 같은동작을 만들어낼 수 있습니다. 프로젝트에서 이미 사용되고 있거나, 가장 잘 활용할 수 있는 라이브러리를 사용하시면 좋습니다.

`XMLHttpRequest` 객체에 대해서도 잘 알아두시면 좋습니다(`fetch`가 나오기 전까지 모든 XHR 통신은 이 객체를 통해 구현되었습니다.)

<https://developer.mozilla.org/ko/docs/Web/API/XMLHttpRequest>

Case 21 : Modal Window 2

케이스 주제

Q. 이미지를 클릭하면 그 이미지 정보를 담고 있는 모달창이 등장하고, 레이어를 클릭하면 모달창이 사라지는 기능을 만드세요.

기능 요구사항

Unsplash의 List photo API를 사용해서 이미지 정보를 리스트로 화면에 띄우고,
특정 이미지를 클릭했을 때, 그 이미지에 대한 설명이 기재된 모달창을 띄우세요.
이미지에 대한 설명은 해당 이미지의 데이터 속성 값으로 리스트 안에 미리 넣어두는 형태로 처리하시오.
모달창을 제외한 외부 여백 공간 클릭 시 모달창이 사라지도록 하세요.

기능 작동 이미지



문제

JavaScript로 해당 기능을 구현하시오.

주요 학습 키워드

객체 형태로 데이터 속성값이 정보를 저장하여 이를 가져오는 방법을 학습합니다.
레이어 팝업을 처리하는 방법을 익히게 됩니다.

작성해주셔야 하는 question 파일경로

q1

./question/1.js/main.js

실행 방법

q1

경로 ./question/1.js

index.html 실행

Case21 : ModalWindow2 - 출제자 해설

JavaScript로 해당 기능을 구현하시오.

A)

Unspalsh API : <https://unsplash.com/documentation#list-photos>

```
const accessKey = "hPyF-tz9tHnxeaoTwb7q0GTw10Wxwr85cD63lk7d7UE";
const imageLists = document.querySelector('.image-lists');
const modal = document.querySelector('.custom-modal');
const thumbnail = modal.querySelector('.thumbnail');
const profileImg = modal.querySelector('.profile-image');
const userName = modal.querySelector('.name');
const insta = modal.querySelector('.insta');
```

```
// API 통신
const imageApi = async () => {
  const res = await axios.get('https://api.unsplash.com/photos/' , {
    params: {
      client_id: accessKey,
    }
  });
  return res;
};
```

```
// 콜백 함수
const render = async (callApi, callTemplate) => {
  const res = await callApi();
  callTemplate(res);
}
```

```
// 이미지 템플릿 및 클릭 이벤트 추가
const imageTemplate = (res) => {
```

```

const images = res.data;
for(const image of images) {
  const { urls, alt_description, user } = image;
  const { name, instagram_username, profile_image: {medium} } = user;
  const li = document.createElement('li');

  li.classList.add('image-list');
  li.dataset.user = JSON.stringify({name, instagram_username, medium});
  li.dataset.image = `${urls.regular}`;
  li.innerHTML = ``;
  imageLists.appendChild(li);

  li.addEventListener('click', function() {
    const obj = {
      image: this.dataset.image,
      ...JSON.parse(this.dataset.user)
    };
    modal.classList.add('active');
    thumbnail.src = obj.image;
    profileImg.src = obj.medium;
    userName.textContent = obj.name;
    insta.textContent = obj.instagram_username;
  })
}
}

```

JSON.stringify(): 자바스크립트 값을 JSON 객체를 String 객체로 변환
 JSON.parse(): String 객체를 JSON 객체로 변환

```

// 이미지 표기 및 modal 이벤트 추가
(async () => {
  await render(imageApi, imageTemplate);
  modal.addEventListener('click', function (e) {
    const target = e.target;
    const isLayer = target.classList.contains('layer');
    isLayer && modal.classList.remove('active');
  });
})();

```

Case 22 : Scroll Indicator

케이스 주제

Q. 요구사항 : 스크롤 시 현재 남의 컨텐츠의 분량을 화면에 표기해 주세요. (e.g
<https://sports.v.daum.net/v/20210123112225915>)

기능 요구사항

스크롤을 내리면 상단에 현재 스크롤이 어느 정도 내려갔는지를 나타내는 상태 표시바를 만드시오.
스크롤이 끝까지 내려가면 Indicator도 끝까지 이동하고, 스크롤을 다시 상단으로 올리면 Indicator도 다시 뒤로 돌아가게 만드시오.

기능 작동 이미지



문제

JavaScript로 해당 기능을 구현하시오.

1. scrollBar의 width값을 변경하는 방식으로 indicator를 적용하시오.
2. 스크롤 시 translateX를 사용하여 scrollBar 위치를 변경하시오.

주요 학습 키워드

q1. 크로스 브라우징을 고려하여 현재 문서의 높이를 가져오기 q2. width 또는 translateX를 사용하여 남은 컨텐츠를 표기하는 방법 학습하기

작성해주셔야 하는 question 파일경로

q1

```
./question/1.js-width/main.js
```

q2

```
./question/2.js-translateX/main.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/1.js-width`

index.html 실행

q2

경로 `./question/2.js-translateX`

index.html 실행

Case22 : ScrollIndicator - 출제자 해설

q1. JavaScript로 해당 기능을 구현하시오 - scrollBar의 width값 변경

A)

```
const scrollBar = document.getElementById('scroll-bar');

window.addEventListener('scroll', function () {
    const scrollTop = document.body.scrollTop || document.documentElement.scrollTop;
    const scrollHeight = document.body.scrollHeight || document.documentElement.scrollHeight;
    const clientHeight = document.body.clientHeight || document.documentElement.clientHeight;

    // contentHeight : 눈에 보이지 않는 남은 범위
    const contentHeight = scrollHeight - clientHeight;
    const percent = (scrollTop / contentHeight) * 100;

    scrollBar.style.width = percent + '%';
})
```

document.documentElement.scrollHeight, document.body.scrollHeight : 전체 문서의 높이
document.documentElement.clientHeight, document.body.clientHeight : 현재 눈에 보이는 브라우저의 높이

q2. JavaScript로 해당 기능을 구현하시오 - translateX

A)

scrollBar의 width 값을 100%로 변경한 다음 최초 위치를 화면 왼쪽 영역 바깥으로 이동
스크롤 시 translateX를 사용하여 scrollBar 위치를 변경

```
const scrollBar = document.getElementById('scroll-bar');

window.addEventListener('scroll', function () {
  const scrollTop = document.body.scrollTop || document.documentElement.scrollTop;
  const scrollHeight = document.body.scrollHeight || document.documentElement.scrollHeight;
  const clientHeight = document.body.clientHeight || document.documentElement.clientHeight;

  const contentHeight = scrollHeight - clientHeight;
  const percent = (scrollTop / contentHeight) * 100;

  scrollBar.style.transition = 'transform 0.3s ease-out';
  scrollBar.style.transform = `translateX(-${100 - percent}%)`;
})
```

```
#scroll-bar {
  position: absolute;
  bottom: 0;
  left: 0;
  height: 4px;
  background-color: blue;

  width: 100%;
  transform: translateX(-100%);
}
```

Case 23 : Loading

케이스 주제

Q. 요구사항 : 회원가입, 로그인, 게시물 작성 등 서버에 데이터를 보낼 경우 전송중인 상태를 화면에 표기합니다.

기능 요구사항

서버에 데이터를 보내는 경우, 데이터를 전송중인 상태를 나타내기 위해 화면에 스피너 UI를 띄우세요.
통신이 완료되었을 시, 완료된 상태를 나타내는 Alert 창을 띄우세요.

기능 작동 이미지



문제

JavaScript로 해당 기능을 구현하시오.

주요 학습 키워드

각 역할별로 콜백함수를 생성하여 코드를 관리하는 방법을 익히고, 통신 성공 / 실패를 효율적으로 관리하는 방법을 익히게 됩니다.

작성해주셔야 하는 question 파일경로

q1

```
./question/1.js/main.js
```

실행 방법 및 의존성 모듈 설치

q1

경로 `./question/1.js`

[index.html 열기](#)

Case23 : Loading - 출제자 해설

q1. JavaScript로 해당 기능을 구현하시오

A)

```
// API 기본 설정
const todoApi = axios.create({
  baseURL: 'https://jsonplaceholder.typicode.com',
  headers: {
    'Content-type': 'application/json; charset=UTF-8',
  },
})

// Post API 설정
const postTodoApi = (data) => {
  return todoApi({
    method: 'post',
    url: '/posts',
    data: data
  })
}

// 응답 상태 처리
const postTodo = async (data, callApi, callback) => {
  try {
    const res = await callApi(data);
    callback(true, res.data, '등록했습니다.');
  } catch (e) {
    callback(false, null, '등록을 실패했습니다.');
    console.error(e);
  }
}

const form = document.querySelector('form');
const title = document.querySelector('#title');
const body = document.querySelector('#body');
const user = document.querySelector('#user');
const loading = document.querySelector('#loading');
```

axios.create() : Axios 기본 설정을 세팅하는 메서드.

```
todoApi.interceptors.request.use(
  function (config) {
    // 요청을 보내기 전에 수행할 일
    loading.style.display = 'block';
    return config;
  },
  function (error) {
    // 오류 요청을 보내기 전에 수행할 일
    return Promise.reject(error);
  }
)

form.addEventListener('submit', async function (e) {
  e.preventDefault();
  const data = {
    title: title.value,
    body: body.value,
    userId: user.value
  }

  await postTodo(data, postTodoApi, result);
});

const result = (isSuccess, data, message) => {
  loading.style.display = 'none';
  if(isSuccess) {
    alert(message);
  } else {
    alert(message);
  }
}
```

검색리스트 Dropdown 구현 및 선택된 이미지 표시하기

문제 1

풀이

```
<!-- solution/2.other_1/1. js/index.html -->
...
<main style="padding-top: 50px;">
    <div class="container">
        <div class="row">
            <div class="col">
                <div class="dropdown">
                    <div class="input-group mb-3">
                        <input type="text" class="form-control" placeholder="Search">
                    </div>
                    <ul class="dropdown-menu">
                    </ul>
                    <script type="text/template" id="dropdown-item-template">
                        <li class="dropdown-item" data-id="<%-id%>">
                            ">
                            <span>
                                <%- description %>
                            </span>
                        </li>
                    </script>
                </div>
            </div>
        <div class="row">
            <div class="col">
                <div class="p-t-50">
                    <img class="image-info" src="" alt="">
                </div>
            </div>
        </div>
    </div>
</main>
<script src="axios/axios.js"></script>
<script src="lodash.js"></script>
<script src="main.js"></script>
...

```

```
// solution/2.other_1/1. js/main.js
const accessKey = "hPyF-tz9tHnxiaoTwb7q0GTw10Wxwr85cD63lk7d7UE";
const inputEl = document.querySelector('input');
const dropdownMenuEl = document.querySelector('.dropdown-menu');
const imageInfoEl = document.querySelector('.image-info');
const template = _.template(document.querySelector('#dropdown-item-template').textContent);
```

```

/**
 * 검색어가 입력됐을때 이벤트
 */
const onInputDebounceQuery = _.debounce(async (e) => {
    const { value: query } = e.target;
    try {
        /**
         * @see https://github.com/axios/axios#response-schema
         */
        const { data } = await axios.get('https://api.unsplash.com/search/photos/' , {
            params: {
                client_id: accessKey,
                query
            }
        });
    }

    const { results } = data;

    /**
     * 조회 결과가 존재하지 않는 경우 항목을 표시하지 않음
     */
    if (!results?.length) {
        dropdownMenuEl.classList.remove('show');
        return;
    }

    /**
     * 드롭다운 메뉴에 항목을 추가
     */
    dropdownMenuEl.innerHTML = results.map((item) => {
        const { id, alt_description: description } = item;
        const { regular: url } = item.urls;
        /**
         * `lodash` 템플릿을 활용하여 조회된 데이터 항목을 템플릿으로 변환
         */
        return template({
            id, description, url
        });
    }).join(' ');
    dropdownMenuEl.classList.add('show');
} catch (e) {
    alert(e.isAxiosError ? '데이터 조회중 에러 발생!' : '데이터 처리중 에러 발생!');
}
}, 250);

/**
 * 드롭다운 아이템 항목 선택시 이벤트
 * [>] 아이템 항목이 눌러졌을때의 이벤트를 `dropdownMenu` 엘리먼트에서 위임받아 처리
 */
const onClickDropdownItem = async (e) => {
    try {
        /**
         * 이벤트가 발생한 엘리먼트로부터 가장 인접한 `.dropdown-item` 엘리먼트를 찾음
         */
        const dropdownItemEl = e.target.closest('.dropdown-item');
        const id = dropdownItemEl.dataset.id;

        const { data } = await axios.get(`https://api.unsplash.com/photos/${id}` , {
            params: {
                client_id: accessKey,
            }
        });
    }
}

```

```

        const {
          urls,
          alt_description: description
        } = data
        const { regular: url } = urls;

        /**
         * 이미지 객체 정보 업데이트
         */
        imageInfoEl.src = url
        imageInfoEl.alt = description

        /**
         * 열려있는 검색결과 내용을 담고 검색어 입력창에 현재 이미지 내용을 입력
         */
        dropdownMenuEl.classList.remove('show');
        inputEl.value = description;
      } catch (e) {
        alert(e.isAxiosError ? '데이터 조회중 에러 발생!' : '데이터 처리중 에러 발생!');
      }
    }

    inputEl.addEventListener('keydown', onInputDebounceQuery);
    dropdownMenuEl.addEventListener('click', onClickDropdownItem);

```

해설

HTML 파일내 아이템 항목 템플릿을 생성하였습니다. 아래 `script` 태그로 감싼 태그 영역은 브라우저에서 랜더링되지 않고, 스크립트에서 이 문자열을 읽어 `lodash` 템플릿으로 활용합니다(자세한 내용은 참고자료를 확인해 주세요).

```

<script type="text/template" id="dropdown-item-template">
<li class="dropdown-item" data-id="<%-id%>">
  ">
  <span>
    <%- description %>
  </span>
</li>
</script>

```

HTML 엘리먼트를 탐색하여 담고있는 객체 네이밍은 다른 변수명과 구분될 수 있도록 `~El` 으로 변경 하였습니다.

`lodash`

에서 제공하는 `debounce`

함수를 이용하여 반복되는 입력이더라도 마지막 입력 후 500ms 경과후에 조회가되도록 구현하였습니다. `debounce`, `throttle`

함수의 개념과 차이를 알아 두시면 좋습니다.

기존 분리되어 구현되어있던 `fetch`

함수를 사용하지 않고, 이벤트 핸들러 내부에서 모든 구현을 처리하였습니다. 재사용될 가능성이 높거나 `fetch`에서 부가적인 처리가 많아진다면 분리하여 사용하는것도 좋은 방법입니다.

`async/await`

비동기 처리를 하실때에는 `try~catch`

를 사용하여 에러발생에 대한 처리를 해주시면 의도하지 않은 동작(예: 서버오류 등..)을 방지하거나 대응하실 수 있습니다.

`DropdownItem` 클릭시 모든 항목에 이벤트를 바인딩하지 않고, `DropdownMenuEl`에 이벤트를 위임하여 처리하였습니다. 이벤트 위임은 항목이 가변적으로 늘어나거나 줄어드는 경우 코드 누락으로 인한 Memory Leaks을 방지할 수 있고, 항목이 많아지는 경우 메모리 사용량을 줄이고 효율적으로 동작할 수 있게 합니다.

참고자료

<https://lodash.com/docs/4.17.15#template>
<https://lodash.com/docs/4.17.15#debounce>
<https://lodash.com/docs/4.17.15#throttle>
https://ui.toast.com/weekly-pick/ko_20160826

문제 2

풀이

```
<!-- solution/2.other_1/2. jq/index.html -->
<!-- 1번 문제풀이 내용과 동일합니다 -->
```

```
// solution/2.other_1/2. jq/main.js
const accessKey = "hPyF-tz9tHnxiaoTwb7q0GTw10Wxwr85cD63lk7d7UE";
const $input = $('input');
const $dropdownMenu = $('.dropdown-menu');
const $imageInfo = $('.image-info');
const template = _.template($('#dropdown-item-template').text());

/**
 * 검색어가 입력됐을때 이벤트
 */
const onInputDebounceQuery = _.debounce(async (e) => {
    const { value: query } = e.target;
    try {
        /**
         * @see https://github.com/axios/axios#response-schema
         */
        const requestUrl = `https://api.unsplash.com/search/photos/?query=${query}&client_id=${accessKey}`;
        const data = await $.ajax(requestUrl, {
            method: 'GET'
        });

        const { results } = data;

        /**
         * 조회 결과가 존재하지 않는 경우 항목을 표시하지 않음
         */
        if (!results?.length) {
            $dropdownMenu.removeClass('show');
            return;
        }

        /**
         * 드롭다운 메뉴에 항목을 추가
         */
        $dropdownMenu.html(
            results.map((item) => {
```

```

        const { id, alt_description: description } = item;
        const { regular: url } = item.urls;
        /**
         * `lodash` 템플릿을 활용하여 조회된 데이터 항목을 템플릿으로 변환
         */
        return template({
            id, description, url
        });
    }).join('')
);
$dropdownMenu.addClass('show');
} catch (e) {
    alert(e.status !== 200 ? '데이터 조회중 에러 발생!' : '데이터 처리중 에러 발생!');
}
}, 250);

/**
 * 드롭다운 아이템 항목 선택시 이벤트
 */
const onClickDropdownItem = async (e) => {
    try {
        const $dropdownItem = $(e.currentTarget);
        /**
         * @see https://api.jquery.com/data/
         */
        const id = $dropdownItem.data('id');

        const requestUrl = `https://api.unsplash.com/photos/${id}/?client_id=${accessKey}`;
        const data = await $.ajax(requestUrl, {
            method: 'GET'
        });
        const {
            urls,
            alt_description: description
        } = data;
        const { regular: url } = urls;

        /**
         * 이미지 객체 정보 업데이트
         * @see https://api.jquery.com/attr/
         */
        $imageInfo.attr('src', url);
        $imageInfo.attr('alt', description);

        /**
         * 열려있는 검색결과 내용을 닫고 검색어 입력창에 현재 이미지 내용을 입력
         * @see https://api.jquery.com/addclass/
         * @see https://api.jquery.com/removeclass/
         */
        $dropdownMenu.removeClass('show');
        /**
         * @see https://api.jquery.com/val/
         */
        $input.val(description);
    } catch (e) {
        alert(e.isAxiosError ? '데이터 조회중 에러 발생!' : '데이터 처리중 에러 발생!');
    }
}

$input.on('keydown', onInputDebounceQuery);
/**
 * [!] 아이템 항목이 눌러졌을때의 이벤트를 `dropdownMenu` 엘리먼트에서 위임받아 처리

```

```
 */
$dropdownMenu.on('click', '.dropdown-item', onClickDropdownItem);
```

해설

1번 문제풀이 내용과 대부분 동작은 비슷하지만 jQuery 라이브러리를 활용하여 DOM을 제어하고 ajax 유ти을 활용하여 코드가 더 간결해졌습니다.
jQuery를 이용할 경우 이벤트 위임을 간단하게 구현하실 수 있습니다(코드 내용을 참고해 주세요)
jQuery 내부적으로 XHR 처리할 수 있는 `$.ajax`
를 지원하여 `axios`
대신 사용하실수 있습니다.

참고자료

<https://api.jquery.com/data/>
<https://api.jquery.com/attr/>
<https://api.jquery.com/addclass/>
<https://api.jquery.com/removeclass/>
<https://api.jquery.com/val/>
<https://api.jquery.com/on/>
<https://api.jquery.com/html/>
<https://api.jquery.com/text/>

결론

jQuery, lodash 등과 같은 라이브러리를 사용하면 코드를 더 쉽고 빠르게 작성하실수 있습니다. 라이브러리를 사용하는것도 좋지만 어떻게 동작하는지 원리를 잘 알고 사용하시면 더 좋을것 같습니다.
사용하시는 라이브러리 API목록을 잘 살펴보시고 알아두시면 나중에 필요한 순간에 빠르게 코드를 작성하실수 있습니다. 오픈소스로 만들어진 라이브러리는 여러명에 의해 확인되고 검증된 기능을 제공하기 때문에 더 안정적이고 더 효율적일 수 있습니다.

Case24 : Search bar

기능 요구사항

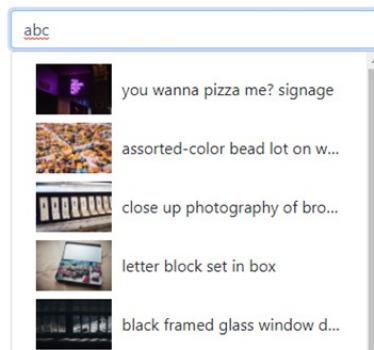
Q. 검색 리스트를 드랍다운 메뉴로 표기하고, 메뉴를 선택하면 선택한 이미지를 화면에 표기합니다.

1. 이미지 API : <https://unsplash.com/documentation#search-photos>
2. debounce : <https://lodash.com/docs/4.17.15#debounce>

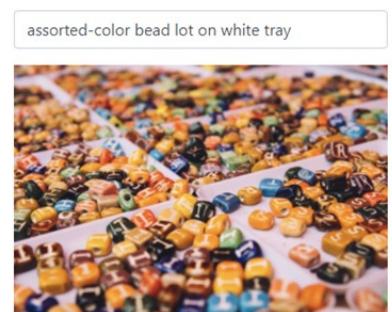
기능 작동 이미지

1. 검색 전

2. 검색어 입력 시



3. 특정 검색어 선택 시



문제

q1. 문제 상황에 대하여 JavaScript로 동작을 구현시킬 수 있는 코드를 작성해보세요

1. 검색 이미지 리스트 가져오기
2. 하나의 이미지 정보만 가져오기
3. 검색 드랍다운 표기
4. 검색 드랍다운 리스트 표기
5. 선택한 이미지 정보 표기

q2. 문제 상황에 대하여 jQuery로 동작을 구현시킬 수 있는 코드를 작성해보세요

1. JavaScript로 짠 코드를 jQuery로 전환
2. Data 속성 활용

주요 학습 키워드

검색어 입력에 따른 API 호출을 제어하는 방법을 학습하게 됩니다.

axios 라이브러리와 async, await를 이용한 API호출

lodash 라이브러리의 debounce()를 사용하여 불필요한 호출 줄이기

반복문을 통해 createElement, innerHTML, appendChild를 사용하여 리스트 표기

jQuery의 data- 속성 활용

작성해주셔야 하는 question 파일경로

q1

./question/1.js/main.js

q2

./question/2.jq/main.js

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/1.js

index.html 열기

q2

경로 ./question/2.jq

터미널

```
$ npm install  
$ npm start
```

Case25 : Combo Box

케이스 주제

드롭다운 리스트는 단순히 클릭하면 리스트가 나오는 형태뿐 아니라, input 태그로 값을 입력하고 input 우측의 버튼을 눌러 값 목록을 드롭다운 리스트로 볼 수 있는 형태로도 만들 수 있습니다. 이를 콤보 박스라고 합니다.

Q. 아래와 같은 스펙을 가진 콤보 박스를 만들어보세요.

기능 요구사항

하나의 input tag를 만든다.

이 input에 타이핑을 해서 값을 입력할 수 있다.

타이핑을 할때 엔터키를 치면 입력한 값을 드롭다운 리스트에 추가한다.

드롭다운 리스트는 안보이다가 input tag 우측의 버튼을 누르면 나온다.

(옵션) 엔터키를 쳐서 값을 드롭다운 리스트에 추가할때 입력이 되었다는 표시를 하는 notification을 만든다.

드롭다운 리스트의 가로 크기는 input + button의 가로이고 세로 크기는 임의의 숫자로 고정이다.

이때 드롭다운 리스트에 height 크기를 넘어갈 정도로 값이 차면 스크롤을 할 수 있도록 한다.

기능 작동 이미지

문제

q1. Javascript - 각각의 이벤트 리스트에 코드를 작성하여 위 기능이 동작되는 콤보 박스를 완성하시오.

주요 학습 키워드

CSS animation을 사용하여 항목이 추가 되었을 때, classList를 add하여 알림메세지 보여주기(CustomEvent 사용)
keyup 이벤트를 사용하여 입력한 항목을 배열에 추가 후, innerHTML을 사용하여 DOM에 동적으로 추가하기
setTimeout을 사용하여 일정시간 후 알림메세지를 자동으로 사라지게 하여 UX개선

작성해주셔야 하는 question 파일경로

q1

`./question/index.js`

실행 방법

q1

경로 `./question`
`index.html` 열기

Case25 : ComboBox - 출제자 해설

Javascript - 각각의 이벤트 리스트에 코드를 작성하여 위 기능이 동작되는 콤보 박스를 완성하시오.

A)

```
const toggleItemList = () => {
  const isVisible = $itemList.style.visibility === "visible";
  $itemList.style.visibility = isVisible ? "hidden" : "visible";
};

$inputTag.addEventListener("keyup", (event) => {
  const { value } = event.target;
  if (!value) {
    return;
  }

  if (event.key === "Enter") {
    items.push(value);
    document.dispatchEvent(new CustomEvent(ITEM_ADDED_EVENTNAME));
    $itemList.innerHTML = items.map((item) => `<li>${item}</li>`).join("");
  }
});

$arrowDown.addEventListener("click", () => {
  toggleItemList();
});

$itemList.addEventListener("click", (event) => {
  if (event.target.nodeName !== "LI") {
    return;
  }

  $inputTag.value = event.target.innerText;
  // 실무에선 변수로 활용
  $currentItem.textContent = `현재 아이템: ${event.target.innerText}`;
  toggleItemList();
});
```

```
document.addEventListener(ITEM_ADDED_EVENTNAME, () => {
  $notification.classList.add("Notification--show");
  $notification.classList.remove("Notification--hide");

  window.setTimeout(() => {
    $notification.classList.add("Notification--hide");
    $notification.classList.remove("Notification--show");
  }, 3000);
});
```

해설

- 콤보 박스 만들기라는 요구 사항을 더 작은 단위로 분해하면 값을 입력하고 드롭다운 리스트에 추가할 수 있는 input tag를 만드는 것과 그렇게 만들어진 값들을 보여줄 수 있는 드롭다운 리스트를 만드는 것으로 나눌 수 있습니다.
- input tag에 값을 입력하고 엔터키를 쳐서 드롭다운 리스트에 값을 추가해야 하므로 기본적으로 input tag에 event listener를 활용해야 합니다.

이때 값이 없는 경우를 처리해야 하고

엔터키가 눌렸을 때 드롭다운 리스트에 추가할 수 있도록 값의 배열에 값을 추가 합니다.

(option) 추가로, 값이 추가되었을 때 추가되었음을 알릴 수 있는 notification도 함께 있으면 UX에 도움이 됩니다.

추가된 값을 드롭다운 리스트에 보여주어야 하는데 여기엔 몇 가지 방법이 있습니다.

- 값을 추가했을 때 곧바로 드롭다운 리스트 dom에 렌더링을 해두고 input tag 우측의 화살표 버튼이 눌렸을 때 CSS를 이용해 보여주는 방법
- 값을 배열로 받아두었다가 화살표 버튼이 클릭되었을 때 dom에 렌더링 하는 방법

- 그 다음 화살표 버튼이 눌렸을 때 드롭다운 리스트를 보여주어야 합니다.

이때도 클릭을 감지해서 보여줘야 하기 때문에 화살표 버튼에 event listener를 활용해야 합니다.

드롭다운 리스트의 화면 표시 여부를 변수로 처리해도 되고

이렇게 상태를 관리하지 않고 단순히 CSS class를 toggle 하는 방법으로 처리해도 됩니다.

- 값들이 몇 개 추가되어 드롭다운 리스트를 열었을 때, 항목 하나하나를 클릭할 수 있어야 하고 클릭하면 현재 선택된 값으로 input tag가 채워져야 합니다.

이때 event listener는 모든 li tag에 각각 다는 것은 자원 낭비이므로 ul tag 하나에 붙여서 event delegation을 활용하는 것이 좋습니다.

클릭 이벤트로 li tag의 text 값을 받았으면 드롭다운 리스트를 다시 toggle 해주고 현재 선택된 값을 input tag에 표시해 줍니다.

실무에선 이 콤보 박스를 활용할때 선택된 값을 상태관리 대상으로 삼아서 다른 로직에 활용합니다.

Case26 : Radio Box

케이스 주제

라디오 박스는 여러 값들이 주어지고 그 중에서 단 하나를 선택할 수 있는 UI입니다.

기본적으로 type이 radio인 input tag를 사용해 만들며, 브라우저가 type이 radio인 input들 중 name 속성이 같은 것들은 하나만 선택되도록 강제하고 그에 따라 렌더링 합니다.

이 라디오 박스는 이렇게 input tag를 사용해도 되지만 다른 tag를 사용해 커스텀하게 작성할 수도 있습니다. 다만 이 경우엔 하나만 선택되고 그에 따라 렌더링이 될 수 있도록 추가적인 작업을 JavaScript로 해주어야 합니다.

Q. 아래와 같은 스펙을 가진 라디오 박스를 만들어보세요.

기능 요구사항

하나의 form tag를 만든다.

이 form tag 안에 연락 수단과 배송 수단을 선택할 수 있는 라디오 박스 UI를 최소 두개를 만든다.

form을 제출하면(type이 submit인 버튼을 누르면) 모든 수단이 선택되었는지 검사한다.

항목 하나라도 선택하지 않았으면 모든 항목에 대한 선택을 해야한다는 정보성 vUI를 만들어 보여준다.

검사를 무사히 통과했으면 현재 form이 최종적으로 어떤 데이터를 보내려는지 form UI 하단에 보여준다.

기능 작동 이미지

연락 수단을 선택해주세요!

Email Phone Email

배송 방법을 선택해주세요!

Ship Plane Car

확인

문제

q1. Javascript로 위 기능을 구현하시오.

주요 학습 키워드

submit event가 발생하면 formData를 이용해 value를 가져오고 배열로 만들어주기
Object.keys를 사용하여 폼을 모두 작성했는지 체크하기
입력한 정보를 innerHTML을 사용하여 DOM에 동적으로 추가하기

작성해주셔야 하는 question 파일경로

q1

`./question/index.js`

실행 방법

q1

경로 `./question/index.html` 열기

Case26 : RadioBox - 출제자 해설

Javascript - 각각의 이벤트 리스트에 코드를 작성하여 위 기능이 동작되는 콤보 박스를 완성하시오.

A)

```
const toggleItemList = () => {
  const isVisible = $itemList.style.visibility === "visible";
  $itemList.style.visibility = isVisible ? "hidden" : "visible";
};

$inputTag.addEventListener("keyup", (event) => {
  const { value } = event.target;
  if (!value) {
    return;
  }

  if (event.key === "Enter") {
    items.push(value);
    document.dispatchEvent(new CustomEvent(ITEM_ADDED_EVENTNAME));
    $itemList.innerHTML = items.map((item) => `<li>${item}</li>`).join("");
  }
});

$arrowDown.addEventListener("click", () => {
  toggleItemList();
});

$itemList.addEventListener("click", (event) => {
  if (event.target.nodeName !== "LI") {
    return;
  }

  $inputTag.value = event.target.innerText;
  // 실무에선 변수로 활용
  $currentItem.textContent = `현재 아이템: ${event.target.innerText}`;
  toggleItemList();
});
```

```
document.addEventListener(ITEM_ADDED_EVENTNAME, () => {
  $notification.classList.add("Notification--show");
  $notification.classList.remove("Notification--hide");

  window.setTimeout(() => {
    $notification.classList.add("Notification--hide");
    $notification.classList.remove("Notification--show");
  }, 3000);
});
```

해설

1. form tag를 활용해 라디오 박스를 만든다고 가정합니다.
2. form과 form이 다루는 데이터를 표현해줄 dom을 html 상에 만들어야 합니다.
3. form이 전체적으로 어떤 데이터를 다루는지(갯수, 종류 등) 알아야 검사 코드를 만들 수 있습니다.

이를 위해 이 정보를 나타내는 객체나 배열을 활용할 수 있습니다. 예제 코드에선 formMap이라는 객체를 만듭니다.

4. 확인 버튼이 클릭되면 서버로 데이터를 보내기 전에 데이터의 유효성 검사를 해야합니다.

이를 위해 form tag에 submit 이벤트를 수신하는 event handler를 등록해야 합니다.

이 핸들러 안에서 검사를 진행합니다.

이 검사를 할 때 위에서 만든 formMap이라는 객체를 활용합니다.

현업에서 검사 코드에는 모든 항목에 대한 입력이나 선택을 했는지, 입력 값의 타입이 정확한지, 입력 값이 요구사항의 조건을 만족하는지 등이 포함될 수 있습니다.

검사를 통과하지 못했다면 사용자가 모든 항목에 대해서 입력 및 선택을 할 수 있도록 안내하는 UI를 만들어 보여줍니다.
검사를 통과했다면 작성 완료된 form 데이터를 보여줍니다(현업에선 API call을 합니다).

5. 이때 form tag는 type이 submit인 버튼이 클릭되면 기본적으로 페이지 새로고침을 하는데, 이 동작이 서비스에서 불필요한 경우엔 이 동작을 하지 않을 필요가 있습니다. 이를 위해 핸들러에서 넘어온 event 객체의 preventDefault 함수를 실행해서 기본 동작을 멎춥니다.
6. 핸들러 내부에서 form이 가진 데이터를 확인하기 위해서 FormData 클래스를 활용합니다. 이때 이 클래스로 만든 인스턴스는 타입이 FormData입니다. iterator protocol이 FormData 내부에 정의되어 있기 때문에 for of loop 이용해 데이터 항목들을 순회할 수 있습니다. for loop를 사용하는 것이 꺼려진다면 Array의 from 함수를 이용해 배열로 만들어 쉽게 순회할 수도 있습니다.

Case27 : Check box

케이스 주제

Q. 아래와 같이 작동하는 체크 박스를 구현하시오.

체크 박스는 여러 값들이 주어지고 그 중에서 여러 값을 선택할 수 있는 UI입니다. 기본적으로 type이 checkbox인 input tag를 사용해 만들며 이 체크 박스는 이렇게 input tag를 사용해도 되지만 다른 tag를 사용해 커스텀하게 작성할 수도 있습니다. 여러개를 선택할 수 있고 그에 따라 여러 체크 박스들이 제대로 렌더링이 될 수 있도록 추가적인 작업을 JavaScript로 해주어야 합니다.

기능 요구사항

요구 사항은 아래와 같다.

1. 하나의 form tag를 만든다.
2. 이 form tag 안에 좋아하는 음식과 관심사를 여러개 선택할 수 있는 체크 박스 UI를 최소 두개를 만든다.
3. form을 제출하면(type이 submit인 버튼을 누르면) 좋아하는 음식과 관심사가 각각 최소 하나씩 선택되었는지 검사한다. 최소 하나씩 선택하지 않았으면 선택을 해야한다는 정보성 UI를 만들어 보여준다.
4. 검사를 무사히 통과했으면 현재 form이 최종적으로 어떤 데이터를 보내려는지 form UI 하단에 보여준다.

첫 화면

좋아하는 음식

Pizza Steak Noodle

관심사

Cooking Bike Swimming Piano Book

아무것도 선택하지 않고 submit 했을때

좋아하는 음식

Pizza Steak Noodle

관심사

Cooking Bike Swimming Piano Book

확인

폼을 모두 작성해주세요.

항목 중 하나만 선택했을때

좋아하는 음식

Pizza Steak Noodle

관심사

Cooking Bike Swimming Piano Book

확인

폼을 모두 작성해주세요.

각 항목당 최소 하나 이상 선택했을때

좋아하는 음식

Pizza Steak Noodle

관심사

Cooking Bike Swimming Piano Book

확인

좋아하는 음식 : steak
관심사 : Bike, swimming

주요 학습 키워드

```
submit event가 발생하면 formData를 이용해 value를 가져오고 배열로 만들어주기  
filter 함수를 사용하여 length 체크하기  
입력한 정보를 innerHTML을 사용하여 DOM에 동적으로 추가하기
```

작성해주세요 하는 question 파일경로

```
./question/src/app.js
```

실행 방법 및 의존성 모듈 설치

경로 ./question

터미널

```
$ yarn install  
$ yarn start
```

해설

1. form tag를 활용해 라디오 박스를 만든다고 가정합니다.
2. form과 form이 다루는 데이터를 표현해줄 dom을 html 상에 만들어야 합니다.
3. form이 전체적으로 어떤 데이터를 다루는지(갯수, 종류 등) 알아야 검사 코드를 만들 수 있습니다.
4. 확인 버튼이 클릭되면 서버로 데이터를 보내기 전에 데이터의 유효성 검사를 해야합니다.

이를 위해 form tag에 submit 이벤트를 수신하는 event handler를 등록해야 합니다.
이 핸들러 안에서 검사를 진행합니다.

현업에서 검사 코드에는 모든 항목에 대한 입력이나 선택을 했는지, 입력 값의 타입이 정확한지, 입력 값이 요구사항의 조건을 만족하는지 등이 포함될 수 있습니다.

검사를 통과하지 못했다면 사용자가 모든 항목에 대해서 입력 및 선택을 할 수 있도록 안내하는 UI를 만들어 보여줍니다.
검사를 통과했다면 작성 완료된 form 데이터를 보여줍니다(현업에선 API call을 합니다).

5. 이때 form tag는 type이 submit인 버튼이 클릭되면 기본적으로 페이지 새로고침을 하는데, 이 동작이 서비스에서 불 필요한 경우엔 이 동작을 하지 않을 필요가 있습니다. 이를 위해 핸들러에서 넘어온 event 객체의 preventDefault 함수를 실행해서 기본 동작을 멈춥니다.
6. 핸들러 내부에서 form이 가진 데이터를 확인하기 위해서 FormData 클래스를 활용합니다. 이때 이 클래스로 만든 인스턴스는 타입이 FormData입니다. iterator protocol이 FormData 내부에 정의되어 있기 때문에 for of loop 이용해 데이터 항목들을 순회할 수 있습니다. for loop를 사용하는 것이 꺼려진다면 Array의 from 함수를 이용해 배열로 만들어 쉽게 순회할 수도 있습니다.

Case28 : Instant search

케이스 주제

Q. 아래와 같이 작동하는 즉시 검색 기능을 구현하시오.

기능 요구사항

요구 사항은 아래와 같다.

1. 하나의 input tag를 만든다.
2. 이 input에 키보드 타이핑이 될때 현재 검색어 기준으로 필요한 API를 호출한다. 이때 API 호출은 즉각 실행해서 결과를 보여준다.
3. 검색어 API가 진행 중일때 input tag 우측에 loading 중임을 표시한다.
4. 검색어 API Response가 도착하면 그 내용을 input tag 아래에 리스트로 보여준다.

첫화면



검색어를 입력하고 로딩 중일때



검색어 API의 Response가 도착했을때

a

Abyssinian
Aegean
American Bobtail
American Curl
Polydactyl

해당하는 검색어가 없을때

abcd|

해당하는 검색어가 없습니다..!

주요 학습 키워드

fetch와 async, await를 이용한 API호출
slice, map 함수를 사용하여 특정조건에 맞는 검색어를 특정갯수만큼 노출 시키는 기능 구현
Element 속성 innerHTML, style을 사용하여 DOM에 보여주기

작성해주셔야 하는 question 파일경로

./question/src/app.js

실행 방법 및 의존성 모듈 설치

경로 ./question

터미널

```
$ yarn install  
$ yarn start
```

설명

1. The Cat API를 사용합니다.
2. 기본적으로 input tag에 이벤트리스너를 등록해서(keypress, keyup 같은 이벤트를 수신) 해결합니다.
3. 이때 사용자가 검색어를 입력하다가 전부 지우는 경우 즉, 이벤트리스너에 전달되는 값이 비어있는 경우가 생깁니다.
이럴땐 API 요청을 하는 것이 네트워크 자원 낭비이므로 이 경우를 제외합니다.
4. 로딩 인디케이터를 만드는 것은 다양한 방법이 있습니다.

코드에 되어있는 것처럼 Pub-Sub 구조를 활용해 로딩 중임을 알리는 이벤트를 활용하는 방법
단순하게, 로딩 인디케이터는 초기에 보이지 않다가 API 요청 이전에 스타일을 이용해 보여주고, API 요청 이후에
(Promise, async await) 스타일을 이용해 숨기는 방법

5. API Request는 브라우저에서 API Request를 위해 기본적으로 제공하는 fetch 함수를 사용하는 방법이 있고 axios 같은 third party를 사용할 수 있습니다.

이때 fetch, axios, superagent 등의 함수는 Promise 기반으로 동작합니다.
따라서 then, catch, finally 함수를 체이닝해서 구현하는 방법과 이 API 요청 함수를 감싸는 함수를 async 함수로 만드는
방법이 있고 현재 코드는 이 방법으로 작성 돼있습니다.

6. 이때 사용자가 타이핑을 한 문자씩 하면 즉각 API를 호출합니다.
7. API Response를 받아서 검색어 결과를 목록을 표현하는 ul tag에 렌더링 합니다.

응답 결과가 있으면 그대로 렌더링하고
응답 결과가 null 이거나 빈 배열인 경우처럼 비어있으면 해당하는 검색어가 없다는 UI를 만들어 사용자에게 친절하게 알
려줍니다.

8. 이때 API가 네트워크, 서버, 인증 등의 이유로 에러를 낼 수 있습니다.

API 처리를 Promise 기반으로 했다면 catch, finally 함수를 체이닝해서 에러가 생겼을때 사용자에게 알립니다.

async await를 사용했다면 API 요청 관련 코드를 try catch로 감싼 뒤 catch에서 동일한 작업을 수행합니다.

Case29 : Auto Complete 1

케이스 주제

검색어 자동완성 기능 만들기

(현업에서 서비스를 만들때 화장품을 검색하거나 영상을 검색할때 등에 사용할 검색 자동완성 기능을 요구하는 경우가 많이 있습니다. 사용자는 이 기능으로 내가 검색한 내용과 관련된 다른 검색어들을 알 수 있게 되어 사용자에게 정보 편의성이 주어지고 서비스 입장에서는 사용자로부터 더 많은 검색을 이끌어낼 수 있습니다.)
Q. 아래와 같은 스펙을 가진 검색 자동완성 기능을 만들어보세요.

기능 요구사항

1. 하나의 input tag를 만든다.
2. 이 input에 키보드 타이핑이 될때 현재 검색어 기준으로 관련 검색어 API를 호출한다.

이때 API 호출은 즉각 실행하길 바라는 경우
타이핑을 멈추고 0.5초 등의 시간이 지나고 요청하는 경우가 있다.

3. 검색어 API가 진행 중일때 input tag 우측에 loading 중임을 표시한다.
4. 검색어 API Response가 도착하면 그 내용을 input tag 아래에 리스트로 보여준다.

기능 작동 이미지



Loading...

문제

q1. Javascript

가장 마지막 타이핑이 일어나고 0.5초 뒤에 API Request를 실행하도록 하는 debounce logic을 작성하시오.

q2. RxJS를 이용해 스트림 구조로 동일한 기능을 작성하시오

주요 학습 키워드

fetch와 async, await를 이용한 API호출

지연시간을 적용하여 마지막 이벤트만 발생시키는 debounce를 이용해, 마지막으로 타이핑을 한 순간에 API를 호출 할 수 있는 기능 구현

slice, map 함수를 사용하여 특정조건에 맞는 검색어를 특정갯수만큼 노출 시키는 기능 구현

작성해주셔야 하는 question 파일경로

q1

`./question/1. Vanilla JavaScript/src/app.js`

q2

`./question/2. RxJS/src/app.js`

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/1. Vanilla JavaScript

터미널

```
$ npm install  
$ npm start
```

q2

경로 ./question/2. RxJS

터미널

```
$ npm install  
$ npm start
```

./solution/2.other

풀이 실행 시

q1

경로 ./solution/2.other/1. Vanilla JavaScript

index.html 열기

q2

경로 ./solution/2.other/2. RxJS

터미널

```
$ npm install  
$ npm run dev
```

Case14 : AutoComplete 1 - 출제자 해설

q1. Vanilla JavaScript

A)

```
const debounce = (targetFunction, debounceTime = 500) => {
  let timerId = null;

  return (...args) => {
    if (timerId) {
      clearTimeout(timerId);
    }

    timerId = setTimeout(() => {
      targetFunction(...args);
    }, debounceTime);
  };
};
```

```
$searchInput.addEventListener(
  "keyup",
  debounce(async (event) => {
    const query = event.target.value;

    if (!query) {
      return;
    }

    document.dispatchEvent(
      new CustomEvent(LOADING_EVENT_NAME, {
        detail: {
          isLoading: true
        }
      })
    );
    // 고양이 검색 API를 검색어 API로 간주합니다
    const response = await fetch(`#${API_URL}?q=${query}`);
  })
);
```

```

const cats = await response.json();
document.dispatchEvent(
    new CustomEvent(LOADING_EVENT_NAME, {
        detail: {
            isLoading: false
        }
    })
);

if (!cats.length) {
    $textList.innerHTML = "";
    $textList.style.visibility = "hidden";
    $infoParagraph.innerHTML = "해당하는 검색어가 없습니다..!";
    return;
}

$textList.innerHTML = cats
    .slice(0, 5)
    .map((cat) => `<li>${cat.name}</li>`)
    .join("");
$textList.style.visibility = "visible";
$infoParagraph.innerHTML = "";
}
);

document.addEventListener(LOADING_EVENT_NAME, ({
    detail: {
        isLoading
    }
}) => {
    $loadingIndicator.style.visibility = isLoading ? "visible" : "hidden";
});

```

해설

- 검색어 API가 따로 없기 때문에 The Cat API를 검색어 API로 가정합니다.
- 기본적으로 input tag에 이벤트리스너를 등록해서(keypress, keyup 같은 이벤트를 수신) 해결합니다.
- 이때 사용자가 검색어를 입력하다가 전부 지우는 경우 즉, 이벤트리스너에 전달되는 값이 비어있는 경우가 생깁니다. 이럴땐 API 요청을 하는 것이 네트워크 자원 낭비이므로 이 경우를 제외합니다.
- 로딩 인디케이터를 만드는 것은 다양한 방법이 있습니다.

코드에 되어있는 것처럼 Pub-Sub 구조를 활용해 로딩 중임을 알리는 이벤트를 활용하는 방법
 단순하게, 로딩 인디케이터는 초기에 보이지 않다가 API 요청 이전에 스타일을 이용해 보여주고, API 요청 이후에
 (Promise, async await) 스타일을 이용해 숨기는 방법

5. API Request는 브라우저에서 API Request를 위해 기본적으로 제공하는 fetch 함수를 사용하는 방법이 있고 axios 같은 third party를 사용할 수 있습니다.

이때 fetch, axios, superagent 등의 함수는 Promise 기반으로 동작합니다.

따라서 then, catch, finally 함수를 체이닝해서 구현하는 방법과

이 API 요청 함수를 감싸는 함수를 async 함수로 만드는 방법이 있고 현재 코드는 이 방법으로 작성 돼있습니다.

6. 이때 사용자가 타이핑을 한 문자씩 할때마다 우리가 등록한 이벤트 핸들러가 실행됩니다. 즉, 현재 상태에서는 사용자의 타이핑 숫자만큼 API Request가 실행됩니다. 우리가 검색 자동완성에서 기대하는 것은 사용자가 유의미한 글자를 모두 입력한 뒤에 API Request를 실행하는 것입니다. 따라서 이벤트 핸들러에 debounce를 걸어서 가장 마지막 타이핑이 일어나고 n초 뒤(코드에서는 0.5초)에 API Request를 실행해서 네트워크 자원을 아끼고 지나치게 자주 로딩 인디케이터와 검색어 목록이 깜빡거리는 점을 차단합니다.
7. API Response를 받아서 검색어 결과를 목록을 표현하는 ul tag에 렌더링 합니다.

응답 결과가 있으면 그대로 렌더링하고

응답 결과가 null 이거나 빈 배열인 경우처럼 비어있으면 해당하는 검색어가 없다는 UI를 만들어 사용자에게 친절하게 알려줍니다.

8. 이때 API가 네트워크, 서버, 인증 등의 이유로 에러를 낼 수 있습니다.

API 처리를 Promise 기반으로 했다면 catch, finally 함수를 체이닝해서 에러가 생겼을때 사용자에게 알립니다.
async await를 사용했다면 API 요청 관련 코드를 try catch로 감싼 뒤 catch에서 동일한 작업을 수행합니다.

q2. RxJS

A)

```
const inputStream = fromEvent($searchInput, "input").pipe(  
  map((event) => event.target.value),  
  debounceTime(500),  
  distinctUntilChanged(),  
  tap(() => ($loadingIndicator.style.visibility = "visible")),  
  switchMap((query) =>  
    ajax(` ${API_URL}?q=${query}` , { method: "GET" }).pipe(  
      ...  
    )  
  )  
)
```

```
        map(({ response }) => response)
    )
),
tap(() => ($loadingIndicator.style.visibility = "hidden"))
);
```

```
inputStream.subscribe({
  next: (cats) => {
    if (!cats.length) {
      $textList.innerHTML = "";
      $textList.style.visibility = "hidden";
      $infoParagraph.innerHTML = "해당하는 검색어가 없습니다..!";
      return;
    }

    $textList.innerHTML = cats
      .slice(0, 5)
      .map((cat) => `<li>${cat.name}</li>`)
      .join("");
    $textList.style.visibility = "visible";
    $infoParagraph.innerHTML = "";
  },
  error: () => {
    $infoParagraph.innerHTML =
      "An error has occurred when fetching search queries.";
  }
});
```

해설

1. Reactive 패러다임은 크게 4가지 개념을 이해하면 됩니다.

Observable : 우리가 애플리케이션을 만들 때 데이터는 API에서도 오고, 여러 DOM tag에 달린 이벤트 리스너에서도 옵니다. Observable은 이런 모든 종류의 데이터의 흐름을 나타냅니다.

Operator : 이렇게 시간순으로 연속적으로 들어오는 데이터를 우리는 그대로 사용하기도 하지만 계산을 하거나 log를 남기기도 합니다. 이렇게 데이터의 흐름 사이 사이에 어떤 특정한 작업을 해야 하는데, 이 작업을 해줄 수 있게 해주는 것이 바로 Operator입니다.

Observer : 이렇게 Observable과 Operator로 데이터가 어떻게 흘러서 어떻게 변하는지를 표현했으니, 이 데이터를 결국 어디에서 활용을 해야 합니다. 그 활용을 하는 주체가 바로 Observer입니다. 그래서 관찰자입니다. 더 정확히는, Observer는 next, error, complete 함수를 프로퍼티로 갖는 객체를 의미합니다. 이 객체를

observableInstance.subscribe(객체) 와 같이 subscribe 함수 안에 넣어주면 데이터를 꺼내오기 시작합니다.

Subscription : 위에서 subscribe 함수를 실행하면 데이터를 꺼내오기 시작하고 이 함수는 Subscription 객체를 반환합니다. 데이터를 꺼내온다는 것은 RxJS가 Browser 어디가에 이벤트 핸들러를 달아놨다는 것을 의미하는데, 더 이상 데이

터를 활용하지 않아야 하는 순간이 오면 사용한 메모리 자원을 다시 반환해야 합니다. 이때 이 Subscription 객체를 이용해 `subscriptionInstance.unsubscribe()` 와 같이 자원을 반환합니다.

2. input tag의 input 행위(keypress, keyup 등의 이벤트)를 `fromEvent` 함수를 이용해 Observable로 만듭니다.
3. 그 다음 Vanilla 버전에서 이벤트 핸들러가 수행했던 행위들(빈 문자 건너뛰기, debounce, 로딩 인디케이터 스타일 바꿔주기, API 요청)을 Operator를 이용해 수행합니다.

`map`, `debounceTime`, `tap`, `ajax` 등의 Operator가 이 작업들에 사용됩니다.

4. 이렇게 만들어진 Observable instance를 구독하기 시작합니다. 그럼 Observer에 검색어 목록 데이터가 전달됩니다. 이 값을 이용해 검색어 목록이 없는 경우와 있는 경우를 각각 구분해서 Vanilla 버전에서와 똑같이 렌더링 합니다.
5. 에러 처리에 대해서는, RxJS는 Observer 객체에 error 핸들러를 제공합니다. Observable에서 에러가 나면 error 핸들러가 수행됩니다. 따라서 에러가 생겼을 때 이 핸들러에 필요한 작업(사용자에게 에러가 났다고 알려주기)을 수행합니다.

Case29 : AutoComplete 1 - 타 개발자님의 해답

q1. Vanilla JavaScript

A)

```
// solution/2.others_1/1. Vanilla JavaScript/src/app.js
const API_URL = "https://api.thecatapi.com/v1/breeds/search";
/***
 * `debounce` 함수에 의해 생성된 함수는 반복적으로 호출되더라도 마지막 `debounceTime` 이후에 1회만 실행됩니다.
 * @param {*} targetFunction
 * @param {*} debounceTime
 */
const debounce = (targetFunction, debounceTime = 500) => {
    let timeoutId = null;
    return (...args) => {
        if (timeoutId) {
            clearTimeout(timeoutId);
        }
        /**
         * setTimeout 함수에 인자값으로 콜백 함수, 시간, 콜백 함수의 인자값을 전달할 수 있습니다.
         * @see https://developer.mozilla.org/ko/docs/Web/API/WindowTimers/setTimeout
         */
        timeoutId = setTimeout(targetFunction, debounceTime, ...args);
    };
};

const LOADING_EVENT_NAME = "loading";
const searchInputEl = document.querySelector(".SearchInput");
const loadingIndicatorEl = document.querySelector(".LoadingIndicator");
const textListEl = document.querySelector(".TextList");
const infoParagraphEl = document.querySelector(".InfoParagraph");

searchInputEl.addEventListener("keyup", debounce(async (event) => {
    /**
     * `currentTarget`, `target` 차이를 알아두시면 좋습니다.
     * [>] `currentTarget`은 이벤트가 바인딩된 객체를 가르키고, `target`은 이벤트버블링 발생시 발생한 객체를 가리킵니다.
     * [>] `currentTarget`은 이벤트가 발생하는 순간에만 사용이 가능하며, `debounce`와 같은 비동기 처리함수에서는 `null` 값이 할당됩니다.
     * @see https://developer.mozilla.org/ko/docs/Web/API/Event/currentTarget
     * @see https://developer.mozilla.org/en-US/docs/Web/API/Event/currentTarget
     * @see https://javascript.info/bubbling-and-capturing
     */
    const { value: query } = event.target;
    /**
     * API 요청을 위해 Fetch API 를 사용합니다.
     * @see https://developer.mozilla.org/ko/docs/Web/API/Fetch_API
     */
    const url = API_URL + `?q=${query}`;
    try {
        /**
         * 로딩 인디케이터 보임
         */
        loadingIndicatorEl.style.visibility = 'visible';
        const response = await fetch(url);
        const data = await response.json();
    }
});
```

```

const isEmptyResponseData = data.length === 0;
/**
 * 검색결과 없는 경우
 */
infoParagraphEl.textContent = isEmptyResponseData ? '검색 결과가 없습니다.' : '';
/**
 * 검색결과 있는 경우 목록 엘리먼트 노출
 */
textListEl.style.visibility = isEmptyResponseData ? 'hidden' : 'visible';
/**
 * 데이터를 HTML 문자열로 변환
 */
const itemsHtml = data.map(({ name }) => `<li>${name}</li>`).join('');
/**
 * 데이터 목록을 화면에 랜더링
 */
textListEl.innerHTML = itemsHtml;
/**
 * 로딩 인디케이터 감춤
 */
loadingIndicatorEl.style.visibility = 'hidden';
} catch (e) {
  infoParagraphEl.textContent = '처리중 에러가 발생하였습니다!';
}
}, 500));

```

해설

debounce, throttle 라이브러리 구현 내용과 동작을 알아두시면 좋습니다.
 debounce, throttle은 반복적으로 발생하는 이벤트등과 같은 처리를 할때 유용하게 사용할 수 있습니다, 위의 문제에서는 서버에 과도한 요청이 발생하지 않게 하기 위해 debounce를 적용하였습니다.
 currnrtTarget, target 차이를 알아두시면 좋습니다.
 currentTarget은 이벤트가 바인딩된 객체를 가르키고, target은 이벤트버블링 발생시 이벤트가 발생한 객체를 가르킵니다.
 currentTarget은 이벤트가 발생하는 순간에만 사용이 가능하며, debounce와 같은 비동기 처리함수에서는 null 값이 할당됩니다.

참고자료

<https://developer.mozilla.org/ko/docs/Web/API/Event/target>
<https://developer.mozilla.org/en-US/docs/Web/API/Event/currentTarget>
<https://javascript.info/bubbling-and-capturing>
https://developer.mozilla.org/ko/docs/Web/API/Fetch_API

q2. RxJS

A)

```

// solution/2.others_1/1. Vanilla JavaScript/src/app.js
...

const API_URL = "https://api.thecatapi.com/v1/breeds/search";
const searchInputEl = document.querySelector(".SearchInput");
const loadingIndicatorEl = document.querySelector(".LoadingIndicator");
const textListEl = document.querySelector(".TextList");
const infoParagraphEl = document.querySelector(".InfoParagraph");

const inputStream = fromEvent(searchInputEl, "input").pipe(

```

```

    /**
     * 전달되는 객체의 속성 추출
     * `map((event) => { event.target.value })` 코드를 대체
     */
    pluck('target', 'value'),
    /**
     * 중복되는 데이터를 제거
     */
    distinctUntilChanged(),
    /**
     * 단어가 입력된 경우에만 처리
     */
    filter((value) => value),
    tap(() => {
        /**
         * 로딩 인디케이터 보임
         */
        loadingIndicatorEl.style.visibility = 'visible';
    }),
    /**
     * 마지막 키보드 입력 발생후 500ms 대기
     */
    debounceTime(500),
    switchMap(
        /**
         * 기존 스트림을 `ajaxGetJSON` 스트림으로 변경
         */
        (query) => ajaxGetJSON(` ${API_URL}?q=${query}`).pipe(
            catchError((e) => {
                infoParagraphEl.textContent = '처리중 에러가 발생하였습니다!';
                return of(e);
            })
        )),
        tap(() => {
            /**
             * 로딩 인디케이터 감춤
             */
            loadingIndicatorEl.style.visibility = 'hidden';
        }),
        filter((value) => {
            /**
             * 에러 객체가 전달된 경우 더이상 전달하지 않음
             */
            return !(value instanceof Error);
        }),
        tap((dataItems) => {
            const isEmptyResponseData = dataItems.length === 0;
            /**
             * 검색결과 없는 경우
             */
            infoParagraphEl.textContent = isEmptyResponseData ? '검색 결과가 없습니다.' : '';
            /**
             * 검색결과 있는 경우 목록 엘리먼트 노출
             */
            textListEl.style.visibility = isEmptyResponseData ? 'hidden' : 'visible';
        })
    );
}

inputStream.subscribe({
    next: (dataItems) => {
        /**
         * 데이터를 HTML 문자열로 변환
         */
        const itemsHtml = dataItems.map(({ name }) => `<li>${name}</li>`).join('');
        /**
         * 데이터 목록을 화면에 렌더링
         */
    }
});

```

```
    textListEl.innerHTML = itemsHtml;
},
/**
 * 스트림 에러가 발생한 경우 스트림이 닫힘
 */
error: (e) => alert(`예상하지 못한 에러 발생! ${e.message}`)
});

```

해설

문제 1의 원리와 동일하게 debounce

기능을 활용하여 과도하게 발생하는 서버 요청을 줄였습니다.

문제 2 코드는 RxJS를 이용하여 작성된 코드입니다.

RxJS(Reactive Extensions For JavaScript)는 기존 코드와 달리 이벤트나 데이터 스트림을 처리하는 방식으로 프로그래밍 합니다.

RxJS에서 제공하는 operators

들의 기능을 잘 알고 활용하면 좋을것 같습니다.

참고자료

<https://www.learnrxjs.io/learn-rxjs/operators>

결론

RxJS를 잘 모른다면 조금 어려울 수 있는 문제 같습니다. 다양한 라이브러리를 사용해 보는 것도 좋은 것 같습니다. 대부분의 라이브러리나 프레임워크에는 만들게 된 동기(Motivation)나, 지향점, 철학 또는 해결하고자 하는 목적이 있는데 라이브러리를 사용하기 전에 한번 찾아보고 공감이 되신다면 금방 익숙해질 것 같습니다.

Case30 : Rating UI

케이스 주제

별점을 평가할 수 있는 Rating UI를 구현하세요.

Q. 아래와 같은 스펙을 가진 Rating UI를 구현하세요.

기능 요구사항

1. 0 ~ N 개의 별을 표시합니다. (0.5점 단위)
2. 각 별은 비어있거나(Empty), 반만 차있거나(Half), 꽉 차있을 수 있습니다(Full). (3가지 각 이미지 파일은 assets 폴더 안에 있습니다.)
3. 별 위로 마우스오버하면 왼쪽부터 마우스 위치까지의 채워진 별로 표시합니다.
4. 클릭시 점수 상태를 업데이트합니다.
5. X 버튼을 클릭하면 별점을 0점으로 초기화합니다.

별의 갯수가 N값만 조정하면 쉽게 변경되도록 작성해보세요.

기능 작동 이미지



문제

q1. N점 만점의 Rating UI를 JavaScript(ES6+)로 구현해보세요.

q2. N점 만점의 Rating UI를 React Component로 구현해보세요.

주요 학습 키워드

DOM Access, DOM Events(mousemove, mouseleave, click, etc.), MouseEvent, Mouse Position Calculation, useState, State Management

작성해주셔야 하는 question 파일경로

q1

./question/q1/index.js

q2

./question/q2/src/components/StarRate.jsx

출제자 강사님 코드 기반으로, 해당 경로에서 요구한 문제사항을 해결해주세요

실행 방법 및 의존성 모듈 설치

q1

경로 ./question/q1

터미널

```
$ npm install  
$ npm start
```

빌드가 완료되면 localhost:3001로 접속합니다.

q2

경로 ./question/q2

터미널

```
$ npm i  
$ npm run start
```

빌드가 완료되면 localhost:3002로 접속합니다.

Case30 : RatingUI - 출제자 해설

q1. N점 만점의 Rating UI를 JavaScript(ES6+)로 구현해보세요.

A)

```
// Score를 받아 별의 점수 표시를 바꿔주는 함수
const setDisplayScore = (score) => {
  const starList = [...$stars.children]
  starList.forEach((star, i) => {
    if (score > i) {
      if (score - i === 0.5) {
        star.className = 'star half'
      } else {
        star.className = 'star full'
      }
    } else {
      star.className = 'star empty'
    }
  })
}

// Score를 받아 점수 상태를 업데이트해주는 함수
const setScore = (score) => {
  setDisplayScore(score)
  $score.textContent = score
  state.score = score
}

// MouseEvent를 받아 Score를 0~N까지 0.5 scale로 점수를 계산해주는 함수
const calculateScore = (e) => {
  const { width, left } = e.currentTarget.getBoundingClientRect()
  const x = e.clientX - left
  const scale = width / MAX_SCORE / 2
  return (Math.floor(x / scale) + 1) / 2
}

// 마우스를 움직일때마다 마우스 위치에 해당하는 점수를 계산하여 별표시를 업데이트해주는 이벤트리스너
$stars.addEventListener('mousemove', (e) => {
```

```

    const score = calculateScore(e)
    setDisplayScore(score)
  })

// 마우스가 별영역을 빠져나가면 기존 점수로 별점수 표시를 업데이트해주는 이벤트 리스너
$stars.addEventListener('mouseleave', () => {
  setDisplayScore(state.score)
})

// 마우스로 별 영역을 클릭하면 마우스 위치에 해당하는 점수를 상태로 업데이트해주는 이벤트 리스너
$stars.addEventListener('click', (e) => {
  const score = calculateScore(e)
  setScore(score)
})

// Reset 버튼을 누르면 점수 상태를 0점으로 초기화해주는 이벤트 리스너
$reset.addEventListener('click', () => {
  setScore(0)
})

```

q2. N점 만점의 Rating UI를 React Component로 구현해보세요.

A)

```

import React, { useState } from 'react'
import './StarRate.scss'
import EmptyStar from '../../assets/star-empty.svg'
import HalfStar from '../../assets/star-half.svg'
import FullStar from '../../assets/star-full.svg'
import Reset from '../../assets/reset.svg'

// 별의 갯수 = Score
const MAX_SCORE = 5

export const StarRate = ({ onChange, score }) => {
  // 표시되는 별의 상태
  const [displayScore, setDisplayScore] = useState(score)

  const handleMove = (e) => {
    const { width, left } = e.currentTarget.getBoundingClientRect()
    let x
    if ('offsetX' in e.nativeEvent) {
      x = e.nativeEvent.offsetX
    } else {

```

```

        x = e.nativeEvent.targetTouches[0].clientX - left
    }
    Array(MAX_SCORE * 2)
        .fill(0)
        .find(
            (_, i) =>
            x <= (width / (MAX_SCORE * 2)) * i + 1 || setDisplayScore((i + 1) / 2)
        )
    }

    return (
        <section>
            <div
                className="stars"
                onMouseMove={handleMove}
                onTouchMove={handleMove}
                onTouchEnd={() => onChange(displayScore)}
                onMouseEnter={() => setDisplayScore(score)}
                onMouseLeave={() => setDisplayScore(score)}
                onClick={() => onChange(displayScore)}
            >
                {[...Array(MAX_SCORE)].map((_, i) => (
                    <Star score={displayScore} idx={i} key={i} />
                )))
            </div>
            <Reset
                className="reset"
                onClick={() => {
                    onChange(0)
                    setDisplayScore(0)
                }}
            />
        </section>
    )
}

const Star = ({ score, idx }) =>
    score > idx ? (
        score - idx === 0.5 ? (
            <HalfStar />
        ) : (
            <FullStar />
        )
    ) : (
        <EmptyStar />
    )

```

```

html,
body,

```

```
#app {  
}  
  
#app {  
    display: grid;  
    margin: 0;  
    height: 100%;  
    span {  
        position: absolute;  
        width: 100%;  
        text-align: center;  
        top: 100px;  
        font-size: 2rem;  
    }  
}  
  
section {  
    justify-self: center;  
    align-self: center;  
    display: inline-flex;  
    position: relative;  
    height: 35px;  
    .stars {  
        height: 52px;  
        cursor: pointer;  
        svg {  
            width: 48px;  
            height: 48px;  
            padding: 2px;  
            position: relative;  
            pointer-events: none;  
        }  
    }  
    .reset {  
        margin-left: 8px;  
        width: 24px;  
        height: 24px;  
        padding: 14px;  
        cursor: pointer;  
    }  
}  
}
```