

REACT

P O R T F O L I O



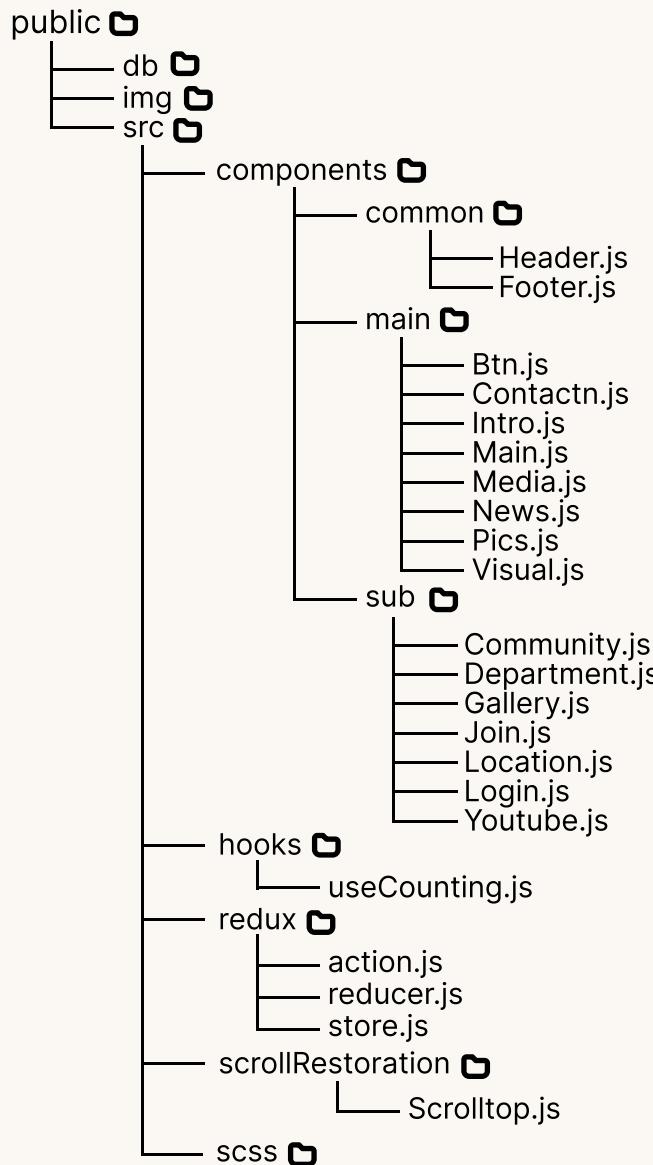
Contents

react portfolio index

Introduction	1
Folder tree structure	2
SCSS structure	3
Responsive layout	4
Main page details	5
Sub page details	6
Department page	6-1
Community page	6-2
Gallery page	6-3
Youtube page	6-4
Location page	6-5
Join page	6-6

Folder tree structure

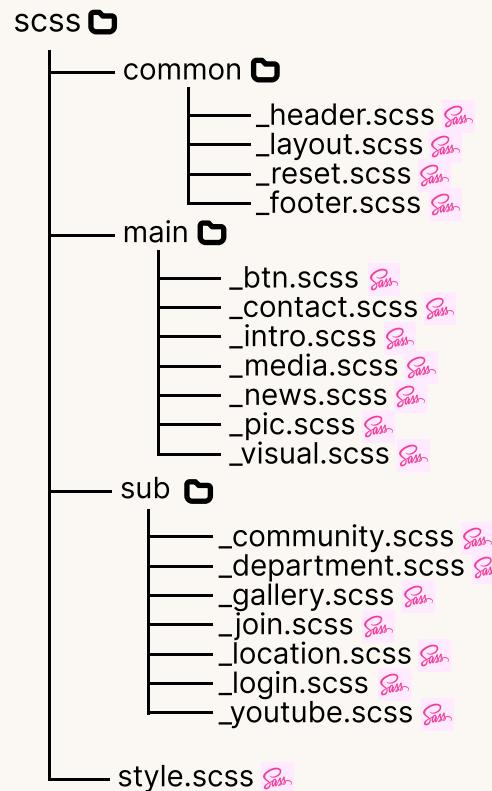
컴포넌트 폴더 구조



각 디렉토리 구성

- db json 목업 파일이 담길 폴더
- img 프로젝트에 사용될 각종 이미지가 담길 폴더
- components 각 페이지를 구성하는 컴포넌트들이 담길 폴더
 - common 모든 페이지에 공통적으로 적용될 컴포넌트가 담길 폴더
 - main 메인페이지를 구성하는 컴포넌트가 담길 폴더
 - sub 서브페이지에 해당하는 컴포넌트가 담길 폴더
- hooks 편의를 위해 제작한 커스텀 흑이 담길 폴더
- redux 상태 관리를 담당하는 폴더 (action.js, reducer.js, store.js)
- scrollRestoration 라우터를 통해 다른 페이지로 이동했을 때 기존 페이지의 스크롤이 그대로 적용되는 것을 방지하기 위해 스크롤을 복원시키는 컴포넌트가 담겨 있음
- scss 컴포넌트 폴더와 마찬가지로 공통 컴포넌트, 메인 컴포넌트, 서브 컴포넌트의 세부 폴더로 나뉘어 있으며, 각 컴포넌트 별로 scss 파일을 분리시킴

SASS 폴더 구조



style.scss 코드

```
@charset 'UTF-8';
```

웹 폰트 불러오기

```
@import url('https://fonts.googleapis.com/css2?
family=Inria+Sans:wght@300;400;700&family=Jost:wght@100;200;300;400;500;600;700&family=Koh+Santepheap:wght@100;300;400;700&family=Nanum+Gothic:wght@400;700&display=swap');
```

```
$desktop: 1179px;
$tablet: 768px;
$mobile: 576px;
```

디바이스 별 break-point 변수 지정

common

```
@import 'common/reset';
@import 'common/layout';
@import 'common/header';
@import 'common/footer';
```

모듈화 하여 분리해 놓은 scss 파일을
style.scss 파일에 병합

main

```
@import 'main/visual';
@import 'main/intro';
@import 'main/news';
@import 'main/btns';
@import 'main/media';
@import 'main/pics';
@import 'main/contact';
```

sub

```
@import 'sub/department';
@import 'sub/community';
@import 'sub/gallery';
@import 'sub/youtube';
@import 'sub/location';
@import 'sub/join';
@import 'sub/login'
```

Responsive page

- 사용 기술
- media query, responsive units
- 구현 기능

pc, tablet, mobile 등 웹을 접속할 수 있는 디바이스의 크기에 따라 최적화된 크기의 화면을 제공하기 위해 미디어 쿼리 문법을 사용하여 반응형으로 제작

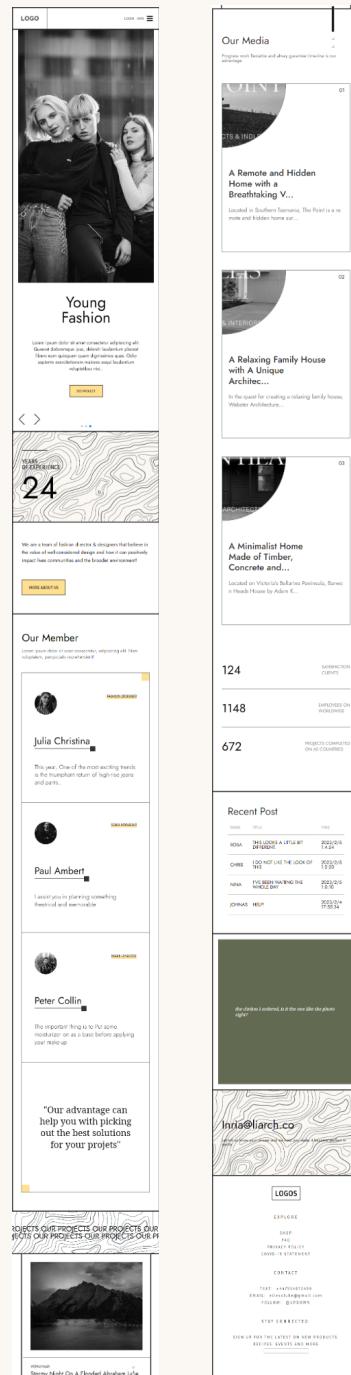
데스크탑 퍼스트 방식으로 진행함

```
// 데스크탑 1180px
@media screen and (max-width: 1179px){
  ...scss code
}
```

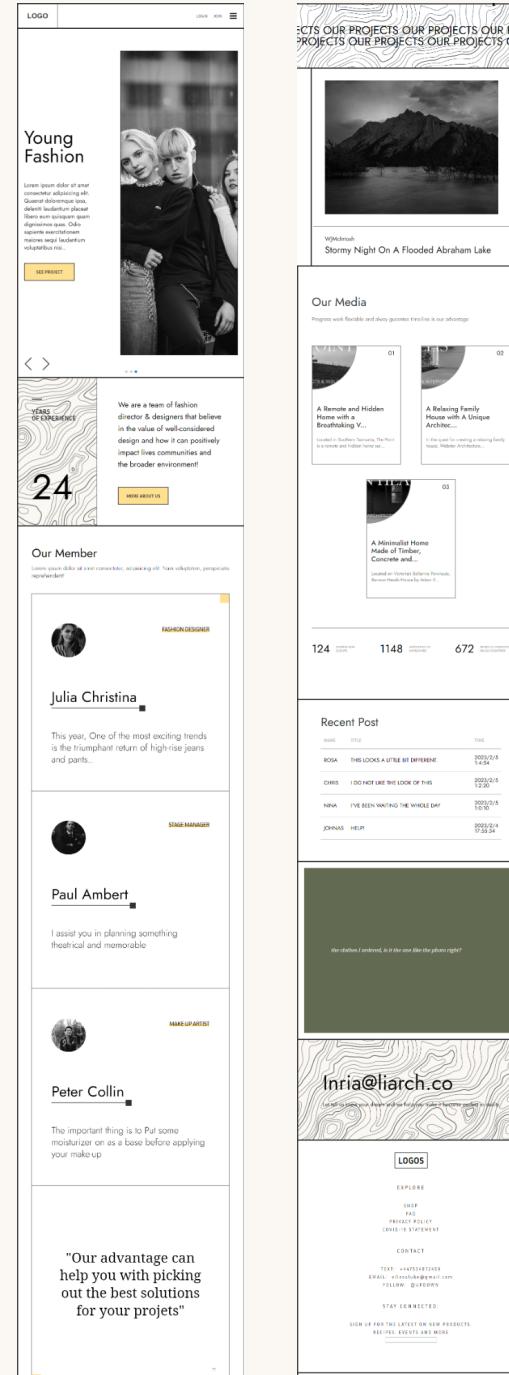
```
// 태블릿 768px
@media screen and (max-width: 768px){
  ...scss code
}
```

```
// 모바일 576px
@media screen and (max-width: 1179px){
  ...scss code
}
```

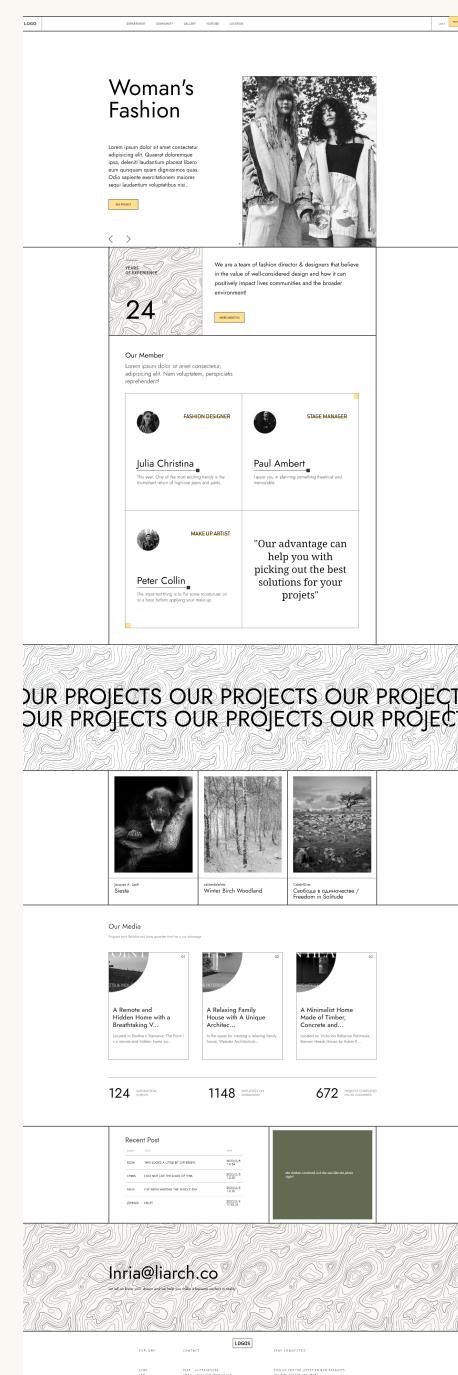
mobile



tablet



desktop



Main page

- 사용 기술 requestAnimationFrame() 등의 Animation API

• 구현 기능

메인 페이지의 인터렉티브 애니메이션 효과는 scrolled state값에 기반을 함

메인 컴포넌트가 마운트 된 후 스크롤 이벤트가 발생할 때마다 바뀐 스크롤 값으로 scrolled state를 업데이트 하며 이 scrolled는 props로 offsetTop값과 함께 하위 각 섹션 컴포넌트에 전달됨.

하위 컴포넌트에서는 scrolled를 계속해서 감지하고 해당 섹션의 offsetTop 위치에 scroll이 도달하였을 때 애니메이션을 수행함

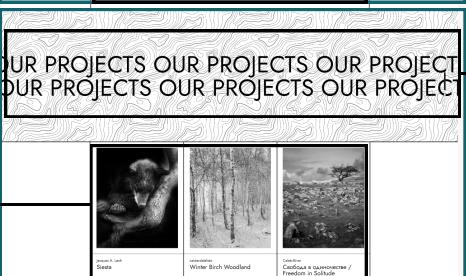
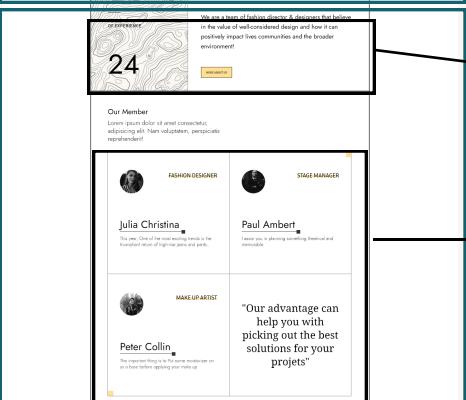
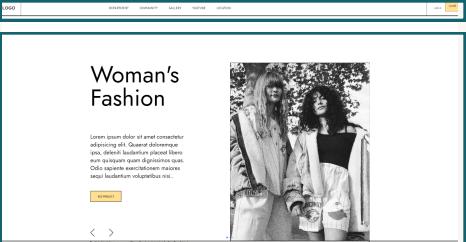
스크롤이 특정 위치에 도달할 시 scale(0)에서 scale(1)로 transition 되도록 설정하여 요소의 크기가 커지도록 함

9 Btns 컴포넌트

메인 컴포넌트가 마운트 되면 메인 컴포넌트 내의 각 섹션(하위 컴포넌트)들의 offsetTop 값을 담은 배열을 pos ref에 저장하고, 버튼을 클릭 시 pos를 참조해서 해당 섹션의 offsetTop 위치로 애니메이션 되어 이동함



7 Contact 컴포넌트



1 Header 컴포넌트

2 Visual 컴포넌트

swiper 라이브러리를 활용하여 슬라이드 구현
flipEffect 모듈을 불러와서 마치 잡지를 보듯 책을 넘기는 효과를 주었음

3 Intro 컴포넌트

타이핑 효과 구현

글자가 하나씩 추가될 때마다 해당 글자의 border-right이 깜빡거리는 애니메이션 주어 마치 커서가 깜빡거리는 효과를 구현함

스크롤이 특정 위치에 도달했을 때 -180deg의 뒷면 상태의 아티클들이 0deg의 앞면 상태로 뒤집히도록 transition 효과를 줌 각각의 아티클에 transition-delay를 다르게 주어 차례대로 뒤집히도록 하였음

4 Pics 컴포넌트

글자들이 옆으로 일정 거리만큼 이동하도록 애니메이션 효과 줌. 요소의 절반 정도가 왼쪽 혹은 오른쪽으로 이동했을 때 다시 원위치로 돌아가도록 설정하여 마치 글자가 끊기지 않고 무한히 움직이는 것 같은 효과를 주었음
스크롤 이벤트가 발생 시 더 빠르게 이동함

5 Media 컴포넌트

requestAnimationFrame을 활용하여 카운팅 애니메이션 구현 애니메이션 구현 코드가 상당히 복잡한 듯 하여 useCounting이라는 hook으로 분리시켜 좀 더 깔끔한 코드를 만들어보려 노력함

6 News 컴포넌트

community 컴포넌트와의 연동
community 컴포넌트에서 생성된 게시글이 localStorage에 저장되고 News 컴포넌트에서는 스토리지에 저장된 게시글을 불러와서 보여줌

8 Footer 컴포넌트

- 사용 기술
 - axios, redux,
- 구현 기능

1) 페이지 전체를 하나의 department 컴포넌트로 구성하였으며, 컴포넌트가 마운트되자마자 직원들의 이름, 직책, 사진이 담긴 json 데이터를 promise 기반의 http 통신 라이브러리인 axios를 이용하여 get함.

2) promise가 이행됨과 동시에 반환된 결과값인 데이터를 디스패치하여 redux에서 전역으로 관리함

```
useEffect(()=>{
  axios
    .get(`/${path}/db/department.json`)
    .then(json=>{
      dispatch(setMembers(json.data.data));
    })
    // 액션 생성 함수
},[]); // 액션 객체와 함께 dispatch하여 리듀서 함수가 호출됨
```

```
export const departmentReducer = (state=initMember, action)=>{
  switch (action.type) {
    case 'SET_MEMBERS' :
      return {...state, members: action.members}
    default:
      return state;
  }
}
```

```
export const setMembers = members=>{
  return {
    type: 'SET_MEMBERS',
    members: members
  }
}
```

MOSS IS A HIDDEN RETREAT ABOVE THE BUSTLING SALAMANCA PLACE, TUCKED WITHIN THE ORIGINAL WAREHOUSES, ONCE HOME TO TRADERS, WHALERS, PUBLICANS, GENTLEMEN AND CONVICTS.



INTERACTIVE
ARTIST!



Lorem ipsum, dolor sit amet consectetur adipiscing elit. Similique dolor magni dolorem provident dicta ducimus et veritatis. Incidunt, recusandae esque adipiscing elit. Similique dolor magni dolorem provident dicta ducimus et veritatis. Incidunt, recusandae esque.



FASHION DESIGNER
STAGE MANAGER
MAKE UP ARTIST
FASHION DIRECTOR
FASHION MERCHANDISER
ADVERTISER

6-2

Community page

- 사용 기술
 - localStorage
- 구현 기능

클라이언트 내 crud 구현.

새로고침하거나 브라우저를 껐다 키더라도 생성한 포스트 데이터가 사라지지 않도록
localStorage에 데이터 저장

1) getLocalItems 함수로 포스트 초기화

community 컴포넌트가 마운트 될 때 로컬스토리지로부터 포스트 데이터를 불러오고,
불러온 데이터를 state에 담아 관리

```
const getLocalItems = ()=>{
  let data = localStorage.getItem('posts'); // localStorage로부터 데이터를 불러옴
  if(data){ // localStorage에 배열 오브젝트를 JSON 문자열 형태로
    return JSON.parse(data); // 저장하기 때문에 불러올 때 객체 형태로 변환 후 반환
  }else {
    return []; // 데이터가 없다면 빈 배열을 반환
  }
}
const [posts, setPosts] = useState(getLocalItems()); // 불러온 데이터를 state로 관리
```

2) localStorage에 포스트 저장

포스트가 업데이트 될 때마다 업데이트 된 데이터를 localStorage에 저장하여 나중에 새로
접속해도 localStorage로부터 데이터를 불러올 수 있도록 코드 작성

```
useEffect(()=>{ // localStorage에는 문자열 형태
  localStorage.setItem('posts', JSON.stringify(posts)); // 만 저장 가능하므로 json문자열
},[posts]); // 형태로 변환하여 저장
```

코멘트 입력란

COMMUNITY

Leave Comment

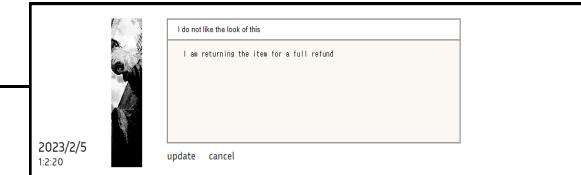
Name _____
Title _____

cancel create

포스트 생성



포스트 수정



Nina

I've been waiting the whole day

When can I receive the clothes I ordered?

2023/2/5
1:010

modify delete



Johnas

Help!

Help! The color ran in the wash. How can I get the stains out?

2023/2/4
17:55:34

modify delete

6-3

Gallery page

- 사용 기술
 - redux, flickr api
- 구현 기능

flickr api를 활용하여 다양한 사진 데이터를 요청할 수 있는 갤러리 구현

1) 첫 마운트 시 main 페이지의 pic 컴포넌트로부터 flickr api로 이미 불려와 redux에 저장되어 관리되고 있는 사진 데이터를 불러와 뿐여줌

```
export default function Gallery(){
  const picData = useSelector(state=>state.flickrReducer.photos); // 첫 마운트 시 리덕스로부터 포토 get
  const [items, setItems] = useState(picData); // 그 후 사진 데이터를 컴포넌트 내에서 state로 관리

  return(
    {items.map((item, idx)=>{ // 사진 데이터가 담겨 있는 배열을 순회하며 개별 article 생성
      return(
        <article>
        ....
        </article>
      )
    })
  )
}
```

2) 상단의 input 검색 시 flickr api 통신을 통해 키워드에 해당되는 사진 불러옴

```
const getFlickr = async opt=>{ // async await 문법을 통해 비동기 통신을 좀 더 편리하게 함
  const api_key = '6695bb82cf9a3db1962df3f386dd83e8';
  const method1 = 'flickr.photos.search';
  let url = ' ';

  if(opt.type === 'search'){ // flickr api 통신을 위한 URI 생성 (쿼리 스트링으로 원하는 데이터의 조건 전달 )
    url = `https://www.flickr.com/services/rest/?method=${method1}&per_page=${opt.count}
    &api_key=${api_key}&format=json&nojsoncallback=1&tags=${opt.tag}` // url에 input에 쓰인 키워드 전달
  }
  const res = await axios.get(url); // 데이터(포토) 요청
  const photos = res.data.photos.photo;
  setItems(photos); // 응답받은 데이터(포토)를 컴포넌트 내에서 state로 관리
}
```



6-4

Youtube page

- 사용 기술
 - youtube api
- 구현 기능

youtube api를 활용하여 재생 목록에 담겨 있는 동영상을 불러와 페이지에 연동
아티클 클릭 시 팝업창이 뜨고 동영상을 클릭하여 시청할 수 있음

1) 첫 마운트가 됨과 동시에 youtube api를 사용하여 데이터를 불러와 화면에 뿌려줌.
데이터는 state로 관리

```
// 나의 재생목록에 있는 동영상을 가져오기 위해 쿼리스트링으로 조건 부여
const url = `https://www.googleapis.com/youtube/v3/playlistItems?part=${part}&key=${key}
&playlistId=${playlistId}&maxResults=${num}`
```

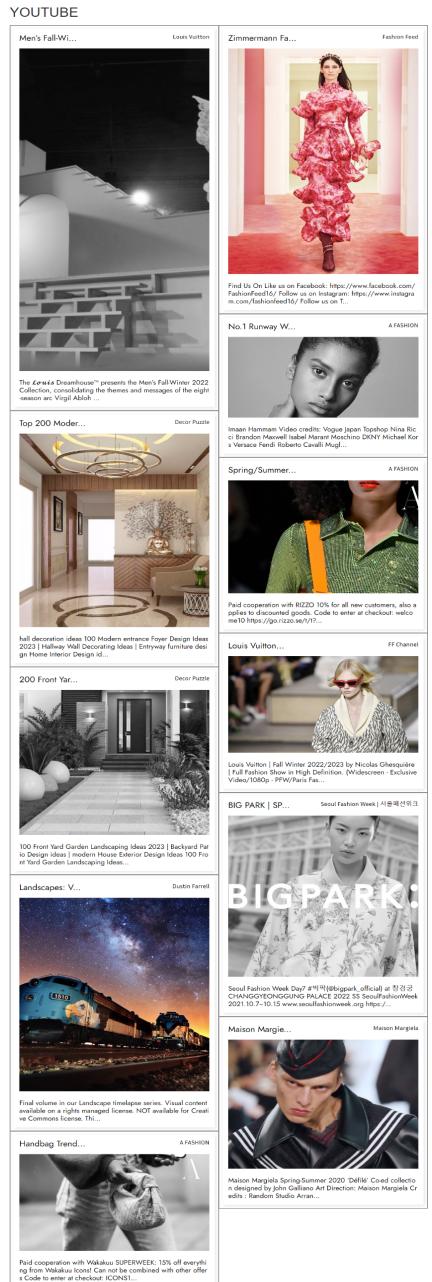
```
useEffect(()=>{
  // axios로 api 통신하여 동영상 데이터 불러옴
  axios.get(url)
  .then(json=>{
    setItems(json.data.items); // 불러온 데이터로 state 초기화
  })
});
```

2) (JSX) 동영상 정보가 담겨 있는 state를 순회하여 아티클 생성

```
{items.map((item, idx)=>{
  // map으로 state를 순회하여 컴포넌트 배열 생성
  return (
    <article key={idx}>
      // 리액트가 효율적으로 트리 변환 작업을 할 수 있도록 key prop 설정
      <div className="inner">
        <h2>{tit_len > 12 ? tit.substr(0, 13) + '...' : tit}</h2>
        <span>{uploader}</span>
        <div className="pic" onClick={()=>{
          setIsPop(true);
          setIndex(idx);
        }}>
          <img src={imgSize} />
        </div>
        <p>{desc_len > 150 ? desc.substr(0, 151) + '...': desc}</p>
      </div>
    </article>
  )
})
```



클릭 시 popup 생성



6-5

Location page

- 사용 기술
 - kakao map api
- 구현 기능

kakao map api를 활용하여 웹 애플리케이션에 지도 기능 구현
3곳의 좌표에 마커를 표시하여 각 지점들의 위치를 설정

1) 카카오 맵 api를 이용해 지도를 생성하고 생성한 지도는 state로 관리

```
export default function Location(){
  const {kakao} = window; // 스크립트 태그로 불러온 api를 리액트에서 인식할 수 있도록 디스트럭처링 할당을
  useEffect(()=>{           // 통해 window 객체에 담겨 있는 kakao 클래스를 직접 뽑아줌
    const options = {
      center: mapInfo[0].latlng, // 맵을 만들 때 필요한 필수 옵션
      level: 3
    }
    const map = new kakao.maps.Map(container.current, options); // 맵 생성
    setMap(map); // state로 맵 관리
  })
}
```

2) 각 지점별 마커 생성

```
export default function Location(){
  const info = [               // 각 지점별 정보를 담은 객체들로 이루어진 배열
    {
      title: "INCHEON"
      latlng: new kakao.maps.LatLng(35.1659875, 129.1355099),
      imgSrc: path+'img/marker.png'
    }
    .....
  ]
  const [mapInfo] = useState(info); // 지점별 정보를 state로 관리
  const [index, setIndex] = useState(0);
  useEffect(()=>{
    new kakao.maps.Marker({ // mapInfo의 특정 index에 해당하는 지점의 마커 생성
      map: map,
      position: mapInfo[index].latlng,
      title: mapInfo[index].title,
      image: new kakao.maps.MarkerImage(mapInfo[index].imgSrc, mapInfo[index].imgSize)
    })
    map.setCenter(mapInfo[index].latlng); // 지도를 해당 지점의 위치로 이동
  },[index]); // index가 바뀔 때마다 지도를 새로 생성하고 해당 index에 해당하는 지점의 마커 생성
}
```

LOCATION



3) (JSX)

```
return(
  <div className="store">
    <div class="inner">
      <h2>LOCATION</h2>
      <ul className="branch">
        {mapInfo.map((data, idx)=>{
          return (
            <li>
              // 각 버튼을 클릭하면 버튼에 해당하는 인덱스로 index state
              // 업데이트
              <button key={idx} onClick={()=>setIndex(idx)}
                style={idx === index ? {background: '#ffdf90'} : null}>
                {data.title}
              </button>
            </li>
          )
        })
      </ul>
    </div>
  </div>
)
```

- 사용 기술 - fontawesome, 정규표현식
- 구현 기능

이미지가 아닌 fontawesome을 사용하여 icon 생성해 꾸며주었으며, 정규표현식을 사용해 회원가입 필드에 대한 유효성 검사 실시

- 각 필드의 값을 모아 하나의 state로 관리
(필드 값이 바뀔 때마다 state가 최신상태로 업데이트 됨)

```
const initVal = {
  userid: '',
  email: '',
  pwd1: '',
  pwd2: '',
  comments: '',
  gender: false,
  interests: false,
  edu: false
}
const [val, setVal] = useState(initVal);
```

```
const [err, setErr] = useState({})
const [isSubmit, setIsSubmit] = useState(false);
const [success, setSuccess] = useState(false);
```

- submit 버튼 클릭 시 클릭 이벤트 핸들러가 호출되어 check(val) 유 효성 검사 함수가 실행되고 반환된 err 객체로 err state가 업데이트 됨

```
const handleSubmit = e=>{
  e.preventDefault();
  setIsSubmit(true);
  setErr(check(val)); // 현재 필드 value 값을 인자로 전달하여 유효성 검사 후 그 결과를 err 객체로 반환
} // err 객체는 필드가 모두 유효하다면 빈 객체
```

- submit 버튼을 누른 후 err state가 업데이트 되어 재렌더링이 되면 다음과 같은 사이드 이펙트 실행

```
useEffect(()=>{
  const len = Object.keys(err).length; // err 객체의 키가 담긴 배열의 length 확인
  if(len === 0 && isSubmit){ // length가 0이면 err 없이 유효하다는 뜻이므로 success state를 true로 업데이트
    setSuccess(true); // success state가 true일 때만 해당 페이지 상단에 '회원가입 성공' 문구가 뜸
  }else {
    setSuccess(false);
  }
},[err, isSubmit]); // err 객체가 업데이트 될 때만 실행
```

JOIN

JOIN

Welcome to visit our homepage

Verification Identify.

Insert Info.

Completion Join.

ESSENTIAL INFO.

USER ID

E-MAIL

PASSWORD

RE PASSWORD

GENDER

INTERESTS

EDUCATION

LEAVE COMMENTS

cancel

send