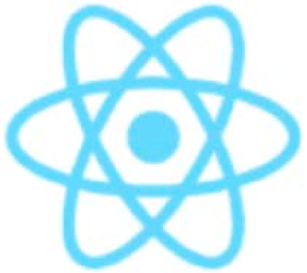
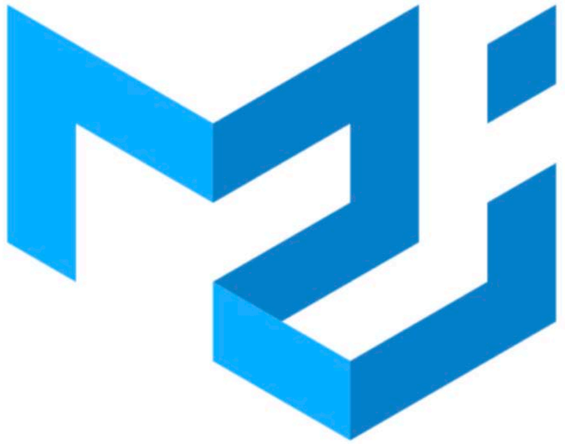
react-kangzoom

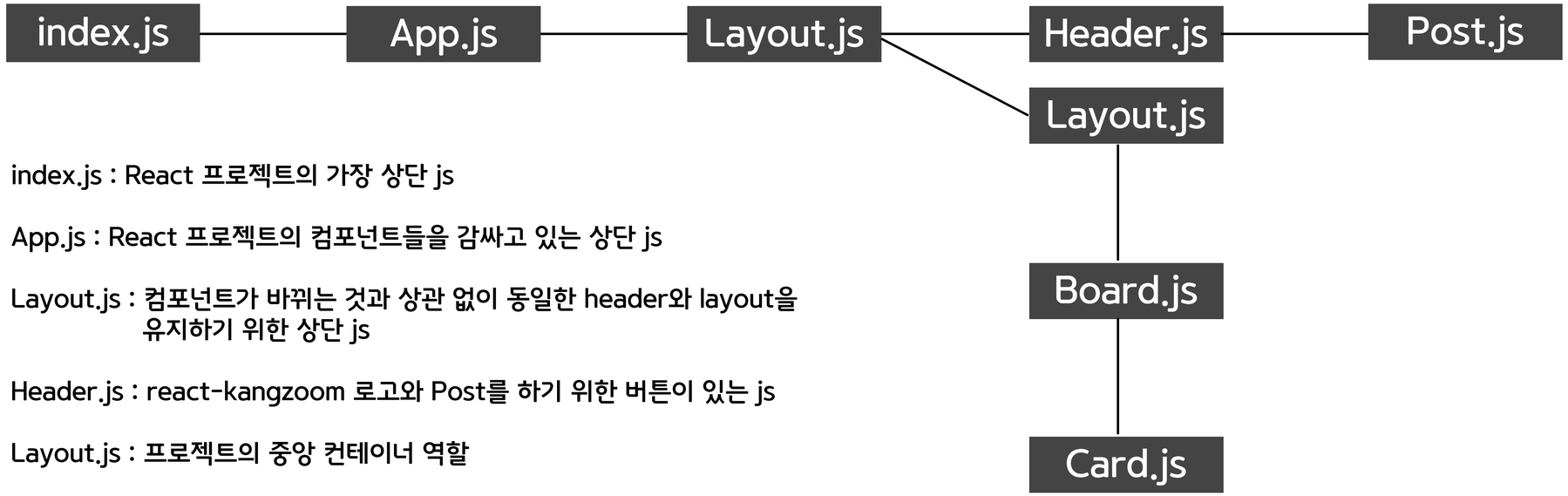
사용 라이브러리 및 기술

- React 라이브러리를 활용한 UI 연습
- Material UI, recoil 라이브러리 활용
- LocalStorage를 이용한 데이터 저장
- JSON stringify, parse 연습





전체적인 구조



index.js : React 프로젝트의 가장 상단 js

App.js : React 프로젝트의 컴포넌트들을 감싸고 있는 상단 js

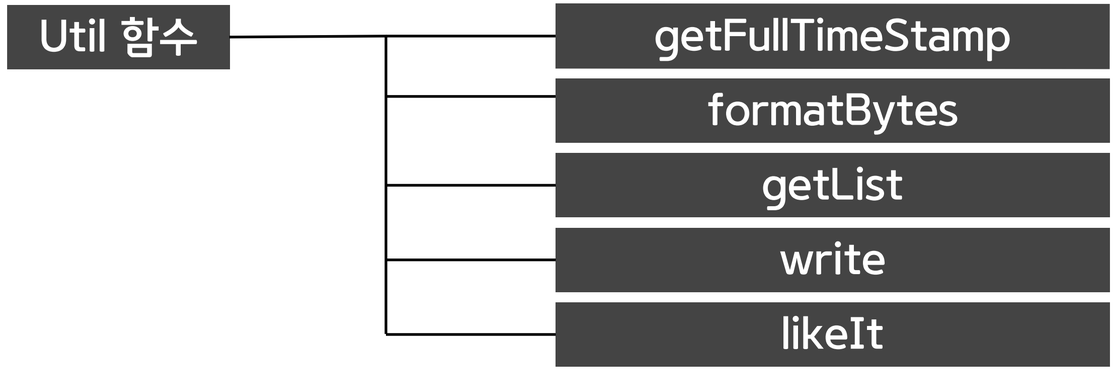
Layout.js : 컴포넌트가 바뀌는 것과 상관 없이 동일한 header와 layout을 유지하기 위한 상단 js

Header.js : react-kangzoom 로고와 Post를 하기 위한 버튼이 있는 js

Layout.js : 프로젝트의 중앙 컨테이너 역할

Board.js : localstorage에 저장돼있는 데이터들을 받아와 리스트로 뿌려주는 js

Card.js : 포스팅된 게시글을 나타내는 js



getFullTimeStamp : date 자료형을 받아와 yyyy.m.DD 오후 HH:MM 형태로 리턴해주는 함수

formatBytes : 파일의 사이즈를 바이트로 받아와 보기 쉬운 단위로 리턴해주는 함수

getList : localstorage의 데이터를 가져오는 함수

write : localstorage에 데이터를 추가하는 함수

likeIt : 포스팅된 게시글의 좋아요를 증가시켜주는 함수

getFullTimeStamp

formatBytes

getList

write

likeIt

```
export const getFullTimeStamp = (value) => {
  const dateTime = new Date(value);
  const years = dateTime.getFullYear();
  const month = dateTime.getMonth() + 1;
  const day = dateTime.getDate();
  const hours = dateTime.getHours();
  const min =
    dateTime.getMinutes() < 10
      ? "0" + dateTime.getMinutes()
      : dateTime.getMinutes();
  const AMOrPM = hours < 12 ? "오전" : "오후";
  return `${years}.${month}.${day} ${AMOrPM} ${hours}:${min}`;
};
```

value : date 타입

date 타입을 받아서
2022.6.10 오후 22:22과 같은 형식으로
리턴해준다.

getFullTimeStamp

formatBytes

getList

write

likeIt

```
export const formatBytes = (byte, decimal = 2) => {
  if (byte === 0) return "0 Byte";

  const k = 1024;
  const dm = decimal < 0 ? 0 : decimal;
  const sizes = ["Byte", "KB", "MB", "GB", "TB", "PB", "EB", "ZB", "YB"];

  const i = Math.floor(Math.log(byte) / Math.log(k));

  return parseFloat((byte / Math.pow(k, i)).toFixed(dm)) + " " + sizes[i];
};
```

value : 숫자 타입

Decimal
~째 자리, 없으면 2숫자 타입을 받아서 ~KB, ~MB 등 보기 좋은 단위로 바꿔준다.

getFullTimeStamp

formatBytes

getList

write

likeIt

```
export const getList = () => {
  const _items = localStorage.getItem("kangzoom-post");
  return JSON.parse(_items);
};
```

localStorage에서 “kangzoom-post”라는 이름의 데이터를 가져오고,
가져온 것을 JSON.parse 함수를 통해 객체 형태로 보내준다.

getFullTimeStamp

formatBytes

getList

write

likeIt

```
export const write = (data) => {
  const tmp = localStorage.getItem("kangzoom-post");
  const prev = tmp ? JSON.parse(tmp) : [];
  const next = {
    id: new Date().getTime(),
    ...data,
    like: 0,
    view: 0,
    createDate: getFullTimeStamp(new Date()),
    image: "./image/1.jpg",
  };
  const result = JSON.stringify([...prev, next]);

  console.log(prev);
  localStorage.removeItem("kangzoom-post");
  localStorage.setItem("kangzoom-post", result);

  return { result };
};
```

data : {
  title : string(제목)
  content : string(내용)
}

localstorage에 kangzoom-post라는 이름의 데이터를 받아와
기존 데이터 삭제, 기존 데이터와 새로 받아온 data를 합쳐서 저장한다.
이번 프로젝트는 React에 학습 목적을 두고
image 업로드, 날짜, 게시글의 id 등 백엔드와 연관된 속성들은 임의의 값으로 저장했다.

getFullTimeStamp

formatBytes

getList

write

likeIt

```
export const likeIt = (id) => {
  const _items = JSON.parse(localStorage.getItem("kangzoom-post"));
  const target = _items.filter(x => x.id === id)[0]
  target.like = target.like + 1;

  const result = [
    ..._items.filter(x => x.id !== id),
    target
  ]

  localStorage.removeItem("kangzoom-post");
  localStorage.setItem("kangzoom-post", JSON.stringify(result));
  return false;
};
```

id : 게시글의 id

받아온 id에 해당하는 게시글을 찾아 좋아요 수를 1씩 증가시켜서 localStorage에 저장한다.

localStorage

localStorage는 컴퓨터별로, 브라우저별로 저장할 수 있다. 따라서 database처럼 어디에서 접속을 하든 같은 데이터를 받아올 수는 없다.

또 localStorage는 개발자 도구의 application - localStorage에서 누구든지 확인이 가능하므로 가능한 보안적인 이슈가 없는 데이터를 저장한다.
유저에 대한 직접적인 정보를 저장하기에는 보안이 너무 취약하다.

localStorage는 간단히 setItem, getItem 2가지로도 충분히 활용히 가능하다.
setItem을 통해 아래와 같이 데이터를 저장할 수 있다.

(작업자가 따로 정의를 하지 않아도 어디서든 호출할 수 있다.)
localStorage.setItem(“react”, “library”);

getItem을 통해 아래와 같이 데이터를 불러올 수 있다.

console.log(“로컬 스토리지”, localStorage.getItem(“react”));
// 로컬 스토리지 library

가장 중요한 것은 localStorage는 문자열 형태만 저장이 된다. 따라서 내가 사용하고 싶은 데이터는 저장하기 위해 별도의 처리를 해주어야 한다.
만약 문자열이 아닌 배열, 객체 등등을 저장하고 getItem으로 불러오면 “[object Object]”와 같이 출력된다.

```
const data = [
    {
        title : “제목”,
        content : “내용”,
        …
    },
    …
];
```

localStorage.setItem(“react”, JSON.stringify(data));

이렇게 stringify로 저장된 데이터를 다시 가져와 사용해야 할 경우, 아래처럼 처리한 뒤에 사용이 가능하다.

JSON.parse(localStorage.getItem(“react”));

Material UI Library

npm

To install and save in your `package.json` dependencies, run the command below using **npm**:

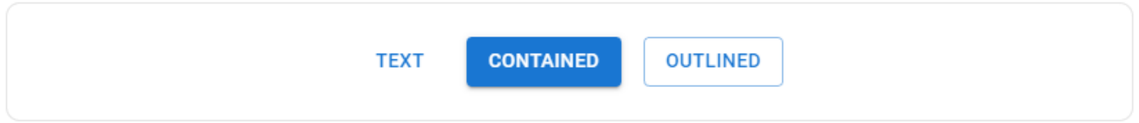
```
npm install @mui/material @emotion/react @emotion/styled
```

Or **yarn**:

```
yarn add @mui/material @emotion/react @emotion/styled
```

Basic button

The `Button` comes with three variants: text (default), contained, and outlined.




```
import * as React from 'react';
import Stack from '@mui/material/Stack';
import Button from '@mui/material/Button';

export default function BasicButtons() {
  return (
    <Stack spacing={2} direction="row">
      <Button variant="text">Text</Button>
      <Button variant="contained">Contained</Button>
      <Button variant="outlined">Outlined</Button>
    </Stack>
  );
}
```

https://mui.com/material-ui/getting-started/installation/

https://mui.com/material-ui/react-button/

package를 npm을 통해 설치하고, import 한 뒤 적절한 속성과 태그를 사용하면 손쉽게 예쁜 버튼을 가져올 수 있다.

버튼 외에도 많은 UI들을 라이브러리로서 제공하고 있다.

Recoil

Recoil 시작하기

React 애플리케이션 생성하기

Recoil는 React를 위한 상태 관리 라이브러리다. 따라서 Recoil를 사용하기 위해서는 React가 설치되어 있어야 한다. React 애플리케이션을 시작하는 가장 쉽고 추천하는 방법은 Create React App을 사용하는 것이다.

```
npx create-react-app my-app
```

> `npx`는 npm 5.2+ 이상에서 함께 제공되는 패키지 실행 도구다. 오래된 버전의 npm은 이 설명를 보면된다.

Create React App을 설치하는 더 많은 방법은 공식 문서를 보면된다.

RecoilRoot

recoil 상태를 사용하는 컴포넌트는 부모 트리 어딘가에 나타나는 `RecoilRoot` 가 필요하다. 루트 컴포넌트가 `RecoilRoot`를 넣기에 가장 좋은 장소다.

```
import React from 'react';
import {
  RecoilRoot,
  atom,
  selector,
  useRecoilState,
  useRecoilValue,
} from 'recoil';

function App() {
  return (
    <RecoilRoot>
      <CharacterCounter />
    </RecoilRoot>
  );
}
```

https://recoiljs.org/ko/docs/introduction/getting-started/

Atom

Atom은 **상태(state)**의 일부를 나타낸다. Atoms는 어떤 컴포넌트에서나 읽고 쓸 수 있다. atom의 값을 읽는 컴포넌트들은 암묵적으로 atom을 구독한다. 그래서 atom에 어떤 변화가 있으면 그 atom을 구독하는 모든 컴포넌트들이 재 렌더링 되는 결과가 발생할 것이다.

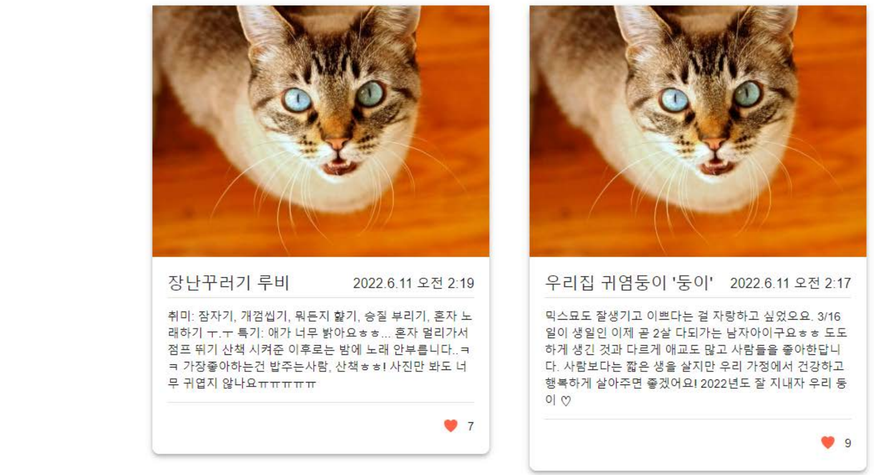
```
const textState = atom({
  key: 'textState', // unique ID (with respect to other atoms/selectors)
  default: '', // default value (aka initial value)
});
```

```
function TextInput() {
  const [text, setText] = useRecoilState(textState);

  const onChange = (event) => {
    setText(event.target.value);
  };

  return (
    <div>
      <input type="text" value={text} onChange={onChange} />
      <br />
      Echo: {text}
    </div>
  );
}
```

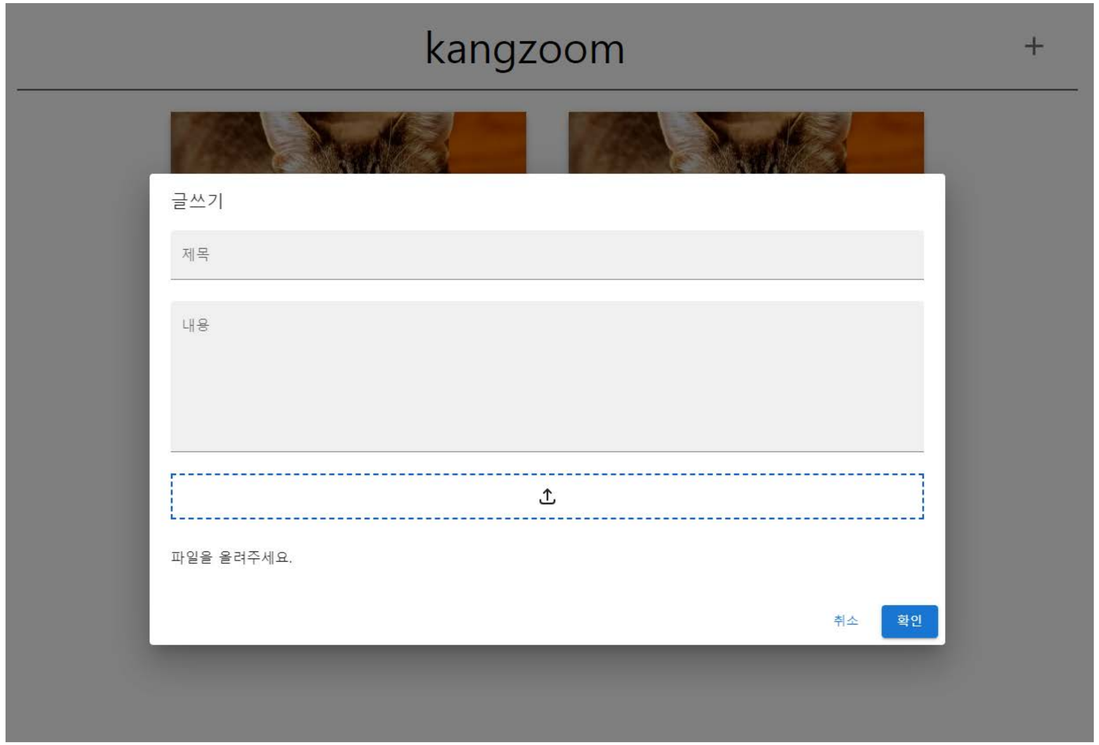
react에서 컴포넌트가 나뉘어져서 관리된다.
만약 컴포넌트가 서로 멀리 떨어져있으면 같은 변수를 서로 동시에 활용하기가 힘들다.
이런 점을 보완해주는 라이브러리이다.
Atom이라는 것을 이용해서 변수에 저장해놓으면, 어떤 컴포넌트든 import해서 사용할 수 있다.

메인 UI/글쓰기 UI

- 글쓰기 후 메인 UI



- 좋아요 클릭 UI



- 글쓰기용 UI