-- components --

<LandingMap.js>

- useEffect, useState 를 "react"로 부터 받아온다

- Input 은 "reactstrap"으로 부터 받아온다

- Label, Row, Col, FormGroup 는 "reactstrap"으로 부터 받아온다

- appkey 는 "../config/appkey.json" 파일로 부터 받아온다

마커 이미지를 생성하고, 각 type 에 해당하는 이미지들이 보여지도록 한다.

카카오 맵에서 위도, 경도를 얻어와 지도에 표시가 되게 해준다.

표시가 되기 위해 마커를 생성하고, 표시할 위치와 이미지를 정해준다.

selected option 으로 선택된 식사, 카페, 술집 별로 알맞은 위치에 마커가 찍히도록 한다.

다른 경로에서도 사용할 수 있도록 export 해준다.

즉, 누군가가 내 파일을 불러올 때(import) LandingMap obj 를 주겠다는 의미

<Loding.js>

- React 는 'react'로 부터 받아온다

- Spinner 은 'reactstrap'로 부터 받아온다 -> 데이터가 아직 로딩되어 있지 않은
상태라면 스핀이 돌면서 데이터가 로드될 때까지 실행된다.

로딩 할 때 의 css 를 코딩

다른 파일에도 불려와지도록 해야하기 때문에 export 해준다.

즉, 누군가가 내 파일을 불러올 때(import) Loading obj 를 주겠다는 의미

<LoginLink.js>

- React 는 'react'로 부터 받아온다

- Container, Jumbotron, Button 은 reactstrap 으로 부터 받아온다

로그인 관련 css 코딩

다른 경로에서도 사용할 수 있도록 export 해준다.

즉, 누군가가 내 파일을 불러올 때(import) LoginLink obj 를 주겠다는 의미

<Map.js>

- React, useEffect 를 'react'로 부터 받아온다

- appkey 는 "../config/appkey.json" 파일로 부터 받아온다

- map 관련 css 코딩

카카오 맵에서 지도 API 를 받아와 위도, 경도를 마커와 마커위치에 할당해준다

다른 경로에서도 사용할 수 있도록 export 해준다.

즉, 누군가가 내 파일을 불러올 때(import) Map obj 를 주겠다는 의미

<MealCard.js>

- React, useState 를 'react'로 부터 받아온다

- Card, CardBody, CardTitle, CardText, CardImg, CardFooter, Button 은 'reactstrap'으로 부터 받아온다

- Modal, ModalHeader, ModalBody, ModalFooter 는 'reactstrap'으로 부터 받아온다

- Map 은 './Map'로 부터 받아온다

- './MealCard.css'를 받아온다

- faAngleRight 는 "@fortawesome/free-solid-svg-icons"로 부터 받아온다

- FontAwesomeIcon 는 "@fortawesome/react-fontawesome"로 부터 받아온다

string 을 object 로 변환해주기 위해 JSON.parse(localstorage key)를 통해 파싱해준다

mypick 페이지에서 user 을 객체로 받아와 user 가 없으면 로그인이 필요하다는 문구가 나오도록 한다.

fetch 함수를 사용해 get 메소드로 네트워크 요청을 보내고 올바른 인증을 거친 user 인지 GET Method 를 통해 확인한다.

올바른 user 라고 판단될 경우 서버에서 응답 헤더를 받자마자 fetch 호출 시 반환받은 response 를 result 파라미터로 받아 MealCard 를 저장하는 POST Method 를

전달한다.

pick 을 클릭했을 때 해당 user mypick 안에 있는 카드 존재여부에 따라 존재를 하지 않으면

'MyPick 에 담겼습니다.' 라는 문구가 뜨며 추가가 되고 이미 존재하면 '이미 MyPick 에 존재합니다.' 라는 문구가 뜬다.

올바른 user 라고 판단되지 않는다면 mypick 페이지에서 '로그인이 필요합니다.'라는 문구가 뜨게 한다.

mealcard 에 대한 css 코딩

다른 경로에서도 사용할 수 있도록 export 해준다.

즉, 누군가가 내 파일을 불러올 때(import) MealCard obj 를 주겠다는 의미

<NavBar.js>

- React, useState 를 "react"로 부터 받아온다

- Container, NavbarText 를 "reactstrap"으로 부터 받아온다

- Collapse, Navbar, NavbarToggler, NavbarBrand, Nav, NavItem, NavLink 를 "reactstrap"으로 부터 받아온다

NavBar 에 해당하는 고민사거리, About, Menu, MyPick, WebProject css 코딩

<PickedCard.js>

- React, useState 를 'react'로 부터 받아온다

- Card, CardBody, CardTitle, CardText, CardImg, CardFooter, Button 를 'reactstrap'으로 부터 받아온다

- Modal, ModalHeader, ModalBody, ModalFooter 를 'reactstrap'으로 부터 받아온다

- Map 을 from './Map'으로 부터 받아온다

- './PickedCard.css'를 받아온다

- faAngleRight 를 "@fortawesome/free-solid-svg-icons"로 부터 받아온다

- FontAwesomeIcon 를 "@fortawesome/react-fontawesome"로 부터 받아온다

string 을 object 로 변화 해주기 위해 JSON.parse(localstorage key)를 해준다

mypick 페이지에서 user 을 객체로 받아와 user 의 토큰이 존재하지 않는다면 '토큰이 만료되었습니다.'라는 문구가 나오도록 한다.

fetch 함수를 사용해 get 메소드로 네트워크 요청을 보내고 유효한 토큰을 받아온다.

서버에서 응답 헤더를 받자마자 fetch 호출 시 반환받은 토큰이 유효하다면 fetch 함수를 사용해 post 메소드로 /api/delete 에 user 정보를 전달한다. Response 의 body 부분에는 cardid 에 props.id 도 같이 실어서 보낸다.

해당 user 의 mypick 페이지에서 pick 된 카드들에 delete 버튼을 클리하면 '삭제완료'라는

문구가 뜨면서 삭제가 된다. delete 에 실패하면 'delete error'가 나오게 한다.

card 에서 viewmore 에 대한 css 코딩

다른 경로에서도 사용할 수 있도록 export 해준다.

즉, 누군가가 내 파일을 불러올 때(import) PickedCard obj 를 주겠다는 의미

<UserCards.js>

- React, useState, useEffect 를 "react"로 부터 받아온다

- Container, Row, Col, Button 를 "reactstrap"로 부터 받아온다

- PickedCard 를 "../components/PickedCard"로 부터 받아온다

user 의 pick 카드 프론트부분을 나타낸것.

pick 카드에는 username, picks 변수가 필요하다.

React Redux 를 활용해 username, picks 에 대한 변수들을 초기화해준다.

user 의 정보를 가져오는 API 를 구현한다. /api/mypicks 로 GET Method 를 보내 pick 애트리뷰트의 데이터를 요청한다.

localStorage 에서 user 를 가져와 인증작업을 거친후 HTTP GET 을 통해 user 정보를 response 에 갱신하고 pick 또한 setPicks 를 통해 값을 갱신한다.

로그아웃이 되면 LogoutHandler 함수를 통해 localStorage 에 있는 값들을 제거해준다.

다른 경로에서도 사용할 수 있도록 export 해준다.

-- rootdirectory --

<App.js>

- React 는 "react"로 부터 받아온다

- LandingPage 는 './pages/LandingPage'로 부터 받아온다.

- AboutPage 는 './pages/AboutPage'로 부터 받아온다.

- MenuPage 는 './pages/MenuPage'로 부터 받아온다.

- SigninPage 는 './pages/SigninPage'로 부터 받아온다.

- SignupPage 는 './pages/SignupPage'로 부터 받아온다.

- MypickPage 는 './pages/MypickPage'로 부터 받아온다.

App.js 는 모든 페이지, 컴포넌트들에 대한 라우터 역할을 한다. 가장 상위 페이지로 그 안에 LandingPage, AboutPage, MenuPage, SigninPage, SignupPage, MypickPage 각각의 컴포넌트들이 담겨있다.

index.js 에서도 사용할 수 있도록 export 해준다.

<setupProxy.js>

react 는 http://localhost:3000

nodejs 는 http://localhost:5000 으로 실행한다.

서로 다른 origin 을 가지기에 react -> nodejs 로 CORS 정책에 의해 올바른 요청이 가지 못한다.

그래서 이러한 문제를 proxy 를 두어 처리한다.

axios 사용시 /posts 라고하면 http://localhost:5000/posts 로 가게 된다.

<server.js>

-- 서버생성을 위한 준비 --

보안을 위해 bcrypt 모듈을 사용한다.

다른 암호화 된 비밀번호를 반환하기 위해 saltRounds 사용한다.

express 서버를 생성하고 port 번호는 5000 으로 웹서버를 연다.

json_data.json(음식점 크롤링)을 읽어와 dataBuffer 변수에 할당해준다.

"./jwt_key.json" 파일에 있는 key 를 jwt_key 변수에 할당해준다.

객체화 시켜준 jwt_key 를 jwt_secret_key 변수에 할당해준다.

-- db connection --

"./database.json"를 data 변수에 할당해준다.

객체화 시킨 data 를 JSON 형태로 파싱하여 conf 변수에 할당해준다.

node.js 와 mysql 을 연동시켜준다.

createConnection 메소드의 인자로 전달되는 객체에 자신의 데이터베이스 정보(유저명, 패스워드, db 등)을 입력하여야 한다.

Node.js 로 서버를 동작하다 보면 DB 와의 연결이 끊어질 때가 있다 이때 handleDisconnect 함수를 통해 DB 에 재연결을 해줄 수 있는 소스 코드를 작성해준다.

body 의 userId 값과 password 를 읽어내기 위해 bodyparser 을 사용합니다.

bodyparser 은 middleware 종류 중 하나로 사용자가 웹사이트로 전달하는 정보들을 검사해준다.

(request 정보에서 form 이나 json 형태로 된 body 를 검사)

-- datas 전달 --

브라우저에서 request 가 왔을 때 서버에서 어떤 작업을 할 지 Router 를 통하여 설정한다.

app.get("/api/datas", (req, res) => {});는 /api/datas 경로에 대한 GET 요청을 처리하기 위한 코드이다.

-- signup --

app.post 를 사용해 /api/signup 경로에서 POST 요청에 응답하게 한다.

SELECT * FROM 을 사용해 username 에서 name 을 가져온다.

mysql 테이블쿼리를 연결하여 username 을 조회한다.

username 이 이미 존재하면 statuscode 함수를 생성해서 code 는 400, 메시지는 'user exist' 로 뜨게한다.

그렇지 않으면 이름과 비밀번호를 기반으로 데이터를 삽입하고 비밀번호의 경우 bcrypt 를 통해 암호화 과정을 거친다. 만약 쿼리 전송 도중 에러가 발생할 경우 code 는 400, 메시지는 "error"라고 뜨게한다.

성공했을 경우 code 는 200, 메시지는 "success"가 뜨게 하고 데이터베이스에 정상적으로 회원 정보가 삽입됨을 확인할 수 있다.

-- signin --

app.post 를 사용해 /api/signin 경로에서 POST 요청에 응답하게 한다.

SELECT * FROM 을 사용해 username 에서 name 과 pw 를 가져온다.

mysql 테이블쿼리를 연결하여 user 의 name 과 pw 를 조회한다.

sql_usercheck 쿼리문을 통해 존재하지 않는 user 라면 데이터의 길이가 0 이 된다. 따라서 user 가 존재하지 않을 경우 statuscode 함수를 생성해서 code 는 400, 메시지는 'user does not exist' 를 전달한다.

sql_usercheck 로 받아온 데이터의 길이가 0 이 아니라면 DB 에 user 가 존재하는 것이다. 그렇다면 sql 변수에 작성해둔 쿼리 문을 DB 에 전송한다. token 기반으로 회원 인증 절차를 밟는다. user 정보에 token 을 추가하여 response 로 보낸다.

비밀번호로 조회하고 틀렸을 경우 code 는 400, 메시지는 "invalid password"라고 뜨게한다.

-- 토큰을 기반으로 회원인증 구현 --

JWT 를 이용해 토큰을 관리해준다.

JWT 는 인증정보를 암호화하여 url 형식으로 전달해 주는 토큰이다.

노드 모듈인 express-jwt 는 두 가지 역할을 수행한다.

(1) 인증된 클라이언트의 엑세스 토큰을 디코딩하고

(2) 인증된 유저정보를 req.user 에 저장한다.

내부적으로는 jsonwebtoken 모듈을 사용하여 .decode() 함수를 호출한다.

req.header 에 토큰을 설정해준다.

-- /api/mypicks 구현 --

app.get("/api/mypicks", (req, res) => {});는 /api/mypicks 경로에 대한 GET 요청을 처리하기 위한 코드

user 변수에 jwt 를 이용한 인증정보를 할당해준다.

username 변수에는 user 의 name 을 할당해준다.

dataBuffer(음식점 크롤링 데이터)를 decode 하여 temp 변수에 할당해준다. (글씨체를 위한 디코딩 과정)

user 테이블에서 해당 username 이 갖고 있는 pick 애트리뷰트를 조회한다.

만약 pick 의 길이가 0 이면 statuscode 를 생성해 code 는 401, 메시지는 'card 0'으로 뜨게 한다.

만약 pick 값 자체가 null 이면 statuscode 를 생성해 code 는 401, 메시지는 'card 0'으로 뜨게 한다.

앞서 조건문에서 필터링이 되지 않았다면 해당 username 의 pick 애트리뷰트에는 값이 존재하는 것이다.

pick 의 내용은 ex) "1,2,4,5..."과 같은 형태로 이루어져있다. 여기서 숫자는 user 가 원하는 식당을 픽했을 때 그 식당의 id 값을 의미한다. 따라서 user_picks 변수에 ','를 제거한 배열로 할당해준다.

기존에 pick 내용은 문자열로 이루어져 있었기 때문에 parseInt 함수를 통해 해당 문자열을 정수형으로 변환해준다.

id 에 맞는 음식점 정보를 결과변수 result 에 할당 후 object 형태로 response 를 전달해준다.

-- /api/pick 구현 --

현재 로그인한 user 를 디코딩하여 name 에 해당하는 데이터를 username 변수에 할당한다.

로그인한 user 의 pick 애트리뷰트를 조회하는 쿼리를 날린다.

만약 데이터가 없다면 401 호출과 함께 'card exist' 메시지를 보낸다.

만약 해당 user 가 존재한다면, pick 에 값이 존재하는지 확인한다. 값이 존재하지 않는다면 newPick 변수에 카드번호와 ','를 할당한다. UPDATE 쿼리를 이용하여 pick 애트리뷰트의 값을 갱신해준다. 성공적으로 갱신되었다면 DB 에는 "1, "과 같은 형태로 저장이 되고 서버에서는 200 호출과 'insertion success' 메시지를 전송한다.

만약 값이 이미 존재한다면, 기존의 값들을 ','를 제거한 후 배열로 만들어 user_picks 에 할당한다. 만약 삽입한적이 있다면 flag 변수를 false 로 할당한 후 401 호출을 보낸다. 하지만 기존 데이터에도 존재하지 않은 id 값이라면 앞서 방법과 마찬가지로 기존 배열 데이터와 새로운 id + ','를 합하여 pick 애트리뷰트에 갱신작업을 해준다. 정상적으로 응답했다면 200 호출과 'insertion success' 메시지를 전송한다.

— /api/delete 구현 —

로그인한 유저가 먼저 USER 테이블에 존재하는지 확인한다.

존재하는 유저라면 pick 애트리뷰트가 null 값인지 확인한다.

만약 pick 값이 null 값이라면 삭제할 것이 없으므로 미리 401 을 호출해 동작을 멈춘다.

만약 클릭한 cardid 가 기존 pick 에 존재한다면 flag 를 true 로 갱신해준다.

만약 flag 가 true 라면 삭제할 데이터가 현재 user 의 pick 애트리뷰트에 존재한다는 것이다. 따라서 flag 를 할당하던 코드에서 따로 만들어둔 newPick 배열 데이터를 새롭게

갱신하여 삭제의 동작처럼 구현한다. 기존 데이터에 해당 id 만 빼고 새로운 데이터를 덮으쓰는 형태이다.

flag 가 false 라면 삭제할 데이터가 기존 데이터에 존재하지 않는 것이므로 401 code 를 호출한다.