Re_pl

프로젝트 보고서

이젠아카데미 풀스택 B - 4조

저희 프로젝트는요

실제 웹 서비스인

“진짜서울(https://jinjja-seoul.com)”의 클론코딩입니다.

유저들이 지도에서 장소를 검색하고,

자신이 추천하고픈 장소를 등록하며

후기와 사진을 남기는 등의 활동이 가능한 사이트입니다.

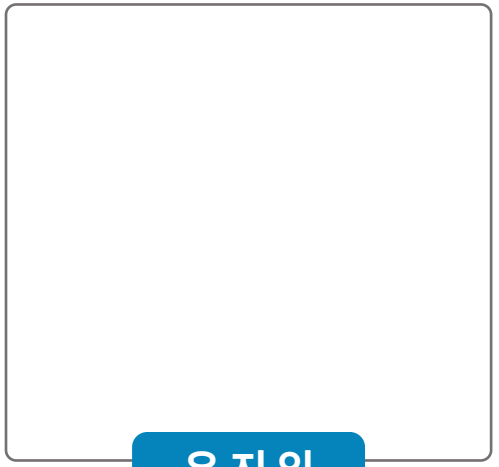
저희 팀원은요

권 채 림

로그인 및 마이페이지, 헤더
프론트엔드 React 및
JSON Server 연동,
Redux Slice 처리

서 보 선

게시판 및 후기 페이지 담당,
프론트엔드 React 및
전체 페이지 Node 연동,
MySQL DB 처리



유 지 인

지도 및 테마 페이지 담당,
프론트엔드 React 및
JSON Server 연동,
Redux Slice

장 윤 신

지도 및 후기 페이지 담당,
KAKAO API 연동
지도 렌더링 React 처리,
Axios Ajax 처리

저희 프로젝트에는요

Front-End

- HTML
- CSS
- JavaScript
- SASS
- React
- Redux

Back-End

- JavaScript
- MySQL
- Node.js

Collabo

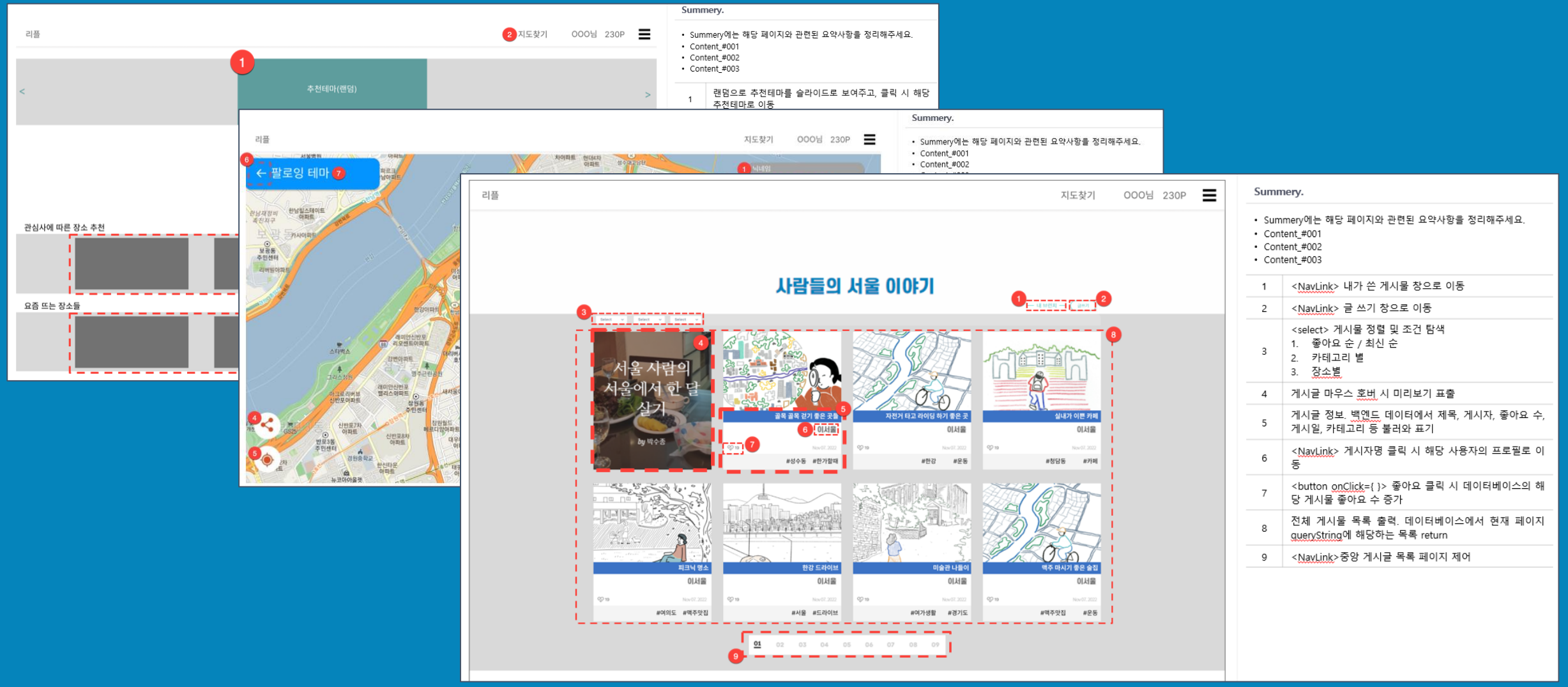
- Figma
- GitHub

Libraries

- react helmet async
- react router dom
- react modal
- react glider
- classnames
- react fontawesome
- ckeditor4
- dayjs
- axios
- lodash
- express
- nodemon
- dotenv
- cookie parser
- body parser
- mysql2
- mybatis mapper
- serve static
- multer
- winston
- ...

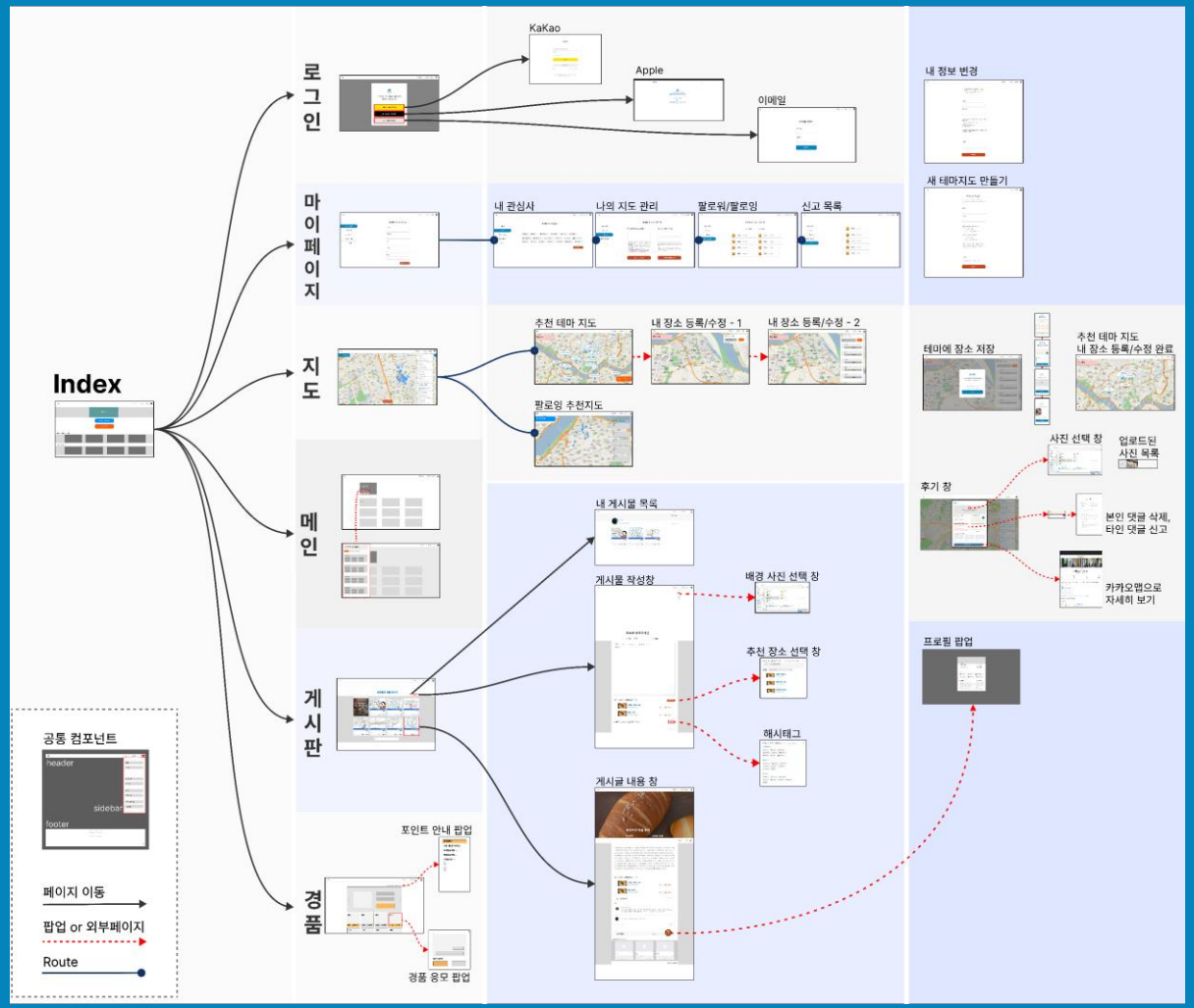
화면설계서

화면 구성을 위해 미리 작업했던 설계서의 일부분

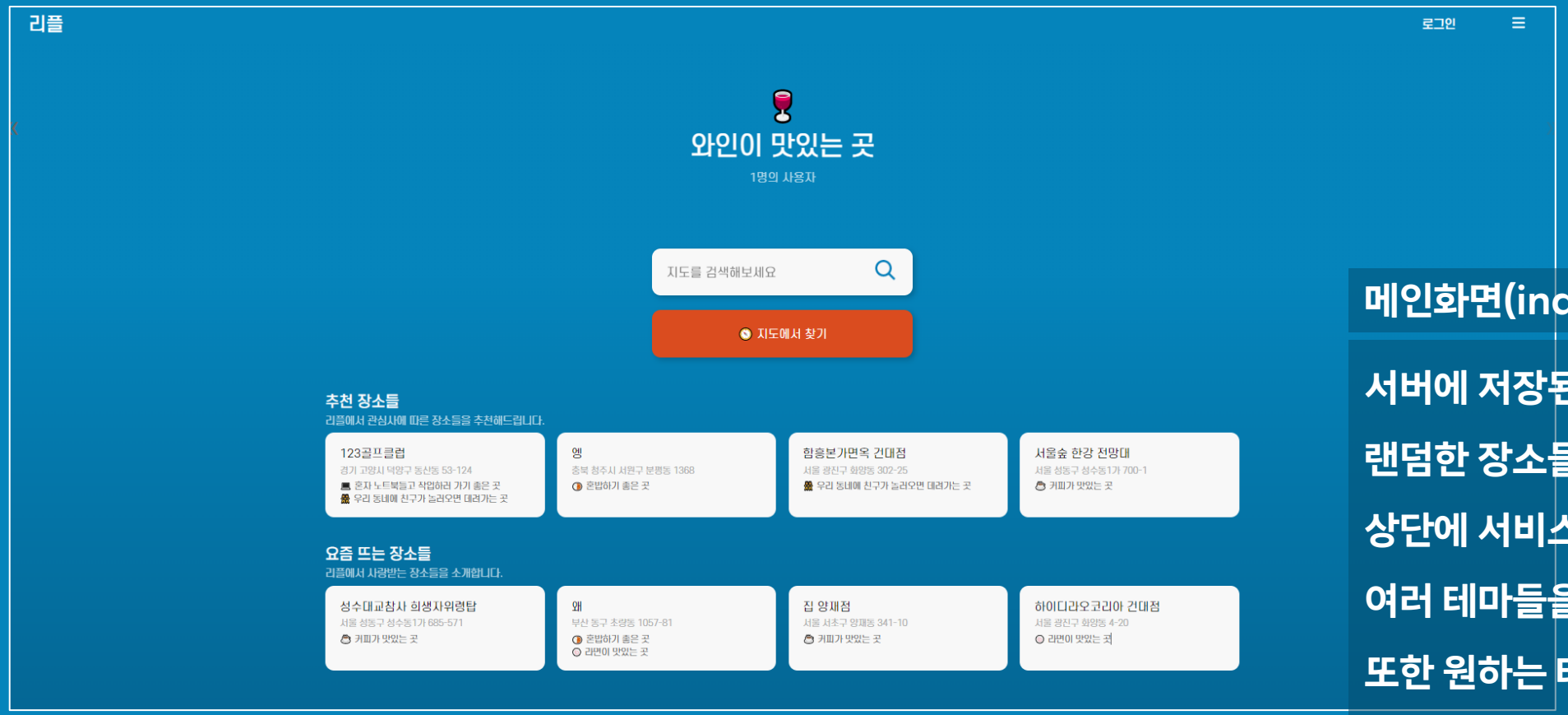


목업 및 마인드맵

피그마(Figma)를 통해 협업한 목업과 서비스 마인드맵



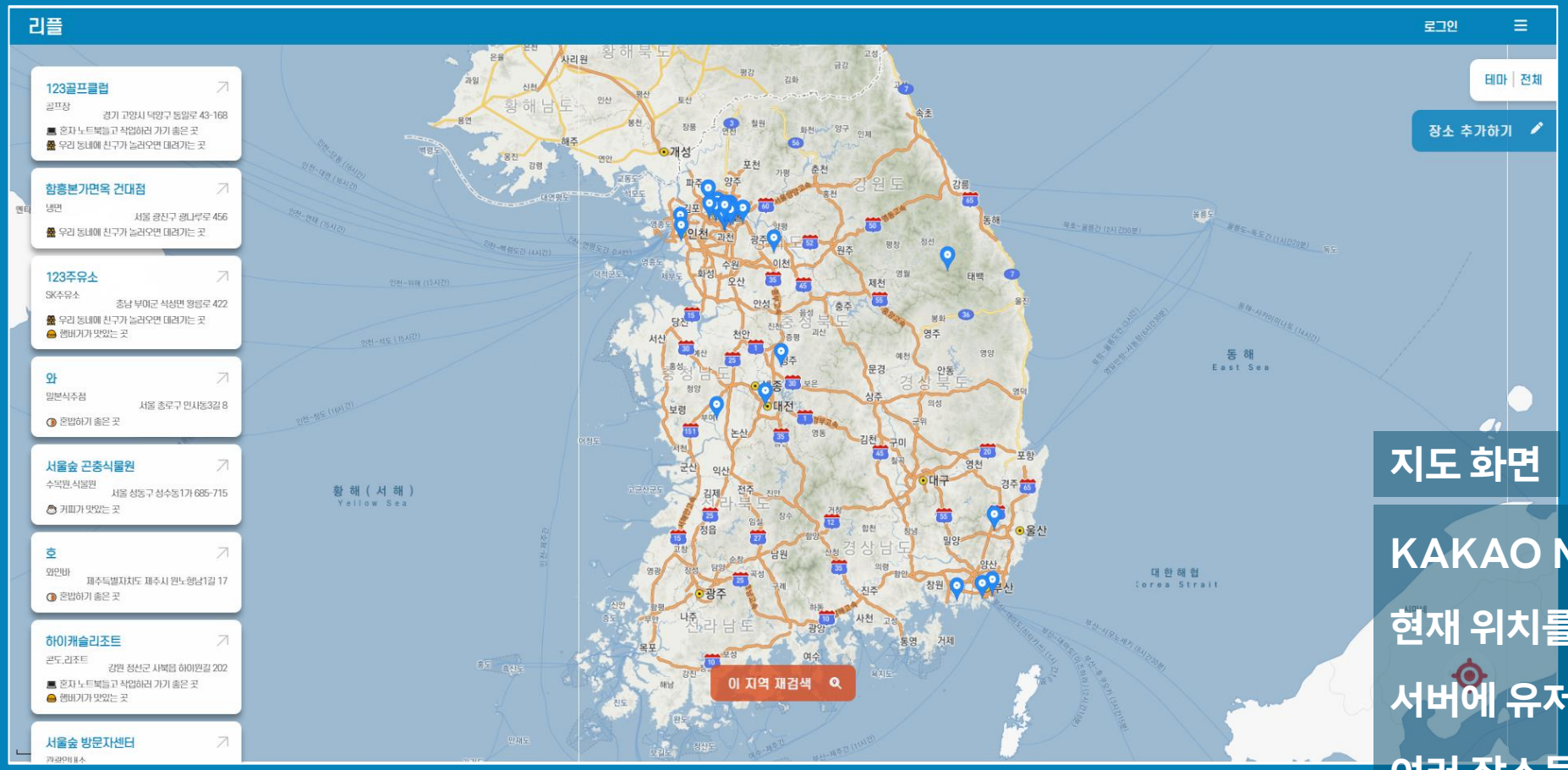
대표 화면별 기능소개



메인화면(index.html)

서버에 저장된 장소 데이터 중 랜덤한 장소들을 추천해주고, 상단에 서비스가 저장 중인 여러 테마들을 슬라이드로 노출, 또한 원하는 테마나 지도를 검색하는 기능을 제공합니다.

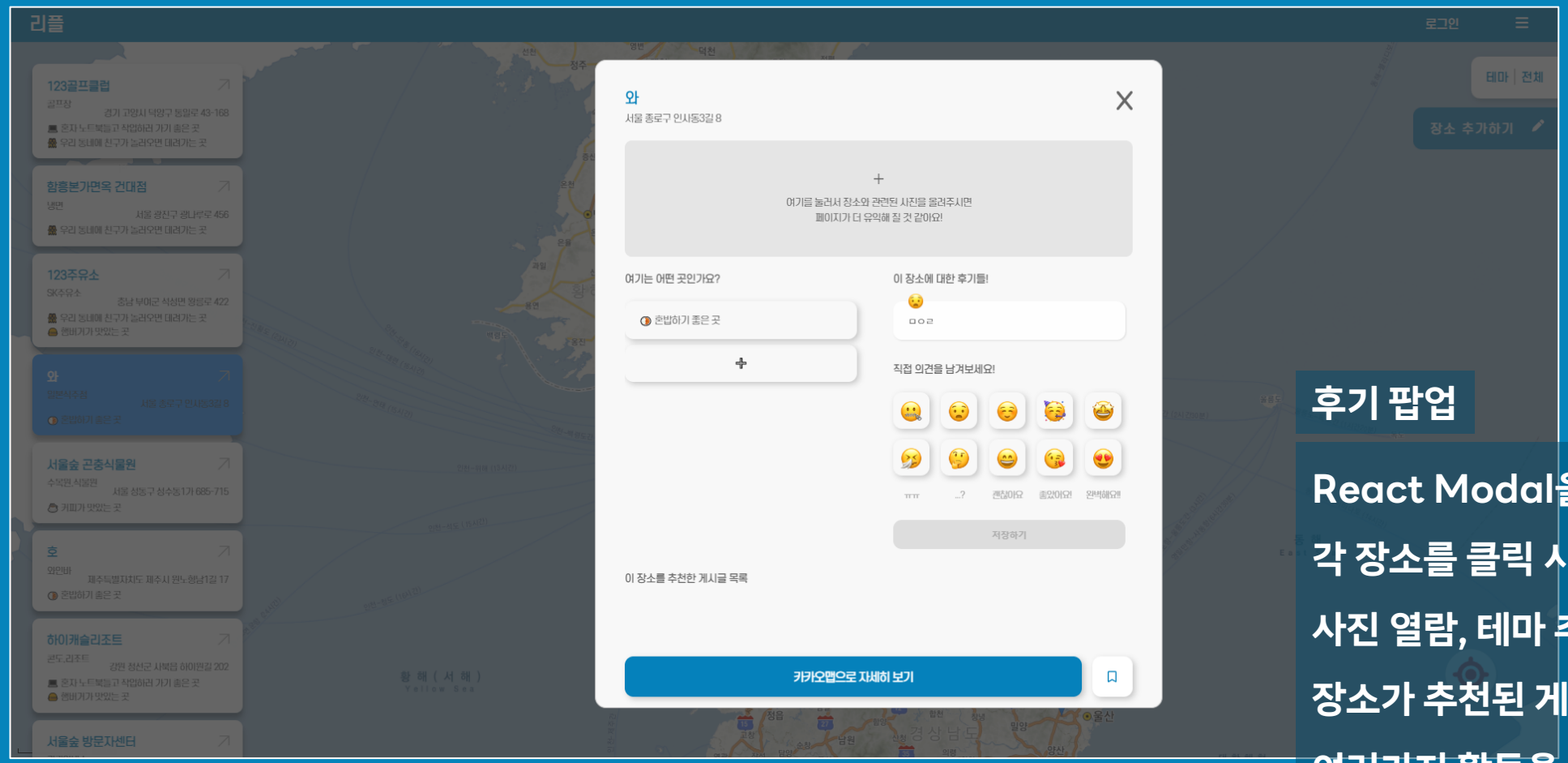
대표 화면별 기능소개



지도 화면

KAKAO MAP API를 연동,
현재 위치를 기반으로 한 지도에서
서버에 유저의 후기가 저장된
여러 장소들을 나열해 노출합니다.

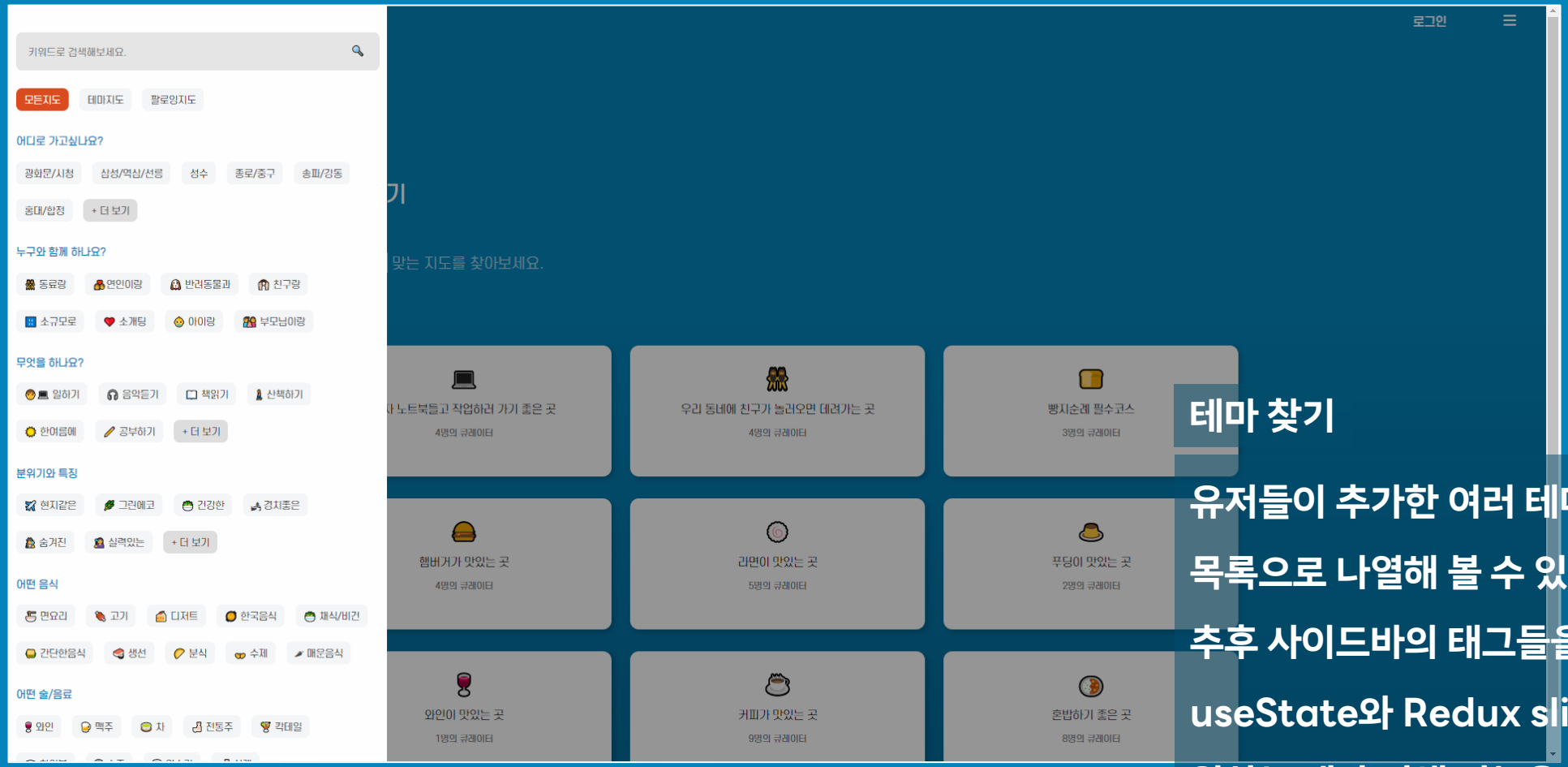
대표 화면별 기능소개



후기 팝업

React Modal을 이용한 컴포넌트로,
각 장소를 클릭 시 후기 작성과
사진 열람, 테마 추가하기,
장소가 추천된 게시물 이동 등
여러가지 활동을 제공합니다.

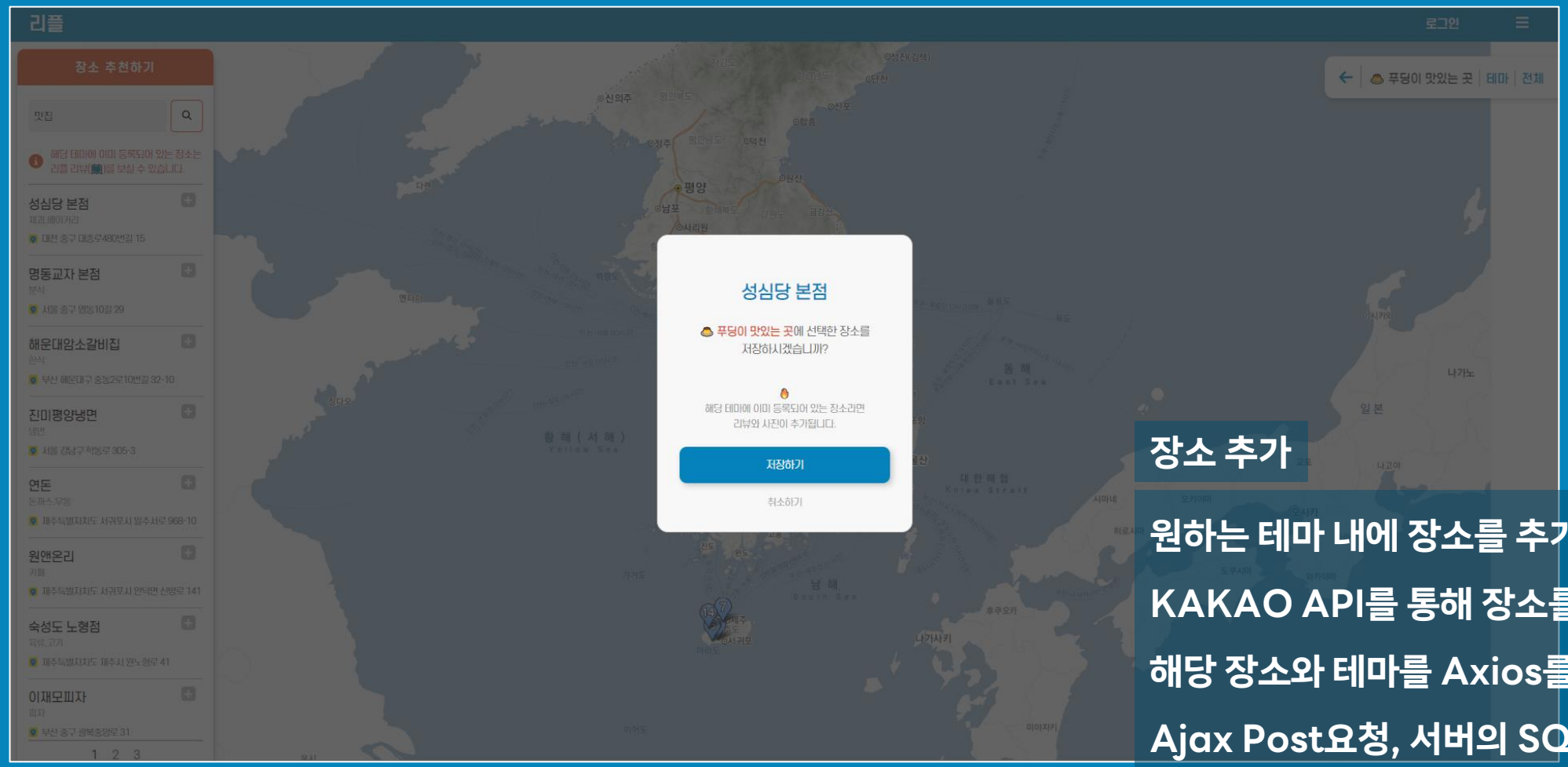
대표 화면별 기능소개



테마 찾기

유저들이 추가한 여러 테마를
목록으로 나열해 볼 수 있습니다.
추후 사이드바의 태그들을 선택할 시
useState와 Redux slide data slice를 통해
원하는 테마 검색 기능을 추가할 예정입니다.

대표 화면별 기능소개



장소 추가

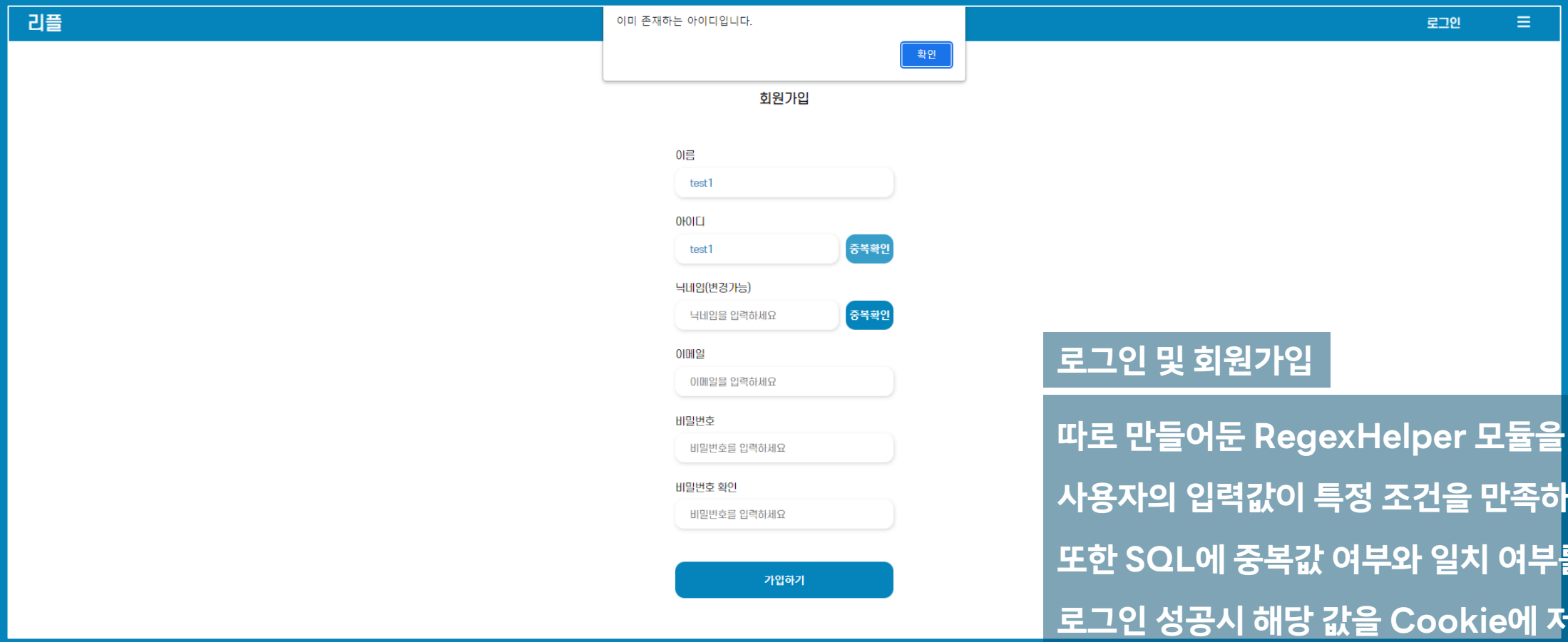
원하는 테마 내에 장소를 추가할 수 있습니다.

KAKAO API를 통해 장소를 검색 후,

해당 장소와 테마를 Axios를 통해

Ajax Post요청, 서버의 SQL DB에 저장합니다.

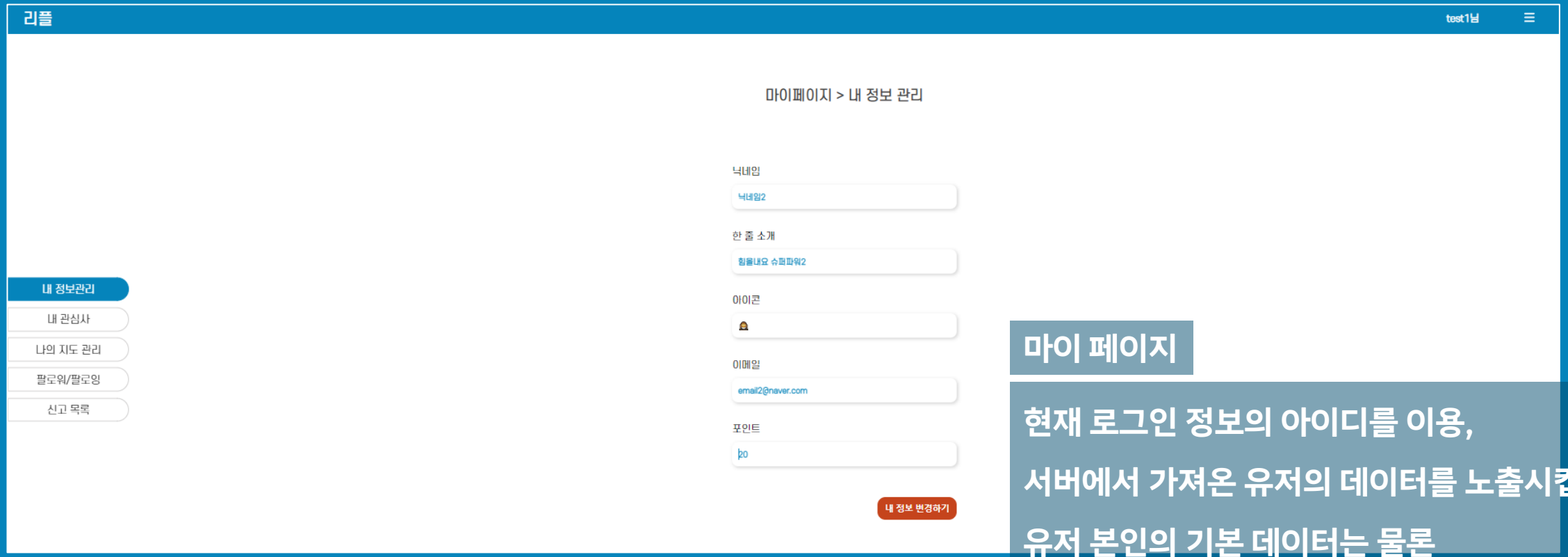
대표 화면별 기능소개



로그인 및 회원가입

따로 만들어둔 RegexHelper 모듈을 이용.
사용자의 입력값이 특정 조건을 만족하는지,
또한 SQL에 중복값 여부와 일치 여부를 판단한 뒤
로그인 성공시 해당 값을 Cookie에 저장해
타 페이지에서 로그인 정보를 이용합니다.

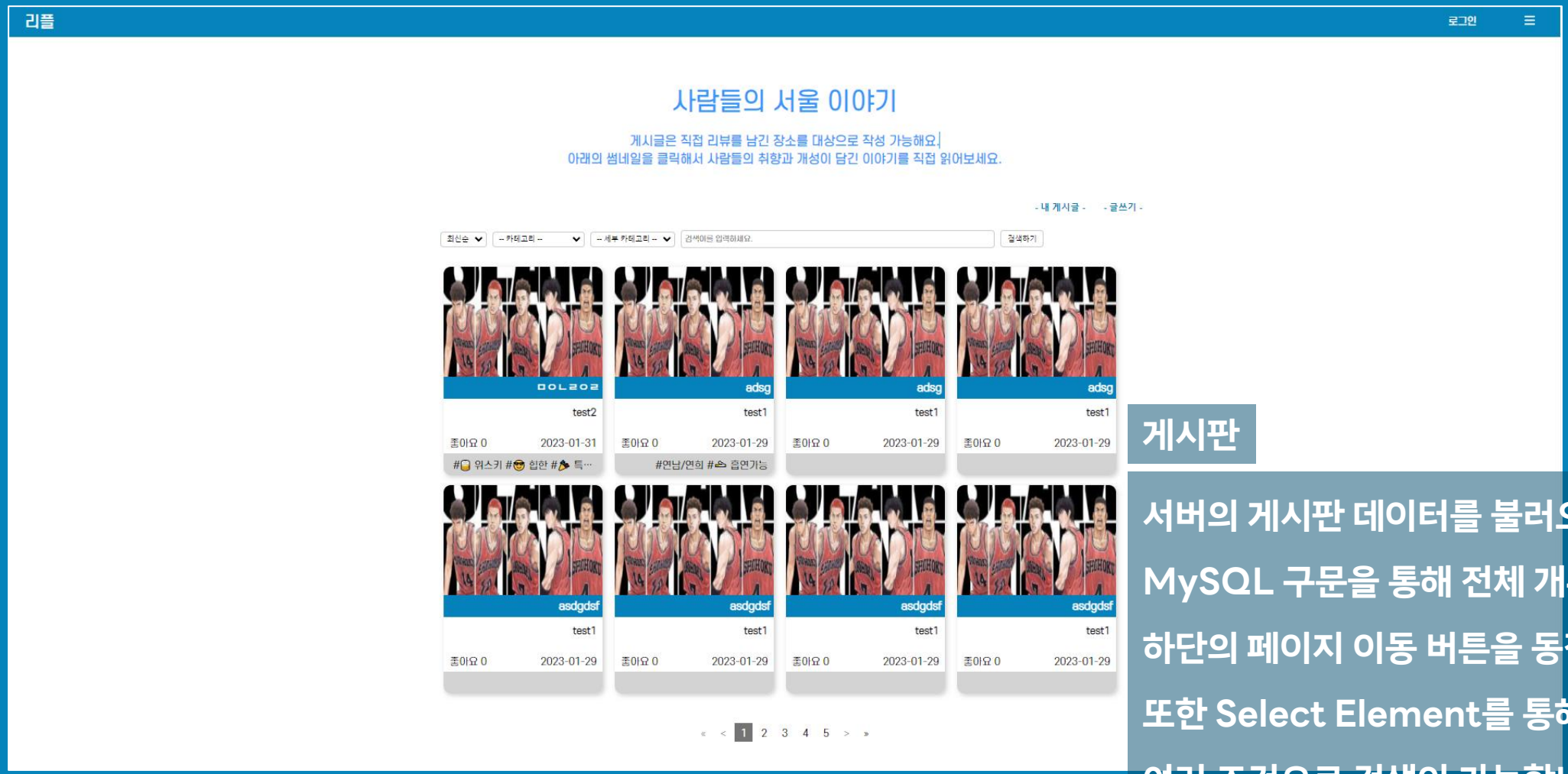
대표 화면별 기능소개



마이 페이지

현재 로그인 정보의 아이디를 이용,
서버에서 가져온 유저의 데이터를 노출시킵니다.
유저 본인의 기본 데이터는 물론
추천한 장소, 게시물 등 여러 페이지로 이동하거나
해당 데이터를 삭제할 수 있습니다.

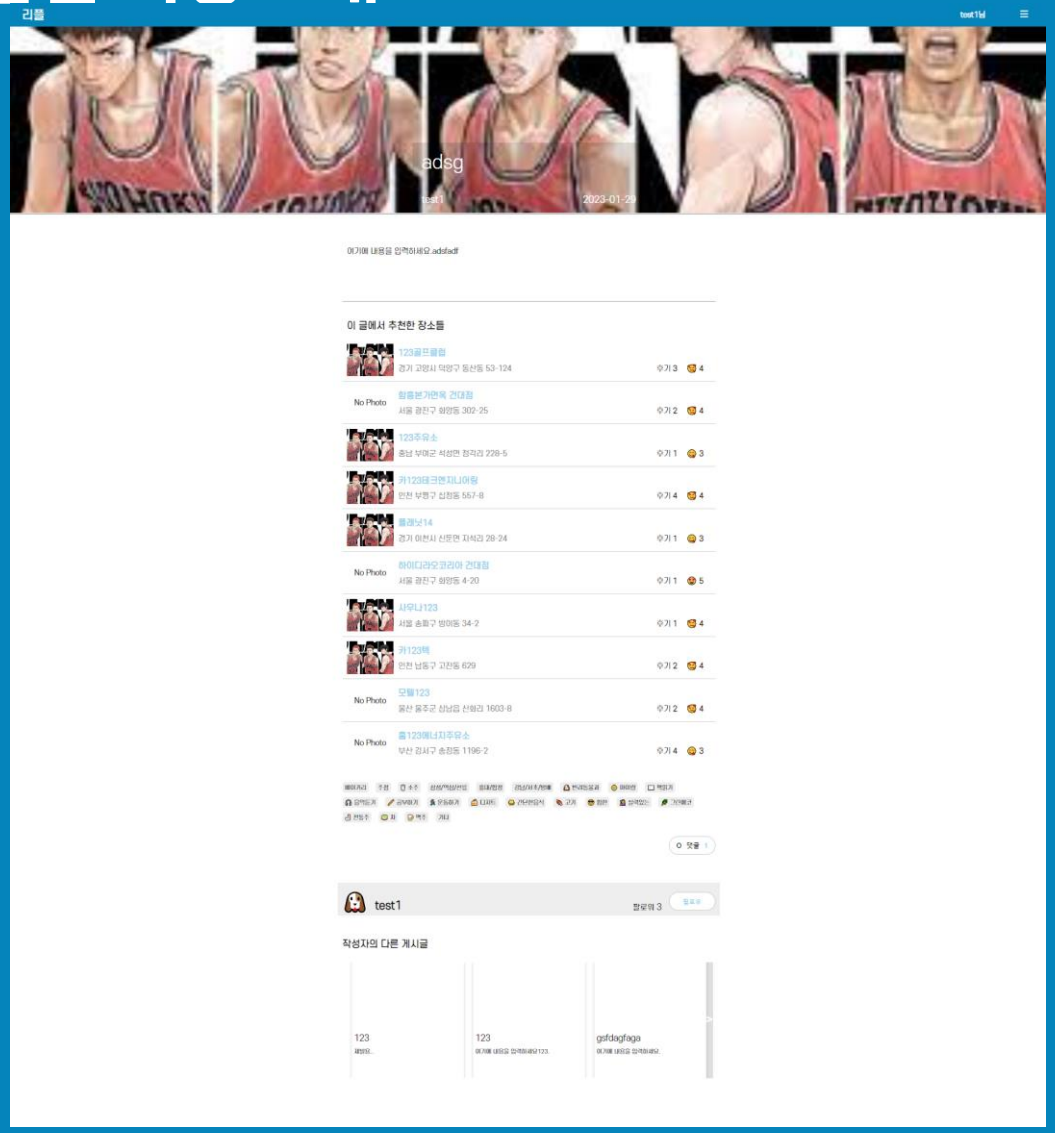
대표 화면별 기능소개



게시판

서버의 게시판 데이터를 불러오며,
MySQL 구문을 통해 전체 개수를 파악해
하단의 페이지 이동 버튼을 동적으로 생성합니다.
또한 Select Element를 통해
여러 조건으로 검색이 가능합니다.

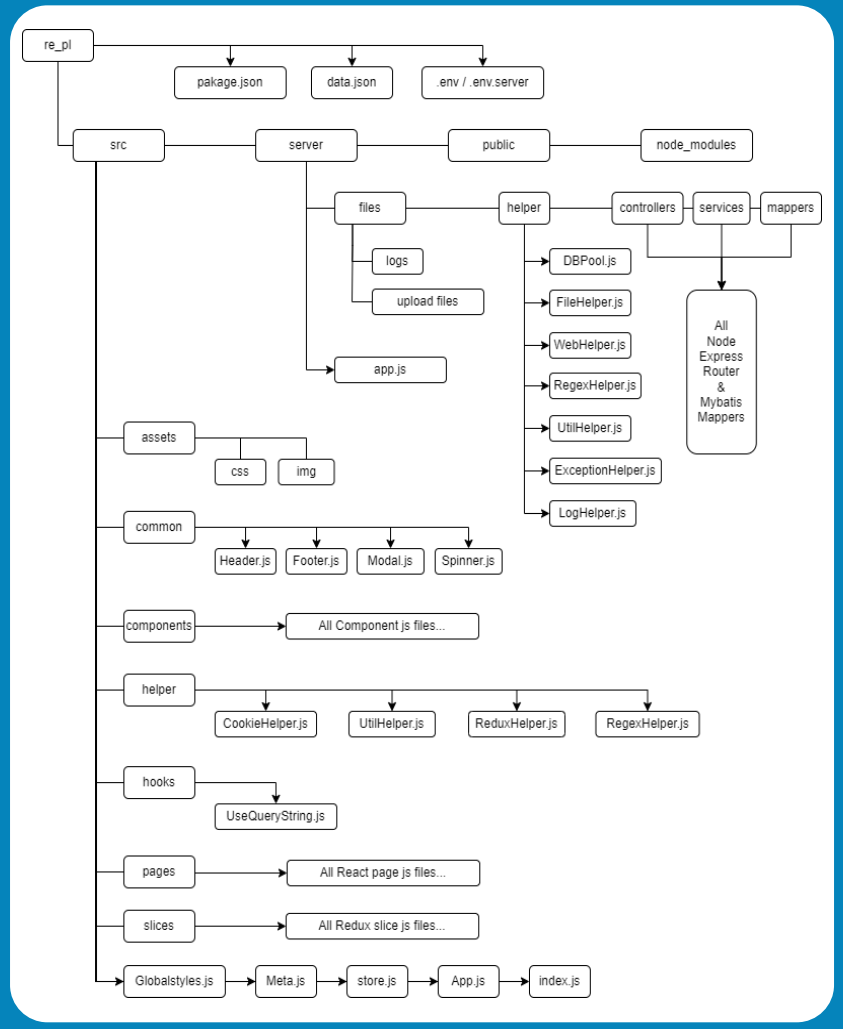
대표 화면별 기능소개



게시판 - 게시글 보기

Node의 Multer 라이브러리를 활용한 파일 처리,
CKEditor를 이용해 HTML로 작성된 본문,
게시글에서 추천된 장소와 태그,
작성자 팔로우와 댓글 남기기 등의
다양한 기능을 이용할 수 있습니다.

프로젝트 파일 구조



프로젝트 구조

React build 전 상태로,
전체 React Front-end 코드는
src 디렉토리 내에
Back-end 코드는
server 디렉토리 내에 배치하였습니다.
자주 쓰이는 코드는 모듈화하여
각 helper 디렉토리에 따로 추출,
전역으로 쓰이는 코드들은
app.js(server),
Header/Footer/Globalstyles.js
(FrontEnd)에서 작성하였습니다.

대표 로직 설명

```jsx
<RecommendPlaceArea>
    <div className='recommend-place-top'>
        <h3>이 글에서 추천한 장소들</h3>
    </div>
    <div className='recommend-place-body'>
        <Spinner loading={loading2} />
        {
            error2 ? (
                <div>추천된 장소 목록을 불러오지 못했습니다.</div>
            ) : (
                <ul>
                    {
                        places && places.map((v, i) => {
                            return (
                                <RecommendListItem
                                    key={i}
                                    id={v.id}
                                    img={v.img}
                                    title={v.place_name}
                                    address={v.address_name}
                                    comment={v.comment}
                                    rating={v.rating}
                                    place_url={v.place_url}
                                    modalOpen={modalOpen}
                                />
                            );
                        })
                    }
                    {
                        isModalOpen ?
                        <LocModal
                            isModalOpen={isModalOpen} closeModal={() => setIsModalOpen(false)}
                            style={{ content: {
                            width: "300px" } }}
                            data={modalContent}
                        /> : <></>
                    }
                </ul>
            )
        }
        ...
```

React Component 예시 코드 (Bulletin.js 中)

```jsx
const RecommendPlaceArea = styled.div`
    width: 800px;
    margin: auto;
    margin-bottom: 40px;

    .recommend-place-top {
        display: flex;
        flex-flow: row nowrap;
        justify-content: space-between;
        align-items: flex-end;
        margin-bottom: 20px;
        padding: 0 10px;

        h3 {
            font-size: 20px;
            font-weight: 600;
        }
    }
```

styled-components 예시

```jsx
const [modalContent, setModalContent] = useState(null);
const [isModalOpen, setIsModalOpen] = useState(false);

// 모달창 이벤트
const modalOpen = useCallback(current => {
    setModalContent({
        id: current.dataset.id,
        place_name: current.dataset.place_name,
        address_name: current.dataset.place_address,
        place_url: current.dataset.place_url });
    setIsModalOpen(true);
}, []);

useEffect(() => {
    if (isModalOpen) {
      document.body.style.cssText = `
            position: fixed;
            top: -${window.scrollY}px;
            overflow-y: scroll;
            width: 100%;
        `;
    } else {
      const scrollY = document.body.style.top;
      document.body.style.cssText = "";
      window.scrollTo(0, parseInt(scrollY || "0", 10) * -1);
    }
  }, [isModalOpen]);
```

각 이벤트 처리

Front-end 렌더링

대부분의 Front-end Component는 다음과 같은 과정을 거칩니다.

1) HTML, Styled-componen를 활용한 외관 제작

2) 표시될 데이터 useEffect + dispatch를 활용해 불러온 뒤 렌더링

3) 각 엘레먼트의 사용자 조작 이벤트 정의

이 과정에서 React 및 Redux의 state, callback, selector, dispatch 등을 활용합니다.

대표 로직 설명

```jsx
// 장소 리뷰 모달
const [modalContent, setModalContent] = useState(0);
const [modalIsOpen, setModalIsOpen] = useState(false);
/**
 * 처음 열릴때 지도를 렌더링하고 전체 데이터를 가져옴 (1회)
 */
useEffect(() => {
  const container = document.getElementById("map");
  const options = {
    // 이젠 아카데미 위도 경도
    center: new kakao.maps.LatLng(37.5025506249856, 127.02485228946493),
    level: zoomLevel,
  };
  const map = new kakao.maps.Map(container, options);
  setMap(map);
  console.log("🗺️ 지도 렌더링");

  // place 데이터 (중복인지 확인 위함)
  dispatch(getMapData());
  // theme 데이터
  dispatch(getThemeData());
  // theme_place 데이터
  dispatch(getTP());

  // 장소 검색 객체를 생성합니다
  const ps = new kakao.maps.services.Places();
  setPs(ps);

  // 검색 결과 목록이나 마커를 클릭했을 때 장소명을 표출할 인포윈도우를 생성합니다
  const infowindow = new kakao.maps.InfoWindow({ zIndex: 1, disableAutoPan: true });
  setInfowindow(infowindow);
}, []);
```

화면 렌더링시 최초 실행되는 useEffect (Map.js 코드 中)

```jsx
/** 모든 장소 데이터를 불러오는 비동기 함수 */
export const getMapData = createAsyncThunk("MapSlice/getMapData", async (payload, { rejectWithValue }) => {
  let result = null;

  try {
    const response = await axios.get("/map");
    result = response.data;
  } catch (err) {
    result = rejectWithValue(err.response);
  }

  if (result.item) {
    return result.item
  } else {
    return result;
  }
});
```

Redux dispatc시 slice의 AsyncThunk를 통해 서버와 Ajax 처리(Axios)

Component -> Redux

만일 이벤트를 통해 서버가 Ajax 요청을 해야 한다면

React Redux의 useCallback 함수 내에 dispatch 함수를 콜,

slice 모듈로 파라미터를 전달 후(payload)

해당 slice에서 Axios 라이브러리를 활용한 Ajax 요청을 통해 Back-end 서버와 통신합니다.

대표 로직 설명

```js
const express = require('express');
const MapService = require('../../service/map/MapService');

module.exports = (() => {
    const url = process.env.MAP_PATH;
    const router = express.Router();

    router.get(`${url}`, async (req, res, next) => {
        let result = null;

        try {
            result = await MapService.getList();
        } catch (err) {
            return next(err);
        }

        res.sendResult({ item: result });
    });
```

server/controllers 내 MapContoller.js

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="MapMapper">
    <select id='selectList'>
        SELECT id, place_name, address_name, road_address_name, lat, lng, category_item_name, phone, place_url
        FROM place
    </select>
```

server/mappers 내 MapMapper.xml

```js
const mybatisMapper = require('mybatis-mapper');
const DBPool = require('../../helper/DBPool');
const { RuntimeExeption } = require('../../helper/ExceptionHelper');

class MapService {
    constructor() {
        mybatisMapper.createMapper([
            './server/mapper/map/MapMapper.xml',
        ]);
    }

    async getList(params) {
        let dbcon = null;
        let data = null;

        try {
            dbcon = await DBPool.getConnection();
            let sql = mybatisMapper.getStatement('MapMapper', 'selectLi
            let [result] = await dbcon.query(sql);
            data = result;
        } catch (err) {
            throw err;
        } finally {
            if (dbcon) dbcon.release();
        }

        return data;
    }
```

server/service 내 MapService.js

Back-end 처리

모든 Back-end(server) 코드는
MySQL DB와 Front-end를
연동하는 작업을 수행합니다.

express의 router 등록을 맡는 모듈을 작성,
미리 지정한 주소에서
Axios CRUD 요청이 발생하면
Mybatis mapper를 통해
직접 SQL에 접속하는 service 모듈을 연동,
원하는 데이터를 서버와 주고 받습니다.

Helper 모듈 설명

```
class CookieHelper {
    /**
     * Singleton
     *
     * @returns CookieHelper 타입 객체
     */
    static #current = null;

    static getInstance() {
        if (CookieHelper.#current === null) {
            CookieHelper.#current = new CookieHelper();
        }

        return CookieHelper.#current;
    }
```

```
export default CookieHelper.getInstance();
```

Front-end helper에서 가장 많이 쓰였던 CookieHelper

자주 쓰이는 모듈

React는 SAS으로, 항상 하나의 어플리케이션 내에서 구동됩니다.
react-router-dom 라이브러리를 활용해 각 url 라우팅을 처리했고
이에 따라오는 모든 하위 컴포넌트 모듈은 memo( ) 함수를 통해 단일 함수를 유지했습니다.

또한 자주 쓰이는 모듈들 역시 구동 중 중복되는 반환을 방지하기 위해
간단한 싱글톤 함수를 통해 서비스 동안 하나의 인스턴스만 유지되도록 처리했습니다.

Helper 모듈 설명

```js
/**
 * 쿠키 저장
 *
 * @param   {string}    name    - 저장할 쿠키 이름
 * @param   {string}    value   - 저장할 쿠키 값
 * @param   {json}      options - 유효시간, 유효경로, 도메인 등 옵션
 */
setCookie(name, value, options = {}) {
    // path 없다면 전역으로 초기화
    if (options.path == undefined) {
        options.path = "/";
    }

    // expire가 있다면 만료일 UTC 표준 문자열로 변환
    if (options.expires instanceof Date) {
        options.expires = options.expires.toUTCString();
    }

    // 초기 쿠키 문자열 생성
    let updateCookie = `${encodeURIComponent(name)}=${encodeURIComponent(value)}`;

    // options의 정보 추가
    for (const optionKey in options) {
        updateCookie += `;${optionKey}`;
        const optionValue = options[optionKey];
        if (optionValue !== true) {
            updateCookie += `=${optionValue}`;
        }
    }

    // cookie 저장
    document.cookie = updateCookie;
}
```

브라우저에 저장된 Cookie를 활용하기 위한 CookieHelper.js

로그인 정보 쿠키 활용

본 프로젝트의 대다수의 페이지(로그인, 댓글, 게시판, 테마 추가 등…)에서
로그인 정보를 Cookie에 저장하고 활용하는 로직이 매우 자주 등장했습니다.

기본 메소드인 document.cookie를 활용,
개발자가 직접 정의한 시간(본 프로젝트에선 1일) 동안
유저 정보를 JSON string으로 저장한 뒤(패스워드 제외)
이 값이 있으면 로그인 상태, 없으면 비로그인 상태로 처리했습니다.

같은 방식으로 만료 기한을 0으로 설정해 쿠키를 즉시 파기하는 메소드와
저장된 Cookie를 입력한 key 파라미터로 검색해 가져오는 메소드를 작성,
폭넓게 활용하였습니다.

Helper 모듈 설명

```js
/**
 * HTTP 상태코드(HTTP Status Code)
 * - 웹에서의 에러 상황을 의미하는 표준화된 숫자값.
 *  - 400 : Bad Request Exception -> 잘못된 요청 (사용자가 지정된 형식으로 입력하지 않은 경우)
 *  - 404 : Page Not Found -> 페이지를 찾을 수 없습니다. (웹 브라우저에 주소 잘못 입력 한 경우)
 *  - 500 : Server Error -> 백엔드 프로그램이 겪는 Runtime Error (백엔드 개발자 잘못)
 *
 * 사용자가 입력값 형식을 지키도록 강제하는 것은 백엔드에게는 보안과 연결되는 중요한 사안이다.
 * 사용자의 입력값을 검사하여 지정된 형식이 아닐 경우 적절한 예외처리를 수행해야 한다.
 */
class BadRequestException extends Error {
    // HTTP 상태코드를 의미하는 멤버변수
    #statusCode;

    // 입력 요소에 대한 selector
    #selector;

    // 입력요소를 두 번째 파라미터로 전달받는다.
    constructor(msg = '잘못된 요청입니다.', selector = null) {
        super(msg);
        super.name = 'BadRequestException';
        this.#statusCode = 400;
        // 멤버변수에 입력요소를 참조시킨다
        this.#selector = selector;
    }

    // 표준화 된 값이기 때문에 setter는 별도로 설정하지 않음
    get statusCode () {
        return this.#statusCode;
    }

    // 입력요소에 대한 getter
    get selector() {
        return this.#selector;
    }
}

export { BadRequestException };
```

개발자가 직접 지정한 오류를 발생시키기 위한 ExceptionHelper.js

개발자 정의 오류

기본적으로 HTML 및 React가 제공하는 오류 외에,
사용자로부터 입력 받은 값이 올바르지 않거나,
서버로부터 응답 받은 결과 코드가 의도와 다르거나,
백엔드 로직 내에서 발생한 오류 등 다양한 오류 처리가 필요했습니다.

Error 기본 객체를 상속한 클래스를 생성,
개발 중 직접 정의한 에러 코드와 메시지를 이용해
try - catch 혹은 if문 등 다양한 조건분기 안에서
올바르지 않은 로직에 대해 모두 예외 처리를 노려 보았습니다.

Helper 모듈 설명

```js
import { BadRequestException } from './ExceptionHelper.js';

/**
 * 정규표현식을 기반으로 입력값에 대한 유효성 검사를 수행하는 클래스.
 * HTML 문서에서 사용하기 위해 input field에 대한 입력값을 검사한다.
 */
class RegexHelper {
    static #current = null;

    static getInstance() {
        if (RegexHelper.#current === null) {
            RegexHelper.#current = new RegexHelper();
        }

        return RegexHelper.#current;
    }

    /**
     * 값의 존재 여부를 검사한다.
     * @param {HTMLElement} field 검사할 대상에 대한 <INPUT>요소의 DOM 객체
     * @param {string} msg     값이 없을 경우 표시할 메세지 내용
     *
     * ex) regexHelper.value('#user_id', '아이디를 입력하세요');
     */
    value(field, msg) {
        const content = field.value;
        if(content == undefined || content == null ||
        (typeof content == 'string' && content.trim().length === 0)) {
            throw new BadRequestException(msg);
        }

        return true;
    }
```

개발자가 직접 지정한 오류를 발생시키기 위한 ExceptionHelper.js

```js
/**
 * 입력값이 정규표현식을 충족하는지 검사한다.
 * @param {HTMLElement} field  검사할 대상에 대한 <INPUT>요소의 DOM 객체
 * @param {string} msg         표시할 메세지
 * @param {object} regexExpr   검사할 정규표현식
 */
field(field, msg, regexExpr) {
    this.value(field,msg);

    // 입력값에 대한 정규표현식 검사가 실패라면?
    if (!regexExpr.test(field.value.trim())) {
        throw new BadRequestException(msg, field);
    }

    return true;
}

/**
 * 숫자로만 이루어 졌는지 검사하기 위해 field()를 간접적으로 호출한다.
 * @param {HTMLElement} field  검사할 대상에 대한 <INPUT>요소의 DOM 객체
 * @param {string} msg         표시할 메세지
 */
num(field, msg) {
    return this.field(field, msg, /^[0-9]*$/);
}
```

입력값 검사 및 예외처리

유저의 로그인 및 가입 처리,

게시판과 후기 창의 댓글 등 입력 폼들에서 빼놓을 수 없는 입력값 검사입니다.

정규 표현식을 활용, 지정한 양식을 충족하지 않는다면

앞서 정의한 ExceptionHelper를 이용, 오류를 throw 처리합니다.

Helper 모듈 설명

```js
/**
 * Redux-Slice에서 반복적으로 사용되는 처리로직을 재사용하기 위해 만든 모듈
 */
const pending = (state, { payload }) => {
    return { ...state, loading: true }
};

const fulfilled = (state, { payload }) => {
    return {
        data: payload?.item ? payload?.item : payload,
        pagenation: payload?.pagenation,
        loading: false,
        error: null
    }
};

const rejected = (state, { payload }) => {
    const err = new Error();

    if (typeof payload.data === "string") {
        err.code = payload.status ? payload.status : 500;
        err.name = "React Error";
        err.message = payload.data;
    } else {
        err.code = payload.data.rtcode;
        err.name = payload.data.rt;
        err.message = payload.data.rtmsg;
    }

    return {
        ...state,
        loading: false,
        error: err
    }
};

export { pending, fulfilled, rejected };
```

Redux Slice에서 반환하는 state값들을 다루기 위한 ReduxHelper.js

Redux Reducer

본 프로젝트 내에서 Redux slice를 폭넓게 사용하였으며,
Redux reducer/extraReducer에서 반환하는
각 action 함수의 결과 처리들을 통합하기 위해
action 함수가 반환하는 객체들을 활용,
자주 쓰이는 Redux slice state 값에 맞게
각 파라미터들을 가공해 반환하도록 정의하였습니다.

```js
extraReducers: {
    [addUser.pending]: pending,
    [addUser.fulfilled]: fulfilled,
    [addUser.rejected]: rejected,

    [checkValue.pending]: pending,
    [checkValue.fulfilled]: fulfilled,
    [checkValue.rejected]: rejected,

    [makeLogin.pending]: pending,
    [makeLogin.fulfilled]: fulfilled,
    [makeLogin.rejected]: rejected,
}
```

```js
import { pending, fulfilled, rejected } from '../../helper/ReduxHelper';
```

이를 활용한 LoginSlice.js의 extraReducer

Helper 모듈 설명

```js
const fs = require('fs');
const dotenv = require('dotenv');
const { join, resolve } = require('path');

class FileHelper {
    static #current = null;

    static getInstance() {
        if (FileHelper.#current == null) FileHelper.#current = new FileHelper();

        return FileHelper.#current;
    }

    mkdirs(target, permission = '0755', data = '') {
        // 대상 경로가 없다면 탈출
        if (target == undefined || target == null) { return; }

        // 윈도우는 경로가 역슬래쉬로 옴(문자열 내부이므로 두개)
        // 이거 리눅스처럼 그냥 슬래쉬로 변경(윈도우는 돌다가능)
        target = target.replace(/\\/gi, '/');
        // Node.js v.17 이상 -> replaceAll('\\', '/');

        let dir = ''; // 폴더 깊이 누적할 변수

        // 경로를 / 단위로 자름
        const target_list = target.split('/');

        // 리눅스/맥 등에선 절대경로가 /로 시작되는 경우가 있음
        // 이 때 target_list의 0번 아이템이 빈칸이 되므로 잘라냄
        // 맨 첫글자가 / 이면 dir 시작을 /으로
        if (target.substring(0, 1) == '/') {
            dir = '/';
        }

        // 윈도우는 하드디스크 문자에 : 붙어있음
        // 그 때 디스크 기준 디렉토리 경로 설정 위해 뒤에 / 붙여줌
        if (target_list[0].indexOf(':') > -1) {
            target_list[0] += '/'
        }

        // 잘라낸 배열만큼 순환하며 디렉토리 생성
        target_list.forEach((v, i) => {
            dir = join(dir, v);

            // 현 폴더를 의미한다면 이번 턴은 중단
            if (v == '.') {
                return;
            }

            // console.debug(dir);
            if (!fs.existsSync(dir)) {
                fs.mkdirSync(dir);
                fs.chmodSync(dir, permission);
            }
        });
    }
```

Back-end 파일 처리를 위한 FileHelper.js

```js
    mkDataFile (data = '') {
        const configFileName = process.env.NODE_ENV !== 'production' ? '.env.server.development' : '.env.server.production';
        const configPath = join(resolve(), configFileName);

        if (!fs.existsSync(configPath)) {
            try {
                throw new Error();
            } catch (e) {
                console.error('┌──────────────────────────────────┐');
                console.error('│      Configuration Init Error    │');
                console.error('└──────────────────────────────────┘');
                console.error('환경설정 파일을 찾을 수 없습니다. 환경설정 파일의 경로를 확인하세요.');
                console.error(`환경설정 파일 경로 : ${configPath}`);
                console.error('프로그램을 종료합니다.');
                process.exit(1);
            }
        }

        dotenv.config({ path: configPath });

        const targetPath = process.env.EDITOR_TEST;
        const targetFile = 'testData.html';

        this.mkdirs(targetPath);

        fs.writeFileSync(`${targetPath}/${targetFile}`, data, err => {
            if (err) {
                console.error(err);
                return;
            } else {
                console.log(`${targetFile} 파일 저장 완료. 경로 : ${targetPath}`);
                console.log(`내용 : ${fs.readFileSync(`${targetPath}/${targetFile}`, 'utf8')}`);
            }
        });
    }
}

module.exports = FileHelper.getInstance();
```

파일 처리

프로젝트 개발 및 서비스 도중 발생하는 다양한 로그 파일들과

사용자가 제출한 파일들을 저장하기 위한 디렉토리 구조를 위한 헬퍼 모듈입니다.

Express fs와 dotenv 라이브러리, path를 활용

개발자가 환경 설정 파일에 지정한 루트에 파일을 저장하며

만약 해당 경로가 없다면 경로를 따라 새로운 디렉토리들을 생성합니다.

Helper 모듈 설명

```js
const logger = require('./LogHelper');

const WebHelper = () => {
    return (req, res, next) => {
        /** Express의 req, res의 기능을 확장 */
        // req.foo = () => { ... };
        // res.bar = () => { ... };

        /** 프론트엔드에게 JSON 결과를 출력하는 기능 */
        res._sendResult = (data, error = null) => {
            /**
             * {
             *     rt: 결과 (OK, 혹은 에러 이름),
             *     rtcode: HTTP 상태코드 (200, 400, 404, 500),
             *     rtmsg: 결과 메세지 (OK 없음. 에러인 경우 에러 내용),
             *     ... JSON 데이터 ...
             *     pubdate: 생성일시
             * }
             */
            const json = {
                rt: 'OK',
                rtcode: 200,
                rtmsg: null
            };

            if (error) {
                json.rt = error.rt || 'Server Error';
                json.rtcode = error.code || 500;
                json.rtmsg = error.message || '요청을 처리하는데 실패했습니다.';

                if (isNaN(json.rtcode)) {
                    json.rtcode = 500;
                }
            }

            if (data) {
                for (const item in data) {
                    json[item] = data[item];
                }
            }
            // 표준시로부터 한국의 시차를 적용하여 ISO 포맷을 생성
            const offset = new Date().getTimezoneOffset() * 60000;
            const today = new Date(Date.now() - offset);
            json.pubdate = today.toISOString();

            res.header('Content-Type', 'application/json; charset=utf-8');
            res.header('name', encodeURIComponent(json.name));
            res.header('message', encodeURIComponent(json.message));
            res.status(json.rtcode || 200).send(json);
        };

        /** 결과가 200(OK)인 경우에 대한 JSON 출력 */
        res.sendResult = data => {
            res._sendResult(data);
        };

        /** 에러처리 출력 */
        res.sendError = error => {
            logger.error(error.stack);
            res._sendResult(null, error);
        };

        // Express의 다음 처리 단계로 제어 이전
        next();
    };
};

module.exports = WebHelper;
```

Express의 response 처리를 확장하기 위한 WebHelper.js

서버 응답 확장

Express Router의 응답을 처리하는 sendResult 함수를 확장한 모듈입니다.
서버의 직접 응답은 지양하고 개발자가 각 통신 결과를 핸들링하기 위한 모듈로,
개발자가 정의한 형식의 응답을 돌려주고 모든 통신을 로그로 남깁니다.

Helper 모듈 설명

```js
const logger = require('./LogHelper');

const WebHelper = () => {
    return (req, res, next) => {
        /** Express의 req, res의 기능을 확장 */
        // req.foo = () => { ... };
        // res.bar = () => { ... };

        /** 프론트엔드에게 JSON 결과를 출력하는 기능 */
        res._sendResult = (data, error = null) => {
            /**
             * {
             *     rt: 결과 (OK, 혹은 에러 이름),
             *     rtcode: HTTP 상태코드 (200, 400, 404, 500),
             *     rtmsg: 결과 메세지 (OK 없음. 에러인 경우 에러 내용),
             *     ... JSON 데이터 ...
             *     pubdate: 생성일시
             * }
             */
            const json = {
                rt: 'OK',
                rtcode: 200,
                rtmsg: null
            };

            if (error) {
                json.rt = error.rt || 'Server Error';
                json.rtcode = error.code || 500;
                json.rtmsg = error.message || '요청을 처리하는데 실패했습니다.';

                if (isNaN(json.rtcode)) {
                    json.rtcode = 500;
                }
            }

            if (data) {
                for (const item in data) {
                    json[item] = data[item];
                }
            }
```

Express의 response 처리를 확장하기 위한 WebHelper.js

```js
            // 표준시로부터 한국의 시차를 적용하여 ISO 포맷을 생성
            const offset = new Date().getTimezoneOffset() * 60000;
            const today = new Date(Date.now() - offset);
            json.pubdate = today.toISOString();

            res.header('Content-Type', 'application/json; charset=utf-8');
            res.header('name', encodeURIComponent(json.name));
            res.header('message', encodeURIComponent(json.message));
            res.status(json.rtcode || 200).send(json);
        };

        /** 결과가 200(OK)인 경우에 대한 JSON 출력 */
        res.sendResult = data => {
            res._sendResult(data);
        };

        /** 에러처리 출력 */
        res.sendError = error => {
            logger.error(error.stack);
            res._sendResult(null, error);
        };

        // Express의 다음 처리 단계로 제어 이전
        next();
    };
};

module.exports = WebHelper;
```

서버 응답 확장

Express Router의 응답을 처리하는 sendResult 함수를 확장한 모듈입니다.
서버의 직접 응답은 지양하고 개발자가 각 통신 결과를 핸들링하기 위한 모듈로,
개발자가 정의한 형식의 응답을 돌려주고 모든 통신을 로그로 남깁니다.

Helper 모듈 설명

```js
/** 필요한 패키지 가져오기 */
const { join, resolve } = require('path');
const dotenv = require('dotenv');
const mysql = require('mysql2/promise');
const logger = require('./LogHelper');

/** 환경설정 파일 로드 --> 데이터베이스 접속정보 가져오기 위함 */
dotenv.config({ path: join(resolve(), '.env.server.development') });

/**
 * DATABASE Connection Pool을 관리하기 위한 SingleTon Class
 */
class DBPool {
    // 싱글톤 객체
    static #current = null;

    // 접속 정보 설정
    static connectionInfo = {
        host: process.env.DATABASE_HOST, // MYSQL 서버 주소 (다른 PC 경우 IP 주소)
        port: process.env.DATABASE_PORT, // MYSQL 포트 번호
        user: process.env.DATABASE_USERNAME, // MYSQL 로그인 가능한 계정 이름
        password: process.env.DATABASE_PASSWORD, // 비밀번호
        database: process.env.DATABASE_SCHEMA, // 사용하고자 하는 데이터베이스 이름
        connectionLimit: process.env.DATA_CONNECTION_LIMIT, // 최대 커넥션 수
        connectTimeout: process.env.DATABASE_CONNECT_TIMEOUT, // 커넥션 타임아웃
        waitForConnections: process.env.DATABASE_WAIT_FOR_CONNECTIONS, // 커넥션 풀이 다 찬 경우 처리
    };

    /** 싱글톤 객체를 생성하여 리턴하는 메소드 */
    static getInstance() {
        if (DBPool.#current == null) {
            DBPool.#current = new DBPool();
        }
        return DBPool.#current;
    }
```

MySQL과의 연결 Pool을 생성하기 위한 DBPool.js

```js
    /**
     * 생성자
     * 데이터베이스 Connection Pool을 생성 및 필요한 이벤트를 정의
     * 각 이벤트는 DBConnection의 생성, 임대, 반납 여부를 모니터링,
     * 데이터베이스에 접속되었을 경우 DB에 전달되는 SQL문을 가로채서 로그로 기록한다
     */
    constructor() {
        // Connection Pool 객체를 멤버변수로서 생성
        this.pool = mysql.createPool(DBPool.connectionInfo);

        // 데이터베이스에 접속된 경우 발생할 이벤트
        this.pool.on('connection', connection => {
            logger.info(`>> DATABASE 접속됨 [threadId=${connection.threadId}]`);

            // 이 객체로 전달되는 SQL 수행 기능을 가로챔
            const oldQuery = connection.query;

            // 가로챈 객체의 기능을 로그 기록 후 SQL을 수행하도록 재정의
            connection.query = function (...args) {
                const queryCmd = oldQuery.apply(connection, args);
                // 1) SQL문에 포함된 모든 줄바꿈문자를 띄어쓰기로 변환
                // 2) SQL문에 포함된 2회 연속 공백 문자를 하나의 공백으로 변환
                // 3) 그 결과를 로그에 기록
                logger.debug(queryCmd.sql.trim().replace(/\n/g, " ").replace(/ +(?= )/g, " "));
            }
        });

        this.pool.on('acquire', connection => {
            logger.info(`>> Connection 임대됨 [threadId=${connection.threadId}]`);
        });

        this.pool.on('release', connection => {
            logger.info(`>> Connection 반납됨 [threadId=${connection.threadId}]`);
        });
    }

    /**
     * Connection Pool에서 하나의 데이터베이스 접속 객체를 임대하는 메서드
     */
    async getConnection() {
        let dbcon = null;

        try {
            dbcon = await this.pool.getConnection();
        } catch (err) {
            if (dbcon) dbcon.release();
            logger.error(err);
            throw err;
        }

        return dbcon;
    }

    /**
     * 데이터베이스 커넥션 풀을 종료함
     */
    close() {
        this.pool.end();
    }
}

// 싱글톤 객체 모듈 내보냄
module.exports = DBPool.getInstance();
```

데이터베이스와 연결

mysql 라이브러리를 이용,
서버(본인 컴퓨터) 내의 MySQL과
Connection Pool을 생성하기 위한 모듈입니다.

1) dotenv 라이브러리 활용, 개발자가 선언해둔
   환경 설정 파일에서 각 정보 동기화
2) 클래스 생성자 내에서 mysql 라이브러리 활용,
   MySQL과의 연결 Pool 객체를 생성해둠
3) 싱글톤 메소드 정의, 후에 필요한 모듈에서
   본 모듈의 mysql connection pool 반환

미완성한 부분

1. 마이페이지 Node.js 미연동
2. 테마 지도 찾기 -> 사이드바 클릭 이벤트 미제작
3. 모든 데이터 삭제 처리 Node.js 환경에서 불가(미제작)
4. SQL Linux 서버에 이식 및 SSH 키 발급을 통한 외부 접속
5. 4번과 연계, 프로젝트 Build 과정을 거쳐 호스팅 처리(Github Page 등)

미흡했던 부분

1. 한 컴포넌트 내에 다수의 슬라이스 selector 배치, 서버 접속 횟수 및 렌더링 소요시간 증가
2. Meta Helmet 처리 빈약, SEO에 취약함
3. SQL 문법의 이해도가 낮아, 불필요한 서버 내 로직이 발생
4. 반응형 최적화가 전무, 디스플레이 환경에 따른 사이트 재배치 필요
5. 강의 코드의 재사용으로 새로운 코드의 학습과 시도에선 미흡함을 보임

본 프로젝트를 마치며

이젠아카데미 풀스택 B 과정을 수강하며 평소 목표로 하던 웹 개발자로써의 역량을 차근히 쌓아 나갔습니다.
과정의 마무리인 본 프로젝트를 수행하며 React를 통한 SAS 제작 및 SEO 최적화, 서브라우팅,
Ajax 처리, DB의 기본인 CRUD 메소드와 함수의 모듈화를 통한 재사용.
Node.js의 개발 환경 구성과 각 Router를 활용한 url 요청 처리,
Redux slice를 이용한 어플리케이션 내 데이터 유지 및 처리와
최종적으로 이 모든 과정을 아우르는 Front-end - Back-end의 연동.
다양한 개발 역량에 대해 폭넓게 수련할 수 있었습니다.
각 개발 파트 간의 협업과 소통이 어떻게 이루어지고 어떠한 자세가 필요한지 역시 체득할 수 있었으며
웹 개발과 서비스에 이르기까지 전반적인 팀원들과의 심도 있는 공동 협업을 통해
개발자로써의 역량을 한 층 끌어올릴 수 있던 뜻깊은 프로젝트였습니다. 함께 수고한 팀원들, 감사합니다.