

Project 3

2023년 2학기

제출일	2023. 12. 15.
과목명	데이터베이스 및 데이터 시각화
담당교수	이기훈

학과	컴퓨터정보공학부	학과	컴퓨터정보공학부
학번	2019202023	학번	2019202074
이름	강병준	이름	김성진
학과	컴퓨터정보공학부	학과	컴퓨터정보공학부
학번	2021202001	학번	2021202056
이름	하지현	이름	방지민

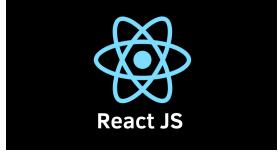
KWANGWOON UNIVERSITY

1. 웹 애플리케이션에 대한 설명

데이터베이스 관리 시스템(DBMS)를 사용하는 따릉이 플랫폼이다. 따라서 해당 웹 애플리케이션은 기본적인 이용자 기능으로 회원가입하고 로그인할 수 있어야 하고, 개인 정보 조회와 게시글 작성 등이 가능해야 한다. 또한 따릉이의 대여소 현황을 조회하거나 위치 정보를 조회하는 것이 가능해야 한다. 관리자 기능으로는 회원 관리와 따릉이 시설 관리, 따릉이 대여소 개설 및 폐쇄와 게시판 관리 기능 등이 기본적으로 들어가 있어야 하며, 따릉이 정보들을 시각화하고 따릉이의 고장(장애) 신고도 가능해야 한다.

위에서 언급한 기본 기능 이외의 추가적인 기능으로는, 해당하는 대여소의 위치에 대한 날씨 정보 확인이나 건강 관련 뉴스 확인, 지도에서 대여소 위치를 확인할 수 있고, 자전거 도로도 확인할 수 있다. 또한 게시글에 좋아요를 하거나 댓글을 달아서 게시글에 대한 사용자의 의견을 표현할 수도 있으며, 이용권을 직접 구입하여 따릉이를 대여하고, 반납하는 것도 가능해야 한다고 생각했다. 추가적으로 사용자의 편의를 위해 소셜 로그인, 이메일 인증을 통한 비밀번호 찾기, 자전거 대여 및 반납 알림, 최근 이용한 대여소 내역이나 대여 기록 확인 등 이 외에도 다양한 기능이 가능하다.

데이터베이스를 사용하는 웹 어플리케이션은 서버와 클라이언트로 구성되어 있기 때문에 클라이언트에서 서버에 API를 요청하게 되면 서버가 클라이언트에게 요청받은 API에 대한 정보를 Database에서 SQL 쿼리를 바탕으로 가져온 뒤, 해당 데이터를 Json 형태로 클라이언트에 전달해 주는 형태로 동작하게 된다. 서버와 클라이언트로 나누어 개발하면서 각각 사용한 기술 stack은 아래와 같다.

Database	Framework	Library	Language
 MySQL	  Spring	 React JS	  Java

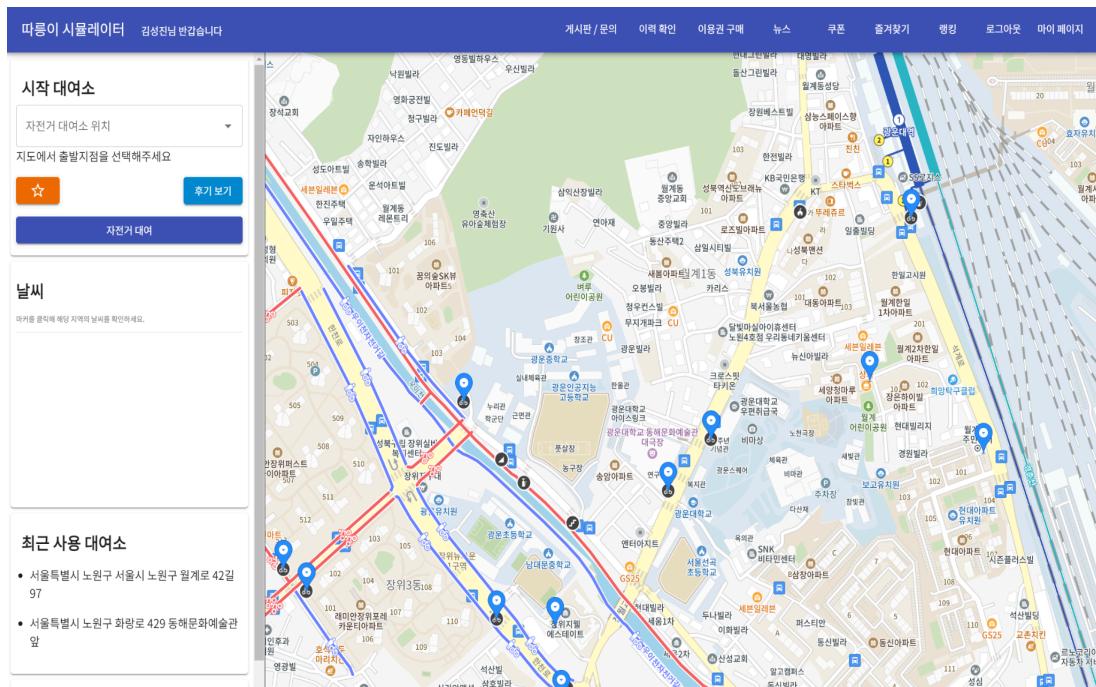
MySQL은 안정적인 관계형 데이터베이스 관리 시스템(RDBMS)으로, 해당 웹 어플리케이션의 Database는 MySQL을 사용하였다.

서버는 Java를 기반으로 Spring Framework를 이용해 구현하였다. Spring은 애플리케이션을 개발할 수 있도록 다양한 기능과 라이브러리를 제공하는 Java Framework이다.

SQL 쿼리문을 직접 사용해야 하기 때문에 **JDBC template** 라이브러리를 활용하여 ORM을 사용하지 않으며, **Prepared Statement** 방식으로 구현을 진행함으로써 SQL injection을 방지할 수 있는 환경을 구축하였다.

클라이언트의 경우에는 JavaScript를 기반으로 **React.js**를 이용해 구현하였다. 이때, **Node.js**를 활용하여 **React** 애플리케이션을 위한 개발 서버를 설정하고 실행하며 개발 환경을 강화하고 효율적으로 만들었다.

[따릉이 웹 애플리케이션 소개]



본 프로젝트에서 작성한 따릉이 웹 애플리케이션은 직접 따릉이를 대여하고 반납하는 과정을 시뮬레이션 할 수 있는 어플리케이션이다. 회원가입 및 로그인을 통해 사용자 정보를 식별할 수 있으며, 지도 내 마커를 이용하거나 검색을 통해 대여소를 찾을 수 있고, 자전거 대여 버튼을 통해 따릉이 대여를 시작할 수 있다.

대여가 시작되면, 도착 지점을 지도 내 마커를 클릭하거나 검색을 통해 지정할 수 있으며, 해당 대여소에 자전거 반납이 가능하다. 이후 대여를 진행하였던 대여소에 대해서 리뷰를 남기거나 평점을 매길 수 있다. 추가적으로, 게시판 기능을 구현하여 관리자 문의, 고장 신고, 자유게시판 등을 이용할 수 있으며 티켓 구매, 쿠폰 사용, 이력 확인, 랭킹 표시 등의 기능이 구현되어 있다.

2. 주요 기능 설명

1. 이메일 방식 회원가입

SCREEN	
<p align="center">회원가입</p> <p align="center">따릉이 시스템에 오신 것을 환영합니다! 아래 정보를 입력해주세요.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">이름 *</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">이메일 *</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">비밀번호 *</div> <div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 10px;">전화번호 * +82</div> <div style="display: flex; justify-content: space-around; align-items: center;"> <div style="border: 1px solid #ccc; padding: 5px; width: 45px;">나이 *</div> <div style="border: 1px solid #ccc; padding: 5px; width: 45px;">무게 *</div> </div> <div style="background-color: #0056b3; color: white; padding: 10px; text-align: center; border-radius: 5px; margin-top: 20px;">회원가입</div> <p align="center" style="font-size: small; margin-top: 10px;">계정이 이미 존재하나요? 로그인</p>	
DESCRIPTION	
<p>이메일 방식의 회원가입을 진행한다. 사용자명, 이메일, 비밀번호, 전화번호, 나이, 무게를 입력하고 회원가입 버튼을 누르면 회원가입을 진행한다. 이메일은 추후 로그인 및 비밀번호 찾기를 할 때 이용한다.</p>	
FRONT END	BACK END
<pre>const onSubmit = async (e) => { e.preventDefault(); // get data const data = new FormData(e.target); const username = data.get('name'); const email = data.get('email'); const password = data.get('password'); const phone_number = data.get('phone').replace(/\^+82 \s/g, ''); const age = data.get('age'); const weight = data.get('weight'); // Check email duplicated try { const response = await fetch(process.env.REACT_APP_API_URL + "/users/email/check-duplicate", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ email, }), }); if (response.status !== 200) { throw new Error("회원가입에 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status === 200) { alert("이미 존재하는 이메일입니다."); return; } } catch (error) { alert("회원가입에 실패하였습니다."); return; } }</pre>	<pre>// 회원가입 @PostMapping("/signup") public ResponseEntity<UserDto> signUp(@RequestBody UserDto userDto) { UserDto user = userService.findUserByEmail(userDto.getEmail()); if(user == null){ userService.signUp(userDto); return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_SIGNUP, result: null)); } else return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_NOT_SIGNUP, result: null)); } // 회원 가입 public void signUp(UserDto userDto) { try { userDto.setUser_type(1); userRepository.insertUser(userDto); } catch (NullPointerException e) { throw new GlobalException(ResponseStatus.INVALID_REQUEST); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre>
<p>회원가입 정보를 입력한 사용자가 회원 가입</p>	

```
// Do signup
try {
  const response = await fetch(process.env.REACT_APP_API_URL + "/users/signup", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      username,
      email,
      password,
      phone_number,
      age,
      weight,
    }),
  });
  if (response.status !== 200) {
    throw new Error("회원가입에 실패하였습니다.");
  }
  const jsonData = await response.json();
  if (jsonData.status === 2016) {
    alert("회원가입에 성공하였습니다.");
    navigate("/login");
    return;
  }
} catch (error) {
  alert("회원가입에 실패하였습니다.");
  return;
}
};
```

입력을 받을 수 있는 기본적인 뷰를 작성하고, 회원가입 버튼을 누를 시 Form Submit을 적용하여 백엔드에 관련 정보를 fetch하여 회원가입을 시도한다. 우선 이메일 중복확인 API를 통해 이메일 중복을 확인하고, 이후 회원가입 API를 통해 회원가입을 진행한다. 정상적으로 회원가입이 완료되었다면 Alert 메시지를 띄우고 로그인 페이지로 리다이렉션한다.

버튼을 눌러 /users/signup API로 요청이 수행된다. 각 필드의 정보를 입력하지 않거나 이미 회원가입이 되어있는 이메일의 경우 정상적으로 회원가입이 수행되지 않으며, 정상적으로 정보를 입력하게 되면 회원가입이 성공하게 된다.

SQL

```
public Boolean insertUser(UserDto userDto) {
    // INSERT
    String sql = "INSERT INTO user (password, username, user_type, email, phone_number, " +
        "weight, age, last_accessed_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
    jdbcTemplate.update(sql,
        userDto.getPassword(),
        userDto.getUsername(),
        userDto.getUser_type(),
        userDto.getEmail(),
        userDto.getPhone_number(),
        userDto.getWeight(),
        userDto.getAge(),
        userDto.getLast_accessed_at());
    return true;
}
```

사용자 관련 데이터를 포함하는 UserDto 타입의 매개변수를 받아서 USER 테이블에 비밀번호, 유저 이름, 유저의 타입(이메일 회원가입, 카카오 회원가입), 이메일, 전화번호, 몸무게, 나이를 입력하여 회원가입을 진행한다. 데이터베이스에 삽입하는 과정에서 오류가 없으면 true를 반환하여 정상적으로 회원 가입이 되었음을 알 수 있다.

2. 비밀번호 찾기

SCREEN	
<p>비밀번호 찾기</p> <p>이메일 *</p> <input type="text"/> <p>인증번호 보내기</p>	<p>비밀번호 찾기</p> <p>이메일 *</p> <p>mirusu400@naver.com</p> <p>인증번호 보내기</p> <p>인증번호 *</p> <p>변경할 비밀번호 *</p> <p>비밀번호 변경하기</p>
DESCRIPTION	
<p>비밀번호를 잊어버렸을 때 비밀번호를 찾는 기능이다. 이메일을 제공 후 “인증번호 보내기” 버튼을 클릭하게 되면 이메일을 전송하게 되며, 이후 인증번호와 변경할 비밀번호를 입력해 비밀번호를 변경할 수 있도록 구현한다.</p>	<p>frontend에서 인증번호 보내기 버튼을 눌렀을 때 /users/send-authcode API로 요청이 수행되면 인증번호를 전송한다. 인증번호를 전송하기 이전에 만약 해당 이메일로 인증 번호를 5분 이내에 전송한 내역이 있다면 해당 인증 번호를 제거하게 된다.</p>
FRONT END	BACK END
<pre>const onSubmitAuthCode = async (e) => { setIsNowLoading(true); (async () => { // Do login try { const response = await fetch(process.env.REACT_APP_API_URL + "/users/send-authcode", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ email, }), credentials: "include", },); if (response.status !== 200) { throw new Error("이메일 전송에 실패하였습니다."); } const jsonData = await response.json(); alert("이메일 전송을 성공하였습니다."); setIsSendAuthCode(true); setIsNowLoading((prev) => !prev); } catch (error) { console.log(error); alert("이메일 전송에 실패하였습니다."); setIsNowLoading((prev) => !prev); return; } })(); };</pre> <p>이메일 인증이 필요하므로, 처음엔 이메일</p>	<pre>// 비밀번호 찾기 후 인증 번호 전송 @PostMapping("/send-authcode") public ResponseEntity<@RequestBody UserDto userDto> sendAuthCode(@RequestBody UserDto userDto) throws MessagingException, UnsupportedEncodingException { EmailAuthDto emailAuthCodeDto = new EmailAuthDto(); // 5분 이내로 다시 이메일 보낼 것을 막기위해서 요청한 이후 번호 보내기 changePasswordService.deleteExistCode(userDto.getEmail()); emailAuthCodeDto.setAuthNum(Integer.parseInt(changePasswordService.sendEmail(userDto.getEmail()))); return ResponseEntity.ok(new JsonResponse<(ResponseStatus.SUCCESS_SENO_AUTHCODE, emailAuthCodeDto.getAuth_num())>()); }</pre> <p>/users/check-authcode API를 통해 입력한</p>

입력 품만 사용자에게 제공을 해 이메일을
입력 후 인증번호 보내기를 누르게 되면
이메일을 보내는 요청을 Backend에
전달한다. 이메일 전송엔 시간이 좀 걸리므로
그동안 Modal 창을 띄워 사용자가 추가적인
Interaction을 하지 못하도록 막는다.

```

136   {isSendAuthCode && (
137     <>
138       <Grid container spacing={2} sx={{ py: 3 }}>
139         <Grid item xs={12}>
140           <TextField
141             variant="outlined"
142             required
143             fullWidth
144             id="authcode"
145             label="인증번호"
146             name="authcode"
147             onChange={({e}) => setAuthcode(e.target.value)}
148           />
149         </Grid>

```

정상적으로 전송이 완료되었다면 인증번호와
변경할 비밀번호를 입력하는 품을 추가로
띄워 입력할 수 있도록 제공한다.

```

const onSubmitPasswordChange = () => {
  (async () => {
    // Do login
    try {
      const response = await fetch(
        process.env.REACT_APP_API_URL + "/users/check-authcode",
        {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            email,
            auth_num: authcode,
          }),
          credentials: "include",
        },
      );
      if (response.status !== 200) {
        throw new Error("인증번호 검사에 실패하였습니다.");
      }
      const jsonData = await response.json();

      if (jsonData.status != 2005) {
        throw new Error("인증번호 검사에 실패하였습니다.");
      }
    }
  })();
}

```

인증번호가 일치하게 되면 변경할 비밀번호를
입력할 수 있다.

```

// 인증 번호 확인 후 비밀 번호 변경
@PutMapping("/change-password")
public ResponseEntity<> changePassword(@RequestBody UserDto userDto){

  changePasswordService.changePassword(userDto.getEmail(), userDto.getPassword());
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_CHANGE_PASSWORD));
}

```

변경할 비밀번호를 입력하고 비밀번호
변경하기 버튼을 눌렀을 때
`/users/change-password` API로 변경할
비밀번호와 함께 요청이 되면 기존의 사용자
정보에 대한 row를 찾아 해당 row에서
비밀번호에 대한 필드인 `password`에 대해
변경 사항이 반영되게 된다.

```

const response2 = await fetch(
  process.env.REACT_APP_API_URL + "/users/change-password",
  {
    method: "PUT",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      email,
      password,
    }),
    credentials: "include",
  },
);
if (response2.status !== 200) {
  throw new Error("비밀번호 변경에 실패하였습니다.");
}
const jsonData2 = await response2.json();
alert("비밀번호 변경을 성공하였습니다.");
navigate("/login");
} catch (error) {
  console.log(error);
  alert("비밀번호 변경에 실패하였습니다. - " + error);
  setIsNowLoading(false);
  return;
}
);
};

```

비밀번호 변경을 신청하면, 정상적인 인증번호를 입력했는지 확인하는 API에 요청을 보내 인증번호를 확인하고, 정상적인 인증번호임이 확인되었으면 비밀번호를 변경하는 API를 통해 비밀번호를 변경한다

SQL

```

public void updatePassword(String email, String newPassword) {
    // UPDATE
    String sql = "UPDATE user SET password = ? WHERE email = ?";
    jdbcTemplate.update(sql, newPassword, email);
}

```

user 테이블에서 특정 이메일을 가진 사용자의 비밀번호를 업데이트한다. 사용자들을 구분할 수 있는 email을 통해 해당 이메일과 일치하는 레코드를 찾아 해당 레코드의 password 필드를 변경할 비밀번호로 update한다.

3, 4. 이메일 방식 로그인 및 로그아웃

SCREEN

	<h3>DESCRIPTION</h3>			
<p>로그인을 하는 품이다. 기존에 회원가입한 이메일 및 비밀번호를 입력 후 로그인을 진행할 수 있으며, 카카오 로그인을 통해서 카카오 ID를 이용한 로그인 또한 이용할 수 있다.</p> <table border="1" data-bbox="283 923 1396 2012"> <thead> <tr> <th data-bbox="283 923 828 990">FRONT END</th><th data-bbox="828 923 1396 990">BACK END</th></tr> </thead> <tbody> <tr> <td data-bbox="283 990 828 2012"> <pre>// Get localStorage token const token = localStorage.getItem("fcm_token"); // Do login try { const response = await fetch(process.env.REACT_APP_API_URL + "/users/signin", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ email, password, fcm_token: token, }), credentials: "include", }); if (response.status !== 200) { throw new Error("로그인에 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status !== 2002) { throw new Error("로그인에 실패하였습니다."); } alert("로그인에 성공하였습니다."); document.location.href = "/"; } catch (error) { console.log(error); alert("로그인에 실패하였습니다."); return; } }</pre> <p>이메일 로그인의 경우, 이메일 및 비밀번호를 입력 후 로그인하기 버튼을 클릭하면 백엔드에 이메일과 비밀번호를 전달해 로그인을 시도하며, 성공하면 Set-Cookie 헤더로 전송되는 쿠키를 이용해 세션 정보를 저장한다. 해당 정보는 계속해서 인증을 위해 쓰인다.</p> <pre>const onClickKakaoLogin = () => { const redirectUri = "http://localhost:3000/oauth/kakao"; document.location.href = "https://kauth.kakao.com/oauth/authorize?response_type=code&client_id=";</pre> </td><td data-bbox="828 990 1396 2012"> <pre>/login @PostMapping("/login") public ResponseEntity<String> signIn(@RequestBody UserDto userDto, HttpServletResponse response){ if (userService.signIn(userDto)) { // 쿠키 설정 Cookie idCookie = new Cookie("id", String.valueOf(userService.findUserIdByEmail(userDto.getEmail()))); Cookie emailCookie = new Cookie("email", userDto.getEmail()); Cookie passwordCookie = new Cookie("password", userDto.getPassword()); Cookie usernameCookie = new Cookie("username", userDto.getUsername()); fcmService.saveTokenByObject(userDto); fcmService.sendIncompleteMessage(userDto.getEmail()); // 쿠키 유효 시간 설정 idCookie.setMaxAge(7 * 24 * 60 * 60); // 7일 emailCookie.setMaxAge(7 * 24 * 60 * 60); passwordCookie.setMaxAge(7 * 24 * 60 * 60); usernameCookie.setMaxAge(7 * 24 * 60 * 60); // 쿠키 경로 설정 idCookie.setPath("/"); emailCookie.setPath("/"); passwordCookie.setPath("/"); usernameCookie.setPath("/"); } } public Boolean signIn(UserDto userdto) { try { userdto.setLast_accessed_at(LocalDateTime.now()); return userRepository.validateUser(userdto); } catch (NullPointerException e) { throw new GlobalException(ResponseStatus.INVALID_REQUEST); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre> <p>frontend에서 이메일 및 비밀번호를 입력 후 로그인하기 버튼을 누르게 되면 /users/signin API로 요청이 수행된다.</p> <p>회원가입이 되어 있는 사용자인지에 대한 검증을 진행하기 위해 validateUser() 메서드를 통해 회원 가입 여부를 판단한다. 검증이 완료된 이메일과 비밀번호라면 정상적으로 로그인이 진행되면서 cookie에 사용자 정보를 저장하여 세션 정보를 유지함과 동시에</p> </td></tr> </tbody> </table>	FRONT END	BACK END	<pre>// Get localStorage token const token = localStorage.getItem("fcm_token"); // Do login try { const response = await fetch(process.env.REACT_APP_API_URL + "/users/signin", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ email, password, fcm_token: token, }), credentials: "include", }); if (response.status !== 200) { throw new Error("로그인에 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status !== 2002) { throw new Error("로그인에 실패하였습니다."); } alert("로그인에 성공하였습니다."); document.location.href = "/"; } catch (error) { console.log(error); alert("로그인에 실패하였습니다."); return; } }</pre> <p>이메일 로그인의 경우, 이메일 및 비밀번호를 입력 후 로그인하기 버튼을 클릭하면 백엔드에 이메일과 비밀번호를 전달해 로그인을 시도하며, 성공하면 Set-Cookie 헤더로 전송되는 쿠키를 이용해 세션 정보를 저장한다. 해당 정보는 계속해서 인증을 위해 쓰인다.</p> <pre>const onClickKakaoLogin = () => { const redirectUri = "http://localhost:3000/oauth/kakao"; document.location.href = "https://kauth.kakao.com/oauth/authorize?response_type=code&client_id=";</pre>	<pre>/login @PostMapping("/login") public ResponseEntity<String> signIn(@RequestBody UserDto userDto, HttpServletResponse response){ if (userService.signIn(userDto)) { // 쿠키 설정 Cookie idCookie = new Cookie("id", String.valueOf(userService.findUserIdByEmail(userDto.getEmail()))); Cookie emailCookie = new Cookie("email", userDto.getEmail()); Cookie passwordCookie = new Cookie("password", userDto.getPassword()); Cookie usernameCookie = new Cookie("username", userDto.getUsername()); fcmService.saveTokenByObject(userDto); fcmService.sendIncompleteMessage(userDto.getEmail()); // 쿠키 유효 시간 설정 idCookie.setMaxAge(7 * 24 * 60 * 60); // 7일 emailCookie.setMaxAge(7 * 24 * 60 * 60); passwordCookie.setMaxAge(7 * 24 * 60 * 60); usernameCookie.setMaxAge(7 * 24 * 60 * 60); // 쿠키 경로 설정 idCookie.setPath("/"); emailCookie.setPath("/"); passwordCookie.setPath("/"); usernameCookie.setPath("/"); } } public Boolean signIn(UserDto userdto) { try { userdto.setLast_accessed_at(LocalDateTime.now()); return userRepository.validateUser(userdto); } catch (NullPointerException e) { throw new GlobalException(ResponseStatus.INVALID_REQUEST); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre> <p>frontend에서 이메일 및 비밀번호를 입력 후 로그인하기 버튼을 누르게 되면 /users/signin API로 요청이 수행된다.</p> <p>회원가입이 되어 있는 사용자인지에 대한 검증을 진행하기 위해 validateUser() 메서드를 통해 회원 가입 여부를 판단한다. 검증이 완료된 이메일과 비밀번호라면 정상적으로 로그인이 진행되면서 cookie에 사용자 정보를 저장하여 세션 정보를 유지함과 동시에</p>
FRONT END	BACK END			
<pre>// Get localStorage token const token = localStorage.getItem("fcm_token"); // Do login try { const response = await fetch(process.env.REACT_APP_API_URL + "/users/signin", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ email, password, fcm_token: token, }), credentials: "include", }); if (response.status !== 200) { throw new Error("로그인에 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status !== 2002) { throw new Error("로그인에 실패하였습니다."); } alert("로그인에 성공하였습니다."); document.location.href = "/"; } catch (error) { console.log(error); alert("로그인에 실패하였습니다."); return; } }</pre> <p>이메일 로그인의 경우, 이메일 및 비밀번호를 입력 후 로그인하기 버튼을 클릭하면 백엔드에 이메일과 비밀번호를 전달해 로그인을 시도하며, 성공하면 Set-Cookie 헤더로 전송되는 쿠키를 이용해 세션 정보를 저장한다. 해당 정보는 계속해서 인증을 위해 쓰인다.</p> <pre>const onClickKakaoLogin = () => { const redirectUri = "http://localhost:3000/oauth/kakao"; document.location.href = "https://kauth.kakao.com/oauth/authorize?response_type=code&client_id=";</pre>	<pre>/login @PostMapping("/login") public ResponseEntity<String> signIn(@RequestBody UserDto userDto, HttpServletResponse response){ if (userService.signIn(userDto)) { // 쿠키 설정 Cookie idCookie = new Cookie("id", String.valueOf(userService.findUserIdByEmail(userDto.getEmail()))); Cookie emailCookie = new Cookie("email", userDto.getEmail()); Cookie passwordCookie = new Cookie("password", userDto.getPassword()); Cookie usernameCookie = new Cookie("username", userDto.getUsername()); fcmService.saveTokenByObject(userDto); fcmService.sendIncompleteMessage(userDto.getEmail()); // 쿠키 유효 시간 설정 idCookie.setMaxAge(7 * 24 * 60 * 60); // 7일 emailCookie.setMaxAge(7 * 24 * 60 * 60); passwordCookie.setMaxAge(7 * 24 * 60 * 60); usernameCookie.setMaxAge(7 * 24 * 60 * 60); // 쿠키 경로 설정 idCookie.setPath("/"); emailCookie.setPath("/"); passwordCookie.setPath("/"); usernameCookie.setPath("/"); } } public Boolean signIn(UserDto userdto) { try { userdto.setLast_accessed_at(LocalDateTime.now()); return userRepository.validateUser(userdto); } catch (NullPointerException e) { throw new GlobalException(ResponseStatus.INVALID_REQUEST); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre> <p>frontend에서 이메일 및 비밀번호를 입력 후 로그인하기 버튼을 누르게 되면 /users/signin API로 요청이 수행된다.</p> <p>회원가입이 되어 있는 사용자인지에 대한 검증을 진행하기 위해 validateUser() 메서드를 통해 회원 가입 여부를 판단한다. 검증이 완료된 이메일과 비밀번호라면 정상적으로 로그인이 진행되면서 cookie에 사용자 정보를 저장하여 세션 정보를 유지함과 동시에</p>			

```

18  (async () => {
19    try {
20      const url = process.env.REACT_APP_API_URL + "/login/oauth2/code/kakao" + "?code=" +
21        query + "&fcm=" + token;
22      const response = await fetch(url, {
23        method: "GET",
24        // headers: { "Content-Type": "application/json" },
25        credentials: "include",
26      });
27      if (response.status !== 200) {
28        throw new Error("로그인에 실패하였습니다.");
29      }
30      const jsonData = await response.json();
31      console.log(jsonData);
32      document.location.href = "/";
33    } catch (error) {
34      alert("로그인에 실패하였습니다.");
35      console.log(error);
36      document.location.href = "/login";
37    }
38  )());
39 }, [token, query]);

```

카카오 로그인의 경우, 외부 카카오 API
로그인 페이지로 리다이렉션을 진행해
로그인을 시도하며, 정상적으로 로그인이
성공되면 다시 리다이렉션을 이용해
백엔드에 로그인 토큰을 전달해 로그인을
시도한다.

로그인이 되었으므로 최근 접속 시간 정보를
갱신한다.

SQL

```

public Boolean validateUser(UserDto userdto) {
    String sql = "SELECT COUNT(*) FROM user WHERE email = ? AND password = ?";
    try {
        Integer count = jdbcTemplate.queryForObject(
            sql, Integer.class, userdto.getEmail(), userdto.getPassword());
        userdto.setId(findUserIdByEmail(userdto.getEmail()));
        updateLastAccessedAt(userdto);
        return count != null && count > 0;
    } catch (EmptyResultDataAccessException e) {
        return false;
    }
}

public void updateLastAccessedAt(UserDto userDto) {
    String sql = "UPDATE user SET last_accessed_at = ? WHERE id = ?";
    jdbcTemplate.update(sql, userDto.getLast_accessed_at(), userDto.getId());
}

```

user 테이블에서 유저가 입력한 이메일과 비밀번호와 일치하는 레코드의 개수를 세어
사용자의 유효성을 검증하고, 해당 유저의 id를 바탕으로 레코드를 찾아 최근 접속 시간에 대한
필드인 `last_accessed_at` 속성을 마지막 접속 시간으로 `update`한다

5, 6, 7. 개인정보 조회 및 수정, 회원 탈퇴

SCREEN	
<p>The screenshot shows the 'My Page' section of a web application. It displays the following user data:</p> <ul style="list-style-type: none"> 이메일 (Email): mirusu400@naver.com 이름 (Name): 김성진 나이 (Age): 24 전화번호 (Phone Number): 01098084536 몸무게 (Weight): 58 보유 금액 (Balance): 50000 <p>Below the form are two buttons: '수정' (Update) in blue and '회원탈퇴' (Logout) in red.</p>	<p>The screenshot shows the 'My Page' section of a web application. It displays the following user data:</p> <ul style="list-style-type: none"> 이메일 (Email): mirusu400@naver.com 이름 (Name): 김성진 나이 (Age): 24 전화번호 (Phone Number): 01098084536 몸무게 (Weight): 58 보유 금액 (Balance): 50000 <p>Below the form are two buttons: '수정' (Update) in blue and '회원탈퇴' (Logout) in red.</p>
DESCRIPTION	
<p>본 기능은 마이 페이지 화면을 통해 개인이 회원가입 시 입력했던 정보를 확인하고, 수정할 수 있는 화면이다. 수정 버튼을 이용해 이름, 나이, 전화번호, 몸무게, 비밀번호 등을 수정할 수 있다. 개인정보 페이지 내 회원탈퇴 버튼을 클릭하면 회원탈퇴를 진행할 수 있으며, 자동으로 계정이 삭제됨과 동시에 로그아웃 처리된다.</p>	<p>개인정보 조회를 위해 마이 페이지에 렌더링하기 위한 해당 사용자의 정보를 반환하기 위한 /users/get-userinfo API로 cookie에 저장된 id를 기반으로 요청을 수행하게 되면 아래 service 코드의 findUserById() 함수를 통해 id를 기반으로 해당 유저의 정보를 반환하기 위한 repository의 메서드를 호출하게 된다.</p> <pre>public UserDto findUserById(int id){ return userRepository.findUserById(id); }</pre>

```

(async () => {
  try {
    const response = await fetch(process.env.REACT_APP_API_URL + "/admin/modify-user", {
      method: "PATCH",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        id: userData.id,
        email: userData.email,
        username: newUserName || userData.username,
        password: newPassword || userData.password,
        phone_number: newPhone || userData.phone_number,
        weight: newWeight || userData.weight,
        age: newAge || userData.age,
        total_money: userData.total_money,
      }),
      credentials: "include",
    });
    if (response.status != 200) {
      throw new Error("회원정보 수정에 실패하였습니다.");
    }
    const jsonData = await response.json();
    if (jsonData.status == 201) {
      alert("회원정보 수정에 성공하였습니다.");
      setEditMode(false);
      setPassword("");
      setNewPassword("");
      setCheckNewPassword("");
      setIsPasswordSame(true);
      setIsCurrentPasswordCorrect(true);
      setNewAge();
      setNewPhone("");
      setNewWeight("");
      fetchData();
      return;
    }
  } catch (error) {
    console.error(error);
  }
});

```

수정 버튼을 눌러 수정 모드에 들어가면, 수정이 가능한 항목들을 **readonly** 옵션을 해제하여 수정 가능하도록 하고, 비밀번호 변경을 위해 비밀번호 입력 **Input** 또한 렌더링을 진행한다. 저장 버튼을 누르면 수정된 값을 통해 API를 통해 백엔드에 전달한다.

```

const handleUserDelete = () => {
  const isDelete = window.confirm("정말로 회원탈퇴를 하시겠습니까?");
  if (isDelete) {
    (async () => {
      try {
        const response = await fetch(process.env.REACT_APP_API_URL + "/users/delete", {
          method: "DELETE",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            id: user.id,
          }),
          credentials: "include",
        });
        if (response.status != 200) {
          throw new Error("회원탈퇴에 실패하였습니다.");
        }
        const jsonData = await response.json();
        if (jsonData.status == 200) {
          alert("회원탈퇴에 성공하였습니다.");
          // Remove all cookie
          document.cookie.split(";").forEach(function (c) {
            c.replace(/\s*,\s*/g, "").replace(/=(.+)/, "=expires=" + new Date().toUTCString() + ";path=/");
          });
          window.location.href = "/";
          return;
        } else {
          throw new Error("회원탈퇴에 실패하였습니다.");
        }
      } catch (error) {
        alert("회원탈퇴에 실패하였습니다.");
        return;
      }
    })();
  }
};

```

회원탈퇴 버튼을 통해 회원을 탈퇴할 때는, 회원을 삭제하는 API를 백엔드에 전달해 회원 정보 삭제를 시도하며, 정상적으로 삭제된 경우 모든 쿠키를 다 지워버려 세션 정보를 없앤 후 페이지를 새로고침하여 로그아웃을 진행한다.

repository를 통해 해당 id를 가진 user의 모든 정보가 담긴 **dto**가 **service**의 **findUserById()** 메서드가 반환받게 되고, 이를 다시 **controller**의 메서드로 반환함으로써 json type으로 **frontend** 측에 반환하게 된다.

```

@PatchMapping("/modify-user")
public ResponseEntity<?> modifyUser(@RequestBody UserDto userDto){
  adminService.modifyUser(userDto);
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_MODIFY_USER_INFO, result: null));
}

```

개인정보 수정을 위해 호출하는 **controller**의 **/modify-user** API이다. 수정할 정보들을 **frontend** 측에서 입력하여 json type으로 요청하게 되면 해당 정보들을 담은 **UserDto** 객체가 만들어진다.

```

public void modifyUser(UserDto userDto) {
  userRepository.modifyUser(userDto);
}

```

service로직의 **modifyUser()** 메서드의 인자로 넘겨주면 **repository**의 **modifyUser()** 메서드로 넘겨주어 수정된 유저의 필드 정보를 바탕으로 각 속성에 대한 업데이트를 수행하게 된다.

```

// 회원탈퇴
@DeleteMapping("/delete")
public ResponseEntity<?> withdrawal(@RequestBody UserDto userDto){
  userService.withdrawal(userDto.getId());
  return ResponseEntity.ok().body(new JsonResponse<>(ResponseStatus.SUCCESS_WITHDRAWAL));
}

```

회원탈퇴의 경우, 프론트엔드 측에서 회원탈퇴 버튼을 클릭했을 때 cookie에 저장되어 있는 해당 user의 id와 함께 /users/delete 엔드포인트로 요청을 수행하게 된다. 해당 컨트롤러에서는 전달받은 user의 id를 **userService**의 **withdrawal()** 메서드의 인자로 전달해준다.

```

public void withdrawal(int id) {
  userRepository.deleteUserById(id);
}

```

withdrawal() 메서드를 통해 전달받은 id를 **userRepository**의 **deleteUserById()** 메서드의 인자로 넘겨줌으로써 해당 id를 가진 user의 정보가 저장되어 있는 row를 삭제함으로써 회원 탈퇴 로직이 동작하게

	된다.
	SQL
	<pre>public UserDto findUserById(int id) { // EXCEPT String sql = "SELECT * FROM user " + "EXCEPT " + "SELECT * FROM user WHERE id <> ?"; try { return jdbcTemplate.queryForObject(sql, new Object[]{id}, new UserRowMapper()); } catch (EmptyResultDataAccessException e) { return null; } }</pre>
	<pre>public void modifyUser(UserDto userDto) { String sql = "UPDATE user SET email = ?, username = ?, password = ?, " + "phone_number = ?, weight = ?, age = ?, total_money = ? WHERE id = ?"; jdbcTemplate.update(sql, userDto.getEmail(), userDto.getUsername(), userDto.getPassword(), userDto.getPhone_number(), userDto.getWeight(), userDto.getAge(), userDto.getTotal_money(), userDto.getId()); }</pre>
	<pre>public void deleteUserById(int id) { // DELETE String sql = "DELETE FROM user WHERE id = ?"; jdbcTemplate.update(sql, id); }</pre>
	<p><code>findUserById()</code> 메서드는 개인정보 조회를 위해 해당 <code>user</code>의 <code>id</code>를 기반으로 정보가 담겨있는 객체인 <code>UserDto</code>를 반환하기 위해 실행하는 SQL 쿼리문이다. <code>USER</code> table의 <code>id</code>와 인자로 전달받은 <code>id</code>가 일치하는 <code>row</code>의 모든 <code>attribute</code>를 <code>select</code>하여 <code>UserDto</code> 객체와 <code>mapping</code>을 수행한 후 반환한다.</p> <p><code>modifyUser()</code> 메서드는 개인 정보 수정을 위해 사용한 SQL 쿼리문이 작성되어 있다. <code>USER</code> table에서 <code>userDto</code>의 <code>id</code>에 해당하는 <code>row</code>들의 속성을 <code>userDto</code>에 저장되어 있는 나머지 속성으로 새롭게 업데이트를 수행한다. 마찬가지로 <code>prepared statement</code>을 통해 안전하게 업데이트를 진행하였다.</p>

`deleteUserById()` 메서드는 유저가 회원탈퇴를 진행할 때 `USER` table에서 주어진 `id`와 일치하는 `row`를 `table`에서 삭제를 진행한다.

8, 9, 10, 11, 12. 자유게시판, 공지사항, 1:1 문의, 따릉이 고장(장애)신고 작성 및 조회

SCREEN

자유게시판

삶이란...



삶은 계란이다..

파일 첨부

뒤로가기

등록

번호	제목	작성자	조회수	작성일
[공지]	나 관리자야 ㅋ	관리자	1	2023-12-03 22:54
[공지]	2316. 살상역 8번출구 대여소 입시 폐쇄 안내	관리자	5	2023-12-03 04:02
[공지]	『고통사망하고 줄이기』 따로이 안전수칙 안내	관리자	6	2023-12-03 03:02
[공지]	출퇴근길 지켜야 할 자전거 에티켓	관리자	4	2023-12-03 03:00
13	삶이란..	김성진	2	2023-12-13 22:37
12	시험기간이라 힐드시죠? [1]	방치민	5	2023-12-03 19:41
11	마름이 너무 좋아요 [1]	강병준	5	2023-12-03 19:31
10	제 이름은	하지만	4	2023-12-03 04:22
9	너라는 데이터베이스에 [2]	병준왕	17	2023-12-03 04:10
8	사과드립니다..	김애풀	3	2023-12-03 04:05
7	파인애플이 죽으면?	김지안	1	2023-12-03 04:01
6	저는 인간 실격입니다..	방치민	13	2023-12-03 03:39
5	???. 행복동은 자전거의 산이 아닙니다.....jpg [2]	김성진	21	2023-12-03 03:29
4	마름을 마르를 버려나세요 [1]	방치민	20	2023-12-03 03:28

이전 1 다음

삶이란...

작성자: 김성진 조회수: 1 작성일: 2023-12-13 22:37



DESCRIPTION

본 기능은 각 게시판에 제목, 본문, 첨부파일 등을 작성해 글을 게시할 수 있고, 게시한 게시물은 게시판 종류에 따라 타인이 조회하거나, 댓글 등을 작성할 수 있다.

각 게시판의 종류마다 별개의 특성을 가지고 있는데, 공지사항은 관리자가 공지사항으로 지정한 글을 모아서 확인할 수 있으며, 자유게시판은 모든 사용자가 글을 작성, 조회, 댓글작성이 가능하다. 고장신고게시판은 모두가 글을 작성, 조회할 수 있지만 관리자만 댓글작성이 가능하며, 1:1 문의는 모두가 글을 작성할 수 있지만 본인이 작성한 글만 확인이 가능하다.

FRONT END	BACK END
<pre> <CKEditor editor={ClassicEditor} data={content} config={{ extraPlugins: [uploadPlugin] }} onReady={(editor) => { // You can store the "editor" and use when it is needed. console.log('Editor is ready to use!', editor); }} onChange={(event, editor) => { setContent(editor.getData()); setParentContent(editor.getData()); console.log({ event, editor, content }); }} onBlur={(event, editor) => { console.log('Blur.', editor); }} onFocus={(event, editor) => { console.log('Focus.', editor); }} /> </pre>	<pre> @PutMapping("/create-post") public ResponseEntity<JsonResponse> createBoard(@RequestBody BoardDto.CreateBoardDto createBoardDto) { BoardService.createBoard(createBoardDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); } @DeleteMapping("/delete-post/{id}") public ResponseEntity<JsonResponse> deleteBoard(@PathVariable("id") Long id) { BoardService.deleteBoard(id); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); } @PostMapping("/file-upload") public ResponseEntity<Map<String, String>> uploadFile(@RequestParam("file") MultipartFile multipartFile) { Map<String, String> response = new HashMap<>(); response = boardService.uploadFile(multipartFile); return ResponseEntity.ok(new JsonResponse(Map.ofString, String)(ResponseStatus.SUCCESS, response)); } @GetMapping("/get-board") public ResponseEntity<JsonResponse> getBoard(@RequestParam int id, @RequestParam int user_id) { BoardDto.GetBoardDto boardDto = boardService.getBoard(id, user_id); if(boardDto == null) throw new GlobalException(ResponseStatus.RESULT_NOT_EXIST); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, boardDto)); } </pre>

게시물 작성, 게시물 리스트, 게시물 조회 세개의 페이지를 작성하였으며 게시물 작성 시 CKEditor 라이브러리를 이용해 에디터를 구현하였다. CKEditor 추가 플러그인을 구현하여 드래그앤 드랍 / 버튼을 이용해 이미지를 첨부할수 있는 기능 또한 구현하였다. 조회 시 데이터베이스에 저장된 본문을 그대로 HTML로 표현하도록 구현하였다.

controller code는 위와 같이 작성되어 있으며, '/board/create-post'나 '/board/file-upload', '/board/get-board'로 요청하면 각각의 해당하는 controller 코드가 실행되게 된다.

우선 게시글 작성의 경우,

```

public void createBoard(BoardDto.CreateBoardDto createBoardDto) {
    try {
        UserDto.UserNameTypeDto userNameTypeDto = userRepository.findNameTypeByNameById(createBoardDto.getUserId());
        if(userNameTypeDto.getIsUser() != 0 && createBoardDto.isNotice() == 0) {
            throw new GlobalException(ResponseStatus.INVALID_AUTHORITY_NOTICE);
        }

        Board board = new Board(
            0, //auto increment id
            createBoardDto.getCategory_id(),
            createBoardDto.getWriter_id(),
            0, //views
            createBoardDto.getTitle(),
            createBoardDto.getContent(),
            createBoardDto.isNotice(),
            createBoardDto.getFile_name(),
            createBoardDto.getUrl(),
            LocalDateTime.now(), //create at
            LocalDateTime.now() //update at
        );
        boardRepository.insertBoard(board);
    }
}

```

위와 같이 Service code가 구현되어 있으며, 카테고리인 category_id로 해당 게시글의 카테고리를 알 수 있다. 2이면 자유게시판, 3이면 1:1 문의, 4이면 따릉이 고장신고이다. 단, user의 id가 관리자가 아니라면 공지사항은 작성하지 못하게 해 둔 것을 확인할 수 있다. 또한, 게시글을 작성할 때 파일을 업로드 할 수 있는데 파일의 경우 별도의 API로 구현해 두었기 때문에 따로 controller 코드가 존재하는 것을 확인할 수 있다.

```

    ...
    public Map<String, String> uploadMultipartFile(MultipartFile multipartFile) {
        try {
            if (!multipartFile.isEmpty()) {
                String fileName = multipartFile.getOriginalFilename();
                String uniqueFileName = UUID.randomUUID() + "-" + fileName;

                String currentPath = System.getProperty("user.dir");
                String filePath = currentPath + File.separator + "src" + File.separator + "main"
                    + File.separator + "resources" + File.separator + "static" + File.separator + uniqueFileName;

                multipartFile.transferTo(new File(filePath));

                Map<String, String> response = new HashMap<>();
                response.put("file_name", fileName);
                response.put("url", uniqueFileName);

                return response;
            } else {
                return null;
            }
        } catch (Exception e) {
            throw new GlobalException(ResponseStatus.UPLOAD_ERROR);
        }
    }
}

```

Service code는 위와 같고, 이 경우 Database가 아닌 로컬에 파일을 저장하고 해당 로컬의 경로를 URL로 프론트에 다시 보내주게 된다.

마지막으로 게시글 조회의 경우

```

@FunctionalInterface
public Supplier<BoardDto> getBoard(int id, int user_id) {
    try {
        BoardDto.BoardWithLike board = boardRepository.findById(id, user_id);
        if(board == null)
            return null;

        boardRepository.increaseViewCount(id);
        String username = "";
        UserDto.UserNameTypeDto userByNameTypeDto = userRepository.findNameTypeByNameById(board.getUserId());
        if (userByNameTypeDto.getUser_type() == 0) {
            username = "관리자";
        } else {
            username = userNameTypeDto.getUsername();
        }

        List<CommentDto> commentList = commentService.getCommentsEachBoardId(user_id);
        BoardDto.GetCategoryBoardId = new BoardDto.GetCategoryBoardId(id, user_id);
        username, board.getWriter_id(), board.getViews(), board.getTitle(),
        board.getContent(), board.isNotice(), board.getFileName(), board.getU_id(),
        board.getLikesCount(), board.isUserLiked(), board.getCreated_at(), board.getUpdated_at(), commentList);

        return board;
    } catch (CaptionNotDataAccessException ex) {
        throw new GlobalException(ResponseStatus.RESULT_NOT_EXIST);
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}

```

이와 같이 Service code가 구현되어 있으며 해당 게시글을 조회한 사람이 관리자이면 관리자라고 출력하고, 아니라면 실제 이름을 출력하게 된다. 이 경우 게시글을 조회하면 조회수를 1 증가시키게 되고, 해당 게시글에 달린 댓글들도 모두 가져와서 함께 반환해주는 것을 확인할 수 있다. 또한, 게시글을 조회할 때에는 사용자의 좋아요 여부 등도 함께 반환해준다.

추가로,

```

@GetMapping("/get-category-titles")
public ResponseEntity<JsonResponse> eachCategoryBoardTitle(@RequestParam int category_id) {
    List<BoardDto>.GetBoardTitleEto> boardTitleEtoList = boardService.getEachCategoryBoardTitle(category_id);
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, boardTitleEtoList));
}

```

이와 같이 각 카테고리에 해당하는 게시글들을 보기 위한 요청이 왔을 때 실행되는 컨트롤러가 존재하는데,

```

public List<BoardDto>.GetBoardTitleEto> artEachCategoryBoardTitle(int categoryId) {
    List<BoardDto>.GetBoardTitleEto> boardTitleEtoList = new ArrayList<>();
    try {
        List<Board>.BoardWithCommentsCount boardList = boardRepository.findListByCategoryId(categoryId);
        for(Board boardWithCommentsCount : boardList) {
            String username = "";
            UserDto.UserNameTypeDto userByNameTypeDto = userRepository.findNameTypeByNameById(boardWithCommentsCount.getWriter_id());
            if (userByNameTypeDto.getUser_type() == 0) {
                username = "관리자";
            } else {
                username = userNameTypeDto.getUsername();
            }

            boardTitleEtoList.add(new BoardDto.GetBoardTitleEto>(boardWithCommentsCount.getCategory_id(),
                boardWithCommentsCount.getWriter_id(), username, boardWithCommentsCount.getViews(),
                boardWithCommentsCount.getTitle(), boardWithCommentsCount.isNotice(),
                boardWithCommentsCount.getCreated_at(), boardWithCommentsCount.getComment_count()));
        }
    }
}

```

	<p>이러한 서비스 코드를 호출한다.</p> <p>이 코드에서는 <code>findTitleByCategoryId</code>를 이용해 해당하는 카테고리의 게시글의 제목, 작성자 등의 정보를 가져온 뒤 작성자가 관리자인 경우 이름을 관리자로 바꿔서 컨트롤러에 반환한다.</p>
SQL	
	<pre>public UserDto.UserNameTypeDto findNameTypeNameById (int user_id){ var userMapper = BeanPropertyRowMapper.newInstance(UserDto.UserNameTypeDto.class); UserDto.UserNameTypeDto userNameTypeDto = jdbcTemplate.queryForObject("SELECT username, user_type FROM `user` WHERE id=?", userMapper, user_id); jdbcTemplate.update("INSERT INTO `board` (`category_id`, `user_id`, `views`, `title`, " + "'content', 'notice', 'file_name', 'url', 'created_at', 'updated_at') VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)" , category_id, user_id, init_views, title, content, notice, file_name, url, create_at, update_at);</pre>
	<pre>public BoardDto.BoardWithLike findBoardById(int id, int userId) { var boardWithLikesMapper = BeanPropertyRowMapper.newInstance(BoardDto.BoardWithLike.class); BoardDto.BoardWithLike boardWithLikes = jdbcTemplate.queryForObject(// EXISTS "SELECT B.*," + "(SELECT COUNT(*) FROM board_like WHERE liked_id = B.id) AS likes_count, " + "EXISTS(SELECT 1 FROM board_like WHERE liked_id = B.id AND user_id = ?) AS user_liked " + "FROM board B WHERE B.id = ?", boardWithLikesMapper, userId, id); } public void increaseViewCount(int id) { jdbcTemplate.update("UPDATE `board` SET views = views + 1 WHERE id = ?", id); }</pre>
	<pre>public List<BoardDto.BoardWithCommentsCount> findTitleByCategoryId(int category_id) { var boardCommentMapper = BeanPropertyRowMapper.newInstance(BoardDto.BoardWithCommentsCount.class); return jdbcTemplate.query("SELECT B.id, B.category_id, B.user_id, B.views, B.title, B.notice, B.created_at, COUNT(C.id) as comment_count " + "FROM comment C RIGHT OUTER JOIN board B " + "ON C.write_id = B.id AND C.category_id = B.category_id " + "WHERE B.category_id=? " + "GROUP BY B.id " + "ORDER BY B.created_at DESC" , boardCommentMapper, category_id); }</pre>
<p>먼저, <code>findNameTypeNameById</code>의 경우, <code>user table</code>에서 <code>id</code>를 통해 <code>username</code>과 <code>user type</code>을 조회하여 반환한다. 이는 글을 작성한 작성자의 태입과 이름을 알고 글 수정이나 삭제 등을 허용할 수 있는지 판단할 때 사용한다.</p> <p>다음으로 글 작성 쿼리인데, 사용자가 작성한 정보들을 이용하여 <code>board table</code>에 <code>insert</code>를 하고 있는 것을 확인할 수 있다.</p>	

`findBoardById`의 경우, 하나의 게시글을 조회하는 부분으로 `board table`에서 `id`를 이용해 해당하는 `board` 정보를 조회한다. 이때, `board_like table`을 이용해 이 게시글을 조회한 사용자의 `id`로 해당 사용자가 이 게시글에 좋아요를 눌렀는지 판단하여 반환하고, `board id`에 해당하는 값을 COUNT하여 총 좋아요의 개수를 반환하는 것을 확인할 수 있다.

`increaseViewCount`의 경우, `board table`에서 이미 존재하는 게시글에 대해 해당 게시글을 조회했을 때 조회수에 1을 더한 형태로 값을 update하여 조회수를 늘리는 동작을 할 때 사용하는 쿼리이다.

마지막으로, 해당 카테고리의 게시글의 `title`들을 조회하는 쿼리인 `findTitleByCategoryId`는 `comment`의 개수도 함께 조회하기 위해 `comment table`과 JOIN을 하는데, `comment`가 없는 게시글도 정보를 확인할 수 있도록 `RIGHT OUTER JOIN`을 수행한다. 이를 통해 게시글의 간단한 정보와 댓글 수 등을 반환할 수 있게 된다.

13, 14. 게시물 수정 및 삭제

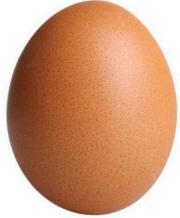
SCREEN

자유게시판 수정

상이란... (수정)

삶은 계란이다.. (수정아님)

파일 첨부 등록

<p>삶이란... (수정)</p> <p>작성자: 김성진 조회수: 3 작성일: 2023-12-13 22:37</p>  <p>삶은 계란이다.. (수정아님)</p> <p>목록으로 이동 수정 삭제</p> <p style="text-align: right;">0</p>					
<p>localhost:3000 내용:</p> <p>삭제되었습니다.</p> <p style="text-align: right;">확인</p>					
<h3>DESCRIPTION</h3>					
<p>게시물 조회 페이지 내에서 본인이 작성하거나, 관리자라면 수정 혹은 삭제 버튼이 존재한다. 수정 버튼을 클릭할 시 기존에 게시한 게시물을 수정할 수 있으며, 삭제 버튼을 클릭하면 게시물을 삭제한다.</p>					
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 5px;">FRONT END</th><th style="padding: 5px;">BACK END</th></tr> </thead> <tbody> <tr> <td style="padding: 10px;"> <pre> <Box sx={{ display: "flex", my: 3, flexDirection: "row", alignItems: "center", width: "100%", justifyContent: "space-between", gap: 1, }}> <Box> <Button variant="contained" sx={{ mt: 3, mr: 1 }} component={RouterLink} to={beforeLink}> 목록으로 이동 </Button> <Button> {boardData.user_name === user.username user.user_type === 0} <Button variant="contained" sx={{ mt: 3, mr: 1 }} color="warning" component={RouterLink} to={editLink + `?id=\${boardData.id}`}> 수정 </Button> <Button variant="contained" sx={{ mt: 3 }} onClick={handleDelete} color="error"> 삭제 </Button> </Button> </Box> </Box> </pre> <p>백엔드에서 제공한 데이터를 바탕으로 수정 혹은 삭제 버튼을 렌더링하도록 구현하였고(사용자명=게시물의 작성자명), 수정 버튼을 누르면 수정 페이지로 이동해 기존 글을 CKEditor로 수정, 삭제의 경우 백엔드에 삭제 API를 전달하여 글 삭제</p> </td><td style="padding: 10px;"> <pre> @PatchMapping("/modify-board") private ResponseEntity<JsonResponse> modifyBoard(@RequestBody BoardDto.ModifyBoardDto modifyBoardDto) { boardService.modifyBoard(modifyBoardDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); } no usages ▾ 마지막 @DeleteMapping("/delete-board") private ResponseEntity<JsonResponse> deleteBoard(@RequestBody BoardDto.BoardDeleteDto boardDeleteDto) { int board_id = boardService.deleteBoard(boardDeleteDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, board_id)); } </pre> <p>Controller 코드로, '/board/modify-board' 또는 '/board/delete-board'로 요청을 받게 되면 해당하는 controller 코드를 수행한다.</p> <p>먼저 게시글 수정의 경우</p> </td></tr> </tbody> </table>	FRONT END	BACK END	<pre> <Box sx={{ display: "flex", my: 3, flexDirection: "row", alignItems: "center", width: "100%", justifyContent: "space-between", gap: 1, }}> <Box> <Button variant="contained" sx={{ mt: 3, mr: 1 }} component={RouterLink} to={beforeLink}> 목록으로 이동 </Button> <Button> {boardData.user_name === user.username user.user_type === 0} <Button variant="contained" sx={{ mt: 3, mr: 1 }} color="warning" component={RouterLink} to={editLink + `?id=\${boardData.id}`}> 수정 </Button> <Button variant="contained" sx={{ mt: 3 }} onClick={handleDelete} color="error"> 삭제 </Button> </Button> </Box> </Box> </pre> <p>백엔드에서 제공한 데이터를 바탕으로 수정 혹은 삭제 버튼을 렌더링하도록 구현하였고(사용자명=게시물의 작성자명), 수정 버튼을 누르면 수정 페이지로 이동해 기존 글을 CKEditor로 수정, 삭제의 경우 백엔드에 삭제 API를 전달하여 글 삭제</p>	<pre> @PatchMapping("/modify-board") private ResponseEntity<JsonResponse> modifyBoard(@RequestBody BoardDto.ModifyBoardDto modifyBoardDto) { boardService.modifyBoard(modifyBoardDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); } no usages ▾ 마지막 @DeleteMapping("/delete-board") private ResponseEntity<JsonResponse> deleteBoard(@RequestBody BoardDto.BoardDeleteDto boardDeleteDto) { int board_id = boardService.deleteBoard(boardDeleteDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, board_id)); } </pre> <p>Controller 코드로, '/board/modify-board' 또는 '/board/delete-board'로 요청을 받게 되면 해당하는 controller 코드를 수행한다.</p> <p>먼저 게시글 수정의 경우</p>	
FRONT END	BACK END				
<pre> <Box sx={{ display: "flex", my: 3, flexDirection: "row", alignItems: "center", width: "100%", justifyContent: "space-between", gap: 1, }}> <Box> <Button variant="contained" sx={{ mt: 3, mr: 1 }} component={RouterLink} to={beforeLink}> 목록으로 이동 </Button> <Button> {boardData.user_name === user.username user.user_type === 0} <Button variant="contained" sx={{ mt: 3, mr: 1 }} color="warning" component={RouterLink} to={editLink + `?id=\${boardData.id}`}> 수정 </Button> <Button variant="contained" sx={{ mt: 3 }} onClick={handleDelete} color="error"> 삭제 </Button> </Button> </Box> </Box> </pre> <p>백엔드에서 제공한 데이터를 바탕으로 수정 혹은 삭제 버튼을 렌더링하도록 구현하였고(사용자명=게시물의 작성자명), 수정 버튼을 누르면 수정 페이지로 이동해 기존 글을 CKEditor로 수정, 삭제의 경우 백엔드에 삭제 API를 전달하여 글 삭제</p>	<pre> @PatchMapping("/modify-board") private ResponseEntity<JsonResponse> modifyBoard(@RequestBody BoardDto.ModifyBoardDto modifyBoardDto) { boardService.modifyBoard(modifyBoardDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); } no usages ▾ 마지막 @DeleteMapping("/delete-board") private ResponseEntity<JsonResponse> deleteBoard(@RequestBody BoardDto.BoardDeleteDto boardDeleteDto) { int board_id = boardService.deleteBoard(boardDeleteDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, board_id)); } </pre> <p>Controller 코드로, '/board/modify-board' 또는 '/board/delete-board'로 요청을 받게 되면 해당하는 controller 코드를 수행한다.</p> <p>먼저 게시글 수정의 경우</p>				

요청을 보내도록 구현하였다.

```
public void modifyBoard(BoardDto modifyBoardDto) {
    try {
        UserDto.userNameTypeDto userTypeNameTypeDto = userRepository.findNameTypeByNameById(modifyBoardDto.getUserId());
        if (userTypeNameTypeDto.getUser_type() != 0) {
            if (modifyBoardDto.isNotice()) {
                throw new GlobalException(ResponseStatus.INVALID_AUTHORITY_NOTICE);
            }
        }
        int user_id = boardRepository.getBoardWriterId(modifyBoardDto.getId());
        if (user_id != modifyBoardDto.getUserId()) {
            throw new GlobalException(ResponseStatus.INVALID_AUTHORITY_MODIFY);
        }
    }
    Board board = new Board(
        modifyBoardDto.getId(),
        modifyBoardDto.getCategory_id(),
        0,
        0,
        modifyBoardDto.getTitle(),
        modifyBoardDto.getContent(),
        modifyBoardDto.isNotice(),
        modifyBoardDto.getFileName(),
        modifyBoardDto.getUnit(),
        null,
        LocalDateTime.now()
    );
    boardRepository.modifyBoard(board);
}
```

위 코드를 살펴보면, 관리자이거나 작성자일 경우에만 글 수정이 가능하기 때문에 해당하는 작성자가 아니면 수정이 불가능하도록 하였고, 작성자거나 관리자일 경우 수정이 가능하도록 구현하였다.

게시글 삭제의 경우

```
public int deleteBoard(BoardDeleteDto boardDeleteDto) {
    try {
        UserDto.userNameTypeDto userTypeNameTypeDto = userRepository.findNameTypeByNameById(boardDeleteDto.getUserId());
        if (userTypeNameTypeDto.getUser_type() != 0) {
            int real_user = boardRepository.getBoardWriterId(boardDeleteDto.getId());
            if (real_user != boardDeleteDto.getUserId()) {
                throw new GlobalException(ResponseStatus.INVALID_AUTHORITY_DELETE);
            }
        }
        boardRepository.deleteBoard(boardDeleteDto.getId());
        return boardDeleteDto.getId();
    } catch (GlobalException e) {
        throw e;
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}
```

동일하게 관리자이거나 작성자가 아니면 삭제가 불가능하도록 구현하였고, 삭제할 수 있는 권한이 있는 사람이면 해당 게시글을 삭제하도록 구현하였다.

SQL

```
public Integer getBoardWriterId(int boardId) {
    return jdbcTemplate.queryForObject(
        "SELECT user_id FROM board WHERE id=?",
        Integer.class,
        boardId
    );
}
```

```

usage: java -jar 게시판.jar
public void modifyBoard(Board board) {
    int id = board.getId();
    int category_id = board.getCategory_id();
    String title = board.getTitle();
    String content = board.getContent();
    boolean notice = board.isNotice();
    String file_name = board.getFile_name();
    String url = board.getUrl();
    LocalDateTime update_at = board.getUpdated_at();

    jdbcTemplate.update("UPDATE `board` SET " +
        "title=?, " +
        "content= ?, " +
        "notice=? , " +
        "file_name=? , " +
        "url=? , " +
        "updated_at=? " +
        "WHERE id=? AND category_id=?"
        , title, content, notice, file_name, url, update_at, id, category_id);
}

usage: java -jar 게시판.jar
public void deleteBoard(int id) {
    jdbcTemplate.update("DELETE FROM `board` WHERE id=?",
        id);
}

```

먼저, 위에서 설명했던 바와 같은 `findNameTypeByNameById`를 사용하고 있다. 이는 코드 설명은 생략하고, 게시글 작성자의 이름과 태입을 반환하는 역할이다.

`getBoardWriterId`의 경우, `board id`를 이용하여 해당하는 게시글의 작성자 `id`를 조회하는 쿼리이다. 이는 게시글의 작성자와 수정/삭제를 시도하는 이용자가 같은지 판단하기 위해 사용한다.

`modifyBoard`는 게시글을 수정하기 위한 쿼리로, 사용자가 입력한 데이터로 게시글 내용을 수정하는 것을 확인할 수 있다.

`deleteBoard`의 경우에는 `board table`에서 `id`에 해당하는 `board` 정보를 삭제하는 쿼리이다.

15. 대여소 현황 조회

SCREEN

	<h3>시작 대여소</h3> <p>자전거 대여소 위치</p> <p>[ST-2074] 서울특별시 노원…</p> <p>자전거 대수 : 1 평점 : 5점</p> <p>★ 후기 보기</p> <p>자전거 대여</p>
DESCRIPTION	
<p>본 기능은 대여소의 자전거 대수, 평균 평점, 즐겨찾기 여부를 확인할 수 있는 기능이다. 메인 페이지 내에서 지도에 반영된 마커를 클릭하면, 해당 대여소의 주소와 평점, 자전거 대수를 확인할 수 있으며, 즐겨찾기와 자전거 대수, 평점은 좌측의 대여 패널에서 확인할 수 있다.</p>	
FRONT END	BACK END
<pre>const infoWindow = new kakao.maps.InfoWindow({ content: ReactDOMServer.renderToString(<div 100,="" 12,="" 175="" 5,="" font-size:="" minheight:="" minwidth:="" padding:="" style="{{" }}=""> [{lendplace_id}]
 {bikeStation.statn_addr2 ? bikeStation.statn_addr2 : bikeStation.statn_addr1}

 <div style={{ display: "flex", flexDirection: "row", justifyContent: "space-between", alignItems: "center", }}> <Typography variant="body1" sx={{ px: 2, py: 0, my: 0 }} style={{ marginTop: 0, marginBottom: 0 }}>평점 : {bikeStation.average_rating}</Typography> <Typography variant="body1" sx={{ px: 2, py: 0, my: 0 }} style={{ marginTop: 0, marginBottom: 0 }}>자전거 수 : {bikeStation.usable_bikes}</Typography> </div> </div>), removable: true, });</pre>	<p>대여소 현황 조회는 하나의 대여소에 대한 자세한 정보를 조회하는 것으로, '/station/get-lendplace-status'로 요청하면</p> <pre>@GetMapping("/station/lendplace-status") public ResponseEntity<JsonResponse> getStationStatus(@RequestParam String lendplace_id, @RequestParam int user_id) { BikeStationDto.BikeStationStatus bikeStationStatus = bikeStationService.getStationStatus(lendplace_id, user_id); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationStatus)); }</pre>
<p>카카오맵 API의 마커 기능을 활용해 필요한 정보를 클릭시 렌더링하도록 이벤트 핸들러를 등록하고, 해당 마커를 맵에 보이게 렌더링하였다. 카카오맵 JS 라이브러리와 리액트간의 호환 문제로 인해 마커 내 버튼 삽입 등은 불가능하였고, 즐겨찾기 설정, 후기 보기 등은 좌측의 대여 패널에서 확인할 수 있도록 조정하였다.</p>	<p>이와 같은 controller 로직이 동작하여 service code를 호출하게 된다.</p> <pre>public BikeStationDto.BikeStationStatus getBikeStationStatus(String lendplaceId, int userId) { try { BikeStationDto.BikeStationStatus bikeStationStatusWithUsername = bikeStationRepository.findStatusBy(lendplaceId, userId); return bikeStationStatusWithUsername; } catch (GlobalException e) { throw new GlobalException(e.getMessage()); } catch (Exception e) { throw new GlobalException(ResponseStatus.RESULT_NOT_EXIST); } }</pre> <p>Service code를 확인하면, 사용자가 controller에게 요청할 때 함께 보냈던 대여소의 id, 즉 <code>lendplace_id</code>와 이를 호출한 사용자의 <code>user_id</code>를 이용해 Repository에서 해당 대여소의 정보를 조회한 뒤, 반환하는 과정이다.</p>
SQL	

```

    public BikeStationDto.BikeStationStatus findStatusById(String lendplaceId, int userId) {
        var bikeMapper = BeanPropertyRowMapper.newInstance(BikeStationDto.BikeStationStatus.class);
        List<BikeStationRatingDto.BikeStationReview> reviews = jdbcTemplate.query(
            "SELECT user_id, rating, review FROM bikestationrating WHERE lendplace_id = ?",
            new BeanPropertyRowMapper<>(BikeStationRatingDto.BikeStationReview.class),
            lendplaceId
        );

        reviews.forEach(review -> {
            String username = jdbcTemplate.queryForObject(
                "SELECT username FROM user WHERE id = ?",
                new Object[]{review.getUserId()},
                String.class
            );

            review.setUsername(username);
        });
    }

    BikeStationDto.BikeStationStatus bikeStationStatus = jdbcTemplate.queryForObject(
        "SELECT BS.lendplace_id, BS.statn_addr1, BS.statn_addr2, BS.startn_lat, BS.startn_lnt, " +
        "      COALESCE(BC.available_bikes, 0) AS usable_bikes, " +
        "      COALESCE(AVG(BR.rating), 0) AS average_rating, " +
        "      CASE WHEN FAV.lendplace_id IS NOT NULL THEN TRUE ELSE FALSE END AS favorite " +
        "FROM bikestationinformation BS " +
        "LEFT JOIN ( " +
        "    SELECT lendplace_id, COUNT(*) AS available_bikes " +
        "    FROM bike " +
        "    WHERE use_status = 0 AND bike_status = 1 " +
        "    GROUP BY lendplace_id " +
        ") AS BC ON BS.lendplace_id = BC.lendplace_id " +
        "LEFT JOIN bikestationrating BR ON BS.lendplace_id = BR.lendplace_id " +
        "LEFT JOIN favorite FAV ON BS.lendplace_id = FAV.lendplace_id AND FAV.user_id = ? " +
        "WHERE BS.lendplace_id = ? " +
        "GROUP BY BS.lendplace_id",
        bikeMapper,
        userId, lendplaceId
    );
    bikeStationStatus.setBikeStationReviews(reviews);
}

```

대여소에 대한 후기 정보도 함께 출력해야 하기 때문에 `bikestationrating` table에서 `lendplace_id`로 후기 정보와 평점, 후기 작성자를 조회한다. 이때 후기 작성자의 이름은 `user_id`가 아닌 `username`으로 반환되어야 하므로 `user` table에서 `user_id`를 이용해 `username`으로 변경한다. 이를 통해 모든 리뷰들을 제공할 수 있다.

이후 쿼리문을 살펴보면, 대여소의 `id`, 주소와 위치정보, 이용 가능한 자전거 수, 평균 평점, 사용자의 즐겨찾기 여부를 반환하고 있는 것을 확인할 수 있다.

우선 내부 `SELECT`문의 경우 각 대여소에서 사용 가능한 자전거의 수를 계산하게 되는데, 여기서 `use_status = 0`은 사용 가능한 상태를, `bike_status = 1`은 자전거가 정상 상태임을 나타낸다. 이 결과와 `bikestationinformation` table을 `lendplace_id`를 바탕으로 `LEFT JOIN`하게 되는데, 즉 이용 가능한 자전거 정보가 없어도 `bikestationinformation` 정보는 포함된다는 것이다.

또한 지금까지의 결과에서 평균 평점 정보를 위해 `bikestationrating` table과 `lendplace_id`를 바탕으로 `LEFT JOIN`하며, 이 결과를 `favorite` table과 `lendplace_id`와 `user_id`를 바탕으로

LEFT JOIN한다. 이러한 결과를 대여소 id인 lendplace_id로 둑어주면 각 대여소의 세부 정보를 조회할 수 있게 된다.

16, 17, 18. 따릉이 이용시간, 이용횟수, 이용거리 랭킹 확인

SCREEN																																																																																																																																						
<p>랭킹</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="4">이용시간 랭킹</th> </tr> </thead> <tbody> <tr> <td>등수</td> <td>이름</td> <td>이메일</td> <td>사용시간</td> </tr> <tr> <td>1</td> <td>병준강</td> <td>kang@naver.com</td> <td>281</td> </tr> <tr> <td>2</td> <td>하지민</td> <td>jimin@naver.com</td> <td>86</td> </tr> <tr> <td>3</td> <td>홍길동</td> <td>hong@gmail.com</td> <td>8</td> </tr> <tr> <td>4</td> <td>마이콜</td> <td>mike@naver.com</td> <td>4</td> </tr> <tr> <td>5</td> <td>하지민</td> <td>but@gmail.com</td> <td>1</td> </tr> <tr> <td>6</td> <td>하지현</td> <td>jihyun@naver.com</td> <td>1</td> </tr> <tr> <td>7</td> <td>강병준</td> <td>lucid_dawn@naver.com</td> <td>1</td> </tr> <tr> <td>8</td> <td>푸바오</td> <td>fubao@naver.com</td> <td>1</td> </tr> <tr> <td>9</td> <td>김지민</td> <td>apple@gmail.com</td> <td>0</td> </tr> </tbody> </table> <p style="text-align: center;">이전 1 다음</p>	이용시간 랭킹				등수	이름	이메일	사용시간	1	병준강	kang@naver.com	281	2	하지민	jimin@naver.com	86	3	홍길동	hong@gmail.com	8	4	마이콜	mike@naver.com	4	5	하지민	but@gmail.com	1	6	하지현	jihyun@naver.com	1	7	강병준	lucid_dawn@naver.com	1	8	푸바오	fubao@naver.com	1	9	김지민	apple@gmail.com	0	<p>랭킹</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="4">이용횟수 랭킹</th> </tr> </thead> <tbody> <tr> <td>등수</td> <td>이름</td> <td>이메일</td> <td>사용횟수</td> </tr> <tr> <td>1</td> <td>병지민</td> <td>jimin@naver.com</td> <td>5</td> </tr> <tr> <td>2</td> <td>하지민</td> <td>but@gmail.com</td> <td>3</td> </tr> <tr> <td>3</td> <td>홍길동</td> <td>hong@gmail.com</td> <td>3</td> </tr> <tr> <td>4</td> <td>마이콜</td> <td>mike@naver.com</td> <td>2</td> </tr> <tr> <td>5</td> <td>김지민</td> <td>apple@gmail.com</td> <td>2</td> </tr> <tr> <td>6</td> <td>병준강</td> <td>kang@naver.com</td> <td>2</td> </tr> <tr> <td>7</td> <td>강병준</td> <td>lucid_dawn@naver.com</td> <td>1</td> </tr> <tr> <td>8</td> <td>하지현</td> <td>jihyun@naver.com</td> <td>1</td> </tr> <tr> <td>9</td> <td>푸바오</td> <td>fubao@naver.com</td> <td>1</td> </tr> </tbody> </table> <p style="text-align: center;">이전 1 다음</p>	이용횟수 랭킹				등수	이름	이메일	사용횟수	1	병지민	jimin@naver.com	5	2	하지민	but@gmail.com	3	3	홍길동	hong@gmail.com	3	4	마이콜	mike@naver.com	2	5	김지민	apple@gmail.com	2	6	병준강	kang@naver.com	2	7	강병준	lucid_dawn@naver.com	1	8	하지현	jihyun@naver.com	1	9	푸바오	fubao@naver.com	1	<p>랭킹</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="4">이용거리 랭킹</th> </tr> </thead> <tbody> <tr> <td>등수</td> <td>이름</td> <td>이메일</td> <td>거리</td> </tr> <tr> <td>1</td> <td>하지현</td> <td>jihyun@naver.com</td> <td>14441</td> </tr> <tr> <td>2</td> <td>병준강</td> <td>kang@naver.com</td> <td>12078</td> </tr> <tr> <td>3</td> <td>하지민</td> <td>jimin@naver.com</td> <td>6944</td> </tr> <tr> <td>4</td> <td>홍길동</td> <td>hong@gmail.com</td> <td>4533</td> </tr> <tr> <td>5</td> <td>하지민</td> <td>but@gmail.com</td> <td>3866</td> </tr> <tr> <td>6</td> <td>마이콜</td> <td>mike@naver.com</td> <td>1362</td> </tr> <tr> <td>7</td> <td>김지민</td> <td>apple@gmail.com</td> <td>1342</td> </tr> <tr> <td>8</td> <td>강병준</td> <td>lucid_dawn@naver.com</td> <td>776</td> </tr> <tr> <td>9</td> <td>푸바오</td> <td>fubao@naver.com</td> <td>392</td> </tr> </tbody> </table> <p style="text-align: center;">이전 1 다음</p>	이용거리 랭킹				등수	이름	이메일	거리	1	하지현	jihyun@naver.com	14441	2	병준강	kang@naver.com	12078	3	하지민	jimin@naver.com	6944	4	홍길동	hong@gmail.com	4533	5	하지민	but@gmail.com	3866	6	마이콜	mike@naver.com	1362	7	김지민	apple@gmail.com	1342	8	강병준	lucid_dawn@naver.com	776	9	푸바오	fubao@naver.com	392
이용시간 랭킹																																																																																																																																						
등수	이름	이메일	사용시간																																																																																																																																			
1	병준강	kang@naver.com	281																																																																																																																																			
2	하지민	jimin@naver.com	86																																																																																																																																			
3	홍길동	hong@gmail.com	8																																																																																																																																			
4	마이콜	mike@naver.com	4																																																																																																																																			
5	하지민	but@gmail.com	1																																																																																																																																			
6	하지현	jihyun@naver.com	1																																																																																																																																			
7	강병준	lucid_dawn@naver.com	1																																																																																																																																			
8	푸바오	fubao@naver.com	1																																																																																																																																			
9	김지민	apple@gmail.com	0																																																																																																																																			
이용횟수 랭킹																																																																																																																																						
등수	이름	이메일	사용횟수																																																																																																																																			
1	병지민	jimin@naver.com	5																																																																																																																																			
2	하지민	but@gmail.com	3																																																																																																																																			
3	홍길동	hong@gmail.com	3																																																																																																																																			
4	마이콜	mike@naver.com	2																																																																																																																																			
5	김지민	apple@gmail.com	2																																																																																																																																			
6	병준강	kang@naver.com	2																																																																																																																																			
7	강병준	lucid_dawn@naver.com	1																																																																																																																																			
8	하지현	jihyun@naver.com	1																																																																																																																																			
9	푸바오	fubao@naver.com	1																																																																																																																																			
이용거리 랭킹																																																																																																																																						
등수	이름	이메일	거리																																																																																																																																			
1	하지현	jihyun@naver.com	14441																																																																																																																																			
2	병준강	kang@naver.com	12078																																																																																																																																			
3	하지민	jimin@naver.com	6944																																																																																																																																			
4	홍길동	hong@gmail.com	4533																																																																																																																																			
5	하지민	but@gmail.com	3866																																																																																																																																			
6	마이콜	mike@naver.com	1362																																																																																																																																			
7	김지민	apple@gmail.com	1342																																																																																																																																			
8	강병준	lucid_dawn@naver.com	776																																																																																																																																			
9	푸바오	fubao@naver.com	392																																																																																																																																			
DESCRIPTION																																																																																																																																						
<p>따릉이 이용시간, 따릉이 이용횟수, 따릉이 이용거리 랭킹을 확인할 수 있다. 가장 많이 사용한 사람 순서대로 내림차순으로 정렬하여 보여주며, 모든 사용자들을 조회하여 순위를 보여준다.</p>																																																																																																																																						
FRONT END		BACK END																																																																																																																																				
<pre>useEffect(() => { // TODO: Fetch ticket data (async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + "/get-highest-useTime", { method: "GET", headers: { "Content-Type": "application/json" }, }); if (response.status != 200) { throw new Error("데이터를 가져오는데 실패하였습니다."); } const jsonData = await response.json(); setRankingData(jsonData.result); setTotalPage(Math.floor(jsonData.result.length / 10) + 1); setShowData(jsonData.result.slice(0, 10)); } catch (error) { alert("데이터를 가져오는데 실패하였습니다."); } })(); });</pre>		<pre>@GetMapping("/get-highest-useTime") public ResponseEntity<> getTopUseTime() { List<VisualizationUserLogtoUserUserInfo> userLog = userLogService.getTopUseTime(); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, userLog)); } no usages ▲ 방지됨 @GetMapping("/get-highest-useCount") public ResponseEntity<> getTopUseCount() { List<VisualizationUserLogtoUserUserInfo> userLog = userLogService.getTopUseCount(); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, userLog)); } no usages ▲ 방지됨 @GetMapping("/get-highest-distance") public ResponseEntity<> getTopUseDistance() { List<VisualizationUserLogtoUserDistanceInfo> userLog = userLogService.getTopUseDistance(); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, userLog)); }</pre>																																																																																																																																				
<p>각 테이블별 API를 백엔드에서 제공받아, 이를 테이블로 렌더링하였다. 순위가 1위에서 10위까지만 나오는것이 아닌, 모든 사용자에 대한 순위가 나오므로 이전/다음 버튼을 통한 페이지네이션을 구현하였다. 또한, 각 탭간의 전환은 상단의 버튼그룹을 이용하여 이동할 수 있도록 구현하였다.</p>		<p>위 코드는 controller의 코드이고, 'get-highest-useTime', 'get-highest-useCount', 'get-highest-distance'로 요청하게 되면 각 컨트롤러 코드가 실행된다. 순서대로 따릉이 이용시간, 이용횟수, 이용거리 랭킹을 반환해 주게 되는데 각각의 코드들은 모두</p>																																																																																																																																				

```

public List<VisualizationUserlogDto.userUseTimeInfo> getTopUseTime() {
    try {
        return userlogRepository.getTopUseTime();
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}

1 usage ▲ 방지됨
public List<VisualizationUserlogDto.userUseCountInfo> getTopUseCount() {
    try {
        return userlogRepository.getTopuseCount();
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}

1 usage ▲ 방지됨
public List<VisualizationUserlogDto.userUseDistanceInfo> getTopUseDistance() {
    try {
        return userlogRepository.getTopUseDistance();
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}

```

서비스 코드는 단순하게 쿼리만 실행하고 그 결과를 다시 사용자에게 반환해주기 위해 위와 같다.

SQL

```

public List<VisualizationUserlogDto.userUseTimeInfo> getTopUseTime() {
    var rowMapper = BeanPropertyRowMapper.newInstance(VisualizationUserlogDto.userUseTimeInfo.class);
    return jdbcTemplate.query("SELECT U.* , SUM(UL.use_time) AS total_use_time " +
        "FROM userlog UL " +
        "JOIN user U ON UL.user_id = U.id " +
        "WHERE UL.user_id IS NOT NULL AND UL.use_time IS NOT NULL " +
        "GROUP BY U.id " +
        "ORDER BY total_use_time DESC; ",
        rowMapper
    );
}

public List<VisualizationUserlogDto.userUseCountInfo> getTopUseCount() {
    var rowMapper = BeanPropertyRowMapper.newInstance(VisualizationUserlogDto.userUseCountInfo.class);
    return jdbcTemplate.query("SELECT U.* , COUNT(UL.user_id) AS total_use_count " +
        "FROM user U LEFT JOIN userlog UL ON U.id = UL.user_id " +
        "WHERE UL.user_id IS NOT NULL " +
        "GROUP BY U.id " +
        "ORDER BY total_use_count DESC; ",
        rowMapper
    );
}

1 usage ▲ 방지됨
public List<VisualizationUserlogDto.userUseDistanceInfo> getTopUseDistance() {
    var rowMapper = BeanPropertyRowMapper.newInstance(VisualizationUserlogDto.userUseDistanceInfo.class);
    return jdbcTemplate.query("SELECT U.* , SUM(UL.use_distance) AS total_use_distance " +
        "FROM user U LEFT JOIN userlog UL ON U.id = UL.user_id " +
        "WHERE UL.user_id IS NOT NULL AND UL.use_distance IS NOT NULL " +
        "GROUP BY U.id " +
        "ORDER BY total_use_distance DESC; ",
        rowMapper
    );
}

```

순서대로 이용 시간, 이용 횟수, 이용 거리 순으로 이용자 정보와 이용 정보를 반환하는 쿼리이다.

우선 이용 시간은 userlog table을 user의 id를 이용해서 user table과 결합하여 두 테이블의 정보를 출력할 수 있도록 한다.

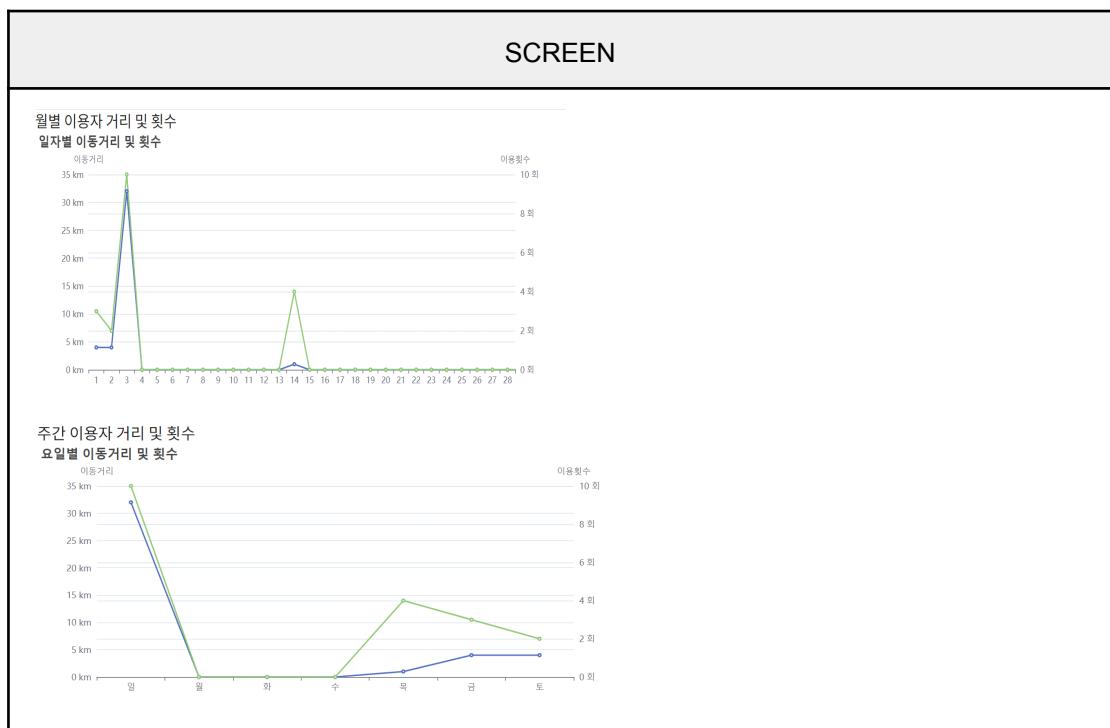
이때, user의 id가 null이거나 use_time이 null인 경우는 제외하고, user의 id로 Group By하여 각각의 GROUP에 대한 use_time값을 SUM으로 더해 user 정보와 함께 출력한다. 단, 순서는 total_use_time, 즉 각 user의 이용 시간이 큰 순서대로 정렬한다.

이용 횟수는 userlog table을 user의 id를 이용해서 user table과 결합하여 두 테이블의 정보를 출력할 수 있도록 한다.

이때, user의 id가 null인 경우는 제외하고, user의 id로 Group By하여 각각의 GROUP내에 log의 개수를 COUNT를 이용하여 측정한다. 이후, 이를 user 정보와 함께 제공한다. 이때 정보의 순서는 count가 큰 데이터 먼저 제공하게 된다.

이용 거리가 긴 순서는 동일하게 userlog table을 user의 id를 이용해서 user table과 결합하여 두 테이블의 정보를 출력할 수 있도록 하며, 이때, user의 id가 null이거나 use_distance 값이 null인 경우는 제외해야 한다. 다음으로 user의 id로 Group By하여 각각의 GROUP에 포함되어 있는 각각의 log의 use_distance 값을 SUM으로 더해 user 정보와 함께 출력한다. 단, 순서는 user의 이용 거리가 큰 순서대로 정렬한다.

19. 관리자 기능 - 대시 보드 (시각화)



DESCRIPTION	
<p>이 기능은 관리자가 따릉이 월별 이용자 거리 및 횟수와 주간 이용자 거리 및 횟수를 그래프로 시각화하여 조회할 수 있는 기능이다.</p>	
FRONT END	BACK END
<pre> <Box sx={{ display: "flex", justifyContent: "start", alignItems: "center" }}> <ButtonGroup> <Button onClick={() => { if (year > 1) { setYear(year - 1); } }} > 이전 년도 </Button> <Button> onClick={() => { setYear(year + 1); }} > 다음 년도 </Button> </ButtonGroup> <ButtonGroup sx={{ pl: 5 }}> <Button onClick={() => { if (month > 1) setMonth(month - 1); }} > 이전 달 </Button> <Button> {month}월 </Button> <Button onClick={() => { if (month < 12) setMonth(month + 1); }} > 다음 달 </Button> </ButtonGroup> </Box> <Divider sx={{ py: 3 }} /> <Typography variant="h5"> 월별 이용자 거리 및 횟수 </Typography> <UserMonthChart data={rawData} /> <Typography variant="h5" sx={{ pt: 10 }}> 주간 이용자 거리 및 횟수 </Typography> <UserWeekChart data={rawData} /> </pre> <p>echarts-for-react 라이브러리를 이용하여 사용자 데이터를 시각화하였다. 원하는 기간을 볼 수 있는 버튼 그룹과 해당 연도, 월을 표시하는 버튼이 있고 UserMonthChart와 UserWeekchart component를 사용하여 월별 이용자 거리 및 횟수와 요일별 이동거리 및 횟수를 차트로 보여준다. 백엔드 API로 부터 받아온 사용자 데이터를 data={rawData}로 차트에 전달한다.</p>	<p>특정 기간의 userlog를 반환하게 되면 그 정보를 이용하여 대시보드의 사용자 이용 시간과 거리 등을 시각화할 수 있기 때문에, 특정 기간을 입력받으면 해당하는 userlog를 반환하는 코드를 구현하였다.</p> <pre> @GetMapping("/getBetweenUserLog") public ResponseEntity<UserLog> getBetweenUserLog(@RequestParam String start, @RequestParam String end) { DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"); LocalDateTime startTime = LocalDate.parse(start, formatter); LocalDateTime endTime = LocalDate.parse(end, formatter); List<VisualizationUserLog> userLogs = userService.getBetweenUserLog(startTime, endTime); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, userLogs)); } </pre> <p>controller 코드는 위와 같이 구현되어 있는데, front end에서 start 날짜, 시간정보와 end 날짜, 시간정보를 String 형태로 전송하면 controller 부분에서 이를 format하여 LocalDateTime 타입으로 변경한다. 이후 이들을 전달하며 서비스 코드를 호출한다.</p> <pre> public List<VisualizationUserLog> getBetweenUserLog(LocalDateTime startTime, LocalDateTime endTime) { try { return userRepository.getBetweenUserLog(startTime, endTime); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } } </pre> <p>이 서비스 코드를 보면 알 수 있듯이, repository의 쿼리 요청만 호출하고, 그 결과를 받아서 다시 사용자에게 전달하는 간단한 로직이다.</p>
SQL	

```

    public List<VisualizationUserlogDto.userLogDto> getBetweenUserlog(LocalDateTime startDateTime, LocalDateTime endDateTime) {
        var rowMapper = BeanPropertyRowMapper.newInstance(VisualizationUserlogDto.userLogDto.class);
        return jdbcTemplate.query(
            "SELECT * FROM userlog WHERE departure_time >= ? AND arrival_time <= ?",
            rowMapper,
            Timestamp.valueOf(startDateTime),
            Timestamp.valueOf(endDateTime)
        );
    }
}

```

getBetweenUserlog는 userlog 테이블에서 특정 시간 범위 내에 있는 log들을 조회하게 된다. 쿼리를 자세히 확인하면 departure_time이 주어진 시작 시간 이후이고, arrival_time이 주어진 종료 시간 이전인 모든 record를 선택하게 된다. 이는 특정 기간 사이에 존재하는 모든 로그 정보를 반환하는 것이며, 이때 LocalDateTime 형식의 데이터를 Timestamp로 변경하여 바인딩하게 된다.

20. 관리자 기능 - 전체 대여 이력 조회

SCREEN																																																																																																																																			
관리자 기능 대시보드 대여 이력 이용권 관리 회원 관리 따릉이 보관소 관리 자전거 관리 쿠폰 관리																																																																																																																																			
<table border="1"> <thead> <tr> <th>번호</th><th>유저명</th><th>자전거명</th><th>출발지</th><th>도착지</th><th>출발시간</th><th>도착시간</th><th>이용 시간</th><th>이용 거리</th><th>이용 결과</th><th></th></tr> </thead> <tbody> <tr><td>57</td><td>36</td><td>7</td><td>ST-1280</td><td>ST-2714</td><td>2023-12-14 20:39:10</td><td>2023-12-14 20:42:02</td><td>2</td><td>356</td><td>대여완료</td><td></td></tr> <tr><td>61</td><td>37</td><td>56</td><td>ST-2074</td><td>ST-10</td><td>2023-12-14 19:29:06</td><td>2023-12-14 20:02:13</td><td>33</td><td>70</td><td>대여완료</td><td></td></tr> <tr><td>60</td><td>37</td><td>33</td><td>ST-2714</td><td>ST-2714</td><td>2023-12-14 19:26:14</td><td>2023-12-14 19:26:25</td><td>0</td><td>0</td><td>대여완료</td><td></td></tr> <tr><td>56</td><td>36</td><td>10</td><td>ST-2074</td><td>ST-1280</td><td>2023-12-14 08:41:51</td><td>2023-12-14 08:42:34</td><td>0</td><td>94</td><td>대여완료</td><td></td></tr> <tr><td>51</td><td>26</td><td>6</td><td>ST-2074</td><td>ST-600</td><td>2023-12-03 20:46:58</td><td>2023-12-03 20:48:22</td><td>1</td><td>776</td><td>대여완료</td><td></td></tr> <tr><td>49</td><td>6</td><td>35</td><td>ST-3122</td><td>ST-2237</td><td>2023-12-03 17:47:17</td><td>2023-12-03 17:51:11</td><td>3</td><td>1173</td><td>대여완료</td><td></td></tr> <tr><td>48</td><td>6</td><td>16</td><td>ST-576</td><td>ST-1280</td><td>2023-12-03 17:37:07</td><td>2023-12-03 17:37:15</td><td>0</td><td>248</td><td>대여완료</td><td></td></tr> <tr><td></td><td></td><td></td><td>ST-2045</td><td>ST-1402</td><td>2023-11-27 22:45:00</td><td>2023-11-27 22:54:00</td><td>9</td><td>1140</td><td>대여완료</td><td></td></tr> <tr><td></td><td></td><td></td><td>ST-2466</td><td>ST-2466</td><td>2023-11-27 14:05:00</td><td>2023-11-27 14:52:00</td><td>47</td><td>3215</td><td>대여완료</td><td></td></tr> <tr><td></td><td></td><td></td><td>ST-150</td><td>ST-170</td><td>2023-11-27 17:15:00</td><td>2023-11-27 17:22:00</td><td>7</td><td>1040</td><td>대여완료</td><td></td></tr> </tbody> </table>											번호	유저명	자전거명	출발지	도착지	출발시간	도착시간	이용 시간	이용 거리	이용 결과		57	36	7	ST-1280	ST-2714	2023-12-14 20:39:10	2023-12-14 20:42:02	2	356	대여완료		61	37	56	ST-2074	ST-10	2023-12-14 19:29:06	2023-12-14 20:02:13	33	70	대여완료		60	37	33	ST-2714	ST-2714	2023-12-14 19:26:14	2023-12-14 19:26:25	0	0	대여완료		56	36	10	ST-2074	ST-1280	2023-12-14 08:41:51	2023-12-14 08:42:34	0	94	대여완료		51	26	6	ST-2074	ST-600	2023-12-03 20:46:58	2023-12-03 20:48:22	1	776	대여완료		49	6	35	ST-3122	ST-2237	2023-12-03 17:47:17	2023-12-03 17:51:11	3	1173	대여완료		48	6	16	ST-576	ST-1280	2023-12-03 17:37:07	2023-12-03 17:37:15	0	248	대여완료					ST-2045	ST-1402	2023-11-27 22:45:00	2023-11-27 22:54:00	9	1140	대여완료					ST-2466	ST-2466	2023-11-27 14:05:00	2023-11-27 14:52:00	47	3215	대여완료					ST-150	ST-170	2023-11-27 17:15:00	2023-11-27 17:22:00	7	1040	대여완료	
번호	유저명	자전거명	출발지	도착지	출발시간	도착시간	이용 시간	이용 거리	이용 결과																																																																																																																										
57	36	7	ST-1280	ST-2714	2023-12-14 20:39:10	2023-12-14 20:42:02	2	356	대여완료																																																																																																																										
61	37	56	ST-2074	ST-10	2023-12-14 19:29:06	2023-12-14 20:02:13	33	70	대여완료																																																																																																																										
60	37	33	ST-2714	ST-2714	2023-12-14 19:26:14	2023-12-14 19:26:25	0	0	대여완료																																																																																																																										
56	36	10	ST-2074	ST-1280	2023-12-14 08:41:51	2023-12-14 08:42:34	0	94	대여완료																																																																																																																										
51	26	6	ST-2074	ST-600	2023-12-03 20:46:58	2023-12-03 20:48:22	1	776	대여완료																																																																																																																										
49	6	35	ST-3122	ST-2237	2023-12-03 17:47:17	2023-12-03 17:51:11	3	1173	대여완료																																																																																																																										
48	6	16	ST-576	ST-1280	2023-12-03 17:37:07	2023-12-03 17:37:15	0	248	대여완료																																																																																																																										
			ST-2045	ST-1402	2023-11-27 22:45:00	2023-11-27 22:54:00	9	1140	대여완료																																																																																																																										
			ST-2466	ST-2466	2023-11-27 14:05:00	2023-11-27 14:52:00	47	3215	대여완료																																																																																																																										
			ST-150	ST-170	2023-11-27 17:15:00	2023-11-27 17:22:00	7	1040	대여완료																																																																																																																										
DESCRIPTION																																																																																																																																			
<p>이 기능은 관리자가 따릉이 자전거의 대여이력을 조회할 수 있는 기능이다. 사용자가 따릉이를 대여하게 되면 어떤 user가 어떤 자전거를 대여했고 출발지에서 출발한 시간과 도착대여소와 도착시간, 이용시간, 총 이용한 거리와 해당 자전거가 반납이 되었는지의 여부를 조회할 수 있다.</p>																																																																																																																																			
FRONT END						BACK END																																																																																																																													

```

useEffect(() => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + `/get-all-userlog`, {
        method: "GET",
        headers: { "Content-Type": "application/json" },
      });
      if (response.status !== 200) {
        throw new Error(`데이터를 가져오는데 실패하였습니다.`);
      }
      const jsonData = await response.json();
      // Reverse the array to show the latest data first
      const reversedData = jsonData.result.reverse();

      setHistoryData(reversedData);
      setTotalPage(Math.ceil(reversedData.length / 10));
      setShowData(reversedData.slice(0, 10));
      setIsLoad(true);
    } catch (error) {
      alert(`데이터를 가져오는데 실패하였습니다.`);
    }
  })();
  useEffect(() => {
    setShowData(historyData.slice((page - 1) * 10, page * 10));
  }, [page, historyData]);
}, []);

```

백엔드 API를 통해 자전거 대여 이력 정보를 table로 렌더링하고 이전/다음 버튼을 이용한 페이지네이션으로 더 많은 대여 이력 정보를 확인할 수 있도록 구현하였다. 대여 이력에는 사용자 ID, 자전거 ID, 출발지, 도착지, 출발시간, 도착시간, 이용거리, 이용 결과등의 정보가 표시되며 데이터 로딩 중에는 로딩 인디케이터가 표시된다.

```

// 모든 유저의 모든 대여 이력을 조회
@GetMapping("/get-all-userlog")
public ResponseEntity<List<UserlogDto>> getAllUserLog() {
  List<UserlogDto> allUserLog = userlogService.getAllUserlog();

  if((allUserLog==null) || (allUserLog.isEmpty()))
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS_GET_ALL_USERLOG, null));
  else
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS_GET_ALL_USERLOG, allUserLog));
}

```

```

public List<UserlogDto> getAllUserlog() {
  return userlogRepository.getAllUserlog();
}

```

관리자의 기능인 전체 대여 이력 조회 시에 모든 유저에 대한 모든 대여 이력을 반환하기 위한 /get-all/userlog API이다.

userlogService의 getAllUserlog() 메서드를 호출하여 모든 유저의 모든 대여 이력에 대한 정보를 리스트로 반환 받는다. 이때 인자를 넘겨주지 않는 이유는 모든 유저에 대한 모든 대여 이력이므로 특별한 조건 사항이 존재하지 않기 때문이다. 모든 유저의 모든 대여 이력 리스트가 null이거나 isEmpty()라면 아무 기록도 존재하지 않는다는 예외처리를 수행하게 된다. 1개 이상이 존재한다면 모든 유저의 모든 대여 기록(userlog) 조회에 성공했다는 response와 함께 대여 이력들을 확인할 수 있다.

SQL

```

public List<UserlogDto> getAllUserlog() {
  String sql = "SELECT * FROM userlog ORDER BY departure_time ASC";
  return jdbcTemplate.query(sql, BeanPropertyRowMapper.newInstance(UserlogDto.class));
}

```

USERLOG table에서 모든 로그 record를 조회해서 출발 시간인 departure_time을 기준으로 그들을 오름차순으로 정렬한 결과를 얻는다.

21, 22, 23. 관리자 기능 - 회원 관리 (회원정보 조회, 수정, 삭제)

SCREEN

관리자 기능		대시보드	대여 이력	이용권 관리	회원 관리	마름이 보금스 관리	자전거 관리	카페 관리	
번호	이메일	이름	전화번호	무게	나이	마지막 접속시간	보유금액	수정	삭제
1	admin	admin	01000000000	100	100	2023-12-14 08:47:48	993000	<button>수정</button>	<button>삭제</button>
2	hong@gmail.com	홍길동	01010001000	67	24	2023-12-03 03:17:05	30000	<button>수정</button>	<button>삭제</button>
3	kim@naver.com	김길동	01011010010	54	40	2023-12-03 03:19:39	50000	<button>수정</button>	<button>삭제</button>
4	jihyeon@naver.com	하지연	01011110001	50	23	2023-12-03 04:20:11	50000	<button>수정</button>	<button>삭제</button>
5	sung@gmail.com	강성진	0102220321	70	24	2023-12-03 03:24:38	50000	<button>수정</button>	<button>삭제</button>
6	jimin@naver.com	방지민	01001339602	50	22	2023-12-04 00:17:09	1000	<button>수정</button>	<button>삭제</button>
9	apple@gmail.com	김자연	01022292284	13	28	2023-12-03 00:00:17	44000	<button>수정</button>	<button>삭제</button>
12	fubao@naver.com	푸바오	01012121212	140	3	2023-12-03 17:24:56	44000	<button>수정</button>	<button>삭제</button>
13	apple@naver.com	강예준	01012934215	34	30	2023-12-03 04:05:28	50000	<button>수정</button>	<button>삭제</button>
14	kang@naver.com	병중강	01049391823	0	0	2023-12-03 04:12:44	40000	<button>수정</button>	<button>삭제</button>

[이전 | 1 | 다음]

회원 정보 수정

유저ID
3

이메일
kim@naver.com

이름
김길동

변경 비밀번호

전화번호
01011010010

나이
40

몸무게
54

보유 금액
50000

수정하기 뒤로가기

DESCRIPTION

위 기능은 관리자가 따릉이 홈페이지에 가입되어 있는 모든 사용자에 대한 정보를 조회할 수 있는 기능이다. 정보에는 사용자가 회원가입할 때 입력한 이메일, 이름, 전화번호, 몸무게, 나이와 가장 최근 따릉이 홈페이지에 접속한 시간 및 티켓을 구매하기 위한 보유 금액이 있다. 관리자는 이 페이지에서 사용자에 대한 정보를 열람하고 수정하거나 사용자의 계정을 삭제 할 수 있다.

FRONT END

```
useEffect(() => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/admin/get-all-users", {
        method: "GET",
        headers: { "Content-Type": "application/json" },
      });
      if (response.status != 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      setUserData(jsonData.result);
      setTotalPage(Math.floor(jsonData.result.length / 10) + 1);
      setShowData(jsonData.result.slice(0, 10));
    } catch (error) {
      alert("데이터를 가져오는데 실패하였습니다.");
    }
  })();
}, [ ]);
```

BACK END

```
@GetMapping("/get-all-users")
public ResponseEntity<User> getAllUsers(){
  List<User> users = adminService.getAllUsers();

  if (users.isEmpty())
    return ResponseEntity.ok(new JsonResponse<>(
      ResponseStatus.SUCCESS_GET_ALL_USERS_INFO_ISEMPTY, result: null));

  return ResponseEntity.ok(new JsonResponse<>(
    ResponseStatus.SUCCESS_GET_ALL_USERS_INFO, users));
}

public List<User> getAllUsers() {
  return userRepository.returnAllUsers();
}
```

해당 페이지에 접근한 관리자는, 백엔드 API를 통하여 사용자 정보를 가져온다. 해당 정보를 테이블로 렌더링하고 이전/ 다음 버튼을 통한 페이지네이션으로 여러 사용자 정보를 볼 수 있도록 구현되었다. 각 사용자의 정보 옆에는 수정과 삭제 버튼이 있다.

관리자가 회원 가입이 되어 있는 모든 유저에 대한 정보를 조회하기 위해 호출하는 /get-all-users API이다. adminService의 getAllUsers() 메서드를 호출하여 repository에서 모든 유저의 정보를 List로 받아와 저장한다. 하지만 List인 users가 empty라면 저장된 유저의 정보가 하나도 없다는 예외처리를 발생시키며, 1명 이상 존재할 경우 해당 List를 반환해줌으로써 모든 유저에 대한 정보를 반환할 수 있다.

```

const submitHandler = () => {
  (async() => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/admin/modify-user", {
        method: "PATCH",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          id: userId,
          username,
          password: userPassword,
          email: userEmail,
          phone_number: userPhone,
          weight: userWeight,
          age: userAge,
          total_money: userMoney
        })
      );
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("수정에 성공하였습니다.");
        navigate("/admin?type=user");
      } else {
        throw new Error("수정에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("수정에 실패했습니다.");
    }
  })();
  return;
}

```

수정버튼을 클릭하면 해당 사용자의 정보를 수정하는 페이지로 이동하여 수정을 진행할 수 있다. 수정 후 수정하기 버튼을 클릭하면 관련 정보를 백엔드 API로 전달하여 수정을 시도하고, 정상적으로 수정이 완료되었다면 다시 유저 관리 페이지로 이동한다.

```

@PatchMapping("/modify-user")
public ResponseEntity<?> modifyUser(@RequestBody UserDto userDto){
  adminService.modifyUser(userDto);
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS MODIFY_USER_INFO, result: null));
}

```

```

public void modifyUser(UserDto userDto) {
  userRepository.modifyUser(userDto);
}

```

관리자가 회원 가입이 되어 있는 유저에 대해 특정 유저에 대한 정보를 수정하는 /admin/modify-user API이다. adminService의 modifyUser() 메서드에 대한 인자로 수정할 정보가 담긴 userDto를 넘겨서 id에 해당하는 user의 나머지 정보들에 대한 수정을 진행한다.

```

@DeleteMapping("/delete-user")
public ResponseEntity<?> deleteUser(@RequestBody UserDto userDto){
  adminService.deleteUser(userDto.getId());
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_DELETE_USER, result: null));
}

```

```

public void deleteUser(int id) {
  userRepository.deleteUserById(id);
}

```

관리자가 회원 가입이 되어 있는 유저에 대해 특정 유저에 대한 삭제를 위한 /admin/delete-user API이다. adminService의 deleteUserById() 메서드에 대한 인자로 해당 유저의 PK인 id를 넘겨주어 삭제를 진행하도록 한다.

SQL

```

public List<UserDto> returnAllUsers() {
  // SOME
  String sql = "SELECT * FROM user WHERE id = SOME (SELECT id FROM user WHERE id > 0)";
  return jdbcTemplate.query(
    sql,
    BeanPropertyRowMapper.newInstance(UserDto.class)
  );
}

```

```
public void modifyUser(UserDto userDto) {  
    String sql = "UPDATE user SET email = ?, username = ?, password = ?, " +  
        "phone_number = ?, weight = ?, age = ?, total_money = ? WHERE id = ?";  
    jdbcTemplate.update(sql,  
        userDto.getEmail(),  
        userDto.getUsername(),  
        userDto.getPassword(),  
        userDto.getPhone_number(),  
        userDto.getWeight(),  
        userDto.getAge(),  
        userDto.getTotal_money(),  
        userDto.getId());  
}
```

```
public void deleteUserById(int id) {  
    // DELETE  
    String sql = "DELETE FROM user WHERE id = ?";  
    jdbcTemplate.update(sql, id);  
}
```

첫번째로 모든 유저에 대한 정보를 반환하는 SQL 쿼리이다. USER table에 대해 id가 0보다 큰 모든 사용자를 선택하여 반환한다.

두번째로 특정 유저에 대한 정보를 수정하는 SQL 쿼리이다. USER table에서 해당 id를 가진 사용자에 대한 정보를 userDto 객체에 저장되어 있는 정보들로 새롭게 UPDATE를 수행한다.

세번째로 특정 유저에 대한 id를 기반으로 해당 유저를 삭제하는 SQL 쿼리이다. USER table에서 해당 id를 가진 사용자의 정보를 DELETE함으로써 해당 유저에 대한 정보를 삭제한다.

24, 25, 26, 27. 관리자 기능 - 대여소 조회, 대여소 개설 및 폐쇄, 대여소 정보 수정

SCREEN

관리자 기능		대시보드	내역 이력	이용권 관리	회원 관리	마름이 보관소 관리	자전거 관리	구분 관리	
ID	전체 주소	이름	위도	경도	최대 차 연기 개 수	현재 차 연기 개 수	활성화 여부	수정	삭제
SF-10	서울특별시 양천구 영등포 로 93	427	37.552746	126.018617	20	2	1	<button>수정</button>	<button>삭제</button>
SF-100	서울특별시 양천구 아동산 로 262	더샵스카이타워 C6 층	37.56667	127.073933	20	0	1	<button>수정</button>	<button>삭제</button>
SF-1000	서울특별시 양천구 신정동 236	서아시아재마 트 건물	37.51038	126.866798	20	1	1	<button>수정</button>	<button>삭제</button>
SF-1001	서울특별시 양천구 남부순 환로길10	서아울호수공 원	0	0	20	0	1	<button>수정</button>	<button>삭제</button>
SF-1002	서울특별시 양천구 목동동 로 316-6	서울시 도로명 주소변경내역	37.5299	126.876541	20	0	1	<button>수정</button>	<button>삭제</button>
SF-1003	서울특별시 양천구 화곡로 59	신월동 이마트	37.53951	126.8283	20	0	1	<button>수정</button>	<button>삭제</button>
SF-1004	서울특별시 양천구 신정동 1004 번1550	신이마트하우 스214동	37.514099	126.831001	20	0	1	<button>수정</button>	<button>삭제</button>
SF-1005	서울특별시 양천구 신정동 310-8	신도리파워 입 구	37.51199	126.856056	20	0	1	<button>수정</button>	<button>삭제</button>
SF-1006	서울특별시 양천구 목동동 로 70 서울영토초등학교	37.536377	126.871513	20	0	1	<button>수정</button>	<button>삭제</button>	
SF-1007	서울특별시 양천구 남부순 환로70길11-9	오승길관현	37.52219	126.8367	20	0	1	<button>수정</button>	<button>삭제</button>

[이전](#) [1](#) [다음](#)

대여소 추가

대여소 ID
SF-10

대여소 주소
서울특별시 마포구 양화로 93

대여소 이름
427

위도
37.552746

경도
126.918617

최대 차
연기 수
20

전체 차연기 수
2

남은 차연기 수
18

대여소 활성화 여부

생성하기
뒤로가기

DESCRIPTION

관리자로써 대여소를 직접 개설하거나 폐쇄하고, 정보를 수정할 수 있다. 관리자 기능 내 끌어온 보관소 관리 페이지로 가면 모든 끌어온이들의 정보를 확인할 수 있고, 추가 버튼을 눌러 직접 대여소를 추가하거나, 수정 버튼을 눌러 대여소의 정보를 변경할 수 있으며, 삭제 버튼을 통해 대여소 정보를 삭제할 수 있다.

FRONT END
BACK END

```
useEffect(() => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/station/get-all-lendplace?user_id=1", {
        method: "GET",
        headers: { "Content-Type": "application/json" },
      });
      if (response.status != 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      setBikeStationData(jsonData.result);
      setTotalPage(parseInt(jsonData.result.length / 10) + 1);
      setShowData(jsonData.result.slice(0, 10));
    } catch (error) {
      alert("데이터를 가져오는데 실패하였습니다.");
    }
  })();
}, []);

```

```
@PostMapping("/create-lendplace")
public ResponseEntity<JsonResponse> createBikeStation(@RequestBody BikeStationDto.BikeStationDetailDto bikeStationCreateDto) {
    bikeStationService.createBikeStation(bikeStationCreateDto);
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}

@GetMapping("/get-all-lendplace")
public ResponseEntity<JsonResponse> getAllStation(@RequestParam int user_id) {
    List<BikeStationDto.BikeStationWithCurrentBike> bikeStationList = bikeStationService.getAllStation(user_id);
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationList));
}

@PatchMapping("/modify-lendplace")
public ResponseEntity<JsonResponse> modifyBikeStation(@RequestBody BikeStationDto.BikeStationDetailDto bikeStationModifyDto) {
    bikeStationService.modifyBikeStation(bikeStationModifyDto);
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}

@DeleteMapping("/delete-lendplace")
public ResponseEntity<JsonResponse> deleteBikeStation(@RequestBody BikeStationDto.BikeStationDeleteDto bikeStationDeleteDto) {
    bikeStationService.deleteBikeStation(bikeStationDeleteDto);
    return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationDeleteDto.getLendplace_id()));
}
```

끌어온이 보관소 관리 페이지로 들어가면, 백엔드의 API를 이용해 대여소의 모든 정보를 가져와 테이블 및 페이지네이션을 이용해 보여준다.

```
const submitHandler = () => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/station/modify-lendplace", {
        method: "PATCH",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          lendplace_id: lendplace_id,
          statn_addr1: statn_addr1,
          statn_addr2: statn_addr2,
          startn_lat: startn_lat,
          startn_lng: startn_lng,
          max_stands: max_stands,
          station_status: station_status ? 1 : 0,
        }),
      });
      if (response.status != 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("수정이 성공하였습니다.");
        navigate("/admin/type:bikestation");
      } else {
        alert("수정에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("수정에 실패했습니다.");
    }
  })();
  return;
};
```

수정 버튼을 누르면 별도의 수정 페이지로 이동해 대여소의 정보를 가져와 기본 입력

이 기능을 확인하면 대여소의 생성, 조회, 수정, 삭제이다.

각 요청을 받았을 경우 해당 컨트롤러 코드가 실행되며, 컨트롤러 코드는 서비스 코드를 호출하여 데이터를 얻어오거나 처리하는 과정을 수행한다.

```
usage = 컨트롤러
public void createBikeStation(BikeStationDto.BikeStationDetailDto bikeStation) {
    try {
        bikeStationRepository.insert(bikeStation);
    }
}

usage = 서비스
public List<BikeStationDto.BikeStationWithCurrentBike> getAllStation(int user_id) {
    try {
        return bikeStationRepository.findAll(user_id);
    }
}

usage = 서비스
public void modifyBikeStation(BikeStationDto.BikeStationDetailDto bikeStation) {
    try {
        bikeStationRepository.update(bikeStation);
    }
}
```

폼에 넣고, 사용자가 수정한 뒤 수정하기
버튼을 누르면 입력 폼의 정보를 백엔드에 전달해 수정을 진행한다.

```
const submitHandler = () => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/station/create-lendplace", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          lendplace_id: lendplace_id,
          statn_addr1: statn_addr1,
          statn_addr2: statn_addr2,
          startn_lat: startn_lat,
          startn_lnt: startn_lnt,
          max_stands: max_stands,
        }),
      });
      if (response.status !== 200) {
        throw new Error(`데이터를 가져오는데 실패하였습니다.`);
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("성공에 성공하였습니다.");
        navigate("/admin?type=BikeStation");
      } else {
        throw new Error("수정에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("수정에 실패했습니다.");
    }
  })();
  return;
};
```

대여소 정보 추가 시 입력 폼으로 이동하며 관련 정보를 입력 후 생성하기를 누르면 정보를 백엔드 API를 통해 전달해 생성을 진행한다.

```
public void deleteBikeStation(BikeStationDto.BikeStationDeleteDto bikeStationDeleteDto) {
  try {
    bikeStationRepository.delete(bikeStationDeleteDto.getLendplace_id());
  } catch (Exception e) {
    throw new GlobalException(ResponseStatus.DATABASE_ERROR);
  }
}
```

하지만 이때 서비스 코드는 별다른 데이터 처리 과정이 필요하지 않기 때문에 repository에서 Database의 조건에 맞는 데이터를 가져온 뒤, 이 서비스 코드를 호출한 컨트롤러에 정보들을 다시 반환하도록 구현하였다.

SQL

```
jdbcTemplate.update("INSERT INTO `bikestationinformation` (`lendplace_id`, `statn_addr1`, `statn_addr2`, `startn_lat`, " +
  "`startn_lnt`, `max_stands`, `station_status`) VALUES (?, ?, ?, ?, ?, ?, ?)"
, bikeStation.getLendplace_id(), bikeStation.getStatn_addr1(), bikeStation.getStatn_addr2(),
bikeStation.getStartn_lat(), bikeStation.getStartn_lnt(), bikeStation.getMax_stands(),
bikeStation.getStation_status());
```

```
I Usage 2. 망사면
public List<BikeStationDto.BikeStationWithCurrentBike> findAll(int userId) {
  var bikeMapper = BeanPropertyRowMapper.newInstance(BikeStationDto.BikeStationWithCurrentBike.class);
  return jdbcTemplate.query(
    "SELECT BS.*," +
      " COALESCE(BC.total_bikes, 0) AS total_bikes, " +
      " COALESCE(BC.usable_bikes, 0) AS usable_bikes, " +
      " COALESCE(AVG(BR.rating), 0) AS average_rating, " +
      " CASE WHEN FAV.lendplace_id IS NOT NULL THEN TRUE ELSE FALSE END AS favorite " +
    "FROM bikestationinformation BS " +
    "LEFT JOIN BikeCounts BC ON BS.lendplace_id = BC.lendplace_id " +
    "LEFT JOIN bikestationrating BR ON BS.lendplace_id = BR.lendplace_id " +
    "LEFT JOIN (SELECT lendplace_id FROM favorite WHERE user_id = ?) FAV ON BS.lendplace_id = " +
    "FAV.lendplace_id " +
    "GROUP BY BS.lendplace_id",
    bikeMapper,
    userId);
}
```

```

public void update(BikeStationDto.BikeStationDetailDto bikeStation) {
    jdbcTemplate.update("UPDATE `bikestationinformation` SET " +
        "statn_addr1= ?, " +
        "statn_addr2= ?, " +
        "startn_lat=?, " +
        "startn_lnt=?, " +
        "max_stands=?," +
        "station_status=? " +
        "WHERE lendplace_id=?"
        , bikeStation.getStatn_addr1(), bikeStation.getStatn_addr2(), bikeStation.getStartn_lat(),
        bikeStation.getStartn_lnt(), bikeStation.getMax_stands(), bikeStation.getStation_status(),
        bikeStation.getLendplace_id());
}

public void delete(String lendplaceId) {
    jdbcTemplate.update("DELETE FROM `bikestationinformation` WHERE lendplace_id=?", lendplaceId);
}

```

관리자가 대여소를 추가하고 싶다면 입력한 대여소 정보를 바탕으로 **INSERT**를 이용하여 **bikestationtable**에 데이터를 추가할 수 있다. 이때, 존재하지 않는 정보의 경우 미리 처리를 해 주고 있지만 주요 코드만 캡쳐하여 첨부하였다.

모든 대여소 세부정보를 한 번에 띄울 수 있도록 모든 대여소 정보 조회, **findAll**의 경우 모든 대여소의 정보 뿐만 아니라 각 대여소의 이용 가능한 자전거 수, 총 자전거 수, 평균 평점 또한 반환해야 하 때문에 **bikestationinformation table** 기반으로 자전거 대여소의 상세 정보와 관련된 여러 데이터를 결합하여 조회하게 된다.

이때, **BikeCounts**는 자전거 수를 가져올 수 있는 VIEW이며, **bikestationinformation**에 자전거 정보를 함께 가져오기 위해 **BikeCount**를, 각 평점을 가져오기 위해 **bikestationrating**을, 즐겨찾기 여부를 출력하기 위해 **favorite table**을 JOIN한다.

대여소 정보를 수정하기 위해 **UPDATE**를 사용해 **bikestationinformation**의 **lendplace_id**가 일치하는 정보를 수정하고 있으며, 대여소 정보를 삭제하기 위해 **DELETE**를 사용해 **bikestationinformation**의 **lendplace_id**가 일치하는 데이터 **row**를 삭제한다.

3. 특별 기능 설명

1. 이메일 인증

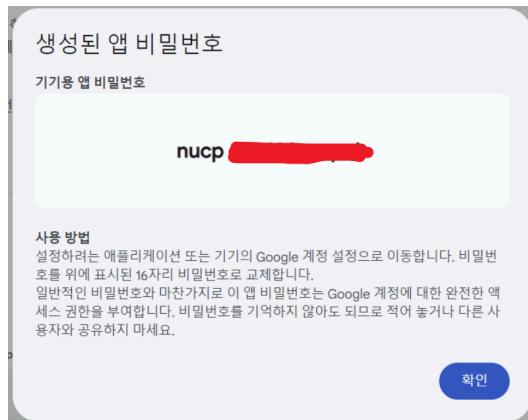
이메일 인증은 클라이언트 측에서 비밀번호 변경 과정 진행 중, 이메일로 전송된 인증번호를 입력하여 인증을 완료하면 비밀번호 변경을 할 수 있도록 할 수 있는 본인 확인 과정에 대한 특수 기능이다.

1000부터 9999까지 랜덤한 난수로 인증번호를 생성하여 해당 이메일로 발급한 인증번호를 DB에 저장하는 기능, 생성한 인증번호를 이메일 인증 html 품에 매핑하여 구글 SMTP를 활용하여 실제 메일로 전송하는 기능, 인증번호 발급 시 5분 뒤에 자동으로 해당 이메일로 발급된 인증번호를 삭제하는 스케줄러, 아직 DB 상에 인증번호가 유지되어 있을 때 다시 인증번호 요청을 보내는 경우에 이를 대체하는 로직으로 구성하였다.

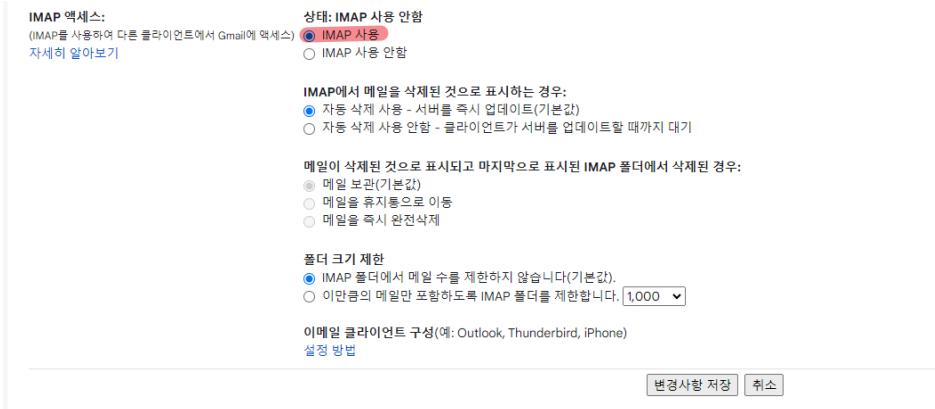
즉, 발급된 인증 번호를 5분 이내에 입력해야 비밀번호 변경이 가능한 인증이 완료된다.

가장 먼저 구글 SMTP를 사용하기 위해 우리 팀의 구글 계정을 생성하여 앱 비밀번호를 아래와 같이 생성해주었다.

생성된 앱 비밀번호는 spring server 측의 application.yml에 명시해주어야 하므로 잘 저장한다.



gmail -> 오른쪽 위 톱니바퀴 모양의 설정 버튼 -> 모든 설정 보기 -> 전달 및 POP/IMAP에서 IMAP 사용을 클릭하여 활성화한다.



SMTP 설정 및 앱 비밀번호 발급을 완료하였으므로, 본격적으로 구현을 진행한다.

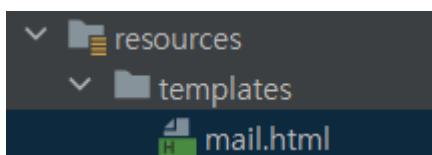
스프링 프로젝트에서 build.gradle에 아래의 종속성을 추가 후 sync를 진행한다.

```
implementation group: 'org.springframework.boot', name: 'spring-boot-starter-mail', version: '2.6.3' // 이메일 인증 코드
implementation 'org.springframework.boot:spring-boot-starter-thymeleaf' // 이메일 품을 위한 태일리프
```

아래와 같이 application.yml 파일을 구성한다. 앱 비밀번호를 생성한 구글 계정의 이메일이 username이며, 앞서 생성한 앱 비밀번호가 password이다.

```
spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: kw2023db@gmail.com
    password: [REDACTED]
    properties:
      mail:
        smtp:
          starttls:
            enable: true
            auth: true
```

resource/templates 폴더 내부에 mail.html 파일을 생성한다.



아래는 mail.html의 내용이다. 이메일 인증 번호 전송 시 사용자에게 보여줄 인증번호 메일 품의 역할을 수행한다.

line 22에서 타임리프 문법으로 ChangePasswordService.java에 구현되어 있는 이메일 인증 번호 생성 이후 메일 전송 로직 수행 시 이메일 인증번호가 매핑이 될 것이다.

```
1  <!DOCTYPE html>
2  <html xmlns="http://www.thymeleaf.org">
3
4  <body>
5  <div style="margin: 95px;">
6
7      <div style="margin: 95px;">
8          
9          <h1>데일리데일리뉴스 이메일 인증 안내 </h1>
10         <br>
11         <span style="color:#000000">인증하세요, </span>
12         <b style="color:#696969">비밀번호 찾기</b>
13         <span style="color:#000000">를 위해 이메일 인증을 진행합니다. </span>
14         <br>
15         <span style="color:#000000">아래 링크를 클릭하여 인증하세요</span>
16         <b style="color:#696969">이메일 인증번호를 복사하거나 직접 입력하여 인증을 완료</b><span style="color:#000000">해주세요. </span>
17         <br><br><br>
18
19     <div align="center" style="border-top: 1px solid black; border-bottom: 1px solid black; font-family:verdana;">
20         <br><br>
21         <b style="color:black; font-size: 24px;">인증번호 </b>
22         <b style="color:red; font-size: 24px;">${text}</b>
23         <br><br><br>
24     </div>
25
26 </div>
27
28 </body>
29 </html>
```

비밀번호 찾기 중 인증 번호 전송에 대한 request를 수행하는 UserController의 코드이다.

비밀번호 변경 관련 로직이 구현되어 있는 changePasswordService에서 메일을 보내는 메서드를 호출하고 있으며, 어떤 이메일로 인증 번호 메일을 전송할지 인자로 담아주었다.

```
// 비밀번호 찾기 중 인증 번호 전송
@PostMapping("/send-authcode")
public ResponseEntity<?> sendAuthCode(@RequestBody UserDto userDto) throws MessagingException, UnsupportedEncodingException {
    EmailAuthDto emailAuthCodeDto = new EmailAuthDto();

    // 5분 이내에 다시 인증 번호 전송을 했다면 앞서 요청한 인증 번호 삭제
    changePasswordService.deleteExistCode(userDto.getEmail());

    emailAuthCodeDto.setAuth_num(Integer.parseInt(changePasswordService.sendEmail(userDto.getEmail())));

    return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_SEND_AUTHCODE, emailAuthCodeDto.getAuth_num()));
}
```

ChangePassWordService에 작성되어 있는 sendEmail() 메서드이다. 마찬가지로 내부에 정의되어 있는 createEmailForm() 메서드를 통해 메일 전송에 필요한 정보를 설정하고 작성된 이메일 품을 전송하게 된다.

```
// 실제 메일 전송 - controller에서 호출
public String sendEmail(String toEmail) throws MessagingException, UnsupportedEncodingException {

    // 메일 전송에 필요한 정보 설정
    MimeMessage emailForm = createEmailForm(toEmail);
    // 실제 메일 전송
    emailSender.send(emailForm);

    return authNum; //인증 코드 반환
}
```

이메일 품에 대한 기본적인 정보를 설정하는 createEmailForm() 메서드이다. 인코딩과 제공 형태가 html임을 명시해주었다. 또한 setContext() 함수를 호출하는데, 이는 앞서 mail.html의

타임리프 변수인 `code`와 매핑시켜주는 역할을 함으로써 생성한 인증번호를 html 품에 보여주게 되는 것이다.

```
// 메일 양식 작성
public MimeMessage createEmailForm(String email) throws MessagingException, UnsupportedEncodingException {

    createCode(email); // 인증 코드 생성
    String setFrom = "kw2023db@gmail.com"; // email-config에 설정한 자신의 이메일 주소(보내는 사람)
    String toEmail = email; // 받는 사람
    String title = "[데비및데비시각화] 인증 코드는 " + authNum + "입니다"; // 제목

    MimeMessage message = emailSender.createMimeMessage();
    message.addRecipients(Message.RecipientType.TO, email); // 보낼 이메일 설정
    message.setSubject(title); // 제목 설정
    message.setFrom(setFrom); // 송신자 이메일
    message.setText(setContext(authNum), charset: "utf-8", subtype: "html");

    return message;
}

// 타임리프를 이용한 context 설정
public String setContext(String code) {
    Context context = new Context();
    context.setVariable("code", code); // 생성한 인증 번호가 th:text="${code}"와 매핑
    return templateEngine.process(template: "mail", context); // mail.html
}
```

랜덤 인증 코드를 생성하는 `createCode()` 메서드이다. 인증 번호의 경우, 4자리로 1000부터 9999의 범위까지의 랜덤한 값으로 제공하고자 한다.

따라서 `Random` 라이브러리를 사용해 생성한 랜덤한 수에 대하여 범위를 지정해줌으로써 4자리 랜덤 인증 번호를 생성하게 된다. 또한 어느 유저에게 발급된 인증 번호인지에 대한 정보를 DB에 저장해야 하므로 `dto`에 값을 저장해주었다.

```
// 랜덤 인증 코드 생성
public void createCode(String email) {
    Random random = new Random();
    authNum = String.valueOf(random.nextInt(bound: 9000)+1000); // 범위: 1000 ~ 9999

    EmailAuthDto emailAuthDto = new EmailAuthDto();
    emailAuthDto.setUser_id(userRepository.findUserIdByEmail(email));
    log.info("email = {}", email);
    emailAuthDto.setEmail(email);
    emailAuthDto.setAuth_num(Integer.parseInt(authNum));
    emailAuthDto.setCreated_at(LocalDateTime.now());

    emailAuthRepository.createAuthCode(emailAuthDto);
}
```

앞서 생성한 `dto`의 값을 DB에 저장하기 위한 코드인 `EmailAuthRepository.java`의 `createAuthCode()` 메서드이다. JDBC template을 사용하여 직접 작성한 SQL 쿼리를 통해 DB에 삽입되는 값을 `dto`의 값들과 매핑해줌으로써 DB에 정보가 담긴 새로운 행을 삽입하게 된다.

```

public Boolean createAuthCode(EmailAuthDto emailAuthDto) {
    String sql = "INSERT INTO emailauth (user_id, auth_num, created_at) VALUES (?, ?, ?)";
    jdbcTemplate.update(sql, emailAuthDto.getUser_id(), emailAuthDto.getAuth_num(), emailAuthDto.getCreated_at());

    return true;
}

```

추가적으로, 실제 실무 서비스에서는 인증 번호 발급 시 일정 시간 이후에 자동으로 만료가 된다. 따라서 우리의 프로젝트에도 실제와 최대한 똑같은 인증 번호 환경을 구축하기 위해 생성된 인증 번호가 5분이 되면 자동으로 만료(DB에서 자동으로 삭제)되는 스케줄러를 아래와 같이 별도로 구현하였다.

```

@Component
public class ScheduledTasks {

    @Autowired
    private ChangePasswordService changePasswordService;

    // 생성한 인증 번호가 5분이 되면 자동으로 만료(DB에서 인증번호 삭제)
    @Scheduled(fixedRate = 5000)
    public void deleteExpiredAuthNum() {
        changePasswordService.deleteExpiredAuthNum();
    }
}

```

구현한 스케줄러를 활성화 하기 위해 `@EnableScheduling` 어노테이션을 사용해주었다.

```

@SpringBootApplication(exclude = SecurityAutoConfiguration.class)
@EnableScheduling
public class DbdbApplication {

    public static void main(String[] args) { SpringApplication.run(DbdbApplication.class, args); }
}

```

실질적으로 DB에 인증번호가 생성된 시간을 기반으로 현재 시간과 비교하여 5분이 지났는지 확인하는 로직은 아래와 같다. 현재 시간 - 인증 번호 생성 시간의 결과를 분으로 반환하여 5분 이상일 경우 `emailAuthRepository`에서 해당 유저에 대해 발급된 인증번호를 DB에서 삭제하도록 구현하였다.

```

public void deleteExpiredAuthNum(){
    List<EmailAuthDto> emailAuthEntityList = emailAuthRepository.findAll();

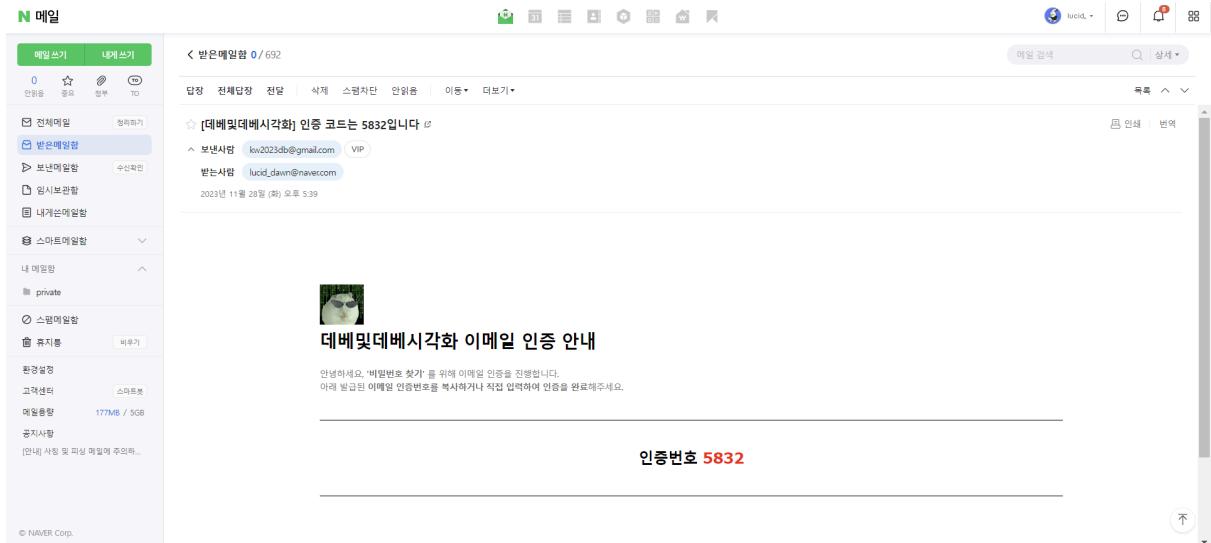
    for(EmailAuthDto emailAuthDto : emailAuthEntityList){
        // 현재 시간 - 인증 번호 발급 시간 계산
        Duration duration = Duration.between(emailAuthDto.getCreated_at(), LocalDateTime.now());

        if(duration.toMinutes() >= 5){
            emailAuthRepository.deleteByUserId(emailAuthDto.getUser_id());
        }
    }
}

```

아래와 같이 클라이언트에서 비밀번호 변경 중 인증 번호를 요청하는 상황이다.

요청한 결과, 아래 이미지와 같이 생성한 인증 번호 기반으로 만든 이메일 품이 실제로 메일로 전송된 것을 확인할 수 있다.

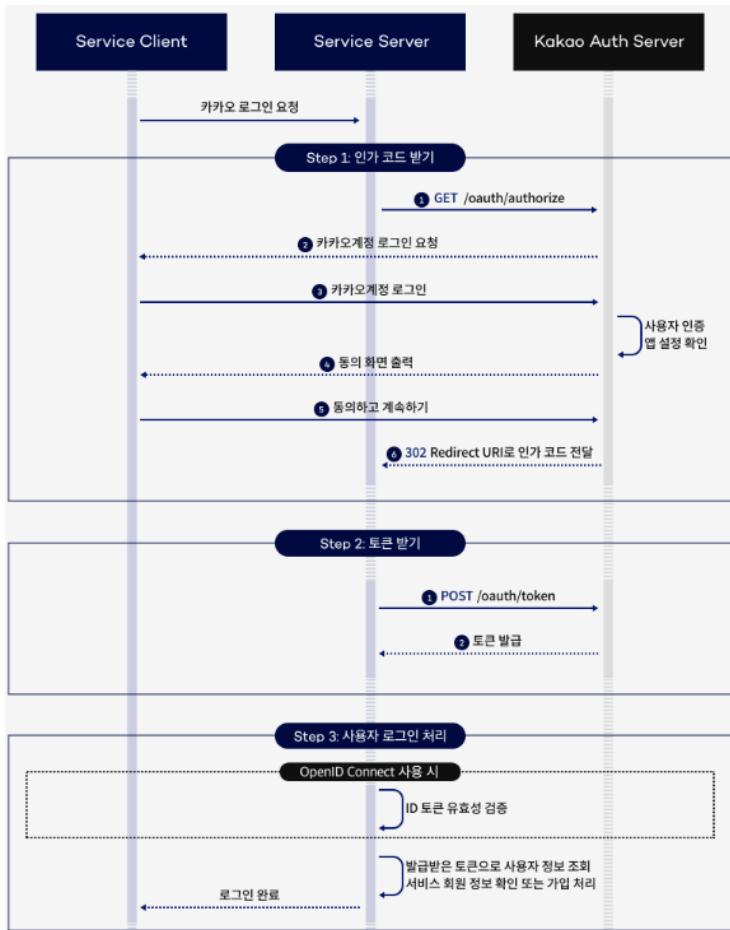


2. 소셜 로그인 (카카오)

소셜 로그인은 기존에 이메일과 비밀번호를 입력하여 로그인하는 방식과는 다르게 naver, kakao, google, apple 등의 소셜 서비스 기반의 로그인 방식이다. 로그인 과정이 간소화되며, 사용자 입장에서 비밀번호를 기억할 필요가 없다는 것이 큰 장점인 특수 기능이다. 그 중에서도 kakao에서 제공하는 소셜 로그인 서비스 구현을 진행할 것이다.

간편하고 편리한 기능인 반면에, 구현을 위해 **kakao developers**에서 제공하는 공식 개발 문서에서 제공하는 시퀀스 다이어그램에 대한 이해를 기반으로 **kakao developers**에서의 환경

설정을 바탕으로 server 측에서 구현을 진행해야 한다.



가장 먼저, kakao developer에 로그인하여 애플리케이션 추가를 진행해야 한다. 아래 이미지와 같이 앱 이름은 dbdbdiv으로, 카테고리의 경우 따릉이 플랫폼이기 때문에 자동차/교통수단으로 설정해주어 애플리케이션을 등록해주었다.

애플리케이션 추가하기

앱 아이콘		파일 선택
		JPG, GIF, PNG 권장 사이즈 128px, 최대 250KB
앱 이름	dbdbdiv	
사업자명	dbdbdiv	
카테고리	자동차/교통수단	▼
<ul style="list-style-type: none"> 입력된 정보는 사용자가 카카오 로그인을 할 때 표시됩니다. 정보가 정확하지 않은 경우 서비스 이용이 제한될 수 있습니다. 		
<input checked="" type="checkbox"/> 서비스 이용이 제한되는 카테고리, 금지된 내용, 금지된 행동 관련 운영정책을 위반하지 않는 앱입니다.		
취소		저장

애플리케이션을 추가한 이후, 아래와 같이 생성된 key들을 잘 저장해주었다. 해당 key들은 server 측에서 인가 코드 및 토큰 발급 시에 사용할 것이다.

 dbdbdiv 

ID 997410 OWNER

앱 키

네이티브 앱 키	[REDACTED]
REST API 키	[REDACTED]
JavaScript 키	[REDACTED]
Admin 키	[REDACTED]

플랫폼

설정된 플랫폼 정보가 없습니다. [플랫폼 설정하기](#)

기본 정보

앱 ID	997410
앱 이름	dbdbdiv
사업자명	dbdbdiv

이어서 Web 플랫폼 등록을 통해 local 환경에서 사용하는 우리 프로젝트의 domain과 port 번호를 명시해주었다.

The screenshot shows the Kakao Developers platform interface. On the left, there's a sidebar with various settings like App Settings, OAuth, General, Business, Keys, and Platforms. The 'Platforms' section is highlighted with a red box. In the main area, there's a list for different platforms: Android, iOS, and Web. Each has a 'Register Platform' button. The 'Web' section is also highlighted with a red box around its 'Register Platform' button. Below this, there's a detailed view of the 'Web Platform Registration' page. It asks for a 'Site Domain' (填写了 http://localhost:8080) and a 'Default domain' (填写了 http://localhost:8080). At the bottom are 'Delete' and 'Save' buttons.

카카오 로그인 API를 활성화를 위해 ON으로 설정하고, redirect URI를 등록해주었다.

이제 사용자가 카카오 로그인을 진행할 때 받을 정보들에 대한 동의항목을 설정해야

한다. 이 과정을 통해 사용자가 로그인을 진행할 때 사용자의 이메일, 닉네임 등의 정보를

얻어 이메일 로그인 방식과 같이 사용자의 정보를 구성하여 자동으로 회원가입을 진행할

것이다.

닉네임에 대한 동의 항목을 필수 동의로 설정으로 변경하고 동의 목적 작성 후에 저장을

완료하였다.

동의 항목 설정

항목

닉네임 / profile.nickname

동의 단계

필수 동의

카카오 로그인 시 사용자가 필수로 동의해야 합니다.

선택 동의

사용자가 동의하지 않아도 카카오 로그인을 원료할 수 있습니다.

이용 중 동의

카카오 로그인 시 동의를 받지 않고, 항목이 필요한 시점에 동의를 받습니다.

사용 안함

사용자에게 동의를 요청하지 않습니다.

동의 목적 [필수]

앱에서 사용할 닉네임 설정

개발자 앱 동의 항목 관리 화면내에 입력하는 사실이 실제 서비스 내용과 다를 경우 API 서비스의 거부사유가 될 수 있습니다.

취소

저장

하지만 이메일의 경우, 바로 필수 동의로 설정할 수 없으므로 비즈니스 등록을 진행해야

한다.

아래와 같이 이메일 필수 동의 목적으로 비즈앱 전환을 진행하여 이메일을 동의 항목

설정이 가능하도록 권한을 부여해주었다.

내애플리케이션 > 앱 설정 > 비즈니스

앱 설정

요약 정보

일반

비즈니스

앱 키

플랫폼

앱 권한 신청

팀 관리

제품 설정

카카오 로그인

동의항목

간편가입

카카오톡 채널

개인정보 국외이전

비즈 앱 정보

이 앱은 비즈 앱이 아닙니다.

사업자 정보를 등록하여 비즈 앱으로 전환할 수 있습니다. 비즈 앱 전환 시 이메일을 필수 동의항목으로 설정할 수 있고, 비즈니스 채널 연결이 가능합니다.

사업자 정보 등록

개인 개발자 비즈 앱

사업자 번호가 없는 개인 개발자는 본인인증과 카카오비즈니스 통합 서비스 약관 동의 완료 후 비즈 앱 전환이 가능합니다.
사업자 정보를 추후 입력할 수 있습니다.

개인 개발자 비즈 앱 전환

개인 개발자 비즈 앱 전환

비즈 앱 전환 목적

비즈 앱으로 전환하려는 목적을 선택해주세요.

이메일 필수 동의	▼
-----------	---

[취소](#) [전환](#)

이제 이메일(카카오계정)에 대한 동의 항목 설정을 필수 동의로 선택할 수 있으므로, 성공적으로 권한이 부여된 것을 확인할 수 있다. 마찬가지로 동의 목적 작성 후 저장해주었다.

동의 항목 설정

항목
카카오계정(이메일) / account_email

동의 단계

필수 동의
카카오 로그인 시 사용자가 필수로 동의해야 합니다.

선택 동의
사용자가 동의하지 않아도 카카오 로그인을 원료할 수 있습니다.

이용 중 동의
카카오 로그인 시 동의를 받지 않고, 항목이 필요한 시점에 동의를 받습니다.

사용 안 함
사용자에게 동의를 요청하지 않습니다.

카카오 계정으로 정보 수집 후 제공

사용자에게 값이 없는 경우 카카오 계정 정보 입력을 요청하여 수집

동의 목적 [필수]

앱에서 사용할 이메일 설정

개발자 앱 동의 항목 관리 화면내에 입력하는 사실이 실제 서비스 내용과 다를 경우 API 서비스의 거부 사유가 될 수 있습니다.

[취소](#) [저장](#)

필요한 정보들을 필수 동의로 설정했을 때 아래와 같이 구성이 이루어진다.

개인정보

항목 이름	ID	상태	
닉네임	profile.nickname	● 필수 동의	설정
프로필 사진	profile.image	● 필수 동의	설정
카카오계정(이메일)	account.email	● 필수 동의 [수집]	설정

이제 본격적으로 kakao developer의 REST API 공식 document에 명시되어 있는 API spec과 시퀀스 다이어그램을 기반으로 서버 측에서 구현을 진행한다.

<https://developers.kakao.com/docs/latest/ko/kakaologin/rest-api>

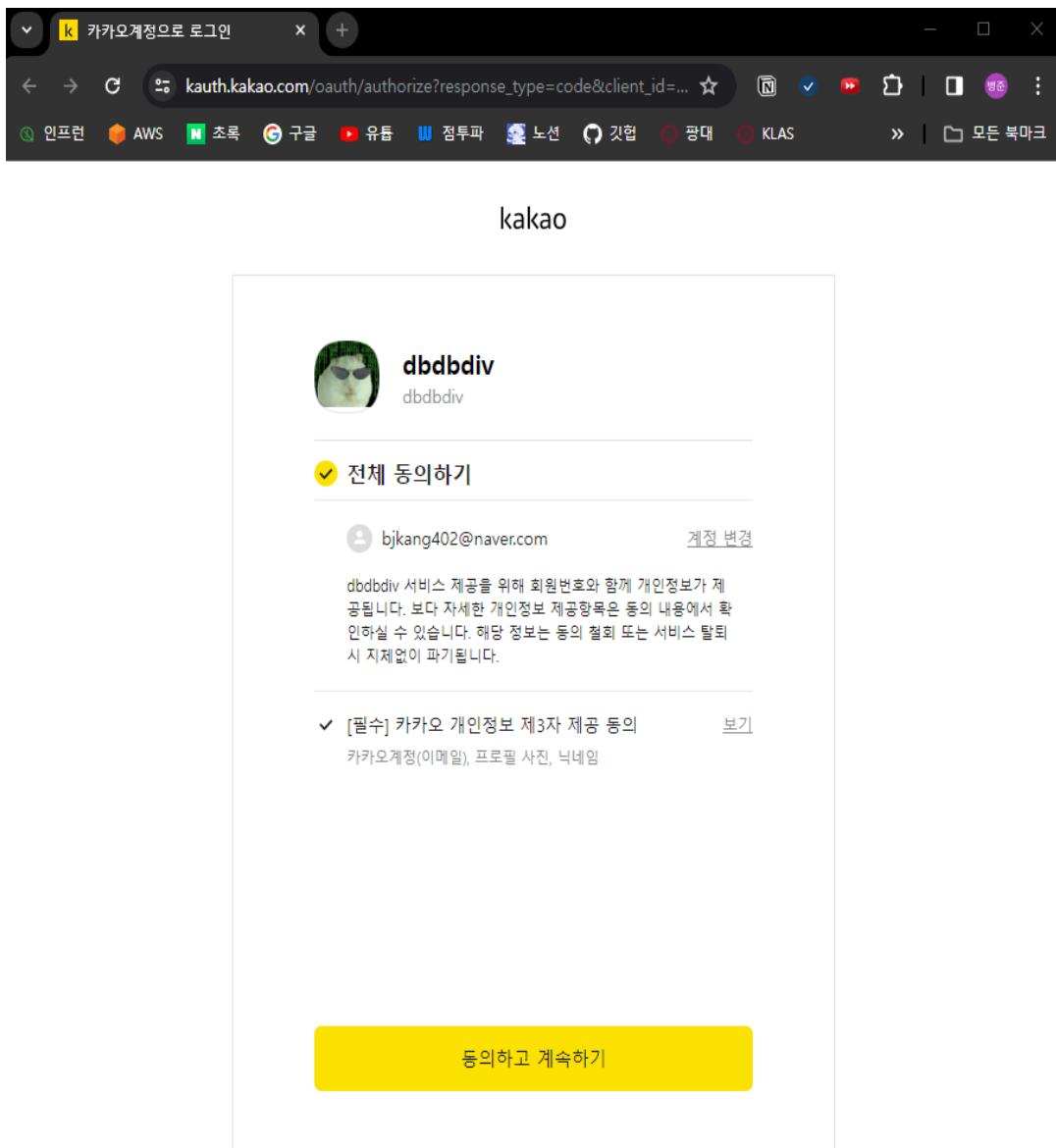
먼저 인가 코드 요청 URL은 아래와 같다.

`https://kauth.kakao.com/oauth/authorize?response_type=code&client_id=${REST_API_KEY}&redirect_uri=${REDIRECT_URI}`

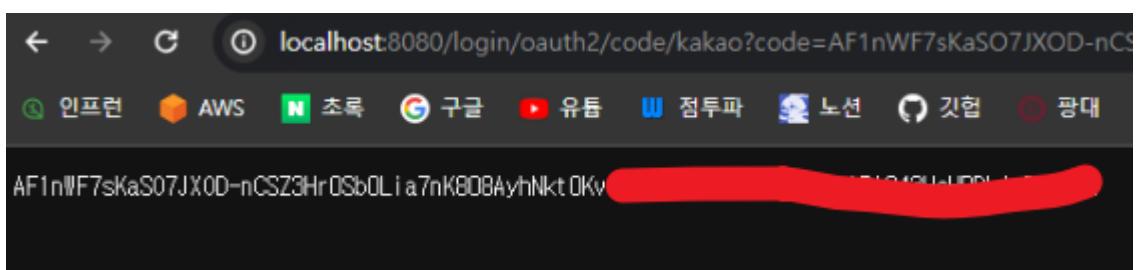
요청

```
https://kauth.kakao.com/oauth/authorize?response_type=code&client_id=${REST_API_KEY}&redirect_
```

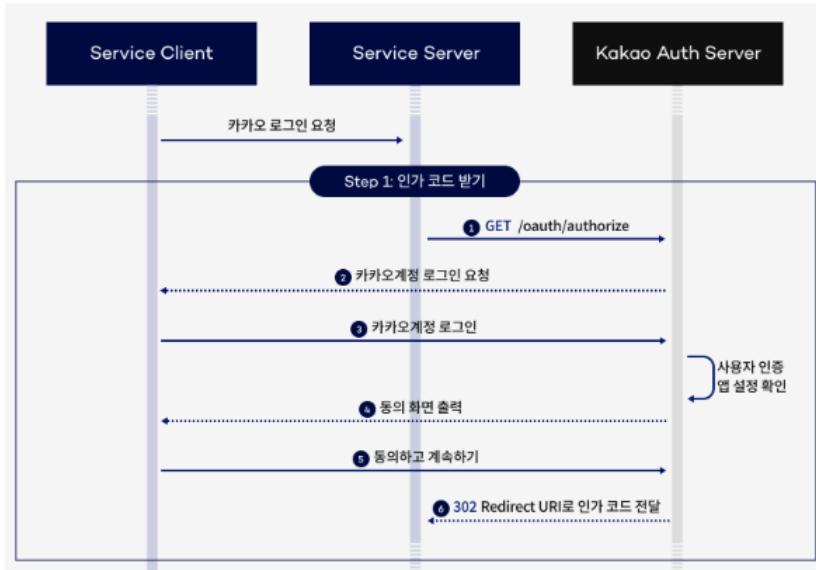
앞서 위에서 저장한 key들을 넣고 URL에 접속하면 아래와 같이 정상적으로 “카카오계정으로 로그인” 페이지가 나오는 것을 확인할 수 있다. 하지만 필수 동의를 한 후에 “동의하고 계속하기”를 눌러도 DB 상에는 저장이 되지 않기 때문에 server 로직 상으로 구현하는 과정이 필요하다. “동의하고 계속하기”를 누르면 아래와 같이 코드가 반환된 것을 확인할 수 있다.



Copyright © Kakao Corp. All rights reserved.



해당 코드는 Step 1: 인가 코드 받기에서 6번(302 REDIRECT URI로 인가 코드 전달)이 수행된 것이다. 성공적으로 코드가 반환되었으므로, Step 1 과정이 완료되었다.



Step2 과정은 아래와 같다.



service 로직부터 설명을 진행하겠다.

먼저, 앞서 얻은 인가 코드를 바탕으로 <https://kauth.kakao.com/oauth/token> URI로 POST 요청을 하여 액세스 토큰을 발급 받아야 한다. 처음 과정에서 저장한 key들과 해당 URI들은 application.yml에 저장하여 보안을 유지하고, Environment 객체를 통해 application.yml과 매팅하여 값을 안전하게 가져와서 사용하였다. document에서 요구하는 http header 탑입을 설정하고, restTemplate으로 resourceUri(<https://kauth.kakao.com/oauth/token>)에 POST 방식으로 request를 수행하여 얻은 response를 반환하는 로직을 getAccessTokenResponse() 메서드에 구현을 진행함으로써 Step2 과정이 완료되었다.

```

// request access token
public JsonNode getAccessTokenResponse(String code) {
    RestTemplate restTemplate = new RestTemplate();

    // generate HttpHeaders object
    HttpHeaders headers = new HttpHeaders();
    headers.add(headerName: "Content-type", headerValue: "application/x-www-form-urlencoded; charset=utf-8");

    // generate HttpBody object
    String clientId = env.getProperty("oauth.kakao.client-id");
    String redirectUri = env.getProperty("oauth.kakao.redirect-uri");
    String resourceUri = env.getProperty("oauth.kakao.resource-uri");

    // log(for check)
    log.info("clientId = {}", clientId);
    log.info("redirectUri = {}", redirectUri);
    log.info("resourceUri = {}", resourceUri);

    // HttpHeaders and HttpBody in one object
    MultiValueMap<String, String> params = new LinkedMultiValueMap<>();
    params.add("grant_type", "authorization_code");
    params.add("client_id", clientId);
    params.add("redirect_uri", redirectUri);
    params.add("code", code);

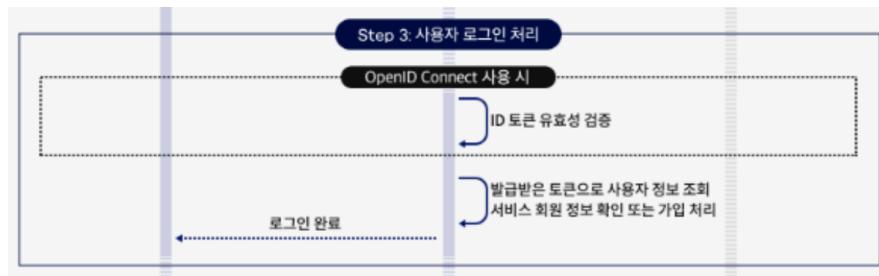
    HttpEntity<MultiValueMap<String, String>> accessTokenRequest = new HttpEntity<>(params, headers);

    JsonNode response = restTemplate.exchange(
        resourceUri,
        HttpMethod.POST,
        accessTokenRequest,
        JsonNode.class
    ).getBody();

    return response;
}

```

마지막 Step3 과정은 아래와 같다.



앞서 얻은 `access_token`을 바탕으로 사용자의 정보를 얻어 자동으로 회원 가입 및 로그인이 수행되도록 할 것이다.

`getAccessTokenResponse()` 메서드를 통해 얻은 액세스 토큰 정보가 담겨 있는 `response`에서 파싱한 `body`를 `parshingAccessToken()` 메서드에서 `access_Token`만을 파싱하여 액세스 토큰을 얻었다.

얻은 액세스 토큰을 인자로 받는 `getUserInfoByAccessTokenResponse()`에서 `access_token`을 기반으로 `user-resource-uri`(<https://kapi.kakao.com/v2/user/m2>)에 POST 방식으로 `request`를 한다. 이 과정을 통해 카카오에서 정식으로 발급된 접근 권한을 가진 `access_token`으로 해당 사용자의 카카오 정보에 접근하여 얻은 `response`의 `body`를 `JsonNode` 객체로 반환할 수 있다.

위에서 반환 받은 유저 정보가 담긴 **JsonNode**에 담겨 있는 유저의 여러 정보들 중에서 유효한 이메일인지 검증을 위해 **is_email_verified**를, 회원 가입 시에 필요한 정보인 **email, nickname**을 파싱해주었다.

```
// Parse and return the access token as it has been obtained
public String parseAccessToken(JsonNode responseBody) { return responseBody.get("access_token").asText(); }

// Request information about users logged in using access tokens
public JsonNode getUserInfoByAccessTokenResponse(JsonNode accessToken) throws JsonProcessingException {

    // Transformation process for processing json objects in Java
    ObjectMapper objectMapper = new ObjectMapper();
    OAuthToken oAuthToken = objectMapper.readValue(accessToken.toString(), OAuthToken.class);
    log.info("accessToken = {}", oAuthToken.getAccess_token());

    RestTemplate restTemplate = new RestTemplate();

    // generate HttpHeaders object
    HttpHeaders headers = new HttpHeaders();
    headers.add(headerName: "Authorization", headerValue: "Bearer "+oAuthToken.getAccess_token());
    headers.add(headerName: "Content-type", headerValue: "application/x-www-form-urlencoded; charset=utf-8");

    HttpEntity<MultiValueMap<String, String>> userInfoRequest = new HttpEntity<>(headers);

    String userResourceUri = env.getProperty("oauth.kakao.user-resource-uri");

    // get user information by json
    return restTemplate.exchange(userResourceUri, HttpMethod.POST, userInfoRequest, JsonNode.class).getBody();
}
```

이어서 이메일에 대한 유효성 검사를 **is_email_verified**로 검증을 진행하고 해당 **email**을 가지고 있는 **user**가 있는지 **userRepository**에서 찾는다. **userDto**가 null이 아니라면 이미 카카오 회원가입을 진행하여 자동 회원가입이 진행된 유저이므로 최근 접속 시간만을 갱신하여 객체를 **controller**로 반환한다. **userDto**가 null이라면 처음 카카오 로그인을 진행한 경우이므로, 앞서 얻은 유저 정보들로 객체를 생성해 **userRepository**를 통해 **user table**에 해당 **user**에 대한 정보를 저장한 후에 객체를 **controller**로 반환한다. 두 경우 모두 **controller** 상에서 **cookie**를 생성하여 로그인 이후의 정보 관리를 수행하게 된다.

위의 **service** 로직에서 호출하는 **userRepository**의 코드는 아래와 같다.

JdbcTemplate을 사용하여 **SQL query**를 통해 데이터베이스 관련 작업을 수행하였다.

이미 카카오 로그인을 하여 자동 회원가입이 된 유저에 대해서는

`updateLastAccessedAt()`을 통해 해당 id를 가지는 `user`의 최근 접속 시간인

`last_accessed_at` 필드를 현재 로그인한 시간으로 `UPDATE`문을 통해 DB에 반영한다.

만약 카카오 로그인을 처음 수행하여 `insertUser()`를 통해 자동 회원가입을 진행하는

유저에 대해서 모든 유저 정보를 바탕으로 `user table`의 모든 `column`에 매핑하여

DB에 해당 `user`에 대한 정보를 `INSERT` 문으로 추가함으로써 회원 가입을 수행하게 된다.

```
@Repository
public class UserRepository {
    @Autowired
    private JdbcTemplate jdbcTemplate;

    public Boolean insertUser(UserDto userDto) {
        String sql = "INSERT INTO user (password, username, user_type, email, phone_number, weight, age, last_accessed_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
        jdbcTemplate.update(sql,
            userDto.getPassword(),
            userDto.getUsername(),
            userDto.getUserType(),
            userDto.getEmail(),
            userDto.getPhone_number(),
            userDto.getWeight(),
            userDto.getAge(),
            userDto.getLast_accessed_at());
        return true;
    }

    public void updateLastAccessedAt(UserDto userDto) {
        String sql = "UPDATE user SET last_accessed_at = ? WHERE id = ?";
        jdbcTemplate.update(sql, userDto.getLast_accessed_at(), userDto.getId());
    }
}
```

아래는 위의 로직들을 호출하는 카카오 로그인 API에 대한 controller 코드이다. 앞선

로직들을 수행하고 반환 받은 `userDto`를 바탕으로 쿠키를 생성하여 로그인 이후 과정을

수행할 수 있게 해준다.

```
// Kakao login
@GetMapping("/login/oauth2/code/kakao")
public ResponseEntity<@RequestParam String code, HttpServletRequest request, HttpServletResponse response> throws JsonProcessingException {
    JsonNode accessTokenResponse = kakaoLoginService.getAccessTokenResponse(code); // code를 통해 얻은 response(access token과 여러 key를 존재)
    String accessToken = KakaoLoginService.parsingAccessToken(accessTokenResponse);
    JsonNode userInfoResponse = KakaoLoginService.getUserInfoByAccessTokenResponse(accessTokenResponse); // access token을 통해 얻은 response(유저 정보 존재)

    UserDto userDto = kakaoLoginService.parsingUserInfo(userInfoResponse);

    HttpSession session = request.getSession();
    session.setAttribute("access_token", accessToken);

    Cookie idCookie = new Cookie("id", String.valueOf(userDto.getId()));
    Cookie emailCookie = new Cookie("email", userDto.getEmail());
    Cookie passwordCookie = new Cookie("password", userDto.getPassword());
    Cookie usernameCookie = new Cookie("username", userDto.getUsername());

    idCookie.setMaxAge(7 * 24 * 60 * 60);
    emailCookie.setMaxAge(7 * 24 * 60 * 60);
    passwordCookie.setMaxAge(7 * 24 * 60 * 60);
    usernameCookie.setMaxAge(7 * 24 * 60 * 60);

    idCookie.setHttpOnly(true);
    emailCookie.setHttpOnly(true);
    passwordCookie.setHttpOnly(true);
    usernameCookie.setHttpOnly(true);

    idCookie.setPath("/");
    emailCookie.setPath("/");
    passwordCookie.setPath("/");
    usernameCookie.setPath("/");

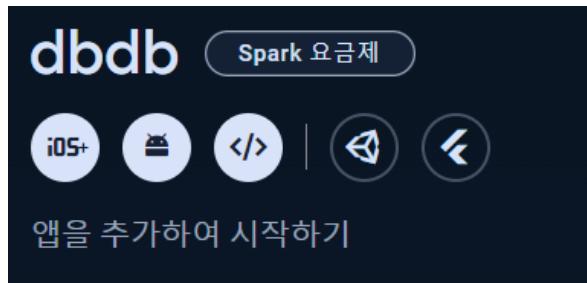
    response.addCookie(idCookie);
    response.addCookie(emailCookie);
    response.addCookie(passwordCookie);
    response.addCookie(usernameCookie);

    return ResponseEntity.ok(new JsonResponse<(ResponseStatus.SUCCESS_KAKAO_LOGIN, result: null)>);
}
```

3. Firebase Cloud Messaging(FCM) 기능

FCM은 Google firebase에서 지원하는 클라우드 상의 메시지 전송 서비스이다.

FCM을 이용하기 위해서 먼저 **firebase** 콘솔에 접속하여 프로젝트를 생성해 주어야 한다. 이후 생성한 프로젝트를 확인하면



위와 같이 앱을 추가하여 시작할 수 있는데, 우리는 웹 개발이기 때문에 </> 모양의 버튼을 클릭하고, 앱 이름을 입력하여 등록한다.

2 Firebase SDK 추가

npm 사용 <script> 태그 사용

이미 npm 및 모듈 번들러(예: webpack 또는 Rollup)를 사용 중인 경우 다음 명령어를 실행하면 최신 SDK를 설치할 수 있습니다(자세히 알아보기).

```
$ npm install firebase
```

그런 다음 Firebase를 초기화하여 사용하려는 제품의 SDK를 사용하세요.

A screenshot of the 'Add Firebase SDK' step in the Firebase setup wizard. It shows two radio button options: 'npm 사용' (selected) and '<script> 태그 사용'. Below the options is a note about using npm or module bundlers like webpack or Rollup, with a command line instruction: '\$ npm install firebase'. At the bottom, it says '그런 다음 Firebase를 초기화하여 사용하려는 제품의 SDK를 사용하세요.' (Then initialize Firebase to use the SDK for the product you're using).

이후, 위와 같이 npm을 선택하여 firebase SDK를 받아온다.

웹 구성

웹 푸시 인증서

웹 푸시 인증서

Firebase 클라우드 메시징은 애플리케이션 ID 키 쌍을 사용하여 외부 푸시 서비스에 연결할 수 있습니다. 자세히 알아보기

키 쌍	추가된 날짜	작업

Generate key pair

기존 키 쌍을 가져올 수도 있습니다.

A screenshot of the 'Web Pusher' configuration screen in the Firebase console. It shows a sidebar with '웹 푸시 인증서' and a main panel with the same title. A note says 'Firebase 클라우드 메시징은 애플리케이션 ID 키 쌍을 사용하여 외부 푸시 서비스에 연결할 수 있습니다.' with a '자세히 알아보기' link. Below is a table for managing key pairs, which is currently empty. A blue button says 'Generate key pair'. At the bottom, it says '기존 키 쌍을 가져올 수도 있습니다.'

또한, Web Push 프로토콜 표준인 VAPID 인증 방식을 사용해야 하기 때문에 프로젝트 설정 내의 클라우드 메시징 탭에서 VAPID 키를 발급받고, 이를 이용해 Web Push 메시지의 토큰을 발급받게 된다.

이제 front에서 firebase 알림이 왔을 때의 처리를 위해 설정해 주어야 한다.

```
import { BrowserRouter } from 'react-router-dom';
import './firebase-messaging-sw.js';
```

firebase-messaging-sw.js에 firebase SDK에 존재하는 정보를 이용해 환경변수로 등록하고, 이를 이용하여 firebase config를 생성한 뒤 해당 파일을 루트 디렉토리의 App.js에 import하여 설정을 마친다.

해당 파일은 아래와 같다.

```
src /--> firebase-messaging-sw.js / ...
1 1 import { initializeApp } from "firebase/app";
2 2 import { getMessaging, getToken, onMessage } from "firebase/messaging";
3
4 3 const firebaseConfig = {
5    apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
6    authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
7    projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
8    storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
9    messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
10   appId: process.env.REACT_APP_FIREBASE_APP_ID
11 };
```

```

12
13  // Initialize Firebase
14  const app = initializeApp(firebaseConfig);
15  const messaging = getMessaging(app);
16
17  async function requestPermission() {
18    console.log("권한 요청 중...");
19
20    const permission = await Notification.requestPermission();
21    if (permission === "denied") {
22      console.log("알림 권한 허용 안됨");
23      return;
24    }
25
26    console.log("알림 권한이 허용됨");
27
28    const token = await getToken(messaging, {
29      vapidKey: process.env.REACT_APP_VAPID_KEY,
30    });
31
32    if (token) console.log("token: ", token);
33    else console.log("Can not get Token");
34
35    onMessage(messaging, (payload) => {
36      console.log("메시지가 도착했습니다.", payload);
37      // ...
38    });
39
40    // Set localStorage token
41    localStorage.setItem("fcm_token", token);
42
43    return token
44  }
45
46  const token = requestPermission();
47
48  export [
49    token
50  ]

```

또한, 이를 `Index.js`에서 아래와 같이 Service worker에 등록해 준다.

```

26
27  if ("serviceWorker" in navigator) {
28    window.addEventListener("load", function () {
29      navigator.serviceWorker
30        .register("/firebase-messaging-sw.js")
31        .then(function (registration) {
32          console.log("Service Worker 등록 성공:", registration.scope);
33        })
34        .catch(function (err) {
35          console.log("Service Worker 등록 실패:", err);
36        });
37    });
38  }
39

```

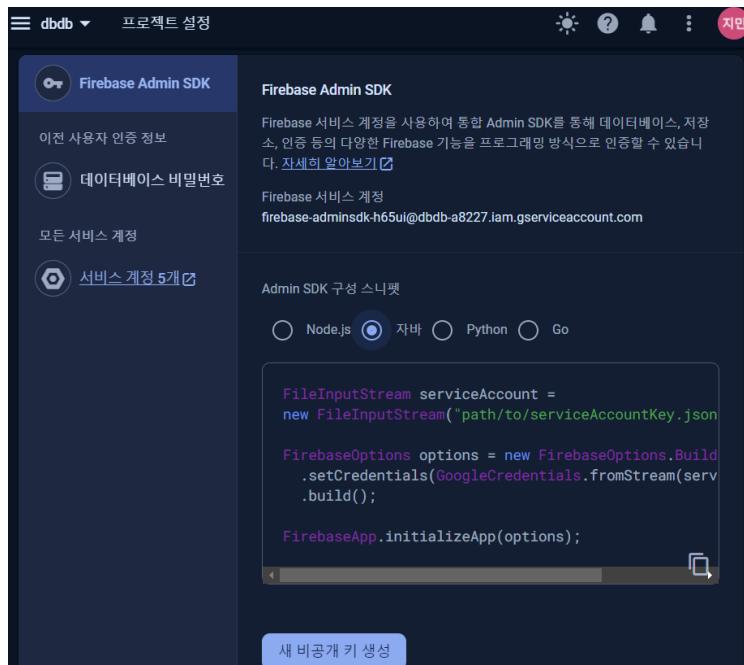
마지막으로, `public` 폴더 내에 `firebase-messaging-sw.js`를 생성하여 아래와 같이 입력한 뒤 Front의 Setting 과정을 끝낸다. 실제로 구현할 때에는 사용자마다 발급되는 FCM Token을 이용하여 해당 Token으로 Backend에서 알림을 전송하면 프론트의 위 코드가 알림을 받게 된다.

```

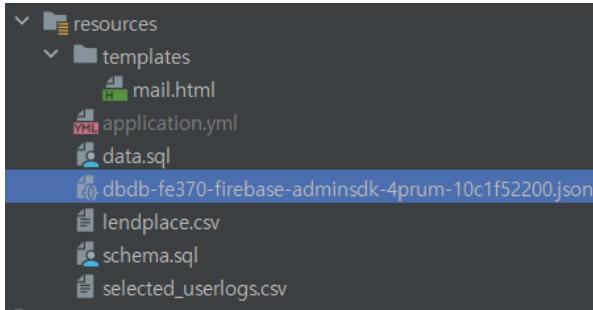
1  self.addEventListener("install", function (e) {
2    console.log("fcm sw install..");
3    self.skipWaiting();
4  });
5
6  self.addEventListener("activate", function (e) {
7    console.log("fcm sw activate..");
8  });
9
10 self.addEventListener("push", function (e) {
11   console.log("push: ", e.data.json());
12   if (!e.data.json()) return;
13
14   const resultData = e.data.json().notification;
15   const notificationTitle = resultData.title;
16   const notificationOptions = {
17     body: resultData.body,
18     icon: resultData.image,
19     tag: resultData.tag,
20     ...resultData,
21   };
22   console.log("push: ", { resultData, notificationTitle, notificationOptions });
23
24   self.registration.showNotification(notificationTitle, notificationOptions);
25 });
26
27 self.addEventListener("notificationclick", function (event) {
28   console.log("notification click");
29   const url = "/";
30   event.notification.close();
31   event.waitUntil(clients.openWindow(url));
32 });
33

```

이제 Backend의 처리를 위해 `firebase` 콘솔로 다시 이동한다.



`firebase`에서 새 비공개 키 생성을 통해 생성된 파일을 `intelliJ`에 넣고, `application.yml`에 FCM을 사용하기 위한 기본적인 프로퍼티들이 담겨 있는 해당 파일을 작성해주었다.



이제 FCM 셋팅 내용을 바탕으로 생성한 리소스로 초기화를 진행하는 설정 로직들을

아래와 같이 작성해주었다. `@Value` 어노테이션을 통해 `application.yml`에 앞서 작성한 `property`를 가져와주었다.

```
15  @Slf4j
16  @Component
17  public class FCMInitializer {
18
19      @Value("${fcm.certification}")
20      private String googleApplicationCredentials;
21
22      @PostConstruct
23      public void initialize() throws IOException {
24          ClassPathResource resource = new ClassPathResource(googleApplicationCredentials);
25
26          try (InputStream is = resource.getInputStream()) {
27              FirebaseOptions options = FirebaseOptions.builder()
28                  .setCredentials(GoogleCredentials.fromStream(is))
29                  .build();
30
31          if (FirebaseApp.getApps().isEmpty()) {
32              FirebaseApp.initializeApp(options);
33              log.info("FirebaseApp initialization complete");
34          }
35      }
36  }
```

이후 아래와 같이 `fcm token`에 대한 CRUD 로직을 바탕으로 아래와 같이 로그인을 성공적으로 완료 했을 때 FCM 알림을 클라이언트에게 송신할 수 있도록 구현해주었다.

아래와 같은 로직으로 로그인 완료 알림 이외에 로그아웃 완료 알림, 이용권 구매 성공 알림, 이용권 구매 실패 알림, 자전거 대여 성공 알림, 자전거 대여 실패 알림, 내 글과 댓글의 좋아요에 대해서 클라이언트에게 알림 기능을 제공해주었다.

```
public void send(Message message) { FirebaseMessaging.getInstance().sendAsync(message); }

public void sendLogincompletedMessage(String email) {
    if (!fcmTokenRepository.hasKey(email)) {
        return;
    }

    String token = fcmTokenRepository.getToken(email);
    Message message = Message.builder()
        .setNotification(Notification.builder()
            .setTitle("로그인 완료 알림")
            .setBody("로그인 되었습니다. ")
            .build())
        .putData("title", "로그인 완료 알림")
        .putData("content", "로그인 되었습니다. ")
        .setToken(token)
        .build();

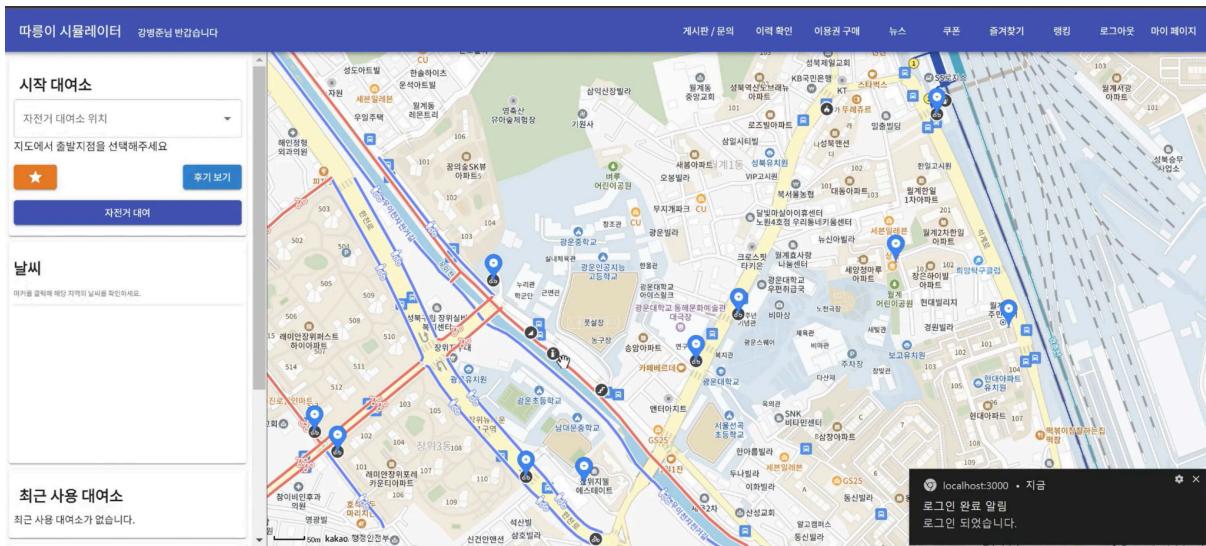
    send(message);
}
```

아래와 같이 각 controller에서 성공적으로 로직 동작 시 아래의 로직이 수행되도록 코드를 작성함으로써 FCM 알림 기능 구현이 완료되었다.

```
fcmService.saveTokenByObject(userDto);
fcmService.sendLogincompletedMessage(userDto.getEmail());
```

FCM은 로그인 완료 알림, 로그아웃 완료 알림, 이용권 구매 실패 알림, 이용권 구매 성공, 자전거 대여 성공, 자전거 대여 실패, 자전거 반납 성공, 자전거 반납 실패, 자신이 작성한 댓글에 좋아요가 눌렸을 때, 자신이 작성한 글에 좋아요가 눌렸을 때 알림이 오도록 적용해주었다.

아래는 로그인은 성공적으로 완료하고 리다이렉트 된 메인 화면에서 로그인을 성공적으로 되었음을 알 수 있는 FCM 로그인 완료 알림이 우측 하단에 생성된 것을 확인할 수 있다.



4, 5. 댓글 작성 및 조회

SCREEN

따릉이 너무 좋아요

작성자: 강병준 조회수: 3 작성일: 2023-12-03 19:31

좋아요~

목록으로 이동
♥
1

댓글
인정합니다
등록

따릉이 너무 좋아요
작성자: 강병준 조회수: 4 작성일: 2023-12-03 19:31



DESCRIPTION

자유게시판 및 1:1 문의 게시판 내에 댓글 기능이 있으며, 해당 기능 내부에서 댓글을 직접 작성 및 작성한 댓글을 조회할 수 있다.

FRONT END	BACK END
<pre> {showReplyInput && (<Box sx={[display: "flex", flexDirection: "row", alignItems: "center", width: "100%", justifyContent: "space-between", py: 2, pb: 10,]} > <TextField id="standard-basic" label="댓글" variant="standard" sx={[width: "100%", mx: 2]} value={myComment} onChange={(e) => setMyComment(e.target.value)} /> <Button variant="contained" sx={[mt: 3]} onClick={submitCommentHandler}> 등록 </Button> </Box>)};); export default BoardRead;</pre> <p>글 조회 페이지 내에 댓글을 작성할 수 있는 폼을 렌더링하고, 댓글 입력 후 등록 버튼을 누르면 백엔드 API에 댓글 작성 요청한다. 만약 댓글이 있다면, 글 조회 API내부에서 댓글도 확인할 수 있으며 댓글이 존재한다면 해당 댓글을 글 조회 페이지 내에서 렌더링한다.</p>	<pre>no usages 티 편지판 @RequestMapping("/post-comments") public ResponseEntity<JsonResponse> getComments(@RequestParam int write_id, @RequestParam int user_id) { List<CommentDto> commentDtos = commentService.getCommentEachBoard(write_id, user_id); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, commentDtos)); } no usages 티 편지판 @PostMapping("/comment-write") public ResponseEntity<JsonResponse> createComment(@RequestBody CommentDto.CreateCommentDto createCommentDto) { commentService.createComment(createCommentDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); }</pre> <p>위 controller를 확인하면, '/board/comment-write', '/board/get-comments'로 요청하면 각 컨트롤러 코드가 동작하게 된다. 먼저 댓글 작성의 경우</p> <pre>no usages 티 편지판 public void createComment(CommentDto.CreateCommentDto createCommentDto) { try { Comment comment = new Comment(0, //id createCommentDto.getUserId(), createCommentDto.getWriter_id(), createCommentDto.getCategory_id(), createCommentDto.getContent(), LocalDateTime.now(), LocalDateTime.now()); commentRepository.insertComment(comment); } }</pre> <p>사용자가 입력한 정보를 이용해 특정 게시글에 comment를 추가하는 것을 확인할 수 있고, 다음으로 댓글 조회의 경우</p>

```

public List<CommentDto.GetCommentDto> getCommentEachBoard(int writeId, int userId) {
    List<CommentDto.GetCommentDto> commentDtoList = new ArrayList<>();
    try {
        List<CommentDto.DBReturnCommentDto> commentList = commentRepository.findCommentByWriteId(writeId, userId);

        for(CommentDto.DBReturnCommentDto comment : commentList) {
            String username = "";
            UserDto.UserNameTypeDto userNameTypeDto = userRepository.findNameTypeByNameById(comment.getUserId());
            if(userNameTypeDto.getter_type() == 0) {
                username = "System";
            } else {
                username = userNameTypeDto.getUsername();
            }

            commentDtoList.add(new CommentDto.GetCommentDto(comment.getId(),
                username, comment.getWriteId(), comment.getCategory_id(),
                comment.getContent(), comment.getLikeCount(), comment.isUserLiked(),
                comment.getCreatedAt(), comment.getUpdatedAt()));
        }
    }
}

```

board id를 통해 comment 정보를 모두 조회한 뒤, 작성자가 관리자일 경우 이름을 관리자로 변경하여 반환해 주었다.
또한, 게시글 하나를 조회할 때 이 service code를 활용하여 해당하는 댓글들을 모두 함께 조회하여 반환하게 된다.

SQL

```

jdbcTemplate.update("INSERT INTO `comment` (`user_id`, `write_id`, `category_id`, `content`, " +
    "`created_at`, `updated_at`) VALUES (?, ?, ?, ?, ?, ?)",
    user_id, write_id, category_id, content, created_at, updated_at);

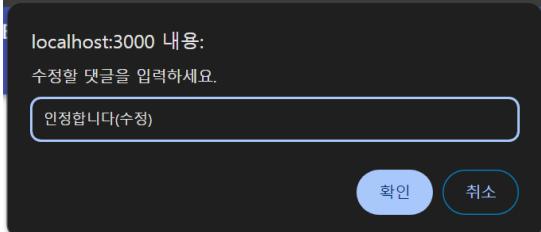
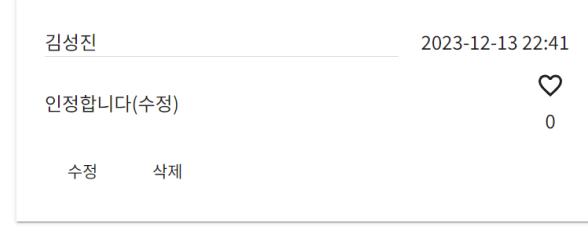
1 Usage ┌ 방지법
public List<CommentDto.DBReturnCommentDto> findCommentByWriteId(int writeId, int userId) {
    var commentMapper = BeanPropertyRowMapper.newInstance(CommentDto.DBReturnCommentDto.class);

    List<CommentDto.DBReturnCommentDto> comments = jdbcTemplate.query(
        "SELECT C.*," +
            "(SELECT COUNT(*) FROM `comment_like` WHERE `liked_id` = C.id) AS `like_count`, " +
            "EXISTS(SELECT 1 FROM `comment_like` WHERE `liked_id` = C.id AND `user_id` = ?) AS `user_liked` " +
            "FROM `comment` C WHERE C.`write_id` = ?",
        commentMapper, userId, writeId
    );
}

```

먼저 댓글 작성의 경우 댓글 정보에 해당하는, 사용자가 입력한 정보들을 INSERT 문을 이용하여 comment table에 추가하고 있는 것을 확인할 수 있다.
다음으로 findCommentByWirteld의 경우, writeld, 즉 게시글의 id와 해당 댓글을 조회하는 user의 id를 함께 입력한다. 이 정보를 이용하여 comment가 속해 있는 게시글 id, 즉 write_id가 입력한 write_id와 같은 comment 정보들을 출력한다. 이때, comment_like table을 이용하여 해당 댓글의 총 좋아요 수와, 입력한 user_id와 일치하는 좋아요가 존재하는지 판단하여 해당 사용자가 이 댓글에 좋아요를 했는지 여부를 알 수 있다.

6, 7. 댓글 수정 및 삭제

SCREEN	
	
	
DESCRIPTION	
<p>본인이 작성한 댓글이라면 댓글 수정 및 삭제 버튼이 존재하며, 수정 버튼을 누르면 프롬프트 창을 이용해 댓글을 수정할 수 있으며, 삭제 버튼을 누르면 댓글이 삭제된다.</p>	
FRONT END	
<pre>{(user.user_type === 0 user.username === comment.username) && (<Box sx={[{ display: "flex", flexDirection: "row", alignItems: "center", width: "100%", },]} > <Button variant="span" sx={{ mr: 1, display: "inline" }} color="warning" onClick={() => { onEditReply(comment.content, comment.id); }} > 수정 </Button> <Button variant="span" sx={{ mr: 1, display: "inline" }} color="error" onClick={() => { onRemoveReply(comment.id); }} > 삭제 </Button>))}</pre>	
BACK END	
<pre>@PatchMapping("/comment-modify") public ResponseEntity<JsonResponse> modifyComment(@RequestBody CommentModifyDto commentModifyDto) { commentService.modifyComment(commentModifyDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); } @DeleteMapping("/comment-delete") public ResponseEntity<JsonResponse> deleteComment(@RequestBody CommentDeleteDto commentDeleteDto) { int id = commentService.deleteComment(commentDeleteDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, id)); }</pre>	
<p>댓글의 수정 및 삭제는 '/board/comment-modify', '/board/comment-delete'이고, 이런 경로로 요청을 하게 되면 각각의 controller가 동작하게 된다.</p> <p>먼저 댓글 수정의 경우에는</p>	

댓글 작성자와 현재 사용자를 비교해 현재 사용자가 쓴 댓글이라면 수정 및 삭제 버튼을 렌더링한다. 수정 버튼을 클릭하면 Prompt 창을 이용해 편집을 할 수 있으며, 확인 버튼을 누르면 입력한 결과를 백엔드 API에 전달하여 수정을 한다. 삭제 버튼을 누르면 삭제가 진행되며, 백엔드 API에 삭제 요청을 보낸다.

```
public void modifyComment(CommentDto commentModifyDto) {
    try {
        UserDto.userNameTypeDto userNameTypeDto = userRepository.findNameTypeByNameById(commentModifyDto.getUserId());
        if (userNameTypeDto.getUserNameType() != 0) {
            int user_id = commentRepository.getCommentWriterId(commentModifyDto.getId());
            if (user_id != commentModifyDto.getUserId()) {
                throw new GlobalException(ResponseStatus.INVALID_AUTHORITY_MODIFY_COMMENT);
            }
        }

        Comment comment = new Comment(
            commentModifyDto.getId(),
            0,
            0,
            0,
            commentModifyDto.getContent(),
            null,
            LocalDateTime.now()
        );
        commentRepository.modifyComment(comment);
    } catch (GlobalException e) {
        throw e;
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}
```

이 로직을 호출한 사용자의 user id를 이용하여 관리자이거나 댓글의 작성자인 경우에만 댓글 수정이 가능하도록 구현하였다.

댓글 삭제의 경우에도 유사하게

```
public void deleteComment(CommentDto commentDeleteDto) {
    try {
        UserDto.userNameTypeDto userNameTypeDto = userRepository.findNameTypeByNameById(commentDeleteDto.getUserId());
        if (userNameTypeDto.getUserNameType() != 0) {
            int realUser = commentRepository.getCommentWriterId(commentDeleteDto.getId());
            if (realUser != commentDeleteDto.getUserId()) {
                throw new GlobalException(ResponseStatus.INVALID_AUTHORITY_DELETE_COMMENT);
            }
        }

        commentRepository.deleteComment(commentDeleteDto.getId());
        return commentDeleteDto.getId();
    } catch (GlobalException e) {
        throw e;
    } catch (Exception e) {
        throw new GlobalException(ResponseStatus.DATABASE_ERROR);
    }
}
```

사용자의 user id를 이용하여 관리자이거나 댓글의 작성자인 경우에만 댓글 삭제가 가능하도록 구현하였다.

SQL

```
public Integer getCommentWriterId(int commentId) {
    return jdbcTemplate.queryForObject(
        "SELECT user_id FROM comment WHERE id=?",
        Integer.class,
        commentId
    );
}
```

```

public void modifyComment(Comment comment) {
    int id = comment.getId();
    String content = comment.getContent();
    LocalDateTime update_at = comment.getUpdated_at();

    jdbcTemplate.update("UPDATE `comment` SET " +
        "content=?," + 
        "updated_at=? " +
        "WHERE id=?"
        , content, update_at, id);
}

public void deleteComment(int id) {
    jdbcTemplate.update("DELETE FROM `comment` WHERE id=?",
        id);
}

```

윗 부분에서 설명한 `findNameTypeByNameById`의 경우 해당 요청을 한 사용자의 user type을 조회하기 위한 쿼리로, 코드 설명은 위에 존재한다. 이 쿼리는 권한 검사를 위해 사용자의 user type을 반환하게 된다.

다음으로 `getCommentWriterId`의 경우 해당하는 댓글의 작성자 `user_id`를 조회하기 위해 댓글의 `id`를 이용해 댓글의 `id`가 일치하는 댓글의 작성자 `user_id`를 조회한다.

`modifyComment`의 경우 `Update`를 이용하여 `comment table`에서 `id`가 현재 댓글 `id`와 일치하는 `comment`에 대한 `attribute` 값 수정을 진행한다.

`deleteComment`의 경우에는 `Delete`를 이용하여 `comment table`에서 `id`가 현재 댓글 `id`와 일치하는 `comment`에 대한, 해당하는 `row` 삭제를 진행한다.

8, 9, 10, 11. 게시물 및 댓글 추천(좋아요) 및 추천해제

SCREEN

너라는 데이터베이스에

작성자: 병준강 조회수: 16 작성일: 2023-12-03 04:10

“나라는 데이터를 저장하고 싶어.”

목록으로 이동
♥
4

푸바오 2023-12-03 04:16

미쳤다.. ♥ 3

하지만 2023-12-03 04:22

미쳤다.. ♥ 1

DESCRIPTION

각 게시물과 댓글 우측 하단에 좋아요 기능이 있으며, 좋아요를 누를 경우 좋아요 횟수가 올라가고 좋아요 개수가 늘어난다. 다시 클릭하면 좋아요가 취소되며, 좋아요 횟수가 줄어든다.

FRONT END
BACK END

```
<Box>
  <Like ? <
    <FavoriteIcon
      className={useAnimate && "heart-animate"}
      sx={[
        {mr: 1,
        cursor: "pointer",
        width: "100%",
        textAlign: "center",
        color: "#FF3040",
        }
      ]}
      onClick={postLikeHandler}
    />
  : <
    <FavoriteBorderOutlinedIcon
      sx={[
        {mr: 1,
        cursor: "pointer",
        width: "100%",
        textAlign: "center",
        }
      ]}
      onClick={postLikeHandler}
    />
  >
  <Typography sx={{ width: "100%", textAlign: "center" }}>{likeCount}</Typography>
</Box>
```

```
@PostMapping("/like")
public ResponseEntity<JsonResponse> likeBoard(@RequestBody BoardLikeDto boardLikeDto) {
  boardService.likeBoard(boardLikeDto);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}

@PostMapping("/comment/like")
public ResponseEntity<JsonResponse> likeComment(@RequestBody CommentLikeDto commentLikeDto) {
  commentService.likeComment(commentLikeDto);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}
```

위 Controller 코드의 경우, 위는 게시글의 좋아요, 아래는 댓글의 좋아요 controller이다. '/board/like'나 '/board/comment/like'로 요청하면 해당 controller 코드가 실행될 것이다.

먼저 게시글 좋아요의 경우 서비스 코드의 주요 부분은

```

<Box sx={{ py: 1.5 }}>
  {comment.userLiked ? (
    <FavoriteIcon
      className={useAnimate && "heart-animate"}
      sx={{
        mr: 1,
        width: "100%",
        textAlign: "center",
        cursor: "pointer",
        color: "#FF3040",
      }}
      onClick={() => postCommentLikeHandler(comment.id)}
    />
  ) : (
    <FavoriteBorderOutlinedIcon
      sx={{
        mr: 1,
        width: "100%",
        textAlign: "center",
        cursor: "pointer",
      }}
      onClick={() => postCommentLikeHandler(comment.id)}
    />
  )}
<Typography sx={{ width: "100%", textAlign: "center" }}>{comment.likeCount}</Typography>
</Box>

```

하트모양 버튼을 렌더링하고, 클릭 시
백엔드에 좋아요 개수를 늘리는 API를
요청한다. 정상적으로 요청이 성공하면 하트
모양을 꽉 찬 하트로 변경하고, 이 때 다시
클릭시 좋아요 개수를 줄어들게 만드는 API를
요청하여 구현하였다.

```

public void likeBoard(BoardDto.BoardLikeDto boardLikeDto) {
  try {
    BoardLike boardLike = new BoardLike(
      boardLikeDto.getUserId(),
      boardLikeDto.getCategory_id(),
      boardLikeDto.getLiked_id()
    );
    boolean exists = boardRepository.findExistByBoardLike(boardLike);
    if(exists) {
      BoardDto.BoardDisplayInfo boardSimpleInfo = boardRepository.findBoardByUserId(boardLikeDto.getUserId());
      fcmService.sendBoardLikeMessage(userService.findUserEmailById(boardSimpleInfo.getUserId()), boardSimpleInfo.getTitle());
      boardRepository.insertBoardLike(boardLike);
    } else {
      boardRepository.deleteBoardLike(boardLike);
    }
  }
}

```

좋아요를 누른 사용자와 게시글 id의 조합을
이용하여 해당 조합이 이미 존재한다면
좋아요를 취소하고, 존재하지 않는다면
좋아요를 등록하게 된다. 이용자가 좋아요를
눌렀다면, 그 글의 작성자에게 fcm 알림을
전송하게 된다.

댓글 좋아요의 경우, 서비스 코드의 주요
부분은 아래와 같다.

```

public void likeComment(CommentDto.CommentLikeDto commentLikeDto) {
  try {
    CommentLike commentLike = new CommentLike(
      commentLikeDto.getUserId(),
      commentLikeDto.getLiked_id()
    );
    boolean exists = commentRepository.findExistByCommentLike(commentLike);
    if(exists) {
      CommentBoardTitleDto commentBoardTitleDto =
        commentRepository.getCommentUserIdAndBoardTitle(commentLikeDto.getLiked_id());
      fcmService.sendCommentLikeMessage(userService.findUserEmailById(commentBoardTitleDto.getUserId()),
        commentBoardTitleDto.getTitle());
      commentRepository.insertCommentLike(commentLike);
    } else {
      commentRepository.deleteCommentLike(commentLike);
    }
  }
}

```

이 부분을 보면, 좋아요를 누른 사용자와 댓글
id의 조합을 이용하여 해당 조합이 이미
존재한다면 좋아요를 취소하고, 존재하지
않는다면 좋아요를 등록하게 된다. 단, 이
때에도 이용자가 좋아요를 눌렀다면 그
댓글의 작성자에게 fcm 알림을 전송하게
된다.

SQL

```

public boolean findExistByBoardLike(BoardLike boardLike) {
  Boolean isExists = jdbcTemplate.queryForObject(
    "SELECT EXISTS ( " +
    "SELECT 1 FROM board_like " +
    "WHERE user_id = ? AND category_id = ? AND liked_id = ? " +
    ")", Boolean.class, boardLike.getUserId(), boardLike.getCategory_id(), boardLike.getLiked_id()
  );
}

```

```

public void insertBoardLike(BoardLike boardLike) {
  int user_id = boardLike.getUserId();
  int category_id = boardLike.getCategory_id();
  int liked_id = boardLike.getLiked_id();

  jdbcTemplate.update("INSERT INTO `board_like` (`user_id`, `category_id`, `liked_id`) VALUES (?,?,?)",

```

```
public void deleteBoardLike(BoardLike boardLike) {
    int user_id = boardLike.getUser_id();
    int category_id = boardLike.getCategory_id();
    int liked_id = boardLike.getLiked_id();

    jdbcTemplate.update("DELETE FROM `board_like` WHERE user_id=? AND category_id=? AND liked_id=?",
        new Object[]{user_id, category_id, liked_id});
}
```

```
public boolean findExistByCommentLike(CommentLike commentLike) {
    Boolean isExists = jdbcTemplate.queryForObject(
        "SELECT EXISTS ( " +
            "    SELECT 1 FROM comment_like " +
            "    WHERE user_id = ? AND liked_id = ? " +
        ") ", Boolean.class, commentLike.getUser_id(), commentLike.getLiked_id());

    return isExists;
}
```

```
+ usage  ▾ 공지판
public void insertCommentLike(CommentLike commentLike) {
    int user_id = commentLike.getUser_id();
    int liked_id = commentLike.getLiked_id();

    jdbcTemplate.update("INSERT INTO `comment_like` (`user_id`, `liked_id`) VALUES (?,?)",
        new Object[]{user_id, liked_id});
}

+ 2 usages  ▾ 공지판
public void deleteCommentLike(CommentLike commentLike) {
    int user_id = commentLike.getUser_id();
    int liked_id = commentLike.getLiked_id();

    jdbcTemplate.update("DELETE FROM `comment_like` WHERE user_id=? AND liked_id=?",
        new Object[]{user_id, liked_id});
}
```

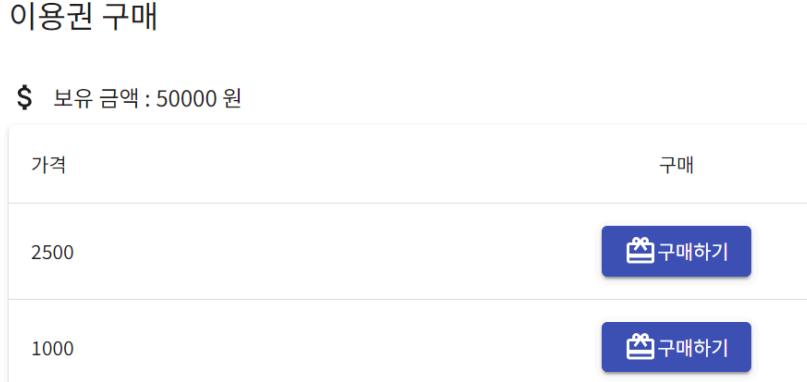
우선, `findExistByBoardLike`는 `EXISTS`를 이용해 `board_like` table에서 현재 사용자의 해당 게시글에 대한 좋아요가 존재하는지 여부를 반환한다. 이를 이용하여 이미 좋아요를 눌렀는지 판단하고 각각 다른 동작을 수행할 수 있다.

`insertBoardLike`의 경우에는 사용자가 해당 게시글에 좋아요를 누르지 않은 경우, 좋아요를 등록하기 위해 사용한다. `INSERT`를 이용해 `board_like` table에 `user_id`와 게시글 번호 등의 정보를 등록한다.

`deleteBoardLike`의 경우에는 사용자가 해당 게시글에 좋아요를 이 누른 경우, 좋아요를 삭제하기 위해 사용한다. `DELETE`를 이용해 `board_like` table에서 `user_id`와 게시글의 조합을 지운다.

댓글 좋아요의 경우에도 동작 자체는 게시글 좋아요와 유사한 것을 확인할 수 있다. `findExistsByCommentLike`는 사용자의 댓글 좋아요 여부 판단을, `insertCommentLike`는 `comment_like`에 댓글 좋아요 등록을, `deleteCommentLike`는 댓글 좋아요 삭제를 수행한다.

12. 이용권 구매

SCREEN	
	<p>이용권 구매는 따릉이를 이용할 수 있는 이용권을 구매하는 기능이다. 위에 표시되어 있는 보유금액을 이용하여 티켓을 구매할 수 있다. 현재 보유중인 이용권은 메인 페이지의 상단에서 이력 확인 - 이용권 이력에서 조회할 수 있다.</p>
FRONT END	BACK END
<pre>const buyTicketHandler = (id) => { (async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + `/users/purchase-ticket?userId=\${userId}&ticketId=\${id}`, { method: 'POST', headers: { 'Content-Type': 'application/json' }, }); if (response.status !== 200) { throw new Error('티켓 구매에 실패했습니다.'); } const jsonData = await response.json(); alert(`티켓 구매에 성공했습니다.`); for (let i = 0; i < ticketList.length; i++) { if (ticketList[i].id === id) { setMoney(money - ticketList[i].ticket_price); break; } } const response2 = await fetch(process.env.REACT_APP_API_URL + `/users/getuserinfo?user_id=\${cookies.id}`, { method: 'GET', headers: { 'Content-Type': 'application/json' }, }); if (response.status !== 200) { throw new Error('티켓 구매에 실패했습니다.'); } const jsonData2 = await response2.json(); dispatch(setUser(jsonData2.result)); } catch (e) { alert(`티켓 구매에 실패했습니다.`); } })(); };</pre> <p>백엔드 API에서 이용권 정보를 가져와 Material-UI 테이블에 렌더링하고 사용자가 티켓을 구매할 수 있도록 한다. 사용자의 보유금액을 표시하고 각 이용권의 옆에 구매버튼으로 티켓구매가 가능하도록 구현하였다.</p>	<pre>// 이용권 구매 @PostMapping("/users/purchase-ticket") public ResponseEntity<?> purchaseTicket(@RequestParam int userId, @RequestParam int ticketId){ boolean isTotalMoneyEnough = ticketService.purchaseTicket(userId, ticketId); if(!isTotalMoneyEnough) { fcmService.sendTicketPurchaseFailedMessage(userService.findUserEmailById(userId)); return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.FAILED_NOT_ENOUGH_TOTAL_MONEY, result: null)); } else { fcmService.sendTicketPurchaseSuccessMessage(userService.findUserEmailById(userId)); return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_PURCHASE_TICKET, result: null)); } }</pre> <p>이용권 구매는 프론트엔드에서 해당 유저에 대한 id와 특정 티켓에 대해 구매하기 버튼을 눌렀을 때 해당하는 티켓의 id를 바탕으로 /users/purchase-ticket API로 요청을 수행했을 때 잔액이 모자르다면 티켓 구매 실패 FCM 알림과 함께 실패 response를 반환하며, 잔액이 충분한 경우 티켓 구매 성공 FCM 알림과 함께 성공 response를 반환한다.</p>

```

public Boolean purchaseTicket(int userId, int ticketId) {
    return ticketRepository.purchaseTicket(userId, ticketId);
}

```

controller에서 user id와 ticket id를 인자로 넘겨준 ticketService의 purchaseTicket() 메서드에서는 ticketRepository()의 purchaseTicket()을 호출해 SQL 쿼리문으로 잔액 조회 후 구매 가능 여부 및 구매 이력을 갱신하도록 구현하였다.

SQL

```

@Transactional
public boolean purchaseTicket(int userId, int ticketId) {
    // Scalar Subqueries
    String priceQuery = "SELECT ticket_price FROM ticket " +
        "WHERE id = (SELECT t.id FROM ticket t WHERE t.id = ?)";
    Integer ticketPrice = jdbcTemplate.queryForObject(
        priceQuery, new Object[]{ticketId}, Integer.class);

    // BETWEEN
    String checkUserBalance = "SELECT COUNT(*) FROM user " +
        "WHERE id = ? AND total_money BETWEEN ? AND 99999999";
    Integer validUser = jdbcTemplate.queryForObject(
        checkUserBalance, new Object[]{userId, ticketPrice}, Integer.class);

    if (validUser == null || validUser == 0)
        return false;

    String updateUser = "UPDATE user SET total_money = total_money - ? WHERE id = ?";
    jdbcTemplate.update(updateUser, ticketPrice, userId);

    String insertHistory = String.format("INSERT INTO paymenthistory (" +
        "user_id, ticket_id, is_used, registration_at) VALUES (%d, %d, 0, '%s')",
        userId, ticketId, LocalDateTime.now().toString());
    jdbcTemplate.update(insertHistory);

    return true;
}

```

가장 먼저 ticketId에 해당하는 티켓 가격을 TICKET table에서 조회를 수행한다. 그리고 구매 가능 여부를 알기 위해 USER 테이블에서 해당 사용자의 잔액인 total_price가 앞서 조회한 티켓 가격 이상인지 확인을 진행한다. 사용자의 잔액이 충분하다면 티켓 가격만큼 잔액을 감소시켜서 USER table에 업데이트를 수행하고, 구매 이력을 기록하기 위해 PAYMENTHISTORY table에 해당 user의 티켓 구매 내역을 추가한다.

13. 쿠폰 등록

SCREEN	
DESCRIPTION	
<p>본 기능은 티켓을 구매하는 대신 쿠폰을 입력하는 것으로 티켓을 얻을 수 있다. 관리자가 등록한 쿠폰 코드를 입력하면, 쿠폰을 등록할 수 있고 쿠폰에 해당하는 티켓을 획득할 수 있다.</p>	<pre>// 사용자의 쿠폰 등록 @PostMapping("/users/registration-coupon") public ResponseEntity<Coupon> registrationCoupon(@RequestParam int userId, @RequestParam String value){ Integer ticketId = couponService.registrationCoupon(userId, value); if(ticketId == null) { return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.FAILED_REGISTRATION_COUPON_ALREADY_USED, null)); } else if(ticketId == -1) { return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.FAILED_REGISTRATION_COUPON_NOT_EXIST, null)); } return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_REGISTRATION_COUPON, null)); }</pre>
<pre>const registerCouponHandler = () => { (async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + `/users/registration-coupon?userId=\${userId}&value=\${couponCode}`); if(response.status === 200) { throw new Error('쿠폰 등록에 실패하였습니다.'); } const jsonData = await response.json(); if(jsonData.status === 200) { alert("쿠폰을 성공적으로 등록하였습니다."); return; } else { alert("쿠폰 등록에 실패하였습니다."); return; } } catch (e) { alert(`쿠폰 등록에 실패하였습니다.`); } })(); };</pre> <p>사용자에게 입력 가능한 쿠폰 화면을 제공해주고, 쿠폰 코드를 입력후 등록 버튼을 클릭하면 해당 쿠폰 코드를 백엔드 API에 전달해 쿠폰 유무를 확인한다. 그리고 결과를 Alert 메시지를 통해 전달한다.</p>	<p>프론트엔드 측에서 사용자가 쿠폰 코드를 입력한 값을 value로 파라미터를 넘겨서 어떤 유저가 어떤 쿠폰 코드를 입력했는지 /users/registration-coupon API로 요청을 수행한다. 이미 사용한 쿠폰이어서 쿠폰 등록을 실패했을 때, 일치하는 쿠폰 코드가 존재하지 않을 때의 예외처리를 수행해주었으며, 쿠폰코드가 일치함과 동시에 사용하지 않은 쿠폰이라면 사용자에게 성공적으로 등록이 수행되게 된다.</p>
SQL	

```

public Integer registrationCoupon(int userId, String value) {
    try {
        CouponDto couponDto = jdbcTemplate.queryForObject(
            sql: "SELECT * FROM coupon WHERE value = ?",
            new Object[]{value},
            new BeanPropertyRowMapper<>(CouponDto.class)
        );

        if (couponDto != null && couponDto.getIs_used() == 1) {
            return null; // 이미 사용한 쿠폰인 경우(is_used가 1인 경우)
        }

        jdbcTemplate.update(sql: "SET @userId = ?;", userId);

        // 쿠폰을 사용했으므로 is_used를 1로 변경
        String sqlUpdate = "UPDATE coupon SET is_used = 1 WHERE value = ?";
        jdbcTemplate.update(sqlUpdate, value);

        return couponDto.getTicket_id();
    } catch (EmptyResultDataAccessException e) {
        return -1; // 쿠폰의 value와 일치하는 쿠폰이 없는 경우
    }
}

```

가장 먼저 사용자가 입력한 쿠폰 코드인 `value`를 COUPON table에서 조회를 수행하여 객체를 얻는다. 만약 해당 객체가 `null` 이거나 객체는 존재하지만 사용한 쿠폰이라면 예외 사항이므로 `null`을 반환한다. 그리고 `@userId` 변수를 해당 사용자의 `userId`로 설정하고, 정상적으로 쿠폰을 사용했으므로 COUPON table에서 해당 쿠폰의 사용 상태인 `is_used`를 1로 바꿈으로써 사용한 쿠폰이라는 정보를 DB에 저장한다.

14, 15. 따릉이 대여 및 반납

SCREEN

도착 대여소

자전거 대여소 위치 [ST-2060] 서울특별시 노원구 석계…

자전거 대수: 3 평점: 4.5점

후기 보기

자전거 반납

DESCRIPTION

자전거를 직접 대여하고, 반납하는 과정을 간단한 버튼으로 시뮬레이션하였다. 지도에서

원하는 대여소를 클릭해 [자전거 대여] 버튼을 클릭하게 되면 자전거가 대여되며, 이후 도착 대여소를 지도에서 선택하고 [자전거 반납] 버튼을 클릭하게 되면 자전거가 반납된다.

FRONT END	BACK END
<pre>const onClickMarker = useCallback((marker, lendplace, userid) => { setIsOnRent((prevIsOnRent) => { if (prevIsOnRent) { getLendPlaceData(lendplace.lendplace_id, false, userid); } else { getLendPlaceData(lendplace.lendplace_id, true, userid); } return prevIsOnRent; }); (async () => { const lat = lendplace.startn_lat; const lon = lendplace.startn_lnt; const date = new Date().getTime(); const datestr = moment(date).format("YYYY-MM-DD"); console.log("=====lendplace"); console.log(lat, lon, lendplace); await fetchWeather(lat, lon, lendplace); })(); }, [startPos, endPos, isOnRent]);</pre>	<pre>// 자전거 대여 @PostMapping("/users/bike-rental") public ResponseEntity<?> bikeRental(@RequestBody UserlogDto userlogDto){ String response = userlogService.bikeRental(userlogDto); if(response.equals("FAILED_NO_VALID_TICKET")){ fcmService.sendBikeRentalFailedMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.FAILED_NO_VALID_TICKET, result: null)); } else if(response.equals("FAILED_INVALID_RENTAL_STATION")){ fcmService.sendBikeRentalFailedMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.FAILED_INVALID_RENTAL_STATION, result: null)); } else if(response.equals("FAILED_INVALID_BIKE")){ fcmService.sendBikeRentalFailedMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.FAILED_INVALID_BIKE, result: null)); } fcmService.sendBikeRentalSuccessMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS_BIKE_RENTAL, result: null)); } public String bikeRental(UserlogDto userlogDto) { return userlogRepository.bikeRental(userlogDto.getUserId(), userlogDto.getDeparture_station()); }</pre>
<p>카카오맵 마커를 클릭시 관련 정보를 백엔드에서 가져와 시작 대여소, 도착 대여소 위치를 가져오고, 자전거 대수 및 평점, 즐겨찾기 정보를 가져온다.</p> <pre>const onRentBike = (lendplace) => { (async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + "/users/bike-rental", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ departure_station: lendplace.lendplace_id, user_id: userid, }), credentials: "include", }); if (response.status != 200) { throw new Error("자전거 대여에 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status == 2026) { alert("자전거 대여에 성공하였습니다."); dispatch(setStartPos(lendplace)); _setStartPos(lendplace); setIsOnRent(true); return; } else { const errorMessage = jsonData.message; alert("자전거 대여 실패: " + errorMessage); // alert("자전거 대여에 실패하였습니다."); } } catch (error) { console.log(error); alert("자전거 대여에 실패하였습니다."); } })(); };</pre> <p>대여 버튼을 클릭하게 되면 백엔드에 대여했다는 정보를 보내며, 대여중인 상태로 변경된다.</p>	<pre>// 자전거 반납 @PostMapping("/users/bike-return") public ResponseEntity<?> bikeReturn(@RequestBody UserlogDto userlogDto){ String response = userlogService.bikeReturn(userlogDto); if(response.equals("FAILED_INVALID_RETURN_STATION")){ fcmService.sendBikeReturnFailedMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.FAILED_INVALID_RETURN_STATION, result: null)); } else if(response.equals("FAILED_OVER_MAX_STANDS")){ fcmService.sendBikeReturnFailedMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.FAILED_OVER_MAX_STANDS, result: null)); } fcmService.sendBikeReturnSuccessMessage(userService.findUserEmailById(userlogDto.getUserId())); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS_BIKE_RETURN, result: null)); } public String bikeReturn(UserlogDto userlogDto) { userlogDto.setArrival_time(LocalDateTime.now()); return userlogRepository.bikeReturn(userlogDto.getUserId(), userlogDto.getArrival_station(), userlogDto.getArrival_time(), userlogDto.getUser_distance()); }</pre> <p>첫 번째로, 따릉이 대여 기능에 대한 controller이다. 프론트엔드 측에서 대여 버튼을 클릭하게 되면 /users/bike-rental API로 요청이 수행되게 된다. 사용 가능한 이용권이 없어 대여가 불가능하거나, 해당 대여소가 비활성화 상태여서 대여가 불가능하거나, 해당 대여소에 이용 가능한 자전거가 없어서 대여가 불가능한 상황이 발생했을 때 FCM으로 자전거 대여 실패 알림이 발생하게 된다. 예외처리를 수행한 경우에 대해서는 자전거 대여에 성공한다. 자전거 대여에 성공하게 되면 FCM으로 자전거 대여 성공 알림과 함께 자전거 대여에 성공구매 이력, 대여소 정보, 자전거, 대여 이력 table에 대한 업데이트가 일어나게 된다.</p> <p>두 번째로, 따릉이 반납 기능에 대한</p>

```

const onReturnBike = (lendplace) => {
  (async () => {
    try {
      const start_lat = startPos.startn_lat;
      const startn_int = startPos.startn_int;
      const end_lat = endPos.startn_lat;
      const endn_int = endPos.startn_int;

      // Calculate distance in meter
      const distance = Math.sqrt(Math.pow(start_lat - end_lat, 2) + Math.pow(startn_int - endn_int, 2)) * 100000;

      const response = await fetch(process.env.REACT_APP_API_URL + "/users/bike-return", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          arrivalStation: endPos.lendplace_id,
          user_id: user_id,
          use_distance: distance ? parseInt(distance) : 0,
        }),
        credentials: "include",
      });
      if (response.status === 200) {
        throw new Error("자전거 반납에 실패했습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.status === 202) {
        alert("자전거 반납에 성공하였습니다.");
        setIsRented(false);
        setStartPos(null);
        setEndPos(null);
        _setStartPos(null);
        _setEndPos(null);
        navigate("/returning");
      }
    } catch (error) {
      console.error(error);
    }
  })();
}

```

대여중 상태에선 도착 대여소 관련 항목만 렌더링이 되며, 도착 대여소를 선택 후 자전거 반납 버튼을 누르면 반납 정보를 백엔드 API를 통해 전달한다. 이때, 시작 장소의 위도와 경도, 도착 장소의 위도와 경도를 계산해 계산 거리를 백엔드에 보내며, 정상적으로 반영이 된 경우 리뷰 페이지로 리다이렉션을 진행한다.

controller이다. 프론트엔드 측에서 반납 버튼을 클릭하게 되면 /users/bike-return API로 요청이 수행되게 된다. 반납하려는 대여소가 비활성화 상태여서 반납이 불가능하거나, 해당 대여소의 최대 거치 가능 자전거 수를 초과한 상황이 발생했을 때 FCM으로 자전거 반납 실패 알림이 발생하게 된다. 예외처리를 수행한 경우에 대해서는 FCM으로 자전거 반납 성공 알림과 함께 자전거 반납에 성공한다. 자전거 반납에 성공하게 되면 구매 이력, 대여소 정보, 자전거, 대여 이력 table에 대한 업데이트가 일어나게 된다.

SQL

```

public String bikeRental(int userId, String departureStation) {
  Integer bike_id, history_id, station_status;

  try {
    // MIN, INNER JOIN
    history_id = jdbcTemplate.queryForObject(
      sql("SELECT MIN(ph.history_id) FROM paymenthistory ph " +
          "INNER JOIN user u ON ph.user_id = u.id " +
          "WHERE u.id = ? AND ph.is_used = 0",
      new Object[]{userId},
      Integer.class
    );
  } catch (EmptyResultDataAccessException e) {
    history_id = -1;
  }

  if (history_id == null || history_id == -1)
    return "FAILED_NO_VALID_TICKET";

  try {
    station_status = jdbcTemplate.queryForObject(
      sql("SELECT station_status FROM bikestationinformation WHERE lendplace_id = ?",
      new Object[]{departureStation},
      Integer.class
    );
  } catch (EmptyResultDataAccessException e) {
    station_status = 0;
  }
}

```

```

    }

    log.info("station_status={}", station_status);

    if (station_status == 0 || station_status == null)
        return "FAILED_INVALID_RENTAL_STATION";

    try {
        // NESTED QUERY
        bike_id = jdbcTemplate.queryForObject(
            sql: "SELECT id FROM (SELECT id FROM bike WHERE lendplace_id = ? " +
                "AND bike_status = 1 AND use_status = 0) AS AvailableBikes " +
                "ORDER BY id LIMIT 1",
            new Object[]{departureStation},
            Integer.class
        );
    } catch (EmptyResultDataAccessException e) {
        bike_id = null;
    }

    log.info("id={}, bike_id");
    if (bike_id == null)
        return "FAILED_INVALID_BIKE";
}

// 차전거 상태 및 결제 이력 업데이트
jdbcTemplate.update(
    sql: "UPDATE bike SET use_status = 1 WHERE id = ?",
    bike_id
);

jdbcTemplate.update(
    sql: "UPDATE paymenthistory SET is_used = 1 WHERE history_id = ?",
    history_id
);

// Userlog 테이블에 정보 삽입
jdbcTemplate.update(
    sql: "INSERT INTO userlog (user_id, bike_id, history_id, departure_station, " +
        "departure_time, return_status) VALUES (?, ?, ?, ?, ?, 0)",
    userId, bike_id, history_id, departureStation, LocalDateTime.now()
);

return "SUCCESS BIKE RENTAL";
}

```

```

public String bikeReturn(int userId, String arrivalStation, LocalDateTime arrivalTime, int useDistance) {

    Integer bike_id, history_id, station_status;
    int max_stands, cur_stands;

    try {
        station_status = jdbcTemplate.queryForObject(
            sql: "SELECT station_status FROM bikestationinformation WHERE lendplace_id = ?",
            new Object[]{arrivalStation},
            Integer.class
        );
    } catch (EmptyResultDataAccessException e) {
        station_status = 0;
    }

    log.info("station_status={}", station_status);

    if (station_status == 0 || station_status == null)
        return "FAILED_INVALID_RETURN_STATION";

    String sql = "SELECT COUNT(*) FROM bike WHERE lendplace_id = ?";
    cur_stands = jdbcTemplate.queryForObject(sql, new Object[]{arrivalStation}, Integer.class);

    // MAX
    String getMaxStandsSql = "SELECT MAX(max_stands) FROM bikestationinformation";
    max_stands = jdbcTemplate.queryForObject(getMaxStandsSql, Integer.class);
}

```

```

    if (cur_stands >= max_stands)
        return "FAILED_OVER_MAX_STANDS";

LocalDateTime departure_time;

try {
    departure_time = jdbcTemplate.queryForObject(
        sql: "SELECT ul.departure_time FROM userlog ul "
            + "WHERE ul.user_id = ? AND ul.return_status = 0 "
            + "ORDER BY ul.departure_time DESC LIMIT 1",
        new Object[]{userId},
        LocalDateTime.class
    );
} catch (EmptyResultDataAccessException e) {
    departure_time = null;
}

LocalDateTime arrival_time = arrivalTime;
Duration duration = Duration.between(departure_time, arrival_time);
long use_time = duration.getSeconds() / 60;

// userlog table에서 user_id에 해당하는 유저의 return_status가 0인 bike_id를 반환하여 bike_id 가져오기
try {
    bike_id = jdbcTemplate.queryForObject(
        sql: "SELECT ul.bike_id FROM userlog ul "
            + "WHERE ul.user_id = ? AND ul.return_status = 0 "
            + "ORDER BY ul.departure_time DESC LIMIT 1",
        new Object[]{userId},
        Integer.class
    );
} catch (EmptyResultDataAccessException e) {
    bike_id = null;
}

// userlog table에서 회원번호 user_id와 같은, 회원번호 bike_id와 bike_id가 같고,
// return_status가 0인 결제수 arrival_station에 회원번호 arrivalStation, arrival_time에 회원번호 user_id
// use_distance에 회원번호 useDistance, return_status는 0인 경우 1로 바꾸기
jdbcTemplate.update(
    sql: "UPDATE userlog SET arrival_station = ?, arrival_time = ?, use_time = ?, use_distance = ?, return_status = 1 "
        + "WHERE user_id = ? AND bike_id = ? AND return_status = 0",
    arrivalStation, arrivalTime, use_time, useDistance, userId, bike_id
);

// 지역 번호 bike_id와 일치하는 bike_table의 id에 해당하는 결제의 use_status를 1에서 0으로 바꾸고, bike_table의 lendplaced_id를 회원번호
jdbcTemplate.update(
    sql: "UPDATE bike SET use_status = 0, lendplace_id = ? WHERE id = ?",
    arrivalStation, bike_id
);

return "SUCCESS_BIKE_RETURN";
}

```

첫 번째로, 사용자의 따름이 대여 요청을 처리하는 쿼리들이다. PAYMENTHISTORY table에서 주어진 userID에 대해 is_used가 0으로써 아직 사용되지 않은 history_id에 대한 조회를 수행한 후에 BIKESTATIONINFORMATION table의 departuerStation에 해당하는 대여소의 상태에 대한 필드인 station_status를 조회한다. 앞서 얻은 attribute 값들을 바탕으로 자전거 중에서 가장 작은 id를 가진 자전거를 BIKE table에서 조회하여 해당 자전거의 사용 상태인 use_status를 1로 변경한 후에 history_id를 통해 해당 결제 이력에 대한 is_used를 1로 변경하여 결제한 티켓이 사용되었음을 업데이트한다. 마지막으로 USERLOG table에 사용자의 id, 자전거의 id, 출발 대여소, 출발 시간, 반환 상태를 의미하는 return_status를 0으로 설정하여 추가한다. 반납이 완료되었을 시에 return_status가 1로 업데이트 될 것이다.

두 번째로, 사용자의 따름이 반납 요청을 처리하는 쿼리들이다. BIKESTATIONINFORMATION table에서 arrivalStation 즉, 도착한 대여소에 대한 활성화 상태인 station_status를 확인한 후에 BIKE table에서 arrivalStation에 있는 현재 자전거 수인 cur_stands를 계산한다. BIKESTATIONINFORMATION table에서 모든 대여소의 최대 자전거 수인 max_stands를 조회해서 cur_stands와 비교를 통해 예외처리를 수행한다. 이제 로그에 대한 정보를 갱신해야 하므로 USERLOG table에서 주어진 userId에 대해 아직 자전거가 반납되지 않았음을 의미하는 return_status가 0일 때와 자전거의 최근 출발 시간인 departure_time을 같이 조회한다. 이제 조회한 출발 시간인 departure_time과 도착 시간인 arrival_time과의 차이인 총 사용 시간을 의미하는 use_time을 계산한다. USERLOG table에서 주어진 userId에 대해 반납되지 않은

자전거의 bike_id를 조회하고 userId를 사용하여 USERLOG table에서 해당하는 row에 대해 대여소, 도착 시간, 사용 시간, 사용 거리를 업데이트하고 반납 상태를 의미하는 return_status를 1로 설정한다. 마지막으로, 조회한 bike_id를 이용해 BIKE table에서 해당 자전거의 use_status를 0으로 변경하고 lendplace_id를 arrivalStation으로 업데이트 함으로써 현재 자전거를 사용하고 있지 않으며(반납 완료), 자전거의 위치를 도착한 대여소로 정보를 갱신해주었다.

16. 대여 이력 조회

SCREEN																				
<div style="border: 1px solid #ccc; padding: 10px; width: 100%;"> <p style="margin: 0;">이력 확인</p> <div style="display: flex; justify-content: space-around; margin-bottom: 5px;"> 대여 이력 이용권 이력 </div> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 20%;">출발지</th> <th style="width: 20%;">출발시간</th> <th style="width: 20%;">도착지</th> <th style="width: 20%;">도착시간</th> <th style="width: 20%;">이동 거리</th> </tr> </thead> <tbody> <tr> <td>서울특별시 노원구 광운로 20 광운 대학교</td> <td>2023-12-14 08:41:51</td> <td>동해문화예술관앞 서울특별시 노원 구 화랑로 429</td> <td>2023-12-14 08:42:34</td> <td>94m</td> </tr> <tr> <td>동해문화예술관앞 서울특별시 노원 구 화랑로 429</td> <td>2023-12-14 20:39:10</td> <td>서울특별시 노원구 서울시 노원구 월계로 42길 97</td> <td>2023-12-14 20:42:02</td> <td>356m</td> </tr> </tbody> </table> <div style="display: flex; justify-content: center; margin-top: 10px;"> 이전 1 다음 </div> </div>						출발지	출발시간	도착지	도착시간	이동 거리	서울특별시 노원구 광운로 20 광운 대학교	2023-12-14 08:41:51	동해문화예술관앞 서울특별시 노원 구 화랑로 429	2023-12-14 08:42:34	94m	동해문화예술관앞 서울특별시 노원 구 화랑로 429	2023-12-14 20:39:10	서울특별시 노원구 서울시 노원구 월계로 42길 97	2023-12-14 20:42:02	356m
출발지	출발시간	도착지	도착시간	이동 거리																
서울특별시 노원구 광운로 20 광운 대학교	2023-12-14 08:41:51	동해문화예술관앞 서울특별시 노원 구 화랑로 429	2023-12-14 08:42:34	94m																
동해문화예술관앞 서울특별시 노원 구 화랑로 429	2023-12-14 20:39:10	서울특별시 노원구 서울시 노원구 월계로 42길 97	2023-12-14 20:42:02	356m																
DESCRIPTION																				
<p>위 기능은 따릉이 자전거 대여 이력을 조회할 수 있는 기능이다. 따릉이로 처음 출발한 대여소와 출발 시간, 따릉이를 반납한 대여소와 도착시간 및 출발 대여소와 도착 대여소 사이의 이동 거리를 측정하여 table에 표시한다.</p>																				
FRONT END	BACK END																			
<pre>useEffect(() => { if (!user.id) { return; } const fetchData = async () => { try { const response = await fetch(`\${process.env.REACT_APP_API_URL}/get-userlog?userId=\${user.id}`, { method: "GET", headers: { "Content-Type": "application/json" }, }); if (response.ok) { throw new Error("데이터를 가져오는데 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status == 2029) { setShowData(true); setTotalPage(1); setTravelHistory([1]); } else { setTravelHistory(jsonData.result); setTotalPage(Math.ceil(jsonData.result.length / 10)); setShowData(jsonData.result.slice(0, 10)); } } catch (e) { // console.error("데이터를 가져오는데 실패하였습니다.", e); setError("데이터를 가져오는데 실패하였습니다."); } }; fetchData(); }, [user.id]);</pre>	<pre>// 해당 유저의 모든 로그 조회 @GetMapping("/get-userlog") public ResponseEntity<> getUserLog(@RequestParam int userId) { List<UserLog> userLog = userLogService.getUserLog(userId); if((userLog==null) (userLog.isEmpty())) return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_GET_USERLOG_EMPTY, null)); return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_GET_USERLOG, userLog)); }</pre> <p>프론트엔드 측에서 이력확인을 위해 해당 템으로 이동하면 해당 유저가 자전거를 대여한 모든 이력들을 반환하는 /get-userlog API이다. service 로직과 repository로부터 가져온 로그 정보들을 json 형식으로 List에 담아서 프론트엔드 측에 반환한다.</p>																			

백엔드 API로부터 사용자의 대여 이력 데이터를 가져와 테이블에 렌더링하며 ‘이전’과 ‘다음’ 버튼을 통해 페이지네이션을 구현하여 데이터를 페이지 별로 나누어서 조회할 수 있다. ‘moment’ library를 사용하여 날짜와 시간을 표시하였다.

```
public List<UserlogDto> getUserlog(int userId) {
    List<UserlogDto> userlogs = userlogRepository.getUserlog(userId);
    Userlog userlog = null;
    Iterator<Userlog> iterator = userlogs.iterator();
    while (iterator.hasNext()) {
        Userlog userlogTemp = iterator.next();
        if (userlogTemp.getDeparture_station() == null) {
            userlogTemp.setDeparture_station_statn_addr1(userlogRepository.getDeparture_station_statn_addr1(userlogTemp.getDeparture_station()));
            userlogTemp.setDeparture_station_statn_addr2(userlogRepository.getDeparture_station_statn_addr2(userlogTemp.getDeparture_station()));
        }
        if (userlogTemp.getArrival1_station() == null) {
            userlogTemp.setArrival1_station_statn_addr1(userlogRepository.getArrival1_station_statn_addr1(userlogTemp.getArrival1_station()));
            userlogTemp.setArrival1_station_statn_addr2(userlogRepository.getArrival1_station_statn_addr2(userlogTemp.getArrival1_station()));
        }
    }
    return userlogs;
}
```

userlogRepository에서 userId를 넘겨 해당 id를 가진 유저의 log들을 모두 반환받는다. 만약 로그가 존재하지 않다면 null로 예외처리를 수행해주었으며, 대여 종이어서 log에 빈 필드가 존재할 수도 있으므로 이에 대한 처리 후에 log들을 반환한다.

SQL

```
public List<UserlogDto> getUserlog(int userId) {
    String sql = "SELECT * FROM userlog WHERE user_id = ?";
    List<UserlogDto> userlogs = jdbcTemplate.query(sql, new Object[]{userId}, new UserlogRowMapper());
    return userlogs.isEmpty() ? null : userlogs;
}
```

USERLOG table에서 userId에 해당하는 모든 log에 대한 record를 반환한다. 만약 해당 유저가 log를 가지고 있지 않아 userlogs가 비어 있다면 null을 반환하고, 존재하면 userlogs를 반환하여 SQL 쿼리 결과에 대한 예외처리를 수행해주었다.

17. 이용권 이력 조회

SCREEN																	
<table border="1"> <thead> <tr> <th>티켓 번호</th> <th>구매 시간</th> <th>사용 여부</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>2023-12-14 08:34:24</td> <td>사용 안료</td> </tr> <tr> <td>1</td> <td>2023-12-14 08:35:44</td> <td>사용 안료</td> </tr> <tr> <td>1</td> <td>2023-12-14 08:35:56</td> <td>사용 전</td> </tr> <tr> <td>1</td> <td>2023-12-14 08:36:39</td> <td>사용 전</td> </tr> </tbody> </table> <p style="text-align: center;">[이전] [1] [다음]</p>			티켓 번호	구매 시간	사용 여부	1	2023-12-14 08:34:24	사용 안료	1	2023-12-14 08:35:44	사용 안료	1	2023-12-14 08:35:56	사용 전	1	2023-12-14 08:36:39	사용 전
티켓 번호	구매 시간	사용 여부															
1	2023-12-14 08:34:24	사용 안료															
1	2023-12-14 08:35:44	사용 안료															
1	2023-12-14 08:35:56	사용 전															
1	2023-12-14 08:36:39	사용 전															
DESCRIPTION																	
<p>위 기능은 사용자가 자신이 구매한 이용권의 내역을 확인할 수 있는 기능이다. 해당 이용권에 대하여 이용권을 구매한 시간과 이용권의 사용여부를 함께 조회할 수 있다. 가장 상단에는 가장 최근에 구매한 이용권이 보여진다.</p>																	

FRONT END	BACK END
<pre> useEffect(() => { const [, setTicket] = useState(null); const [, setTicketHistory] = useState(null); const response = await fetch(process.env.REACT_APP_API_URL + '/users/get-ticket/payment-history?userId=\${user.id}', { method: "GET", headers: { "Content-Type": "application/json" }, }); if (response.status === 200) { throw new Error("데이터를 가져오는데 실패하였습니다."); } else { const jsonData = await response.json(); if (jsonData.status === 200) { setShowData(true); setTotalPage(1); setTicketHistory([]); } else { setTicketHistory(jsonData.result); setTotalPage(Math.ceil(jsonData.result.length / 10)); setShowData(jsonData.result.slice(0, 10)); } } catch (e) { alert("데이터를 가져오는데 실패하였습니다."); } }, []); </pre> <p>백엔드 API를 통해 사용자의 티켓 구매 이력을 가져와 테이블에 렌더링한다. 티켓번호, 구매시간, 사용 여부를 표시하며 '이전', '다음' 버튼을 이용한 페이지네이션으로 데이터를 페이지별로 나누어서 확인할 수 있다.</p>	<pre> // 이용권 구매 이력 조회 @GetMapping("/users/get-ticket/payment-history") public ResponseEntity<PaymentHistoryDto> purchaseTicket(@RequestParam int userId){ List<PaymentHistoryDto> paymentHistories = paymentHistoryService.getAllPaymentHistory(userId); if(paymentHistories.isEmpty()) return ResponseEntity.ok(new JsonResponse<ResponseStatus>(ResponseStatus.SUCCESS_GET_ALL_PAYMENT_HISTORY_INFO, null)); else return ResponseEntity.ok(new JsonResponse<ResponseStatus>(ResponseStatus.SUCCESS_GET_ALL_PAYMENT_HISTORY_INFO, paymentHistories)); } public List<PaymentHistoryDto> getAllPaymentHistory(int userId) { return paymentHistoryRepository.returnAllPaymentHistories(userId); } </pre> <p>프론트엔드 측에서 해당 유저의 모든 티켓 구매 이력을 조회할 때 요청하는 /users/get-ticket/payment-history API이다. 유저의 cookie에 저장되어 있는 해당 user의 id를 통해 paymentHistoryService의 getAllPaymentHistory() 메서드의 인자로 넘겨주어 얻은 결과를 반환한다. 만약 티켓 구매 이력이 isEmpty()가 true가 되어 비어있다면 해당 유저의 티켓 구매 내역 정보가 없다는 예외를 던지고, 만약 하나 이상 존재하면 모든 티켓 구매 이력을 반환한다.</p>
SQL	
<pre> // 이용권 구매 이력 조회 public List<PaymentHistoryDto> returnAllPaymentHistories(int userId) { String sql = "SELECT * FROM paymenthistory WHERE user_id = ?"; return jdbcTemplate.query(sql, new Object[]{userId}, BeanPropertyRowMapper.newInstance(PaymentHistoryDto.class)); } </pre> <p>PAYMENTHISTORY table에서 특정 사용자의 ID에 해당하는 모든 티켓 구매 이력에 대한 record를 조회하여 얻은 결과를 PaymentHistoryDto 객체의 리스트로 변환 후에 반환을 진행한다.</p>	

18, 19, 20, 21. 관리자 기능 - 이용권 관리 (이용권 조회, 수정, 삭제, 추가)

SCREEN

관리자 기능

- 대시보드
- 대여 이력
- 이용권 관리
- 회원 관리
- 따릉이 보관소 관리
- 자전거 관리
- 쿠폰 관리

번호	가격	수정	삭제
1	2500	<button style="border: 1px solid #007bff; padding: 2px 10px;">수정</button>	<button style="border: 1px solid #dc3545; background-color: #fff; color: #dc3545; padding: 2px 10px;">삭제</button>
2	1000	<button style="border: 1px solid #007bff; padding: 2px 10px;">수정</button>	<button style="border: 1px solid #dc3545; background-color: #fff; color: #dc3545; padding: 2px 10px;">삭제</button>

[추가](#)

이전
1
다음

티켓 번호

티켓 가격

[수정하기](#)
[뒤로가기](#)

티켓 정보 추가

티켓 가격

[추가하기](#)
[뒤로가기](#)

DESCRIPTION

위 기능은 관리자가 따릉이를 대여할 수 있는 이용권을 관리할 수 있는 기능이다. 어떤 종류의 이용권이 있는지 조회할 수 있고 그 이용권의 가격과 이용권 번호를 함께 볼 수 있다. 이용권의 정보를 수정하거나 삭제, 추가할 수 있다.

FRONT END

BACK END

백엔드 API를 통해 이용권 정보를 가져와 테이블로 렌더링하고 이전/다음 버튼을 이용한 페이지네이션을 통해 더 많은 데이터를 볼 수 있도록 구현하였다. 이용권 정보에는 이용권 ID와 가격이 있다. 관리자는 각 이용권별로 옆에 ‘수정’, ‘삭제’ 버튼을 이용할 수 있다.

```

const submitHandler = () => {
  (async() => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/admin/modify-ticket",
        {
          method: "PATCH",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            id: ticketId,
            ticket_price: ticketPrice,
          })
        }
      );
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (!jsonData.success) {
        alert("수정에 성공하였습니다.");
        navigate("/admin?type=ticket");
      } else {
        throw new Error("수정에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("수정에 실패했습니다.");
    }
  })();
  return;
}

```

수정 버튼을 누르면 별도의 수정 페이지로 이동하게 되며, 이 때 금액을 수정하여 반영을

```

// 모든 이용권 종류 조회(유저는 관리자 권한)
@GetMapping("/get-all-ticket")
public ResponseEntity<> getAllTicket() {
  List<TicketDto> tickets = ticketService.getAllTicket();
  if(tickets.isEmpty())
    return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_GET_ALL_TICKETS_INFO_ISEMPTY, result: null));
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_GET_ALL_TICKETS_INFO, tickets));
}

public List<TicketDto> getAllTicket() {
  return ticketRepository.returnAllTickets();
}

```

현재 구매 가능한 모든 이용권의 종류를 조회하는 `/get-all-ticket` API이다. `ticketService`의 `getAllTicket()` 메서드를 호출하여 모든 `ticket`들의 종류를 `TicketDto`에 대한 `List`로 반환한다. 만약 생성되어 있는 티켓이 하나도 없어서 `tickets`가 `null`인 상황이면 저장된 티켓의 정보가 하나도 없다는 예외를 발생시킨다. 티켓이 하나 이상 존재하여 `tickets`가 `empty`가 아니라면 반환함으로써 모든 이용권의 종류를 반환한다.

할 수 있다. 금액을 수정후 수정하기 버튼을 클릭하면 백엔드 API에 수정할 이용권의 ID과 금액이 반영된다.

```
const submitHandler = () => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/admin/create-ticket", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          ticket_price: ticketPrice,
        }),
      });
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("성공적으로 추가되었습니다.");
        navigate("/admin?type=ticket");
      } else {
        throw new Error("추가에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("추가에 실패했습니다.");
    }
  })();
  return;
};
```

수정 버튼을 누르면 별개의 추가 페이지로 이동하며, 이때 금액을 지정한 후 새로운 이용권을 만들 수 있다. 금액을 지정하면 백엔드 API에 이용권의 가격이 전달이 되며, 정상적으로 요청이 성공하였을 경우 티켓 목록을 보는 화면으로 리다이렉션한다.

```
// 이용권 종류 정보 수정
@PatchMapping("/admin/modify-ticket")
public ResponseEntity<> modifyTicket(@RequestBody TicketDto ticketDto) {
  ticketService.modifyTicket(ticketDto.getId(), ticketDto.getTicket_price());
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS MODIFY TICKET, result: null));
}
```

```
public void modifyTicket(int id, int newTicketPrice) {
  ticketRepository.modifyTicket(id, newTicketPrice);
}
```

현재 존재하는 이용권의 정보를 수정하는 /admin/modify-ticket API이다. ticketService의 modifyTicket() 메서드에 변경할 이용권의 id와 변경할 가격을 인자로 넘겨주어 ticketRepository에 넘겨서 SQL 쿼리를 통해 변경을 적용하게 된다. 따라서 특정 이용권의 금액을 변경하는 로직을 수행하게 되는 것이다.

```
// 이용권 종류 정보 삭제
@DeleteMapping("/admin/delete-ticket")
public ResponseEntity<> deleteTicket(@RequestBody TicketDto ticketDto) {
  ticketService.deleteTicket(ticketDto.getId());
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_DELETE TICKET, result: null));
}
```

```
public void deleteTicket(int id) {
  ticketRepository.deleteTicket(id);
}
```

현재 존재하는 이용권을 삭제하는 /admin/delete-ticket API이다. ticketService의 deleteTicket() 메서드에 삭제할 이용권의 id를 인자로 넘겨주어 ticketRepository에 넘겨서 SQL 쿼리를 통해 해당 이용권에 대한 삭제를 진행하게 된다.

```
// 이용권 종류 추가
@PostMapping("/admin/create-ticket")
public ResponseEntity<> createTicket(@RequestBody TicketDto ticketDto) {
  ticketService.createTicket(ticketDto.getTicket_price());
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_CREATE TICKET, result: null));
}
```

```
public void createTicket(int ticketPrice) {
  ticketRepository.createTicket(ticketPrice);
}
```

새로운 금액을 가진 새로운 이용권의 종류를 추가하는 /admin/create-ticket API이다. ticketService의 createTicket() 메서드에 새로 추가할 이용권의 금액을 인자로 넘겨서 SQL 쿼리를 통해 새로운 금액으로 이용권을 생성하게 된다.

SQL

```
public List<TicketDto> returnAllTickets() {  
    String sql = "SELECT * FROM ticket";  
    return jdbcTemplate.query(  
        sql,  
        BeanPropertyRowMapper.newInstance(TicketDto.class)  
    );  
}  
  
public void modifyTicket(int id, int newTicketPrice) {  
    String sql = "UPDATE ticket SET ticket_price = ? WHERE id = ?";  
    jdbcTemplate.update(sql, newTicketPrice, id);  
}  
  
public void deleteTicket(int id) {  
    String sql = "DELETE FROM ticket WHERE id = ?";  
    jdbcTemplate.update(sql, id);  
}  
  
public void createTicket(int ticket_price) {  
    String sql = "INSERT INTO ticket (ticket_price) VALUES (?)";  
    jdbcTemplate.update(sql, ticket_price);  
}
```

첫 번째로 모든 종류의 티켓을 반환하는 쿼리이다. TICKET table에서의 모든 속성을 TicketDto의 필드와 매핑시켜서 반환을 수행한다.

두 번째로 특정 티켓에 대한 수정을 진행하는 쿼리이다. 가격을 수정할 수 있으므로 인자로 받은 가격에 대한 정보를 주어진 티켓 id에 대해 적용함으로써 가격에 대한 업데이트를 수행한다.

세 번째로 특정 티켓에 대한 삭제를 진행하는 쿼리이다. TICKET table에서 특정 ticket id를 가진 ticket을 delete를 수행하여 해당 티켓에 대한 정보를 삭제한다.

네 번째로 새로운 티켓을 생성하는 쿼리이다. TICKET table에서 인자로 받은 가격에 대한 정보를 바탕으로 테이블에 삽입하여 새로운 티켓을 생성한다.

22, 23, 24, 25. 관리자 기능 - 자전거 관리 (자전거 조회, 자전거 생성, 수정, 삭제)

SCREEN

관리자 기능					
	대여소 위치	사용 여부	자전거 상태	수정	삭제
0	대여소 위치	사용 여부	자전거 상태	<button>수정</button>	<button>삭제</button>
56	ST-10	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>
55	ST-2074	미사용중	사용 불가능	<button>수정</button>	<button>삭제</button>
53	ST-1280	미사용중	사용 불가능	<button>수정</button>	<button>삭제</button>
52	ST-2232	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>
51	ST-914	미사용중	사용 불가능	<button>수정</button>	<button>삭제</button>
50	ST-3327	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>
49	ST-2731	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>
48	ST-913	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>
47	ST-2731	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>
46	ST-3193	미사용중	사용 가능	<button>수정</button>	<button>삭제</button>

[이전 | 1 | 다음]

자전거 정보 추가

자전거 대여소 위치

자전거 번호

사용 여부

자전거 상태

생성하기 뒤로가기

수정하기 뒤로가기

DESCRIPTION

이 기능은 관리자 기능 중 하나로, 자전거에 대한 정보를 확인하고 추가 및 편집할 수 있는 기능이다. 자전거 관리 페이지에 들어가면 현재 존재하는 모든 자전거에 대한 정보를 확인할 수 있으며, 추가 버튼을 통해 자전거를 생성, 수정 버튼을 통해 자전거의 대여소 위치, 사용 여부, 상태 등을 수정, 삭제를 통해 자전거를 삭제할 수 있다.

FRONT END

```
useEffect(() => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/bike/get-all", {
        method: "GET",
        headers: { "Content-Type": "application/json" },
      });
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      setBikeData(jsonData.result);
      setTotalPage(parseInt(jsonData.result.length / 10) + 1);
      setShowData(jsonData.result.slice(0, 10));
    } catch (error) {
      alert("데이터를 가져오는데 실패하였습니다.");
    }
  })();
});
```

BACK END

```
@PostMapping("/create")
public ResponseEntity<JsonResponse> createBike(@RequestBody BikeDto bikeCreateDto) {
  bikeService.createBike(bikeCreateDto);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}

no usages. ↳ 빙자됨
@GetMapping("/get-all")
public ResponseEntity<JsonResponse> getAllBike() {
  List<Bike> bikeAllDtoList = bikeService.getAllBike();
  if(bikeAllDtoList.isEmpty()) throw new GlobalException(ResponseStatus.RESULT_NOT_EXIST);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeAllDtoList));
}

no usages. ↳ 빙자됨
@PatchMapping("/modify")
public ResponseEntity<JsonResponse> modifyBike(@RequestBody BikeDto bikeModifyDto) {
  bikeService.modifyBike(bikeModifyDto);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}

no usages. ↳ 빙자됨
@DeleteMapping("/delete")
public ResponseEntity<JsonResponse> deleteBike(@RequestBody BikeDto bikeDeleteDto) {
  bikeService.deleteBike(bikeDeleteDto);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeDeleteDto.getId()));
}
```

자전거 관리 페이지로 들어가면 백엔드 API를 통해 가져온 자전거 정보를 테이블로 렌더링하고 페이지네이션을 통해 모든 등록되어 있는 자전거 정보를 볼 수 있다. 자전거의 ID, 현재 자전거가 있는 대여소의 위치, 사용 여부, 자전거 상태에 대한 정보를 조회할 수 있으며 관리자는 각 자전거 옆에 있는 버튼을 통해 수정 혹은 삭제, 생성을 할 수 있다.

자전거 조회, 생성, 수정, 삭제 기능이 존재해야 한다.

따라서 순서대로 생성, 조회, 수정, 삭제이며 프론트에서 각 보이는 경로로 요청을 하면 해당하는 컨트롤러 코드가 실행된다. 우선 자전거 생성을 하고자 하는 주요 코드를 살펴보면

```
public void createBike(BikeDto.BikeCreateDto bikeCreateDto) {
  try {
    Bike bike = new Bike(
      0, //auto increment id
      bikeCreateDto.getLendPlace_id(),
      bikeCreateDto.getUse_status(),
      bikeCreateDto.getBike_status()
    );
    bikeRepository.insertBike(bike);
  }
```

관리자가 입력한 자전거 정보의 형태를 변환하고, 이를 database에 저장할 수 있도록

```

const submitHandler = () => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/bike/modify", {
        method: "PATCH",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          id: bikeId,
          lendplace_id: lendplaceId,
          use_status: useStatus ? 1 : 0,
          bike_status: bikeStatus ? 1 : 0,
        }),
      });
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("수정에 성공하였습니다.");
        navigate("/admin?type=bike");
      } else {
        throw new Error("수정에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("수정에 실패했습니다.");
    }
  })();
  return;
};

```

수정 버튼을 누르면 별도의 수정 페이지로 이동해 자전거의 정보를 가져와 화면에 렌더링한다. 대여소 위치와 사용중 여부, 자전거 상태를 수정할 수 있으며 수정하기 버튼을 누르면 해당 정보를 백엔드 API에 전달한다.

```

const submitHandler = () => {
  (async () => {
    try {
      const response = await fetch(process.env.REACT_APP_API_URL + "/bike/create", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          lendplace_id: lendplaceId,
          use_status: 0,
          bike_status: 1,
        }),
      });
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("생성에 성공하였습니다.");
        navigate("/admin?type=bike");
        mirusu400, 2주 전 · feat: Admin 페이지 수정
      } else {
        throw new Error("생성에 실패했습니다.");
      }
    } catch (error) {
      console.log(error);
      alert("생성에 실패했습니다.");
    }
  })();
  return;
};

```

추가 버튼을 누르면 추가 페이지로 이동한다. 이때는 대여소의 위치만 지정하며 생성하기 버튼을 누르면 해당 대여소 위치에 자전거가 한개 생성되도록 백엔드 API에 요청을 전송한다. 정상적으로 수정이 완료되었다면 자전거 관리 페이지로 리다이렉션된다.

쿼리를 호출한다.

다음으로 자전거 정보 조회는 모든 자전거를 한번에 조회할 수 있도록 하며,

```

1 usage ▲ 방지인
` public List<Bike> getAllBike() {
    try {
        return bikeRepository.findAll();
    }
}
```

이 또한 쿼리를 호출하여 결과를 받고, 그 결과를 다시 service code를 호출한 곳에 전달한다.

자전거 정보 수정과 삭제의 경우에도

```

1 usage ▲ 방지인
` public void modifyBike(BikeDto.BikeModifyDto bikeModifyDto) {
    try {
        bikeRepository.update(bikeModifyDto);
    }
} catch (Exception e) {
    throw new GlobalException(ResponseStatus.DATABASE_ERROR);
}

1 usage ▲ 방지인
public void deleteBike(BikeDto.BikeDeleteDto bikeDeleteDto) {
    try {
        bikeRepository.delete(bikeDeleteDto.getId());
    }
} catch (Exception e) {
    throw new GlobalException(ResponseStatus.DATABASE_ERROR);
}
}
```

단순하게, 자전거 정보를 수정하고 삭제하는 repository의 코드만 호출하도록 간단하게 구현하였다.

SQL

```

public void insertBike(Bike bike) {
    jdbcTemplate.update("INSERT INTO `bike` (`lendplace_id`, `use_status`, `bike_status`) VALUES (?, ?, ?)"
        , bike.getLendplace_id(), bike.getUse_status(), bike.getBike_status());
}

public List<Bike> findAll() {
    var bikeMapper = BeanPropertyRowMapper.newInstance(Bike.class);

    return jdbcTemplate.query(
        "SELECT * FROM bike " +
        "ORDER BY id DESC", bikeMapper
    );
}

public void update(BikeDto.BikeModifyDto bikeModifyDto) {
    jdbcTemplate.update("UPDATE `bike` SET "
        + "lendplace_id=?," +
        "use_status= ?, " +
        "bike_status=? " +
        "WHERE id=?"
        , bikeModifyDto.getLendplace_id(), bikeModifyDto.getUse_status(), bikeModifyDto.getBike_status(), bikeModifyDto.getId());
}

// 공지판
■ public void delete(int id) { jdbcTemplate.update("DELETE FROM `bike` WHERE id=?", id); }

```

`insertBike`는 `INSERT`를 이용해 관리자가 추가하고자 입력한 자전거 정보를 `bike table`에 저장한다.

`findAll`은 기본적인 `SELECT`를 이용해 `bike table`의 모든 정보를 조회하는데, 이때 자전거의 `id`가 큰 순서, 즉 최근에 생성된 순서대로 반환한다.

`update`는 `UPDATE`를 이용해 `bike table`에서 자전거 `id`가 일치하는 자전거 정보를 관리자가 입력한 정보로 변경한다.

`delete`는 `DELETE`를 이용해 `bike table`에서 해당하는 `id`의 자전거 정보를 삭제한다.

26, 27, 28, 29. 관리자 기능 - 쿠폰 관리(쿠폰 조회, 수정, 삭제, 발급)

SCREEN

번호	이름	가격	사용 여부	수정	삭제
1	교수님의은총	2500	사용됨	<button>수정</button>	<button>삭제</button>
1	꾸뿐	2500	사용됨	<button>수정</button>	<button>삭제</button>
1	따릉따릉	2500	사용됨	<button>수정</button>	<button>삭제</button>
1	따릉이2500	2500	사용됨	<button>수정</button>	<button>삭제</button>
1	따릉이쿠폰	2500	사용됨	<button>수정</button>	<button>삭제</button>
2		1000	사용안됨	<button>수정</button>	<button>삭제</button>

발급
이전
1
다음

쿠폰 발급

발급하기
뒤로가기

티켓 정보 수정

수정하기
뒤로가기

DESCRIPTION

위 기능은 관리자가 쿠폰에 대한 정보를 관리할 수 있는 기능이다. 관리자는 발급했던 모든 쿠폰들을 조회 할 수 있고 쿠폰의 정보에는 쿠폰 번호, 이름, 가격과 쿠폰이 사용되었는지의 여부가 있다. 수정 버튼을 통해 쿠폰의 정보를 수정할 수 있고 삭제 버튼을 통해 해당 쿠폰을 삭제할 수 있으며 사용자에게 새롭게 쿠폰을 발급할 수 있다.

FRONT END	BACK END
<pre>useEffect(() => { // TODO: Fetch ticket data (async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + "/get-all-coupon", { method: "GET", headers: { "Content-Type": "application/json" }, }); if (response.status !== 200) { throw new Error("데이터를 가져오는데 실패하였습니다."); } const jsonData = await response.json(); setTicketData(jsonData.result); setTotalPage(parseInt(jsonData.result.length / 10) + 1); setShowData(jsonData.result.slice(0, 10)); } catch (error) { alert("데이터를 가져오는데 실패하였습니다."); } })(); }, []);</pre>	<pre>// 코드 구조 표기 @GetMapping("/get-all-coupon") public ResponseEntity<JoinTicketDto> getAllCoupon(){ List<CouponFullOuterJoinTicketDto> allCoupon = couponService.getAllCoupon(); if(allCoupon == null allCoupon.isEmpty()) return ResponseEntity.ok(new JsonResponse<>().setStatus(ResponseStatus.SUCCESS_GET_ALL_COUPON_EMPTY, "result: null")); return ResponseEntity.ok(new JsonResponse<>().setStatus(ResponseStatus.SUCCESS_GET_ALL_COUPON, allCoupon)); } public List<CouponFullOuterJoinTicketDto> getAllCoupon() { return couponRepository.getAllCoupon(); }</pre>
<p>백엔드 API를 통해 쿠폰정보를 가져와</p>	<p>현재 생성되어 있는 모든 쿠폰을 조회하는 /get-all-coupon API이다. couponService의 getAllCoupon() 메서드를 호출하여 모든 coupon들의 종류를 List에 반환한다. 만약</p>

테이블로 렌더링하고 이전/ 다음 버튼을 이용한 페이지네이션을 통해 더 많은 쿠폰 정보를 조회할 수 있다. 쿠폰 정보에는 쿠폰 ID, 가격, 해당 쿠폰 사용 여부등이 포함되어 있고 관리자는 각 쿠폰의 옆에 있는 ‘수정’, ‘삭제’ 버튼이 존재한다.

```
const submitHandler = () => {
  (async () => {
    try {
      console.log(ticketId, ticketPrice, isUsed, couponValue);
      const response = await fetch(process.env.REACT_APP_API_URL + "/modify-coupon", {
        method: "PATCH",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify({
          ticket_id: ticketId,
          // ticket_price: ticketPrice,
          // is_used: isUsed,
          value: couponValue,
        }),
      });
      if (response.status !== 200) {
        throw new Error("데이터를 가져오는데 실패하였습니다.");
      }
      const jsonData = await response.json();
      if (jsonData.success) {
        alert("수정에 성공하였습니다.");
        navigate("/admin?type=coupon");
      } else {
        throw new Error("수정에 실패했습니다.");
      }
    } catch (error) {
      alert("수정에 실패했습니다.");
    }
  })();
  return;
};
```

쿠폰 수정 버튼을 누르면 쿠폰을 수정할 수 있으며, 수정 완료 후 수정하기 버튼을 누르면 관련 정보가 백엔드 API로 전달이 된다.

```
const submitHandler = () => {
  (async () => {
    const response = await fetch(process.env.REACT_APP_API_URL + `/create-coupon`, {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({
        value: couponValue,
        ticket_id: ticketId,
      }),
    });
    if (response.status !== 200) {
      throw new Error("쿠폰 발급에 실패했습니다.");
    }
    if (response.status != 200) {
      throw new Error("쿠폰 발급에 실패했습니다.");
    }
    const jsonData = await response.json();
    if (jsonData.success) {
      alert("쿠폰을 발급하였습니다.");
      navigate("/admin?type=coupon");
    } else {
      throw new Error("쿠폰 발급에 실패했습니다.");
    }
  })();
  return;
};
```

발급하기 시에는, 쿠폰의 이름과 쿠폰의 가격을 명시할 수 있으며 발급하기 버튼을 누르면 이름과 쿠폰 정보가 백엔드 API에 전달되어 생성을 시도할 수 있다. 정상적으로 발급이 되었다면 쿠폰 관리 페이지로 리다이렉션된다.

생성되어 있는 쿠폰이 하나도 없어서 allCoupon이 null인 상황이면 저장된 쿠폰의 정보가 하나도 없다는 예외를 발생시킨다. 쿠폰이 하나 이상 존재하여 allCoupon이 empty가 아니라면 반환함으로써 모든 쿠폰의 종류를 반환한다.

```
// 쿠폰 수정
@PatchMapping("/modify-coupon")
public ResponseEntity<?> modifyCoupon(@RequestBody CouponDto couponDto){

  couponService.modifyCoupon(couponDto);
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_MODIFY_COUPON, result: null));
}

public void modifyCoupon(CouponDto couponDto) {
  couponRepository.modifyCoupon(couponDto);
}
```

현재 존재하는 쿠폰의 정보를 수정하는 /modify-coupon API이다. couponService의 modifyCoupon() 메서드에 변경할 값이 저장되어 있는 couponDto를 인자로 넘겨주어 couponRepository에 넘겨서 SQL 쿼리를 통해 변경을 전용하게 된다. 따라서 특정 이용권에 대한 쿠폰 코드를 변경하는 로직을 수행하게 되는 것이다.

```
// 쿠폰 삭제
@DeleteMapping("/delete-coupon")
public ResponseEntity<?> deleteCoupon(@RequestBody CouponDto couponDto){

  couponService.deleteCoupon(couponDto);
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_DELETE_COUPON, result: null));
}

public void deleteCoupon(CouponDto couponDto) {
  couponRepository.deleteCoupon(couponDto);
}
```

현재 존재하는 특정 쿠폰을 삭제하는 /delete-coupon API이다. couponService의 deleteCoupon() 메서드에 couponDto를 넘겨주면 삭제할 쿠폰의 id를 바탕으로 SQL 쿼리를 이용해 해당 쿠폰에 대한 삭제를 진행하게 된다.

```
// 쿠폰 추가
@PostMapping("/create-coupon")
public ResponseEntity<?> createCoupon(@RequestBody CouponDto couponDto){

  couponService.createCoupon(couponDto);
  return ResponseEntity.ok(new JsonResponse<>(ResponseStatus.SUCCESS_CREATE_COUPON, result: null));
}

public void createCoupon(CouponDto couponDto) {
  couponRepository.createCoupon(couponDto);
}
```

새로운 쿠폰을 생성하는 /create-coupon API이다. couponService의 createCoupon()

메서드를 호출하여 새로운 쿠폰 코드를
바탕으로 새로운 쿠폰을 생성하게 된다.

SQL

```
public List<CouponFullOuterJoinTicketDto> getAllCoupon() {
    // LEFT OUTER JOIN, UNION, RIGHT OUTER JOIN, FULL OUTER JOIN(LEFT OUTER JOIN+RIGHT OUTER JOIN)
    String sql = "SELECT * FROM coupon LEFT OUTER JOIN ticket ON coupon.ticket_id = ticket.id " +
        "UNION " +
        "SELECT * FROM coupon RIGHT OUTER JOIN ticket ON coupon.ticket_id = ticket.id";
    List<CouponFullOuterJoinTicketDto> coupons = jdbcTemplate.query(
        sql, BeanPropertyRowMapper.newInstance(CouponFullOuterJoinTicketDto.class));

    Iterator<CouponFullOuterJoinTicketDto> iterator = coupons.iterator();
    while (iterator.hasNext()) {
        CouponFullOuterJoinTicketDto coupon = iterator.next();
        if (coupon.getValue() == null) {
            iterator.remove();
        }
    }

    return coupons != null ? coupons : new ArrayList<>();
}
```

```
public void modifyCoupon(CouponDto couponDto) {
    String sql = "UPDATE coupon SET ticket_id = ? WHERE value = ?";
    jdbcTemplate.update(sql, couponDto.getTicket_id(), couponDto.getValue());
}
```

```
public void deleteCoupon(CouponDto couponDto) {
    String sql = "DELETE FROM coupon WHERE value = ?";
    jdbcTemplate.update(sql, couponDto.getValue());
}
```

```
public void createCoupon(CouponDto couponDto) {
    String sql = "INSERT INTO coupon (value, is_used, ticket_id) VALUES (?, ?, ?)";
    jdbcTemplate.update(sql, couponDto.getValue(), couponDto.getIs_used(), couponDto.getTicket_id());
}
```

첫 번째는 모든 쿠폰의 정보를 반환하는 SQL 쿼리이다. COUPON table과 TICKET table을 LEFT OUTER JOIN으로 결합하여 COUPON table의 모든 record와 일치하는 TICKET table의 record를 반환하고 COUPON과 TICKET을 RIGHT OUTER JOIN으로 결합하여 TICKET table의 모든 record와 일치하는 COUPON table의 레코드를 반환함으로써 두 결과를 UNION을 통해 합침으로써 중복을 제거한다. 즉, 중복 없이 모든 종류의 쿠폰에 대한 정보를 full outer join으로 반환하게 된다.

두 번째는 특정 쿠폰에 대한 정보를 수정하는 SQL 쿼리이다. COUPON table을 통해 수정할 ticket의 id를 기반으로 쿠폰 코드에 대한 변경을 진행하게 된다. 따라서 새로 바뀐 쿠폰 코드로 입력을 하지 않으면 등록을 할 수 없게 된다.

세 번째는 특정 쿠폰에 대한 삭제를 진행하는 SQL 쿼리이다. COUPON table에서 쿠폰 코드인 value에 해당하는 row를 삭제를 진행하게 된다.

네 번째는 새로운 쿠폰을 생성하는 SQL 쿼리이다. COUOPN table에 쿠폰 코드, 사용 여부, 티켓 ID 등의 정보를 바탕으로 새로운 쿠폰에 대한 정보를 삽입한다.

30, 31. 출발지 및 도착지 별점 및 리뷰 기능

SCREEN	
<p>별점</p> <p>여행은 즐거우셨나요? 즐겨우셨다면 리뷰를 남겨주세요! 😊</p> <p>출발지</p> <p>서울특별시 노원구 광운로 20 광운대학교</p> <p>★ ★ ★ ★ ★</p> <p>후기를 입력해주세요</p> <p>작성하기</p>	
DESCRIPTION	
위 기능은 따릉이를 사용 후 각 대여소에 대한 리뷰를 매길 수 있는 기능으로, 별점을 남기고 후기를 직접 입력할 수 있는 기능이다. 사용자가 직접 작성한 후기 및 다른 사용자가 작성한 후기는 즐겨찾기에 들어가서 대여소를 확인하거나 검색하여 평점과 후기를 볼 수 있다.	
FRONT END	BACK END

```

const onSubmitStart = (startRating, startComment) => {
  (async () => {
    const response = await fetch(process.env.REACT_APP_API_URL + "/rating/lendplace-review", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        user_id: user.id,
        lendplace_id2: startPosInfo.lendplace_id,
        rating2: startRating,
        review2: startComment,
      }),
    });
    if (response.status !== 200) {
      throw new Error("리뷰 작성에 실패했습니다.");
    }
    const jsonData = await response.json();
    if (!jsonData.success) {
      throw new Error("리뷰 작성에 실패했습니다.");
    }
    setIsStartSubmitted(true);
  }) catch (error) {
    console.log(error);
    alert("리뷰 작성에 실패했습니다 : " + error);
  }
};
```

이 코드는 사용자가 따릉이의 출발 대여소와 도착 대여소에 대한 별점 평가와 리뷰를 남길 수 있다. 리뷰 작성이 끝나면 백엔드 API를 통하여 서버로 전송한다. **user interface**는 **material-UI component**를 이용하여 구성하였다.

사용자가 자전거를 반납할 때, 대여하고 반납한 대여소에 평점과 후기를 남길 수 있다. 따라서 프론트에서는 사용자가 자전거를 반납할 때 두 대여소에 대한 평점을 매길 수 있는 창을 띄우고, 그 창에서 사용자가 값을 입력한 후 확인 버튼을 누르면 서버로 요청이 되는데,

```

@PutMapping("/lendplace/review")
public ResponseEntity<JsonResponse> reviewRating(@RequestBody BikeStationRatingDto bikeStationRatingRequest) {
  bikeStationRatingService.createRating(bikeStationRatingRequest);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null));
}
```

해당 컨트롤러 코드가 실행될 것이다.

이 코드는

```

public void createRating(BikeStationRatingRequest bikeStationRatingRequest) {
  try {
    if (bikeStationRatingRequest.getLendplace_id1() != null && !bikeStationRatingRequest.getLendplace_id1().isBlank()) {
      BikeStationRating bikeStationRating = new BikeStationRating(
        bikeStationRatingRequest.getLendplace_id1(),
        bikeStationRatingRequest.getRating(),
        bikeStationRatingRequest.getWriter_id(),
        bikeStationRatingRequest.getReview1(),
        bikeStationRatingRequest.getReview2()
      );
      bikeStationRatingRepository.insert(bikeStationRating);
    }
    if (bikeStationRatingRequest.getLendplace_id2() != null && !bikeStationRatingRequest.getLendplace_id2().isBlank()) {
      BikeStationRating bikeStationRating = new BikeStationRating(
        bikeStationRatingRequest.getLendplace_id2(),
        bikeStationRatingRequest.getWriter_id(),
        bikeStationRatingRequest.getRating(),
        bikeStationRatingRequest.getReview2()
      );
      bikeStationRatingRepository.insert(bikeStationRating);
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
}
```

이와 같은 서비스 코드를 호출하게 되며, 서비스 코드를 확인하면 각 데이터가 존재하는 경우 각각 출발 대여소와 도착 대여소에 대한 후기와 평점을 등록하는 것을 확인할 수 있다.

SQL

```

String sql = "INSERT INTO bikestationrating (`lendplace_id`, `user_id`, `rating`, `review`) VALUES (?, ?, ?, ?)";

jdbcTemplate.update(
  sql,
  bikeStationRating.getLendplace_id(), bikeStationRating.getUser_id(), bikeStationRating.getRating(), bikeStationRating.getReview());

```

이 경우 간단하게 **bikestationrating table**에 **INSERT**를 이용해 데이터를 저장하는 쿼리로, 출발/도착 대여소를 나타내는 **lendplace_id**와 함께 후기를 등록한 사용자 정보, 사용자가 입력한 평점 및 후기를 **bikestationrating table**에 저장하게 된다.

32. 대여소 리뷰 확인 기능

SCREEN

대여소 리뷰

자전거 대여소 위치

ST-2714

평균 평점: ★★★★★★ 4.5 점

홍길동



학교 근처라서 대여하기 편해요~

김지만



오 여기 넘 조아용

DESCRIPTION

위 기능은 위에서 따릉이를 대여하여 출발한 대여소와 도착해 따릉이를 반납한 대여소에 대해 사용자 본인 및 다른 사용자들이 남긴 평점과 리뷰를 조회할 수 있는 페이지이다. 해당 대여소에 남긴 모든 평점의 평균이 보여지고 사용자들이 남긴 개별적인 리뷰와 평점 또한 확인 할 수 있다.

FRONT END

```
useEffect(() => {
  if (!query || !userId) return;
  const query = `&${query}&${userId}`;
  const response = await fetch(process.env.REACT_APP_API_URL + `/station/get-lendplace-status?lendPlace_id=${query}&user_id=${userId}&lendPlace_id>${query}&user_id>${userId}`);
  if (response.status === 200) {
    const jsonData = await response.json();
    setAvgRating(jsonData.result.average_rating);
    setReviewRating(jsonData.result.average_rating);
    alert(`리뷰를 가져왔습니다.`);
  } else {
    new Error(`리뷰를 가져오는데 실패했습니다.`);
  }
  const response2 = await fetch(process.env.REACT_APP_API_URL + `/station/get-all-lendplace?user_id=${userId}`, {
    method: "GET",
    headers: { "Content-Type": "application/json" },
  });
  if (response2.status === 200) {
    const jsonData2 = await response2.json();
    const metaData = jsonData2.result.map((data) => {
      const lendPlaceId = data.lendPlace_id;
      const statAddr = `${data.lendPlace_addr} ${data.state_addr2}`;
      const statName = `${data.lendPlace_name} ${data.state_name}`;
      return { ...data, statName, statAddr };
    });
    setLendPlaceData(metaData);
  } else {
    alert(`리뷰를 가져오는데 실패했습니다.`);
  }
});
```

사용자가 대여소를 검색하고 선택하여 해당 자전거 대여소에 대한 리뷰와 평균 평점을 조회할 수 있다. 백엔드 API를 통해 대여소의 리뷰 데이터를 가져와 화면에 표시한다. 대여소 검색 및 선택은 ‘Autocomplete’을 이용하여 구현했다.

BACK END

대여소 현황조회를 하게 될 때, 해당 대여소의 후기 데이터도 함께 프론트에 전달하게 된다. 이때 대여소 현황조회는 하나의 대여소에 대한 자세한 정보를 조회하는 것이며

‘/station/get-lendplace-status’로 요청하면

```
@GetMapping("/get-lendplace-status")
public ResponseEntity<JsonResponse> getStationStatus(@RequestParam String lendPlace_id, @RequestParam int user_id) {
  BikeStationDtos.BikeStationInfoStatus bikeStationStatus = bikeStationService.getStationStatus(lendPlace_id, user_id);
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationStatus));
}
```

이와 같은 controller로직이 동작한다.

```
public BikeStationDtos.BikeStationInfoStatus getBikeStationStatus(String lendPlaceId, int userId) {
  try {
    BikeStationDtos.BikeStationInfoStatus bikeStationStatusWithUsername = bikeStationRepository.findStatusByUserId(lendPlaceId, userId);
    return BikeStationDtos.BikeStationInfoStatus.bikeStationStatus = bikeStationService.getBikeStationStatus(lendPlace_id, user_id);
  } catch (EmptyResultDataAccessException e) {
    throw new GlobalException(ResponseStatus.RESULT_NOT_EXIST);
  } catch (Exception e) {
    throw new GlobalException(ResponseStatus.DATABASE_ERROR);
  }
}
```

Service code를 확인하면, 위에서 언급했던 바와 같이 사용자가 controller에게 요청할 때 함께 보냈던 대여소의 id와 이를 호출한 사용자의 id를 이용해 해당 대여소 정보를 조회하게 된다.

SQL

```

public BikeStationDto.BikeStationStatus findStatusById(String lendplaceId, int userId) {
    var bikeMapper = BeanPropertyRowMapper.newInstance(BikeStationDto.BikeStationStatus.class);
    List<BikeStationRatingDto.BikeStationReview> reviews = jdbcTemplate.query(
        "SELECT user_id, rating, review FROM bikestationrating WHERE lendplace_id = ?",
        new BeanPropertyRowMapper<>(BikeStationRatingDto.BikeStationReview.class),
        lendplaceId
    );

    reviews.forEach(review -> {
        String username = jdbcTemplate.queryForObject(
            "SELECT username FROM user WHERE id = ?",
            new Object[]{review.getUserId()},
            String.class
        );

        review.setUsername(username);
    });
}

```

위의 대여소 현황 조회에서 다뤘던 쿼리는 제외하고 후기와 평점 데이터를 반환하는 부분을 살펴보면, 위와 같은 쿼리를 확인할 수 있다.

해당 쿼리는 특정 대여소에 대한 후기와 평점 정보를 모두 확인하기 위해 `bikestationrating` table에서 `lendplace_id`가 일치하는 정보를 찾아 해당 대여소의 후기 데이터들을 모두 조회하게 된다. 후기를 작성한 사용자의 `id`, 별점, 후기 문구를 조회하게 되는 것인데, 이때 프론트에서 사용자의 `id`가 아닌, 사용자 이름을 띄우기 때문에 `user`의 `id`를 `name`으로 변경하여 반환하게 된다.

33, 34. 대여소 즐겨찾기 설정 및 해제 기능

SCREEN

★ 서울특별시 노원구 광운로 20 광운대학교

★★★★★ 5점
후기 보기

대여소 즐겨찾기

DESCRIPTION

위 기능은 자주 사용하거나 근처의 대여소를 즐겨찾기로 추가할 수 있는 기능이다. 즐겨찾기

추가는 메인페이지에서 시작 대여소를 선택하고 아래의 별 버튼을 누르면 ‘즐겨찾기 설정에 성공하였습니다.’라는 메시지가 뜨며 즐겨찾기에 등록된다. 한번 더 클릭하면 즐겨찾기가 해제된다. 즐겨찾기된 대여소는 메인페이지의 상단에서 즐겨찾기 페이지를 들어가면 목록을 확인할 수 있다.

FRONT END	BACK END
<pre>const onClickFavorite = () => { (async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + "/favorite/favorite-lendplace", { method: "POST", headers: { "Content-Type": "application/json" }, body: JSON.stringify({ lendplace_id: isOnClient ? endPos.lendplace_id : startPos.lendplace_id, user_id: user_id, }), credentials: "include", }); if (response.status != 200) { throw new Error("즐겨찾기 설정에 실패하였습니다."); } const jsonData = await response.json(); if (jsonData.status != 2025) { setIsFavorite(!isFavorite); if (isFavorite) { alert("즐겨찾기 해제에 성공하였습니다."); } else { alert("즐겨찾기 설정에 성공하였습니다."); } } return; } else { alert("즐겨찾기 설정에 실패하였습니다."); } } catch (error) { console.log(error); alert("즐겨찾기 설정에 실패하였습니다."); } })(); localStorage.setItem("isFavorite", !isFavorite); setIsFavorite(!isFavorite); };</pre>	<pre>@PostMapping("/favorite-lendplace") public ResponseEntity<JsonResponse> favoriteLendplace(@RequestBody FavoriteRequestDto requestDto) { favoriteService.createFavorite(requestDto); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, null)); }</pre>
<p>위 코드는 controller 코드로, 프론트에서 '/favorite/favorite-lendplace'로 요청을 하게 되면 해당 controller가 요청을 감지하고 Service에 요청을 하게 된다.</p>	
<p>즐겨찾기 페이지로 들어가면 백엔드 API로부터 사용자의 즐겨찾기 대여소 목록을 가져와 각 대여소의 정보 및 평점을 Material-UI 카드에 렌더링하여 사용자가 즐겨찾기를 toggle 할 수 있다. 즐겨찾기 페이지에는 즐겨찾기로 등록된 대여소의 주소, 대여 가능한 자전거 수, 평균 평점에 대한 정보를 조회할 수 있고 대여소 옆에 ‘후기 보기’버튼을 누르면 해당 대여소의 리뷰 페이지로 리다이렉션된다.</p>	
SQL	
<pre>public boolean findByUserIdAndLendplaceId(Favorite favorite) { Boolean isExists = jdbcTemplate.queryForObject("SELECT EXISTS (" + " SELECT 1 FROM favorite " + " WHERE user_id = ? AND lendplace_id = ? " + ")", Boolean.class, favorite.getUser_id(), favorite.getLendplace_id()); return isExists; }</pre>	

```

public void insert(Favorite favorite) {
    jdbcTemplate.update("INSERT INTO `favorite` (`user_id`, `lendplace_id`) VALUES (?,?)",
        favorite.getUser_id(), favorite.getLendplace_id());
}

❸ 방지민
public void delete(Favorite favorite) {
    jdbcTemplate.update("DELETE FROM `favorite` WHERE user_id=? AND lendplace_id=?",
        favorite.getUser_id(), favorite.getLendplace_id());
}

```

위 SQL을 살펴보면, `findByIdAndLendplaceId`의 경우 `favorite table`에 `user_id`와 `lendplace_id`의 조합이 존재하는지 판단하는 Query이고, 이를 사용자의 해당 대여소 즐겨찾기 여부를 알기 위해 사용할 수 있다.

`insert`의 경우에는 `favorite table`에 `user_id`와 `lendplace_id`를 저장하고, `delete`의 경우에는 삭제한다. `insert`는 즐겨찾기 등록에, `delete`는 즐겨찾기 등록 해제에 사용한다.

35. 날씨 확인 기능

SCREEN

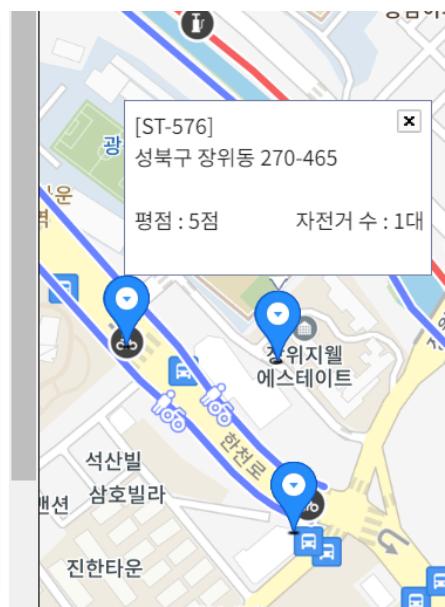
날씨

성북구 장위동 270-465의 날씨



Rain : 실비

현재 온도 7.78도	최저 온도 4.61도	최고 온도 7.97도
----------------	----------------	----------------



[ST-576]
성북구 장위동 270-465
평점 : 5점 자전거 수 : 1대

경위지웰 에스테이트
석산빌
삼호빌라
한천로
진한타운

DESCRIPTION

마커 클릭 시, 해당 지역 근처의 날씨를 확인할 수 있는 기능이다.

FRONT END

```
const fetchWeather = async (lat, lon, lendplace) => {
  const API_KEY = "6f96069f9b5896f4eadf1a221d0333ab";
  try {
    const response = await fetch(
      `https://cors-anywhere.herokuapp.com/https://api.openweathermap.org/data/2.5/weather?lat=${lat}&lon=${lon}&appid=${API_KEY}&units=metric`
    );
    if (response.status !== 200) {
      throw new Error("날씨 정보를 가져오는데 실패하였습니다.");
    }
    const jsonData = await response.json();
    jsonData.lendplace = lendplace;
    setWeatherInfo(jsonData);
  } catch (error) {
    console.log(error);      mirusu400, 2주 전 · fix: 검색 시에도 날씨 정보 가져오게끔 수
  }
};
```

마커를 클릭한 지점의 위도와 경도를 이용해, Openweathermap의 API를 이용해 날씨 정보를 가져와 이를 렌더링해 주었다. OpenWeatherMap의 CORS 오류를 해결해 주기 위해 중간에 Proxy Middleware를 추가해 문제를 해결하였다.

36. 뉴스 확인 기능

SCREEN

뉴스

건강과 관련된 실시간 뉴스를 확인할 수 있습니다.
출처 : 헬스조선

나이 들면 아침잠 없어지지만… ‘이 수 준’이면 질병 의심?

나이가 들면 자연스레 아침잠이 줄어든다. 실제 노인들은 아침에 잠이 일찍 깨 이른 새벽부터 활동하는 경우가 많다. 이는 뇌 노화로 인한 자연스...

2023-12-14

아이가 말이 늦어요… 병원 가봐야 할까요?

코로나 이후 발달지연 아동이 증가했다. 여러 원인이 거론되지만 마스크가 입을 가리면서 아이들이 어른들의 입모양으로 언어를 배울 기회를 갖지 못했기 때문이라는 주장에 무게가 실린다. 언어는...

2023-12-14

“주말 곤이라고 방심은 금물”… 음주운전 사망사고 ‘목요일’에 집중

경찰청이 송년 모임 등 각종 술자리가 늘어나는 연말연시를 앞두고 음주운전을 집중 단속한다. 경찰이 올해 음주운전 근절을 위해 지속적인 단속과 상습 음주운전자 차량 압수·몰수 등을 추진한 결과, 음주운전 교통사고...

2023-12-14

스마트 체중계 100% 믿지 마세요…제품마다 정확도 달라

최근 모바일 어플리케이션(앱)을 활용해 체중이나 체지방을 측정뿐만 아니라 건강관리, 디어트 기능 등이 가능해 체중계가 헬스케어 제품으로 각광받고 있다. 집에서 편리하게 신체 기능을 기록, 관...

2023-12-14

DESCRIPTION

따릉이 자전거 사용 및 건강과 관련된 실시간 뉴스를 본 홈페이지를 통해 확인할 수 있는 기능이다.

FRONT END

```
useEffect(() => {
  window.feednami
    .load("https://health.chosun.com/site/data/rss/rss.xml")
    .then((feed) => {
      const entries = feed.entries;           mirusu400, 2주 전 • fix: 외관 변경 요구사항
      // Each entry if idx % 2 === 0, add left side, else add right side
      const leftResult = [];
      const rightResult = [];
      entries.map((entry, idx) => {
        if (idx % 2 === 0) {
          leftResult.push({
            ...entry,
            thumbText:
              entry.description.slice(0, parseInt(Math.random() * 50) + 75) +
              "...",
            side: "left",
          });
        } else {
          rightResult.push({
            ...entry,
            thumbText:
              entry.description.slice(0, parseInt(Math.random() * 50) + 75) +
              "...",
            side: "right",
          });
        }
      });
      setLeftData(leftResult);
      setRightData(rightResult);
    });
}, []);
```

헬스조선 RSS 피드 정보를 실시간으로 가져와, 이를 Material UI를 통해 실시간으로 렌더링한다. RSS 피드를 파싱하기 위해서 feednami 라이브러리를 활용하였고, 제목과 설명, 게시 시간을 파싱하여 렌더링하도록 구현하였다.

37. 자전거도로 확인 기능

SCREEN

시작 대여소

자전거 대여소 위치

지도에서 출발지점을 선택해주세요

★ 후기 보기

자전거 대여

날씨

미커를 클릭해 해당 지역의 날씨를 확인하세요.

DESCRIPTION

본 기능은 애플리케이션 지도 내 자전거 도로를 확인할 수 있는 기능으로, 자전거 도로를 하이라이팅하여 표시해 쉽게 길을 확인할 수 있는 기능이다.

FRONT END

```

useEffect(() => {
  const container = document.getElementById("map");
  const options = {
    center: new kakao.maps.LatLng(lat, long),
    level: 3,
  };
  const map = new kakao.maps.Map(container, options);
  setMap(map);

  kakao.maps.event.addListener(map, "dragend", () => {
    const leftTop = map.getBounds().getSouthWest();
    const rightBottom = map.getBounds().getNorthEast();
    updateMarker(map, leftTop, rightBottom);
  });

  kakao.maps.event.addListener(map, "zoom_changed", () => {
    const leftTop = map.getBounds().getSouthWest();
    const rightBottom = map.getBounds().getNorthEast();
    updateMarker(map, leftTop, rightBottom);
  });
  map.addOverlayMapTypeId(kakao.maps.MapTypeId.BICYCLE);
  updateMarker(map, map.getBounds().getSouthWest(), map.getBounds().getNorthEast());
}, [isLoaded]);

```

KakaoMap API의 `addOverlayMapTypeId` 옵션을 이용해 자전거 도로 하이라이팅 기능을 활성화하여 해당 시각화를 시현하였다.

38. 대여소 검색 기능

SCREEN	
<p>The screenshot shows a search interface with the title "시작 대여소" (Start Rental Station). A dropdown menu is open under the input field "자전거 대여소 위치" (Bicycle Rental Station Location) containing the following search results:</p> <ul style="list-style-type: none"> [ST-1000] 서울특별시 양천구 신정동 236 서부식자재마트 건너편 [ST-1001] 서울특별시 양천구 남부순환로4 길20 서서울호수공원 [ST-1002] 서울특별시 양천구 목동동로 316-6 서울시 도로환경관리센터 [ST-1003] 서울특별시 양천구 화곡로 59 신월동 이마트 [ST-1004] 서울특별시 양천구 신정이펜1로 50 신정이펜하우스314동 [ST-1005] 서울특별시 양천구 신정동 310-8 신트리공원 입구 [ST-1006] 서울특별시 양천구 목동중앙로 70 서울영도초등학교 	<h3>SCREEN</h3> <p>시작 대여소 자전거 대여소 위치 양천구</p> <p>[ST-1000] 서울특별시 양천구 신정동 236 서부식자재마트 건너편</p> <p>[ST-1001] 서울특별시 양천구 남부순환로4 길20 서서울호수공원</p> <p>[ST-1002] 서울특별시 양천구 목동동로 316-6 서울시 도로환경관리센터</p> <p>[ST-1003] 서울특별시 양천구 화곡로 59 신월동 이마트</p> <p>[ST-1004] 서울특별시 양천구 신정이펜1로 50 신정이펜하우스314동</p> <p>[ST-1005] 서울특별시 양천구 신정동 310-8 신트리공원 입구</p> <p>[ST-1006] 서울특별시 양천구 목동중앙로 70 서울영도초등학교</p>
DESCRIPTION	
<p>자전거 대여 시 원하는 대여소를 찾을 때 검색을 통하여 대여소를 찾을 수 있는 기능이다. 해당 검색 기능을 통해 사용자는 쉽게 대여소를 선택할 수 있다.</p>	<p>대여소를 검색하기 위해 단어를 입력한 뒤 요청을 하면</p> <pre>@GetMapping("/search/lendPlace") public ResponseEntity<JsonResponse> searchStation(@RequestParam String name) { List<BikeStationDto.BikeStationDetailDto> bikeStationDetailDtoList = bikeStationService.searchStation(name); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationDetailDtoList)); }</pre> <p>이러한 컨트롤러 코드를 수행하게 된다. 이러한 컨트롤러 로직은 아래와 같은 서비스 코드를 호출하는 것을 확인할 수 있는데,</p> <pre>public List<BikeStationDto.BikeStationDetailDto> searchStation(String name) { try { return bikeStationRepository.findDetailByName(name); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre> <p>이때 이 서비스 코드 내에서는 repository를 호출하여 사용자가 입력한 이름이 포함되는 대여소를 찾아 반환하고, 최종적으로 이 정보들을 초기에 호출한 사용자에게 전달하게</p>

39. 최근 사용 대여소 확인 기능

SCREEN

최근 사용 대여소

- 서울특별시 노원구 서울시 노원구 월계로 42길97
- 서울특별시 노원구 화랑로 429 동해문화예술관
앞

DESCRIPTION

따릉이 대여 시 최근에 사용한 대여소를 확인 후 선택할 수 있는 기능이다

FRONT END	BACK END
<pre>if (user && user.id) { const recentResponse = await fetch(process.env.REACT_APP_API_URL + '/station/get-recent-lendplace?user_id=' + user.id) , { method: 'GET', headers: { 'Content-Type': "application/json" }, credentials: 'include', }); if (recentResponse.status !== 200) { throw new Error('최근 대여소 정보를 가져오는데 실패하였습니다.'); } const recentJsonData = await recentResponse.json(); _setRecent(recentJsonData.result); }</pre>	사용자가 최근에 대여한 대여소를 확인하기 위해 요청을 하면
따릉이 대여 화면에서 사용자를 기준으로 최근 대여소를 확인하는 API를 이용해 최근 사용 대여소 정보를 알아내고, 이를 화면에 렌더링한다.	<pre>@GetMapping("/user/recentLocation") public ResponseEntity<JsonResponse> getRecentStation(@UserParam int user_id) { List<BikeStationDto.BikeStationSimpleWithState> bikeStationStatusList = bikeStationService.getRecentStation(user_id); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationStatusList)); }</pre>
	이러한 코드가 실행된다. 이 코드는 서비스 코드를 호출하고, 서비스 코드에서는 데이터베이스에 접근하여 원하는 데이터를 가져와 반환해주게 된다.
	<pre>public List<BikeStationDto.BikeStationSimpleWithState> getRecentStation(int userId) { try { return bikeStationRepository.findRecentByUserId(userId); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre>
	서비스 코드를 확인하면 repository 를 이용해 Database의 데이터를 가져와서 이를 단순히 반환해 주고 있는 것을 확인할 수 있다.
SQL	

```

public List<BikeStationDto.BikeStationSimpleWithState> findRecentlyUsed(int userId) {
    var stationMapper = BeanPropertyRowMapper.newInstance(BikeStationDto.BikeStationSimpleWithState.class);
    return jdbcTemplate.query(
        // DISTINCT, COALESCE, Subqueries in the FROM clause, ALL, GROUP BY, ORDER BY, LIMIT, NULL
        "SELECT DISTINCT BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time, COALESCE(AVG(BR.rating), 0) AS average_rating " +
        "FROM (" +
        "    SELECT departure_station AS station, departure_time AS time " +
        "    FROM userlog " +
        "    WHERE user_id = ? AND departure_time IS NOT NULL " +
        "    UNION ALL " +
        "    SELECT arrival_station AS station, arrival_time AS time " +
        "    FROM userlog " +
        "    WHERE user_id = ? AND arrival_time IS NOT NULL " +
        ") AS UL " +
        "JOIN bikestationinformation BSI ON BSI.lendplace_id = UL.station " +
        "LEFT JOIN bikestationrating BR ON BSI.lendplace_id = BR.lendplace_id " +
        "GROUP BY BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time " +
        "ORDER BY UL.time DESC " +
        "LIMIT 2",
        stationMapper,
        userId,
        userId
    );
}

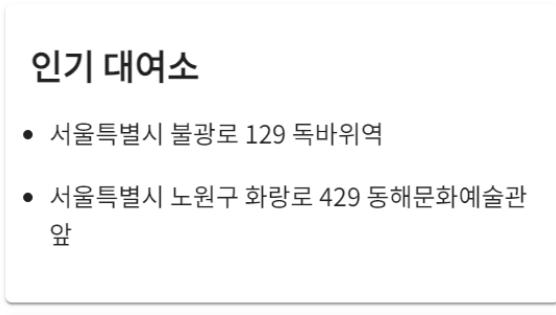
```

이는 메인 화면 좌측에 사용자에 관련된 정보를 띠워주는데, 그 중 사용자가 최근 이용한 두 개의 대여소를 출력하기 위해 사용하는 쿼리이다.

이 쿼리는 대여소 정보와 대여소 평점 정보를 조회하게 된다. 최근 2개를 조회하기 위해 각 로그를 `departure_time`이 존재하면 이 값으로, 존재하지 않으면 `arrival_time` 값으로 가장 최신의 대여소를 판단한다. 단 이때 최대 2개를 가장 최신순으로 반환해야 하기 때문에 `ORDER BY`와 `LIMIT`을 이용하게 된다.

이때 대여소 평점 정보 또한 함께 조회해야 하기 때문에 `bikestationrating` table을 JOIN하는 것을 확인할 수 있다.

40. 인기 대여소 확인 기능

SCREEN	
	
DESCRIPTION	
<p>위 기능은 사용자들이 자주 사용하는 인기 대여소를 확인 후 선택할 수 있는 기능이다.</p>	
FRONT END	BACK END

```
// 인기 대여소 정보 가져오기
const popularResponse = await fetch(process.env.REACT_APP_API_URL + "/station/get-popular-lendplace", {
  method: "GET",
  headers: { "Content-Type": "application/json" },
  credentials: "include",
});
if (popularResponse.status !== 200) {
  throw new Error("인기 대여소 정보를 가져오는데 실패하였습니다.");
}
const popularJsonData = await popularResponse.json();
setPopular(popularJsonData.result);
```

따릉이 대여 화면에서 백엔드의 자주 사용하는 인기 대여소 API를 이용해 대여소 정보를 알아내고, 이를 화면에 렌더링한다.

인기 대여소 조회는 각 대여소의 평균 평점을 이용하여 평균 평점이 가장 높은 두 개의 대여소를 사용자에게 제공하는 기능이다.

```
@GetMapping("/get-popular-lendplace")
public ResponseEntity<JsonResponse> getPopularStation() {
  List<BikeStationDto.BikeStationSimpleState> bikeStationStatusList = bikeStationService.getPopularStation();
  return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, bikeStationStatusList));
}
```

해당 정보를 알고 싶다는 요청이 들어오면 이 컨트롤러 코드가 실행되며,

```
public List<BikeStationDto.BikeStationSimpleState> getPopularStation() {
  try {
    return bikeStationRepository.findPopular();
  } catch (Exception e) {
    throw new GlobalException(ResponseStatus.DATABASE_ERROR);
  }
}
```

서비스 코드가 호출되는데 이는 repository의 코드를 호출한 뒤 값을 받아 반환한다.

SQL

```
usage ━ 방시민
▼public List<BikeStationDto.BikeStationSimpleState> findPopular() {
  var stationMapper = BeanPropertyRowMapper.newInstance(BikeStationDto.BikeStationSimpleState.class);
  return jdbcTemplate.query(
    "SELECT BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, BR.average_rating " +
    "FROM (" +
    "  SELECT lendplace_id, AVG(rating) AS average_rating " +
    "  FROM bikestationrating " +
    "  GROUP BY lendplace_id " +
    "  ORDER BY average_rating DESC " +
    "  LIMIT 2" +
    ") AS BR " +
    "LEFT JOIN bikestationinformation BSI ON BR.lendplace_id = BSI.lendplace_id",
    stationMapper);
}
```

이 쿼리를 살펴보면 대여소의 간결한 정보와 대여소의 평점 정보를 조회하고 있다. 이때 각 대여소의 평균 평점이 높은 두 개의 대여소 정보와 그 평점을 얻기 위해 FROM 내의 Sub Query를 이용하여 bikestationrating table에서 lendplace_id로 그룹핑하여 평점을 구하고, 평점이 높은 순서대로 정렬하여 두개까지만 반환하도록 한다.

추가로 이렇게 구한 두 개의 대여소에 대해 대여소 정보 또한 반환해야 하기 때문에 위 결과에 LEFT JOIN을 통해 대여소 정보가 있는 경우 대여소 정보를 출력할 수 있도록 결합해 주었다.

41. 즐겨찾기 대여소 확인 기능

SCREEN

대여소 즐겨찾기

- 서울특별시 노원구 광운로 20 광운대학교
- 서울특별시 성북구 석관동 375-97

대여소 즐겨찾기

★ 서울특별시 노원구 광운로 20 광운대학교
대여가능 자전거: 2

★★★★★ 5점

후기 보기

★ 서울특별시 성북구 석관동 375-97
대여가능 자전거: 0

☆☆☆☆☆ 0점

후기 보기

DESCRIPTION

이 기능은 사용자가 즐겨찾기 설정한 대여소를 확인할 수 있는 기능으로 메인페이지에서는 출발 대여소로 선택할 수 있고 대여소 즐겨찾기 페이지에는 해당 대여소의 정보 및 평점, 리뷰를 확인할 수 있다.

FRONT END	BACK END
<pre>const fetchData = async () => { try { const response = await fetch(process.env.REACT_APP_API_URL + `/favorite/get-lendplace?user_id=\${user.id}`, { method: "GET", headers: { "Content-Type": "application/json" }, credentials: "include", }); if(response.status != 200) { throw new Error('자전거 대여소 정보를 가져오는데 실패하였습니다.'); } const jsonData = await response.json(); console.log(jsonData.result); setFavoriteStation(jsonData.result); } catch (error) { console.log(error); } };</pre> <p>사용자를 기준으로 즐겨찾기한 대여소를 API를 통하여 알아내고, 이를 화면에 렌더링 한다.</p>	<p>서버에서는 해당 사용자가 즐겨찾기한 모든 대여소를 반환해 주는 기능이 필요한데, 해당 정보가 필요하다고 요청을 받게 되면</p> <pre>@GetMapping("/get-lendplace") public ResponseEntity<JsonResponse> getFavoriteLendPlace(@RequestParam int user_id) { List<FavoriteDto.FavoriteAllDto> favoriteAlltoList = favoriteService.findAllLendPlace(user_id); return ResponseEntity.ok(new JsonResponse(ResponseStatus.SUCCESS, favoriteAlltoList)); }</pre> <p>해당 코드가 실행되며</p> <pre>public List<FavoriteDto.FavoriteAllDto> findAllLendplace(int userId) { try { return favoriteRepository.findAllLendplaceIdByUserId(userId); } catch (Exception e) { throw new GlobalException(ResponseStatus.DATABASE_ERROR); } }</pre> <p>이 코드를 다시 호출하게 되고, 해당 코드</p>

	내에서는 repository에서 반환받은 데이터를 다시 컨트롤러에 반환하게 된다.
SQL	
	<pre> public List<FavoriteDto.FavoriteAllDto> findAllLendplaceIdByUserId(int userId) { var favoriteMapper = BeanPropertyRowMapper.newInstance(FavoriteDto.FavoriteAllDto.class); return jdbcTemplate.query(// IN, COUNT "SELECT BSI.*, " + " (SELECT COUNT(*) FROM bike WHERE bike.lendplace_id = BSI.lendplace_id) AS total_bikes, " + " (SELECT COUNT(*) FROM bike WHERE bike.lendplace_id = BSI.lendplace_id AND bike.use_status = 0 " + "AND bike.bike_status = 1) AS usable_bikes, " + " (SELECT COALESCE(AVG(rating), 0) FROM bikestationrating WHERE lendplace_id = BSI.lendplace_id) AS average_rating, " + " TRUE AS isFavorite " + "FROM bikestationinformation BSI " + "WHERE BSI.lendplace_id IN (SELECT lendplace_id FROM favorite WHERE user_id = ?)", favoriteMapper, userId); } </pre>

사용자가 즐겨찾기한 대여소의 대여소 정보를 출력하는 쿼리로, 대여소 정보와 대여소의 총
자전거 수, 대여소에 존재하는 사용 가능한 자전

4. SQL 설명

사용한 SQL 종류는 총 42개입니다.

No	SQL 문 종류	SQL 문
1	INSERT	<pre> // INSERT String sql = "INSERT INTO user (password, username, user_type, email, phone_number, " + "weight, age, last_accessed_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?)"; jdbcTemplate.update(sql, userDao.getPassword(), userDao.getUsername(), userDao.getUser_type(), userDao.getEmail(), userDao.getPhone_number(), userDao.getWeight(), userDao.getAge(), userDao.getLast_accessed_at()); return true; </pre>

		<pre> 1 { 2 "email" : "bbb@gmail.com", 3 "username" : "bbb", 4 "password" : "aaa", 5 "phone_number" : "01010101010", 6 "weight" : "50", 7 "age" : "20" 8 } </pre> <p>Body Cookies Headers (8) Test Results</p> <p>Pretty Raw Preview Visualize JSON ↻</p> <pre> 1 { 2 "timeStamp": "2023-12-14 19:13:43", 3 "status": 2016, 4 "message": "회원 가입에 성공했습니다", 5 "result": null, 6 "success": true 7 } </pre>
<p>회원 가입시 사용하는 SQL이다.</p> <p>입력 받은 회원 정보(Password, Username, Email, Phone number, weight, age)와 직접 설정해준 user의 type, 최근 접속 시간을 INSERT문을 통해 데이터베이스 user table에 입력하는 Query이다.</p> <p>해당 쿼리를 실행하면 사용자가 회원가입을 위해 입력한 정보들과 서버에서 따로 설정해준 정보들이 함께 데이터베이스에 저장되는 것을 확인할 수 있다.</p>		

No	SQL 문 종류	SQL 문
2	DELETE	<pre>// DELETE String sql = "DELETE FROM user WHERE id = ?"; jdbcTemplate.update(sql, id);</pre> <p>Body Cookies Headers (8) Test Results</p> <p>Pretty Raw Preview Visualize JSON ↻</p> <pre> 1 { 2 "id" : 38 3 } </pre> <pre> 1 { 2 "timeStamp": "2023-12-14 19:14:29", 3 "status": 2008, 4 "message": "회원 탈퇴를 성공하였습니다", 5 "result": null, 6 "success": true 7 } </pre> <p>회원 탈퇴시 사용하는 SQL이다.</p>

		<p>사용자가 본인의 회원 탈퇴 요청을 할 때, 사용자의 id를 이용하여 Delete 요청을 하게 된다. 이때 id를 통해 해당하는 사용자의 정보를 지우게 되는데, 그 때 DELETE SQL을 사용하게 된다.</p> <p>user table에서 요청한 id와 일치하는 데이터를 찾은 뒤, 해당 데이터를 삭제되는 것을 확인할 수 있다.</p>
--	--	--

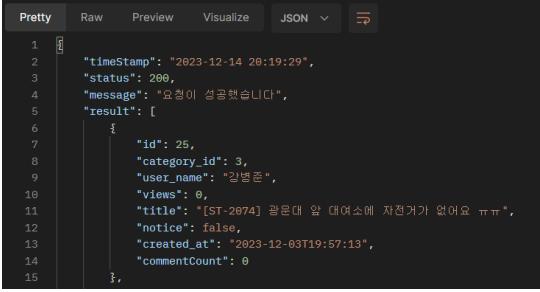
No	SQL 문 종류	SQL 문
3	UPDATE	<p>// UPDATE</p> <pre>String sql = "UPDATE user SET password = ? WHERE email = ?"; jdbcTemplate.update(sql, newPassword, email);</pre> <p>The screenshot shows a REST API testing interface. The request URL is partially visible as /user. The body of the request contains a JSON object:</p> <pre>1 {"email": "jimin@naver.com", 2 "password": "jiminjimin" 3 }</pre> <p>The response tab is selected, showing the following JSON data:</p> <pre>1 {"timeStamp": "2023-12-14 19:15:26", 2 "status": 2007, 3 "message": "비밀번호 변경을 성공하였습니다.", 4 "result": null, 5 "success": true 6 }</pre> <p>The response status code is 2007, indicating success. The message is "비밀번호 변경을 성공하였습니다." (Password change was successful.). The result field is null, and the success field is true.</p> <p>비밀번호 변경 시 사용하는 SQL이다. 사용자가 비밀번호를 잊어버렸을 경우, 사용자 정보에 등록된 email을 통해 인증 과정을 거치고 비밀번호를 변경할 수 있게 된다. 따라서 사용자의 email을 통해 일치하는 정보를 찾고, 해당하는 사용자의 정보 중 password를 사용자가 입력한 비밀번호로 UPDATE SQL을 사용하여 변경하게 된다. 즉 데이터베이스의 user table에서 해당하는 email의 유저 정보 중, password가 변경되는 것을 확인할 수 있다.</p>

No	SQL 문 종류	SQL 문
----	----------	-------

		<pre>// MIN, INNER JOIN history_id = jdbcTemplate.queryForObject(sql: "SELECT MIN(ph.history_id) FROM paymenthistory ph " + "INNER JOIN user u ON ph.user_id = u.id " + "WHERE u.id = ? AND ph.is_used = 0", new Object[]{userId}, Integer.class);</pre>
4	INNER JOIN	<pre> 1 2 "user_id" : 37, 3 "departure_station" : "ST-2074" 4 y Cookies Headers (8) Test Results retty Raw Preview Visualize JSON ↻ 1 2 "timeStamp": "2023-12-14 19:29:07", 3 "status": 2026, 4 "message": "자전거 대여에 성공하였습니다.", 5 "result": null, 6 "success": true 7 </pre>
사용자가 bike를 대여할 때 사용하는 Query 중 하나이다. 사용자의 id를 통해 특정 사용자가 자전거 대여 요청을 보냈을 때, paymenthistory 테이블에서 특정 사용자의 사용하지 않은(is_used = 0) 결제 기록 중 가장 오래된 것(MIN(ph.history_id))을 찾는데 사용된다. 이때, user id를 기준으로 paymenthistory table과 user table을 INNER JOIN하여 해당 사용자와 해당 사용자의 이용권 구매 이력을 연결한다.		

No	SQL 문 종류	SQL 문
5	LEFT OUTER JOIN	<pre>// LEFT OUTER JOIN, UNION, RIGHT OUTER JOIN, FULL OUTER JOIN(LEFT OUTER JOIN+RIGHT OUTER JOIN) String sql = "SELECT * FROM coupon LEFT OUTER JOIN ticket ON coupon.ticket_id = ticket.id " + "UNION " + "SELECT * FROM coupon RIGHT OUTER JOIN ticket ON coupon.ticket_id = ticket.id"; List<CouponFullOuterJoinTicketDto> coupons = jdbcTemplate.query(sql, BeanPropertyRowMapper.newInstance(CouponFullOuterJoinTicketDto.class)); return coupons != null ? coupons : new ArrayList<>();</pre> <pre> 1 2 "timeStamp": "2023-12-14 19:37:53", 3 "status": 2035, 4 "message": "[관리자] 모든 쿠폰 조회를 성공하였습니다.", 5 "result": [6 { 7 "value": "교수님의은총", 8 "is_used": 1, 9 "ticket_id": 1, 10 "id": 1, 11 "ticket_price": 2500 12 }, 13 { 14 "value": "우연", 15 "is_used": 1, 16 "ticket_id": 1, 17 "id": 1, 18 "ticket_price": 2500 19 }] </pre>

		<p>관리자 기능에서 존재하는 모든 쿠폰 조회에 사용한 Query이다.</p> <p>Coupon과 해당하는 Ticket을 연결하여 모든 Coupon과 Ticket Table에 해당하는 정보를 조회한다, 이때 LEFT OUTER JOIN의 경우 ticket_id로 JOIN하여 모든 coupon 정보를 출력하면서 해당하는 ticket 정보도 함께 출력하게 되는데, LEFT OUTER JOIN이므로 ticket 정보가 존재하지 않는 경우에도 coupon 정보를 출력하게 된다.</p>
--	--	---

No	SQL 문 종류	SQL 문
6	RIGHT OUTER JOIN	<pre> return jdbcTemplate.query("SELECT B.id, B.category_id, B.user_id, B.views, B.title, B.notice, B.created_at, COUNT(C.id) as comment_count " + "FROM comment C RIGHT OUTER JOIN board B " + "ON C.write_id = B.id AND C.category_id = B.category_id " + "WHERE B.category_id=? " + "GROUP BY B.id " + "ORDER BY B.created_at DESC" , boardCommentMapper, category_id); </pre>  <pre> { "category_id": 3 } </pre>  <pre> { "timeStamp": "2023-12-14 20:19:29", "status": 200, "message": "요청이 성공했습니다", "result": [{ "id": 25, "category_id": 3, "user_name": "강별준", "views": 0, "title": "[ST-2074] 광운대 앞 대여소에 자전거가 없어요ㅠㅠ", "notice": false, "created_at": "2023-12-03T19:57:13", "commentCount": 0 }] } </pre> <p>특정 category에 해당하는 게시글의 title들을 조회할 때 사용한다.</p> <p>comment table과 board table을 join한 뒤, 해당하는 category에 대한 board 정보들과 그 board의 댓글 수를 출력하게 된다.</p> <p>이 때, 게시글의 title을 사용자에게 보여주는 경우 해당 게시글에 달린 댓글의 수 또한 함께 제공해야 하기 때문에 OUTER JOIN을 이용하며, comment table과 board table을 board의 id와 category를 이용하여 JOIN한다.</p> <p>만약, comment가 존재하지 않는 board라도 board 정보를 출력하게 하기 위해 RIGHT OUTER JOIN을 이용한 것이다.</p>

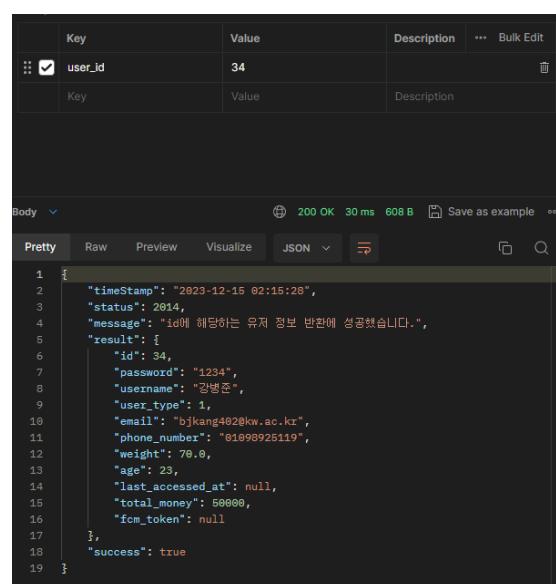
No	SQL 문 종류	SQL 문
7	FULL OUTER JOIN	<pre>// LEFT OUTER JOIN, UNION, RIGHT OUTER JOIN, FULL OUTER JOIN(LEFT OUTER JOIN+RIGHT OUTER JOIN) String sql = "SELECT * FROM coupon LEFT OUTER JOIN ticket ON coupon.ticket_id = ticket.id " + "UNION " + "SELECT * FROM coupon RIGHT OUTER JOIN ticket ON coupon.ticket_id = ticket.id;"; List<CouponFullOuterJoinTicketDto> coupons = jdbcTemplate.query(sql, BeanPropertyRowMapper.newInstance(CouponFullOuterJoinTicketDto.class)); return coupons != null ? coupons : new ArrayList<>();</pre> <pre>1 ↴ 2 "timeStamp": "2023-12-14 19:37:53", 3 "status": 2035, 4 "message": "[관리자] 모든 쿠폰 조회를 성공하였습니다.", 5 "result": [6 { 7 "value": "교수님의등총", 8 "is_used": 1, 9 "ticket_id": 1, 10 "id": 1, 11 "ticket_price": 2500 12 }, 13 { 14 "value": "꼬꼬", 15 "is_used": 1, 16 "ticket_id": 1, 17 "id": 1, 18 "ticket_price": 2500 19 }, 20], 21 "count": 2 22}</pre>

No	SQL 문 종류	SQL 문
8	SET / UNION	<pre>// LEFT OUTER JOIN, UNION, RIGHT OUTER JOIN, FULL OUTER JOIN(LEFT OUTER JOIN+RIGHT OUTER JOIN) String sql = "SELECT * FROM coupon LEFT OUTER JOIN ticket ON coupon.ticket_id = ticket.id " + "UNION " + "SELECT * FROM coupon RIGHT OUTER JOIN ticket ON coupon.ticket_id = ticket.id;"; List<CouponFullOuterJoinTicketDto> coupons = jdbcTemplate.query(sql, BeanPropertyRowMapper.newInstance(CouponFullOuterJoinTicketDto.class)); return coupons != null ? coupons : new ArrayList<>();</pre>

		<pre> 1 * 2 * "timeStamp": "2023-12-14 19:37:59", 3 * "status": 2035, 4 * "message": "[관리자] 모든 쿠폰 조회를 성공하였습니다.", 5 * "result": [6 { 7 "value": "교수님의은총", 8 "is_used": 1, 9 "ticket_id": 1, 10 "id": 1, 11 "ticket_price": 2500 12 }, 13 { 14 "value": "쭈骡", 15 "is_used": 1, 16 "ticket_id": 1, 17 "id": 1, 18 "ticket_price": 2500 19 }] </pre>
		<p>모든 쿠폰 조회에 사용된 쿼리이다. Coupon과 Coupon에 명시된 ticket id를 이용해 Ticket을 연결하여 모든 Coupon과 그에 해당하는 Ticket Table의 정보를 조회한다, UNION의 경우, 두 쿼리의 결과를 합쳐서 하나의 결과로 만드는 과정으로, 중복되는 결과는 제거한다. 이 경우 LEFT OUTER JOIN과 RIGHT OUTER JOIN의 결과를 합쳐 하나로 만들기 위해 사용된다.</p>

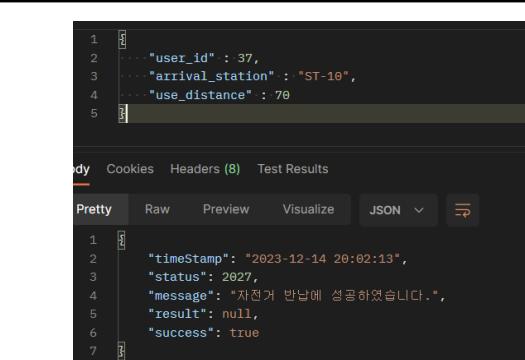
No	SQL 문 종류	SQL 문
9	SET / INTERSECT	<div style="display: flex; justify-content: space-between;"> <div style="flex: 1;"> <pre> // INTERSECT String sql = "SELECT * FROM user " + "INTERSECT " + "SELECT * FROM user WHERE email = ?"; try { return jdbcTemplate.queryForObject(sql, new Object[]{email}, new UserRowMapper()); } catch (EmptyResultDataAccessException e) { return null; } </pre> </div> <div style="flex: 1;"> <p>The screenshot shows a POST request to 'http://localhost:8080/api/user'. The 'Body' tab is selected, showing a JSON payload:</p> <pre> { "email": "bbb@gmail.com", "username": "bbb", "password": "aaa", "phone_number": "01010101010", "weight": "60", "age": "28" } </pre> <p>The response status is 200 OK, with a timestamp of 2023-12-15 02:09:06, a status of 2017, and a message indicating the user already exists.</p> </div> </div> <p>이미 회원가입이 되어 있는 이메일로는 회원가입이 진행되면 안된다. 따라서 이메일에 대한 중복확인을 진행해야 하므로 user table에서 해당 이메일에 대한 필드인 email을 기반으로 전체 user에 대한 정보 집합과</p>

		해당 이메일을 가진 user에 대한 정보 집합에 대해 교집합 연산인 INTERSECT 쿼리를 사용하여 구해주었다.
--	--	---

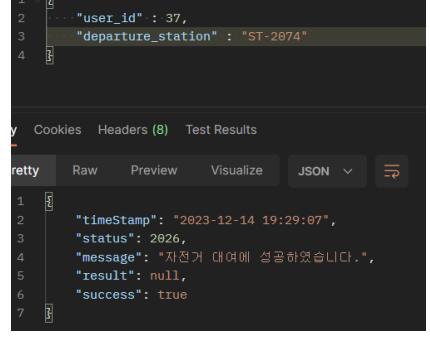
No	SQL 문 종류	SQL 문
10	SET / EXCEPT	<pre>// EXCEPT String sql = "SELECT * FROM user " + "EXCEPT " + "SELECT * FROM user WHERE id <> ?"; try { return jdbcTemplate.queryForObject(sql, new Object[]{id}, new UserRowMapper()); } catch (EmptyResultDataAccessException e) { return null; }</pre>  <p>The screenshot shows a REST API response with a 200 OK status. The response body is a JSON object representing a user record:</p> <pre> 1 { 2 "timeStamp": "2023-12-15 02:15:28", 3 "status": 2014, 4 "message": "id로 해당하는 유저 정보 반환에 성공했습니다.", 5 "result": [6 { 7 "id": 34, 8 "password": "1234", 9 "username": "강병준", 10 "user_type": 1, 11 "email": "bjkang402@kw.ac.kr", 12 "phone_number": "01098925119", 13 "weight": 70.0, 14 "age": 23, 15 "last_accessed_at": null, 16 "total_money": 50000, 17 "fcm_token": null 18 }, 19], 20 "success": true 21 }</pre> <p>Below the JSON response, there is explanatory text in Korean:</p> <p>user_id를 기반으로 USER table에서 해당 유저에 대한 모든 정보를 반환할 때 EXCEPT를 사용하여 주어진 user_id에 해당하지 않는 모든 레코드를 제외시키고 남은 레코드인 해당 user_id를 가진 레코드에 대한 정보를 반환한다.</p>

No	SQL 문 종류	SQL 문
11	SOME	<pre>// SOME String sql = "SELECT * FROM user WHERE id = SOME (SELECT id FROM user WHERE id > 0)"; return jdbcTemplate.query(sql, BeanPropertyRowMapper.newInstance(UserDto.class));</pre>

		<pre>{ "timeStamp": "2023-12-15 02:23:28", "status": 2009, "message": "[관리자] 모든 유저의 정보가 반환되었습니다.", "result": [{ "id": 1, "password": "admin", "username": "admin", "user_type": 0, "email": "admin", "phone_number": "01000000000", "weight": 100.0, "age": 100, "last_accessed_at": "2023-12-14T20:59:01", "total_money": 993000, "fcm_token": "cwGO-NaVzN74a_arbC0Awm:APA91bEN-Mkg7Aa9CbEe5GI1Qavfnoc" }, { "id": 2, "password": "honghong", "username": "홍길동", "user_type": 1, "email": "hong@gmail.com", "phone_number": "01010001000", "weight": 67.0, "age": 24, "last_accessed_at": "2023-12-03T03:17:05", "total_money": 30000, "fcm_token": null }] }</pre>
		<p>관리자의 기능인 모든 유저 조회 기능을 위해 USER table에서 SOME를 사용하여 id가 0보다 큰 모든 id를 선택한 부분집합에 속하는 레코드를 조회한다. id가 0보다 크기 때문에 모든 유저에 대한 정보가 반환된다.</p>

No	SQL 문 종류	SQL 문
12	MAX	<pre>// MAX String getMaxStandsSql = "SELECT MAX(max_stands) FROM bikestationinformation"; max_stands = jdbcTemplate.queryForObject(getMaxStandsSql, Integer.class);</pre>  <p>The screenshot shows the Java code for retrieving the maximum value from the 'max_stands' column of the 'bikestationinformation' table. Below the code, a terminal window displays the JSON response from the database query. The response includes a timestamp, status, message, and a result object containing a single item with an ID of 1, a user ID of 37, an arrival station of 'ST-10', and a use distance of 70.</p> <pre> { "timeStamp": "2023-12-14 20:02:13", "status": 2027, "message": "자전거 반납에 성공하였습니다.", "result": null, "success": true } </pre> <p>자전거 반납에 사용하는 쿼리 중 일부이며, 대여소 들 중 가장 많은 자전거를 보관할 수 있는 대여소의 보관 가능 자전거 수를 조회한다. 이는 bikestationinformation 테이블에서 max_stands의</p>

		<p>최댓값을 MAX를 이용해 조회하여 얻을 수 있다. 이 값을 이용하여 현재 대여소에 보관된 자전거 수(<code>cur_stands</code>)가 최대 보관 가능 수(<code>max_stands</code>) 이상이면, 오류가 발생했다는 것을 알 수 있을 것이다.</p>
--	--	---

No	SQL 문 종류	SQL 문
13	MIN	<pre>// MIN, INNER JOIN history_id = jdbcTemplate.queryForObject(sql: "SELECT MIN(ph.history_id) FROM paymenthistory ph " + "INNER JOIN user u ON ph.user_id = u.id " + "WHERE u.id = ? AND ph.is_used = 0", new Object[]{userId}, Integer.class);</pre>  <p>이 또한 사용자가 자전거를 대여할 때 사용하는 Query들 중 하나이다. 위에서 설명했던 바와 같이, 사용자의 id를 통해 특정 사용자가 자전거 대여 요청을 보냈을 때, paymenthistory 테이블에서 특정 사용자의 사용하지 않은 결제 기록들 중 가장 먼저 구매한 이용권을 찾는다. 이때 <code>MIN(ph.history_id)</code>를 사용한다. 즉, MIN의 경우 구매한 여러개의 이용권 중 가장 오래 된 이용권부터 순차적으로 사용하기 위해 <code>history_id</code>가 가장 작은 이용권 하나를 얻어올 때 사용하게 된다.</p>

No	SQL 문 종류	SQL 문
----	----------	-------

14

ALL

```

return jdbcTemplate.query(
    "DISTINCT COALESCE, Subqueries in the FROM clauses, ALL, GROUP BY, ORDER BY, LIMIT, NULL
    "SELECT DISTINCT BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time, COALESCE(AVG(BR.rating), 0) AS average_rating "
    "FROM (" +
        " SELECT departure_station AS station, departure_time AS time " +
        " FROM userlog " +
        " WHERE user_id = ? AND departure_time IS NOT NULL " +
        " UNION ALL " +
        " SELECT arrival_station AS station, arrival_time AS time " +
        " FROM userlog " +
        " WHERE user_id = ? AND arrival_time IS NOT NULL " +
    ") AS UL " +
    "JOIN bikestationinformation BSI ON BSI.lendplace_id = UL.station " +
    "LEFT JOIN bikestationrating BR ON BSI.lendplace_id = BR.lendplace_id " +
    "GROUP BY BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time " +
    "ORDER BY UL.time DESC " +
    "LIMIT 2",
    stationMapper,
    userId,
    userId
);

```

Key	Value
<input checked="" type="checkbox"/> user_id	2

Body Cookies Headers [8] Test Results

Pretty Raw Preview Visualize JSON ↗

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
{
  "timeStamp": "2023-12-14 19:49:23",
  "status": 200,
  "message": "요청이 성공했습니다",
  "result": [
    {
      "lendplace_id": "ST-869",
      "statn_addr1": "서울특별시 강북구 한천로 935",
      "statn_addr2": "반동 386-17",
      "time": "2023-12-01T03:11:04",
      "average_rating": 3.0
    },
    {
      "lendplace_id": "ST-2401",
      "statn_addr1": "서울특별시 강북구 오현로 28길 46",
      "statn_addr2": "",
      "time": "2023-12-01T03:10:56",
      "average_rating": 2.0
    }
  ],
  "success": true
}

```

사용자의 최근 대여소를 일부 조회할 때 사용하는 Query이다.

즉, 사용자의 편의를 위해 메인 화면에 사용자가 최근 이용한 두 개의 대여소를 출력하기 위해 사용하는 Query이다.

우선, userlog table을 통해 해당 사용자의 log를 조회한 후, 가장 최근에 이용한 대여소를 추출하기 위해 log의 arrival_time을, 아직 도착하지 않았다면 해당 log의 departure_time을 이용하여 가장 최근에 이용한 겹치지 않는 두 대여소를 조회한다.

최신에 이용한 순서대로 정보를 얻기 위해 departure_time 또는 arrival_time 순서로 정렬하고, 2개까지만 띄워주기 위해 두개로 제한한다. 이때 arrival_time을 이용해 조회한 대여소와 departure time을 이용해 조회한 대여소를 모두 확인하기 위해 UNION ALL을 사용한다.

No	SQL 문 종류	SQL 문
----	----------	-------

15

IN

```

    /> IN, COUNT(
        SELECT BS1.*
        WHERE bike.lendplace_id = BS1.lendplace_id) AS total_bikes,
        +
        (SELECT COUNT(*)
        FROM bike
        WHERE bike.lendplace_id = BS1.lendplace_id AND bike.use_status = 0 AND bike.bike.status > 1) AS usable_bikes,
        +
        (SELECT COALESCE(AVG(rating), 0) FROM bikestationrating
        WHERE lendplace_id = BS1.lendplace_id) AS average_rating,
        +
        TRUE AS isFavorite
    )
    WHERE BS1.lendplace_id IN (SELECT lendplace_id
    FROM favorite
    WHERE user_id = ?),
    favoriteMapper,
    userId
);

```

```

{
    "lendplace_id": "ST-1280",
    "statn_addr1": "서울특별시 노원구 화랑로 429",
    "statn_addr2": "동해문화예술관앞",
    "startn_lat": 37.619381,
    "startn_lnt": 127.057854,
    "max_stands": 20.0,
    "station_status": 1,
    "total_bikes": 0,
    "usable_bikes": 0,
    "isFavorite": true,
    "average_rating": 4.5
}
,
```

즐겨찾기한 대여소 조회에 사용하게 되는 쿼리이다.
즉, 특정 사용자가 즐겨찾기한 자전거 대여소들에 대한 상세 정보를 조회하기 위한 쿼리이다.
사용자의 고유 user_id를 사용하여 favorite 테이블에서 해당 사용자가 즐겨찾기한 대여소의 lendplace_id를 먼저 추출한 뒤 이어서, bikestationinformation 테이블에서 조회하고자 하는 대여소의 lendplace_id가 사용자의 즐겨찾기 목록에 포함된 대여소의 lendplace_id와 일치하는지 확인한다. 이 일치 여부를 판단하기 위해 SQL의 IN 연산자를 사용한다.
이 쿼리를 통해 사용자에게 중요한 정보인 즐겨찾기한 대여소의 상세 데이터를 효율적으로 추출할 수 있다.

16

LIMIT

```

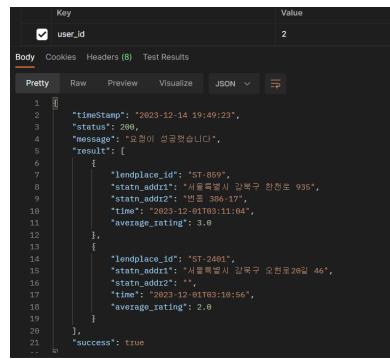
return jdbcTemplate.query(
    // DISTINCT, COALESCE, Subqueries in the FROM clauses, ALL, GROUP BY, ORDER BY, LIMIT, NULL
    "SELECT DISTINCT BS1.lendplace_id, BS1.statn_addr1, BS1.statn_addr2, UL.time, COALESCE(AVG(BR.rating), 0) AS average_rating "
    "FROM (* "
    "    SELECT departure_station AS station, departure_time AS time "
    "    FROM userLog "
    "    WHERE user_id = ? AND departure_time IS NOT NULL "
    "    UNION ALL "
    "    SELECT arrival_station AS station, arrival_time AS time "
    "    FROM userLog "
    "    WHERE user_id = ? AND arrival_time IS NOT NULL "
    ") AS UL "
    "JOIN bikestationinformation BST ON BS1.lendplace_id = UL.station "
    "LEFT JOIN bikestationrating BR ON BS1.lendplace_id = BR.lendplace_id "
    "GROUP BY BS1.lendplace_id, BS1.statn_addr1, BS1.statn_addr2, UL.time "
    "ORDER BY UL.time DESC "
    "LIMIT 2",
    stationMapper,
    userId,
    userId
);

```

		<pre> Key Value user_id 2 Body Cookies Headers [B] Test Results Pretty Raw Preview Visualize JSON ▾ 1 2 "timeStamp": "2023-12-14 19:49:23", 3 "status": 200 4 "message": "성공하였습니다", 5 "result": [6 { 7 "lendPlace_id": "ST-059", 8 "statn_addr1": "서울특별시 강북구 한천로 935", 9 "statn_addr2": "반포 386-17", 10 "time": "2023-12-01T03:11:04", 11 "average_rating": 3.0 12 }, 13 { 14 "lendPlace_id": "ST-2401", 15 "statn_addr1": "서울특별시 강북구 오현로 20길 46", 16 "statn_addr2": "", 17 "time": "2023-12-01T03:10:56", 18 "average_rating": 2.0 19 } 20], 21 "success": true </pre>
<p>이 또한 사용자의 편의를 위해 메인 화면에 사용자가 최근 이용한 두 개의 대여소를 출력하기 위해 사용하는 Query이다.</p> <p>우선, userlog table을 통해 해당 사용자의 log를 조회한 후, arrival_time 또는 해당 정보가 존재하지 않는 경우 departure_time을 이용하여 가장 최근에 이용한 겹치지 않는 두 대여소를 조회한다. 이 정보와 대여소 정보를 함께 얻기 위해 이 둘을 JOIN하는데, 대여소 정보와 함께 대여소 별점 정보 또한 조회하기 위해 bikestationrating table 또한 JOIN한다. 이때 최대 2개로 제한하기 위해 LIMIT을 사용하게 된다.</p>		

No	SQL 문 종류	SQL 문
17	AVG	<pre> // IN, COUNT SELECT BSI.*, (SELECT COUNT(*) FROM bike WHERE bike.lendPlace_id = BSI.lendPlace_id) AS total_bikes, (SELECT COUNT(*) FROM bike WHERE bike.lendPlace_id = BSI.lendPlace_id AND bike.use.status = 0 AND bike.bike.status = 1) AS usable_bikes, (SELECT COALESCE(AVG(rating), 0) FROM bikestationrating WHERE lendPlace_id = BSI.lendPlace_id) AS average_rating, TRUE AS isFavorite FROM bikestationinformation BSI WHERE BSI.lendPlace_id IN (SELECT lendPlace_id FROM favorite WHERE user_id = ?), favoriteMapper, user </pre> <pre> { "lendplace_id": "ST-1280", "statn_addr1": "서울특별시 노원구 화랑로 429", "statn_addr2": "동해문화예술관앞", "startn_lat": 37.619381, "startn_lnt": 127.057854, "max_stands": 20.0, "station_status": 1, "total_bikes": 0, "usable_bikes": 0, "isFavorite": true, "average_rating": 4.5 } </pre> <p>사용자가 본인의 즐겨찾기한 모든 대여소와 그 정보를</p>

		<p>조회할 때, 대여소 정보를 출력하기 위한 쿼리이다. <code>user_id</code>를 이용하여 <code>favorite</code> table에서 해당 <code>user_id</code>의 사용자가 즐겨찾기한 대여소의 <code>lendplace_id</code>들을 추출한 후, <code>bikestationinformation</code> 테이블에서 <code>lendplace_id</code>를 이용하여 찾는다. 이렇게 해당하는 모든 대여소에 대한 총 자전거 수, 사용 가능한 자전거 수, 평균 평점 등을 함께 반환한다. 이때, 평균 평점은 각 대여소의 <code>rating</code> 값을 <code>AVG</code>를 이용하여 구하게 되는 것을 확인할 수 있다.</p>
--	--	---

No	SQL 문 종류	SQL 문
18	DISTINCT	<pre> return jdbcTemplate.query("SELECT DISTINCT BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time, COALESCE(AVG(BR.rating), 0) AS average_rating " + "FROM (" + " SELECT departure_station AS station, departure_time AS time " + " FROM userlog " + " WHERE user_id = ? AND departure_time IS NOT NULL " + " UNION ALL " + " SELECT arrival_station AS station, arrival_time AS time " + " FROM userlog " + " WHERE user_id = ? AND arrival_time IS NOT NULL " + ") AS UL " + "JOIN bikestationinformation BSI ON BSI.lendplace_id = UL.station " + "LEFT JOIN bikestationrating BR ON BSI.lendplace_id = BR.lendplace_id " + "GROUP BY BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time " + "ORDER BY UL.time DESC " + "LIMIT 2", stationMapper, userId, userId); </pre>  <p>메인 화면에 사용자가 최근 이용한 두 개의 대여소를 출력하기 위해 사용하는 쿼리이다. 가장 최근 대여소 두 개를 조회하기 위해 <code>user id</code>와 <code>userlog</code> table을 <code>join</code>한 뒤, 해당하는 <code>userlog</code>의 데이터 중 <code>arrival_time</code> 또는 <code>departure_time</code>을 이용하여 가장 최근에 이용했던, 중복되지 않는 두 대여소를 조회한다. 이 정보와 함께 대여소 정보 등을 위해 추가 <code>JOIN</code>을 수행하고 있는 것 또한 확인할 수 있다. 이때, 내부 <code>FROM</code> 절에서 <code>arrival_time</code> 또는 <code>departure_time</code>을 이용하여 겹치는 대여소 정보도 포함해 조회해 주었기 때문에 출력할 때 <code>DISTINCT</code>를</p>

		이용하여 겹치는 대여소는 제거하고, 중복되지 않는 대여소를 최대 두 개 사용자에게 제공하게 된다.
--	--	--

No	SQL 문 종류	SQL 문
19	COUNT	<pre> IN, COUNT "SELECT RS1.*, " + " + (SELECT COUNT(*) FROM bike WHERE bike.lendplace_id = RS1.lendplace_id) AS total_bikes, " + " + (SELECT COUNT(*) FROM bike WHERE bike.lendplace_id = RS1.lendplace_id AND bike.use_status = 0 AND bike.bike_status = 1) AS usable_bikes, " + " + (SELECT COALESCE(Avg(rating), 0) FROM bikestationrating WHERE lendplace_id = RS1.lendplace_id) AS average_rating, " + " + TRUE AS isFavorite " + "FROM bikestationinformation RS1 " + "WHERE RS1.lendplace_id IN (SELECT lendplace_id FROM favorite WHERE user_id = ?)". favoriteMapper, userId </pre> <p> { "lendplace_id": "ST-1280", "statn_addr1": "서울특별시 노원구 화랑로 429", "statn_addr2": "동해문화예술관앞", "startn_lat": 37.619381, "startn_lnt": 127.057854, "max_stands": 20.0, "station_status": 1, "total_bikes": 0, "usable_bikes": 0, "isFavorite": true, "average_rating": 4.5 }, </p> <p>사용자가 즐겨찾기한 대여소를 조회할 때 해당 Query를 이용하게 된다. user_id를 활용하여 favorite 테이블에서 사용자가 즐겨찾기한 대여소의 lendplace_id들을 추출한 뒤 bikestationinformation 테이블에서 사용자의 즐겨찾기 목록에 포함된 대여소의 lendplace_id와 일치하는 대여소들을 찾게 된다. 이를 이용하여 대여소의 세부 정보를 출력한다. 이를 위해 bike 테이블에서 COUNT 함수를 사용하여 각 대여소의 lendplace_id에 해당하는 row의 수, 즉 해당 대여소에 존재하는 자전거의 수를 계산하여 반환하게 된다. 이 쿼리를 통해 사용자에게 중요한 정보인 즐겨찾기한 대여소의 상세 데이터를 효율적으로 추출할 수 있다.</p>

No	SQL 문 종류	SQL 문
----	----------	-------

20

SUM

```

return jdbcTemplate.queryForObject(
    // WITH, SUM
    "WITH BikeCount AS (" +
        "SELECT lendplace_id, COUNT(*) AS total_bikes, " +
        "SUM(CASE WHEN use_status = 0 AND bike_status = 1 THEN 1 ELSE 0 END) AS usable_bikes " +
        "FROM bike " +
        "GROUP BY lendplace_id ) " +
    "SELECT BS.*, COALESCE(BC.total_bikes, 0) AS total_bikes, (BS.max_stands - COALESCE(BC.total_bikes, 0)) AS empty_stands, " +
    "COALESCE(BC.usable_bikes, 0) AS usable_bikes " +
    "FROM bikeStationInformation BS " +
    "LEFT JOIN BikeCount BC ON BS.lendplace_id = BC.lendplace_id " +
    "WHERE BS.lendplace_id = ? ", bikeMapper, lendplaceId
);

```

The screenshot shows a table with two rows:

Key	Value
<input checked="" type="checkbox"/> lendplace_id	ST-10
Key	Value

Below the table, there is a JSON response:

```

{
  "Body": {
    "Pretty": [
      1: "timeStamp": "2023-12-15 02:14:34",
      2: "status": 200,
      3: "message": "요청이 성공했습니다.",
      4: "result": [
        5: {
          6: "lendplace_id": "ST-10",
          7: "statn_addr1": "서울특별시 마포구 양화로 93",
          8: "statn_addr2": "427",
          9: "startn_lat": 37.552746,
          10: "startn_lnt": 126.918617,
          11: "max_stands": 20.0,
          12: "station_status": 1,
          13: "total_bikes": 3,
          14: "usable_bikes": 1,
          15: "empty_stands": 17
        },
        16: {
          17: "success": true
        }
      ]
    ],
    "Cookies": null,
    "Headers": [
      "Content-Type": "application/json; charset=UTF-8"
    ],
    "Test Results": null
  },
  "Cookies": null,
  "Headers": [
    "Content-Type": "application/json; charset=UTF-8"
  ],
  "JSON": "true"
}

```

하나의 대여소 정보를 조회할 때 사용하는 쿼리로, 해당 쿼리는 `lendplace_id`를 이용해 대여소와 관련 정보들을 출력하기 위해 사용한다. 여기서 `WITH`를 사용하여 `lendplace_id`의 대여소에 존재하는 `bike`의 정보들을 얻게 되는데, `WITH` 내에서 `bike_status`가 1로 이용 가능한 자전거이며 `use_status`가 0이어서 아무도 이용하지 않고 있는, 이용 가능한 자전거 수를 계산하기 위해 `SUM`과 그 내부에 `CASE`를 사용하여 이용 가능한 자전거이면 각각 1을 더할 수 있도록 해 주었다.

No	SQL 문 종류	SQL 문
21	GROUP BY	<pre> return jdbcTemplate.query("SELECT B.id, B.category_id, B.user_id, B.views, B.title, B.notice, B.created_at, COUNT(C.id) as comment_count " + "FROM comment C RIGHT OUTER JOIN board B " + "ON C.write_id = B.id AND C.category_id = B.category_id " + "WHERE B.category_id=? " + "GROUP BY B.id " + "ORDER BY B.created_at DESC" , boardCommentMapper, category_id); </pre>

		<pre> 1 "timeStamp": "2023-12-14 20:19:29", 2 "status": 200, 3 "message": "요청이 성공했습니다", 4 "result": [5 { 6 "id": 25, 7 "category_id": 3, 8 "user_name": "김병준", 9 "views": 0, 10 "title": "[ST-2074] 질문대 입 대여소에 자전거가 없어요ㅠㅠ", 11 "notice": false, 12 "created_at": "2023-12-03T19:57:13", 13 "commentCount": 0 14 } 15] </pre>
		<p>특정 category에 해당하는 게시글의 title들을 조회할 때 사용한다.</p> <p>comment table과 board table을 join한 뒤, 해당하는 category에 대한 board 정보들과 그 board의 댓글 수를 출력하게 된다.</p> <p>comment table과 board table을 board의 id와 category를 이용하여 JOIN할 때 comment가 존재하지 않는 board라도 board 정보를 출력하게 하기 위해 RIGHT OUTER JOIN을 이용한다. 이러한 join을 수행하게 되면 게시글과 게시글에 해당하는 댓글 정보가 모두 출력되어야 하기 때문에 동일한 게시글이 여러번 조회되게 되는데, 이를 해결하기 위해 GROUP BY를 이용하여 동일한 게시글들을 묶어 주었다.</p>

No	SQL 문 종류	SQL 문
22	CASE	<pre> // VIEW, CASE "CREATE VIEW IF NOT EXISTS BikeCounts AS " + "SELECT <u>lendplace_id</u>, COUNT(*) AS total_bikes, " + "COUNT(CASE WHEN use_status=0 AND bike_status=1 THEN 1 END) AS usable_bikes " + "FROM bike " + "GROUP BY <u>lendplace_id</u>"); </pre>

	lendplace_id	total_bikes	usable_bikes
▶	ST-10	3	1
	ST-1000	1	1
	ST-103	1	1
	ST-1032	1	1
	ST-1204	1	1
	ST-1280	6	5
	ST-1322	5	5
	ST-1327	1	1
	ST-2060	3	3
	ST-2061	1	1
	ST-2074	1	0

No	SQL 문 종류	SQL 문
23	NESTED QUERY	<pre>// NESTED QUERY bike_id = JdbcTemplate.queryForObject(sql: "SELECT id FROM (SELECT id FROM bike WHERE lendplace_id = ? + "AND bike_status = 1 AND use_status = 0) AS AvailableBikes + "ORDER BY id LIMIT 1", new Object[]{departureStation}, Integer.class);</pre>

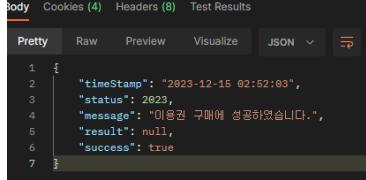
		<pre> 1 "user_id": 37, 2 "departure_station": "ST-2074" 3 4 </pre> <p>pretty Cookies Headers (8) Test Results</p> <p>Raw Preview Visualize JSON</p> <pre> 1 { 2 "timeStamp": "2023-12-14 19:29:07", 3 "status": 2026, 4 "message": "자전거 대여에 성공하였습니다.", 5 "result": null, 6 "success": true 7 </pre>
자전거 대여 과정에서 Nested Query를 사용하여 BIKE table에서 자전거의 상태가 활성화이면서 사용하고 있고 있다는 조건을 만족하는 자전거들을 찾고, 그 중 ID가 가장 작은 하나의 자전거를 선택함으로써 자전거 대여 시 어떤 자전거가 선택되어야 하는지 ID를 반환해주었다.		

No	SQL 문 종류	SQL 문
24	EXISTS	<pre> BoardDto.BoardWithLike boardWithLikes = jdbctemplate.queryForObject(// EXISTS "SELECT B.* , " + "(SELECT COUNT(*) FROM board_like WHERE liked_id = B.id) AS likes_count , " + "EXISTS(SELECT 1 FROM board_like WHERE liked_id = B.id AND user_id = ?) AS user_liked " + "FROM board B WHERE B.id = ?", boardWithLikesMapper, userId, id); </pre> <p>"timeStamp": "2023-12-15 02:41:29", "status": 200, "message": "요청이 성공했습니다", "result": { "id": 1, "category_id": 2, "user_name": "관리자", "user_id": 1, "views": 3, "title": "출퇴근길 지켜야 할 자전거 애티켓", "content": "<p>출퇴근길 지켜야 할 자전거 애티켓</p><p>따를라면 이것도 라이딩을 위해
</p><p>모두 함께 배려하고 안전주행해요</p>", "notice": true, "file_name": "자전거 주행 애티켓(평일용).jpg", "url": "9a8d891f-880e-4371-b349-be1fe00e628_자전거 주행 애티켓(평일용).jpg", "likeCount": 0, "userLiked": false, "created_at": "2023-12-03T03:00:56", "updated_at": "2023-12-03T03:00:56", "commentDtoList": [] }, "success": true </p> <p>한 사용자가 게시글을 확인할 때 사용하는 쿼리이다. board table에서 특정 게시물에 대한 정보를 조회하고, 그 게시물에 대한 좋아요 수와 이 게시물을 조회한 사용자가 해당 게시물을 좋아요 했는지 여부를 함께 반환하게 된다. 이때 board_like table을 이용하여 해당 id의 게시글에</p>

		<p>좋아요 한 사람들 중 해당 사용자 id가 존재한다면 이 게시글에 좋아요를 누른 것이기 때문에 EXISTS를 이용하여 해당 정보를 반환하게 된다.</p>
--	--	---

No	SQL 문 종류	SQL 문
25	UNIQUE	<pre>CREATE TABLE if not exists `board_like` (`user_id` int NOT NULL, `category_id` int NOT NULL, `liked_id` int NOT NULL, UNIQUE (`user_id`, `liked_id`), CONSTRAINT `board_like_ibfk_1` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`) ON DELETE CASCADE ON UPDATE CASCADE, CONSTRAINT `board_like_ibfk_2` FOREIGN KEY (`liked_id`, `category_id`) REFERENCES `board` (`id`, `category_id`) ON DELETE CASCADE ON UPDATE CASCADE);</pre> <p>board_like와 comment_like table의 user_id와 liked_id의 조합의 경우, primary key는 아니지만 한 게시물에 한 사람은 단 한 번만 좋아요를 누를 수 있기 때문에 Unique해야 한다. 따라서 board_like table을 create할 때 UNIQUE를 이용해 해당 조합이 하나씩만 존재할 수 있게 해 주었다.</p>

No	SQL 문 종류	SQL 문
26	Subqueries in the FROM clauses	<pre>return jdbcTemplate.query(// DISTINCT, COALESCE, Subqueries in the FROM clauses, ALL, GROUP BY, ORDER BY, LIMIT, NULL "SELECT DISTINCT BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time, COALESCE(AVG(BR.rating), 0) AS average_rating " "FROM (" + "SELECT departure_station AS station, departure_time AS time " + "FROM userlog " + "WHERE user_id = ? AND departure_time IS NOT NULL " + "UNION ALL " + "SELECT arrival_station AS station, arrival_time AS time " + "FROM userlog " + "WHERE user_id = ? AND arrival_time IS NOT NULL " + ") AS UL " + "JOIN bikestationinformation BSI ON BSI.lendplace_id = UL.station " + "LEFT JOIN bikestationrating BR ON BSI.lendplace_id = BR.lendplace_id " + "GROUP BY BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time " + "ORDER BY UL.time DESC " + "LIMIT 2", stationMapper, userId, userId);</pre>

		<table border="1"> <thead> <tr> <th>Key</th><th>Value</th></tr> </thead> <tbody> <tr> <td><input checked="" type="checkbox"/> userId</td><td>1</td></tr> <tr> <td><input checked="" type="checkbox"/> ticketId</td><td>2</td></tr> <tr> <td><input type="checkbox"/></td><td></td></tr> <tr> <td>Key</td><td>Value</td></tr> </tbody> </table>  <p>body Cookies (4) Headers (8) Test Results</p> <p>Pretty Raw Preview Visualize JSON ↗</p> <pre> 1 { 2 "timeStamp": "2023-12-15 02:52:03", 3 "status": 2023, 4 "message": "이용권 구매에 성공하였습니다.", 5 "result": null, 6 "success": true 7 }</pre>	Key	Value	<input checked="" type="checkbox"/> userId	1	<input checked="" type="checkbox"/> ticketId	2	<input type="checkbox"/>		Key	Value
Key	Value											
<input checked="" type="checkbox"/> userId	1											
<input checked="" type="checkbox"/> ticketId	2											
<input type="checkbox"/>												
Key	Value											
		<p>사용자가 이전에 사용한 자전거 대여소의 정보를 반환하는 기능 수행 시에 FROM 절로 subquery를 생성하여 USERLOG table에서 사용자 id에 대한 레코드를 필터링하고 departure_time이 NULL이 아니라는 조건에서 사용자의 출발 지점인 deperature_staiion과 출발 시간인 departure_time을 선택한 집합과 동일한 사용자 ID에 대해 arrival_time이 NULL이 아니라는 조건에서 USERLOG table에서 사용자의 도착 지점 arrival_station과 도착 시간 arrival_time을 선택한 집합을 UNION ALL을 통해 결합하여 반환한다.</p>										

No	SQL 문 종류	SQL 문
27	WITH	<pre> return jdbcTemplate.queryForObject(// #ITH, SUM "WITH BikeCount AS (" + " SELECT lendplace_id, COUNT(*) AS total_bikes, " + " SUM(CASE WHEN use_status = 0 AND bike_status = 1 THEN 1 ELSE 0 END) AS usable_bikes " + " FROM bike " + " GROUP BY lendplace_id) " + "SELECT BS.* , COALESCE(BC.total_bikes, 0) AS total_bikes, (BS.max_stands - COALESCE(BC.total_bikes, 0)) AS empty_stands, " + "COALESCE(BC.usable_bikes, 0) AS usable_bikes " + "FROM bikeLocationInformation BS " + "LEFT JOIN BikeCount BC ON BS.lendplace_id = BC.lendplace_id " + "WHERE BS.lendplace_id = ? ", bikeMapper, lendplaceId); </pre>

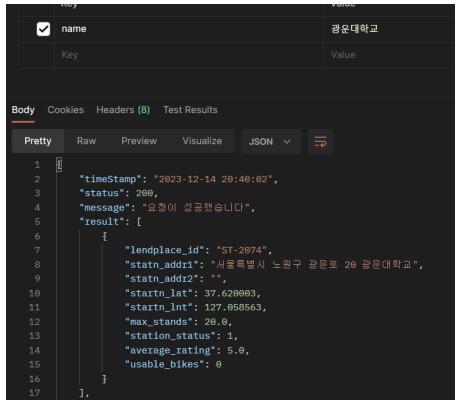
		<p>The screenshot shows a JSON response from a REST API. At the top, there's a table with a single row containing a checked checkbox labeled 'lendplace_id' and the value 'ST-10'. Below this is a 'Body' tab showing the full JSON response:</p> <pre> 1 "timeStamp": "2023-12-15 02:14:34", 2 "status": 200, 3 "message": "요청이 성공했습니다.", 4 "result": { 5 "lendplace_id": "ST-10", 6 "statn_addr1": "서울특별시 마포구 양화로 93", 7 "statn_addr2": "427", 8 "statn_lat": 37.552746, 9 "statn_lng": 126.918617, 10 "max_stands": 26, 11 "station_status": 1, 12 "total_bikes": 3, 13 "usable_bikes": 1, 14 "empty_stands": 17 15 }, 16 "success": true 17 }</pre>
<p>하나의 대여소에 해당하는 세부적인 정보를 확인하기 위한 쿼리로, 대여소에 존재하는 자전거 정보를 얻기 위해 WITH을 이용하고 있다. 이때 WITH는 중간 결과를 생성하는 것으로, lendplace_id를 이용하여 각 대여소에 존재하는 bike정보를 가져오며, 총 bike의 개수, 이용 가능한 자전거의 수 등의 결과를 반환하게 된다.</p>		

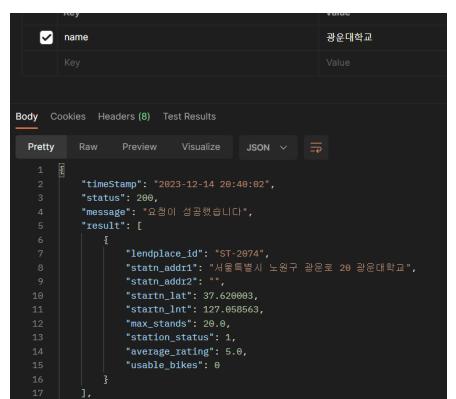
No	SQL 문 종류	SQL 문
28	SCALAR Subqueries	<p>The screenshot shows a JSON response from a REST API. At the top, there's a table with two rows: one checked checkbox labeled 'userId' with value '1' and another checked checkbox labeled 'ticketId' with value '2'. Below this is a 'Body' tab showing the full JSON response:</p> <pre> 1 { 2 "timeStamp": "2023-12-15 02:52:03", 3 "status": 2023, 4 "message": "미용권 구매에 성공하였습니다.", 5 "result": null, 6 "success": true 7 }</pre> <p>scalar subquery를 사용하여 ticked_id에 해당하는 1개의 record로부터 1개의 컬럼인 ticket_price 값만을 반환함으로써 티켓 구매를 진행할 때, 티켓의 가격을</p>

		조회하는데 사용하였다.
--	--	--------------

No	SQL 문 종류	SQL 문
29	VIEW	<pre>// VIEW, CASE "CREATE VIEW IF NOT EXISTS BikeCounts AS " + "SELECT <u>lendplace_id</u>, COUNT(*) AS total_bikes, " + "COUNT(CASE WHEN use_status=0 AND bike_status=1 THEN 1 END) AS usable_bikes " + "FROM bike " + "GROUP BY <u>lendplace_id</u>");</pre> <p>1 ● SELECT * FROM dbdb.BikeCount;</p> <p>해당 VIEW는 각 대여소에 존재하는 자전거 관련 정보 결과를 저장하고 있는 가상 테이블이다. 각 대여소에 존재하는 자전거 정보는 자주 사용되기 때문에 VIEW로 생성하였으며, CREATE VIEW와 IF NOT EXISTS를 이용하여 BikeCounts VIEW를 존재하지 않을 경우에만 생성할 수 있도록 하였다.</p>

No	SQL 문 종류	SQL 문
30	STRING OPERATION	<pre>String likePattern = "%" + name + "%"; return jdbcTemplate.query(// STRING OPERATION.likePattern), LIKE "SELECT BSI.*, COALESCE(AVG(BSR.rating), 0) AS average_rating " + // 평균 평점을 계산합니다. "FROM bikestationrating BSR ON BSI.lendplace_id = BSR.lendplace_id " + // bikestationrating과 조인합니다. "LEFT JOIN bikestationrating BSR ON BSI.lendplace_id = BSR.lendplace_id " + BSR.rating은 조인합니다. "WHERE BSI.state_and1 LIKE ? OR BSI.state_and2 LIKE ? " + "GROUP BY BSI.lendplace_id", // lendplace_id에 대해 그룹화합니다. bikeMapper, likePattern, likePattern);</pre>

		 <p>The screenshot shows a JSON response from a REST API. The 'Body' tab is selected, displaying the following data:</p> <pre> 1 "timeStamp": "2023-12-14 20:40:02", 2 "status": 200, 3 "message": "모델이 성공했습니다", 4 "result": [5 { 6 "lendplace_id": "ST-2074", 7 "statn_addr1": "서울특별시 노원구 광운로 20 광운대학교", 8 "statn_addr2": "", 9 "startn_lat": 37.628003, 10 "startn_lnt": 127.058563, 11 "max_stands": 20.0, 12 "station_status": 1, 13 "average_rating": 5.0, 14 "usable_bikes": 0 15 } 16], 17 </pre>
		<p>사용자가 검색한 단어를 이용하여 대여소를 검색할 때 사용하는 query이다. 이를 위해 대여소의 두개의 addr 중 하나라도 사용자가 입력한 단어를 포함한다면 해당 대여소의 정보를 반환해주게 된다. 이때, 사용자가 입력한 단어와 정확히 일치하는 것 뿐만 아니라 사용자가 입력한 단어를 포함하는 대여소도 함께 검색할 수 있도록 하기 위해 %와일드카드를 이용하여 검색어 앞뒤로 어떤 단어가 추가되어도 검색되도록 구현하였다.</p>

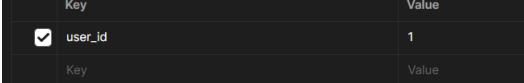
No	SQL 문 종류	SQL 문
31	LIKE	 <p>The screenshot shows a JSON response from a REST API. The 'Body' tab is selected, displaying the following data:</p> <pre> String likePattern = "%" + name + "%"; return jdbcTemplate.query(// STRING OPERATION(likePattern), LIKE "SELECT BSI.* , COALESCE(AVG(BSR.rating), 0) AS average_rating " + // 평균 평점은 계산합니다. "FROM bikestationinformation BSI " + "LEFT JOIN bikestationrating BSR ON BSI.lendplace_id = BSR.lendplace_id " + // bikestationrating은 조인합니다. "WHERE BSI.statn_addr1 LIKE ? OR BSI.statn_addr2 LIKE ? " + "GROUP BY BSI.lendplace_id", // lendplace_id에 대해 그룹화합니다. bikeMapper, likePattern, likePattern); </pre>  <p>The screenshot shows a JSON response from a REST API. The 'Body' tab is selected, displaying the following data:</p> <pre> 1 "timeStamp": "2023-12-14 20:40:02", 2 "status": 200, 3 "message": "모델이 성공했습니다", 4 "result": [5 { 6 "lendplace_id": "ST-2074", 7 "statn_addr1": "서울특별시 노원구 광운로 20 광운대학교", 8 "statn_addr2": "", 9 "startn_lat": 37.628003, 10 "startn_lnt": 127.058563, 11 "max_stands": 20.0, 12 "station_status": 1, 13 "average_rating": 5.0, 14 "usable_bikes": 0 15 } 16], 17 </pre>

이 또한 사용자가 검색한 단어를 이용하여 대여소를 검색할 때 사용하는 **query**이다. 이 경우에도 대여소의 두개의 **addr**과 사용자가 입력한 단어를 비교하여 둘 중

		어디에나 사용자 입력 단어를 포함한다면 해당하는 대여소의 정보를 반환한다. 이때 사용자가 검색한 단어와 대여소의 주소 정보는 둘 다 STRING으로, STRING을 비교하기 위해 LIKE를 사용한다.
--	--	---

No	SQL 문 종류	SQL 문
32	ORDER BY	<p>return jdbcTemplate.query("SELECT B.id, B.category_id, B.user_id, B.views, B.title, B.notice, B.created_at, COUNT(C.id) as comment_count " + "FROM comment C RIGHT OUTER JOIN board B " + "ON C.write_id = B.id AND C.category_id = B.category_id " + "WHERE B.category_id=? " + "GROUP BY B.id " + "ORDER BY B.created_at DESC" , boardCommentMapper, category_id);</p> <pre> 1 2 "timeStamp": "2023-12-14 20:19:29", 3 "status": 200, 4 "message": "요청이 성공했습니다", 5 "result": [6 { 7 "id": 25, 8 "category_id": 3, 9 "user_name": "김별준", 10 "views": 0, 11 "title": "[ST-2074] 칼문대 앞 대여소에 자전거가 없어요ㅠㅠ", 12 "notice": false, 13 "created_at": "2023-12-03T19:57:13", 14 "commentCount": 0 15 } </pre> <p>이 또한 특정 category에 해당하는 게시글의 title들을 조회할 때 사용한다. comment table과 board table을 category와 id로 join한 뒤, 해당하는 board 정보들과 그 board의 댓글 수를 출력하게 된다. 모든 board의 일부 정보들을 출력할 때, 최신에 작성한 글이 가장 상위에 오도록 해야 하기 때문에 created_at 정보가 작은 순서로 정렬하여 제공하는 것을 확인할 수 있다.</p>

No	SQL 문 종류	SQL 문
----	----------	-------

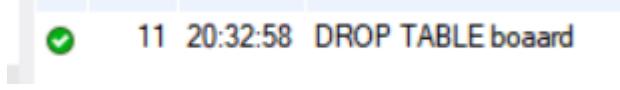
		<pre> return jdbcTemplate.query("SELECT BSI.* " + "FROM bikestationinformation BSI " + "JOIN (" + " SELECT UL.departure_station " + " FROM userlog UL " + " WHERE UL.user_id = ? " + " AND UL.arrival_station IS NULL " + ") AS DepartureInfo ON BSI.lendplace_id = DepartureInfo.departure_station", stationMapper, userId) </pre>
33	NULL	 <p>Body Cookies Headers (8) Test Results</p> <p>Pretty Raw Preview Visualize JSON</p> <pre> 1: { 2: "timeStamp": "2023-12-15 03:07:21", 3: "status": 200, 4: "message": "요청이 성공했습니다", 5: "result": [], 6: "success": true 7: } </pre>
		<p>해당 쿼리는 사용자가 아직 반납하지 않은 자전거가 존재한다면, 그 자전거의 출발 대여소 정보를 조회하게 된다. 이는 사용자가 현재 대여중인 상태이면 추가 대여를 막기 위해 사용된다.</p> <p>userlog table에서 해당하는 사용자의 log 중, arrival_station이 NULL인, 즉 반납을 하지 않은 log가 존재한다면 그 log의 departure_station 정보를 추출하여 해당 대여소의 정보를 조회하게 된다. 이때, NULL인지 아닌지를 확인할 때 IS NULL이 사용된다.</p>

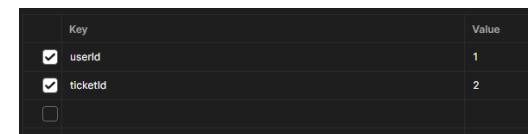
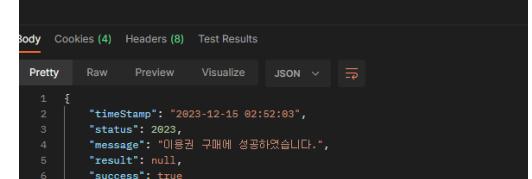
No	SQL 문 종류	SQL 문																		
34	TRIGGER	<pre> DELIMITER // CREATE TRIGGER AfterCouponUpdate AFTER UPDATE ON coupon FOR EACH ROW BEGIN IF NEW.is_used = 1 THEN INSERT INTO paymenthistory (user_id, ticket_id, is_used, registration_at) VALUES (@userId, NEW.ticket_id, 0, NOW()); END IF; END; </pre> <table border="1"> <tr> <td>Name</td> <td>Event</td> <td>Timing</td> <td>Created</td> <td>SQL Mode</td> <td>Definer</td> <td>Client Character</td> <td>Connection Collat...</td> <td>Database Collat...</td> </tr> <tr> <td>AfterCouponUpdate</td> <td>UPDATE</td> <td>AFTER</td> <td>2023-12-03 03:4...</td> <td>STRICT_TRANS...</td> <td>admin@%</td> <td>utf8mb4</td> <td>utf8mb4_genera...</td> <td>utf8mb4_genera...</td> </tr> </table> <p>AfterCouponUpdate라는 이름의 트리거를 생성하여 COUPON table의 특정 행에 대해 쿠폰이 사용되어 사용 여부인 is_used가 1이된 경우에 사용된 쿠폰에 대한</p>	Name	Event	Timing	Created	SQL Mode	Definer	Client Character	Connection Collat...	Database Collat...	AfterCouponUpdate	UPDATE	AFTER	2023-12-03 03:4...	STRICT_TRANS...	admin@%	utf8mb4	utf8mb4_genera...	utf8mb4_genera...
Name	Event	Timing	Created	SQL Mode	Definer	Client Character	Connection Collat...	Database Collat...												
AfterCouponUpdate	UPDATE	AFTER	2023-12-03 03:4...	STRICT_TRANS...	admin@%	utf8mb4	utf8mb4_genera...	utf8mb4_genera...												

		정보를 PAYMENTHISTORY table에 자동으로 기록하는 용도로 사용하였다.
--	--	--

No	SQL 문 종류	SQL 문
35	ASSERTION	<pre>public void run(String... args) { // 트리거 존재 여부 확인 String checkTrigger = "SELECT COUNT(*) FROM INFORMATION_SCHEMA.TRIGGERS WHERE TRIGGER_SCHEMA = ? AND TRIGGER_NAME = ?"; Integer count = jdbcTemplate.queryForObject(checkTrigger, new Object[]{schema, "assertion_user"}, Integer.class); // 트리거가 존재하지 않으면 생성 if (count == null count == 0) { String createTrigger = "CREATE TRIGGER assertion_user " + "BEFORE INSERT ON user " + "FOR EACH ROW " + "BEGIN " + " IF NEW.user_type NOT IN (0, 1, 2) THEN " + " SIGNAL SQLSTATE '45000' " + " SET MESSAGE_TEXT = 'user_type은 0, 1 or 2이어야 합니다.' "; + " END IF; " + "END"; jdbcTemplate.execute(createTrigger); } }</pre> <p>The screenshot shows the 'Triggers' tab in MySQL Workbench. A new trigger named 'assertion_user' is being created. It is defined as an 'INSERT' trigger on the 'user' table, running before each row is inserted. The trigger body contains logic to check if the 'user_type' value is not one of 0, 1, or 2, and if so, to signal an error with the message 'user_type은 0, 1 or 2이어야 합니다.'.</p>

No	SQL 문 종류	SQL 문
36	CONSTRAINT	<pre>CREATE TABLE if not exists 'coupon' ('value' varchar(255) UNIQUE NOT NULL, 'is_used' int NULL DEFAULT NULL, 'ticket_id' int NOT NULL, PRIMARY KEY ('value'), CONSTRAINT 'coupon_ibfk_1' FOREIGN KEY ('ticket_id') REFERENCES 'ticket' ('id') ON DELETE CASCADE ON UPDATE CASCADE);</pre> <p>The screenshot shows the 'Foreign keys' tab in MySQL Workbench. A new foreign key constraint named 'coupon_ibfk_1' is being created. It defines a relationship between the 'ticket_id' column in the 'coupon' table and the 'id' column in the 'ticket' table. The constraint uses the 'ON DELETE CASCADE ON UPDATE CASCADE' rule.</p>

No	SQL 문 종류	SQL 문
37	DROP	  <p>DROP 경우, 잘못 생성한 TABLE이나 SCHEMA를 삭제하기 위해 사용하였다. Spring에서 직접적으로 사용하지는 않았으며 위와 같이 오타를 낸 booard table을 삭제하는 것을 확인할 수 있다. 또한, 서버를 실행할 때 해당 Table이 존재하지 않으면 Create를 자동으로 하도록 설정해 두었기 때문에 데이터를 비우고 싶을 때 DROP SCHEMA를 수행하고 서버를 다시 실행하기도 하였다.</p>

No	SQL 문 종류	SQL 문
38	BETWEEN	<pre>// BETWEEN String checkUserBalance = "SELECT COUNT(*) FROM user " "WHERE id = ? AND total_money BETWEEN ? AND 99999999"; Integer validUser = jdbcTemplate.queryForObject(checkUserBalance, new Object[]{userId, ticketPrice}, Integer.class);</pre>   <p>USER table에서 특정 사용자의 id와 매개변수로 주어진 userId와 일치하고, 해당 사용자의 잔액인 total_money가 티켓 가격인 ticketPrice와 99999999 사이에 있는지 확인을 진행한다. 즉, 티켓을</p>

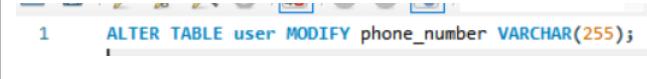
		구매할 만큼의 잔액을 소유하고 있는지 판단하기 위해 BETWEEN을 사용하였다.
--	--	---

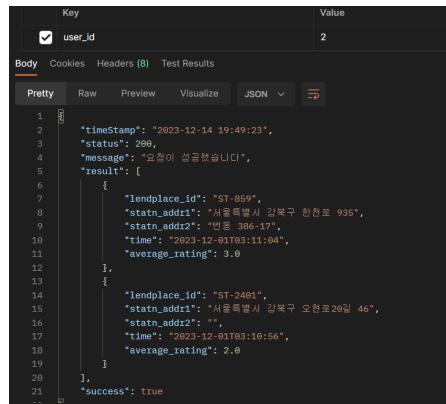
No	SQL 문 종류	SQL 문
39	BASIC SELECT	<pre> return jdbcTemplate.query("SELECT * FROM bike " + "ORDER BY id DESC", bikeMapper); </pre> <pre> 1 ↴ { 2 "timeStamp": "2023-12-15 03:15:54", 3 "status": 200, 4 "message": "요청이 성공했습니다", 5 "result": [6 { 7 "id": 57, 8 "lendplace_id": "ST-2074", 9 "use_status": 1, 10 "bike_status": 1 11 }, 12 { 13 "id": 56, 14 "lendplace_id": "ST-10", 15 "use_status": 0, 16 "bike_status": 1 17 }] </pre> <p>이 경우, 기본 SELECT를 통해 정보를 조회하는 쿼리이다. 즉 bike table에 존재하는 모든 정보를 조회하게 되며, 결과를 조회한 뒤 id로 정렬만 수행하게 된다.</p>

No	SQL 문 종류	SQL 문
40	LOAD DATA INFILE	<pre> 1 LOAD DATA LOCAL INFILE './src/main/resources/lendplace.csv' 2 INTO TABLE bikestationinformation 3 FIELDS TERMINATED BY ',' 4 ENCLOSED BY "" 5 LINES TERMINATED BY '\r\n' 6 IGNORE 1 LINES 7 (lendplace_id, statn_addr1, statn_addr2, startn_lat, startn_lnt) 8 SET max_stands = 20, 9 station_status = 1; </pre>

lendplace_id	statn_addr1	statn_addr2	statn_lat	statn_lnt	max_stands	station_status
ST-10	서울특별시 마포구 양화로 93	427	37.552746	126.918617	20	1
ST-100	서울특별시 광진구 아차산로 262	더샵스타시티 C동 앞	37.536667	127.073593	20	1
ST-1000	서울특별시 광진구 신길동 236	서부식자마트 건너편	37.51038	126.866798	20	1
ST-1001	서울특별시 광진구 남부순환로 470	서서울호수공원	0	0	20	1
ST-1002	서울특별시 양천구 목동들로 216-6	서울시 도로환경관리센터	37.5299	126.876541	20	1
ST-1003	서울특별시 양천구 목동들로 59	신길동 이마트	37.539551	126.828204	20	1
ST-1004	서울특별시 양천구 신정아리원로50	신정아리원아우스314동	37.514094	126.831001	20	1
ST-1005	서울특별시 양천구 신정동 210-8	신분리공원 입구	37.51395	126.856056	20	1
ST-1006	서울특별시 양천구 목동중앙로 70 서울영도초...	37.536377	126.871513	20	1	
ST-1007	서울특별시 양천구 남부순환로 70길 11-9	오수길공원	37.52219	126.83677	20	1
ST-1008	서울특별시 양천구 신길로 894	강수공원	37.52256	126.84948	20	1
ST-1009	서울특별시 양천구 신길동 263	신월 정소년 문화센터	0	0	20	1
ST-1010	서울특별시 광진구 목동로 596	신자초고등학교교자로	37.53297	127.075935	20	1
ST-1011	서울특별시 광진구 신길동 263	신월사거리	37.536201	126.827797	20	1
ST-1012	서울특별시 양천구 가봉공원로 133	으뜸공원	37.536361	126.831711	20	1
	서울특별시 양천구 신정동 258	화곡로 입구 교차로	37.53952	126.825401	20	1

이 경우 따릉이 정보공개 서비스에서 실제 따릉이 대여소 정보나 사용자 이용 데이터 등의 현실 세계의 데이터를 받아온 뒤, 이를 Database에 입력하여 해당 정보를 이용하기 위한 작업이다. 해당 쿼리의 경우, LOCAL FILE인 csv 파일의 형식을 고려하여 원하는 형태로 Database의 Bikestationinformation Table에 입력해 준 결과를 확인할 수 있다.

No	SQL 문 종류	SQL 문																																																
41	ALTER	 <pre>1 ALTER TABLE user MODIFY phone_number VARCHAR(255);</pre> <table border="1"> <thead> <tr> <th>Column</th><th>Type</th><th>Nullable</th><th>Indexes</th></tr> </thead> <tbody> <tr> <td>id</td><td>int(11)</td><td>NO</td><td>PRIMARY</td></tr> <tr> <td>password</td><td>varchar(255)</td><td>NO</td><td></td></tr> <tr> <td>username</td><td>varchar(255)</td><td>NO</td><td></td></tr> <tr> <td>user_type</td><td>int(11)</td><td>NO</td><td></td></tr> <tr> <td>email</td><td>varchar(255)</td><td>YES</td><td></td></tr> <tr> <td>phone_number</td><td>varchar(255)</td><td>YES</td><td></td></tr> <tr> <td>weight</td><td>double</td><td>YES</td><td></td></tr> <tr> <td>age</td><td>int(11)</td><td>NO</td><td></td></tr> <tr> <td>last_accessed_at</td><td>datetime</td><td>YES</td><td></td></tr> <tr> <td>total_money</td><td>int(11)</td><td>NO</td><td></td></tr> <tr> <td>fcm_token</td><td>varchar(255)</td><td>YES</td><td></td></tr> </tbody> </table> <p>ALTER는 데이터베이스의 구조를 변경하는 데 사용되는데, 초기에 user table을 생성할 때 table의 phone_number column의 데이터 형식을 double로 잘못 설정하여 이를 올바르게 변경하기 위해 ALTER를 사용하였다. 즉, user table의 phone_number를 double에서 VARCHAR(255)로 변경한 것이다.</p>	Column	Type	Nullable	Indexes	id	int(11)	NO	PRIMARY	password	varchar(255)	NO		username	varchar(255)	NO		user_type	int(11)	NO		email	varchar(255)	YES		phone_number	varchar(255)	YES		weight	double	YES		age	int(11)	NO		last_accessed_at	datetime	YES		total_money	int(11)	NO		fcm_token	varchar(255)	YES	
Column	Type	Nullable	Indexes																																															
id	int(11)	NO	PRIMARY																																															
password	varchar(255)	NO																																																
username	varchar(255)	NO																																																
user_type	int(11)	NO																																																
email	varchar(255)	YES																																																
phone_number	varchar(255)	YES																																																
weight	double	YES																																																
age	int(11)	NO																																																
last_accessed_at	datetime	YES																																																
total_money	int(11)	NO																																																
fcm_token	varchar(255)	YES																																																

No	SQL 문 종류	SQL 문
42	COALESCE	<pre> return jdbcTemplate.query(// DISTINCT, COALESCE, Subqueries in the FROM clauses, ALL, GROUP BY, ORDER BY, LIMIT, NULL "SELECT DISTINCT BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time, COALESCE(AVG(BR.rating), 0) AS average_rating " + "FROM (" + " SELECT departure_station AS station, departure_time AS time " + " FROM userlog " + " WHERE user_id = ? AND departure_time IS NOT NULL " + " UNION ALL " + " SELECT arrival_station AS station, arrival_time AS time " + " FROM userlog " + " WHERE user_id = ? AND arrival_time IS NOT NULL " + ") AS UL " + "JOIN bikestationinformation BSI ON BSI.lendplace_id = UL.station " + "LEFT JOIN bikestationrating BR ON BSI.lendplace_id = BR.lendplace_id " + "GROUP BY BSI.lendplace_id, BSI.statn_addr1, BSI.statn_addr2, UL.time " + "ORDER BY UL.time DESC " + "LIMIT 2", stationMapper, userId, userId); </pre>  <p>사용자가 최근 이용한 두 개의 대여소를 출력하기 위해 사용하는 Query에서 사용된다. log의 arrival_time을, 아직 도착하지 않아서 arrival_time이 없는 경우 해당 log의 departure_time을 이용하여 가장 최근에 이용한 두 대여소를 조회한다. 대여소 정보와 함께 대여소 별점 정보 또한 조회하기 위해 bikestationrating table을 JOIN하게 되는데, 이때 별점을 조회하게 될 경우 별점이 없는 대여소가 존재해 NULL이 될 수도 있는데, 만약 NULL인 경우 대신 0을 출력하기 위해 COALESCE를 사용해 주었다.</p>

5. 데이터 시각화

5-1. 관리자 대시보드 기능

교통데이터



교통

활용사례(갤러리) 등록 URL 복사 목록 이동

서울시 따릉이 대여소별 대여/반납 승객수 정보

서울시 1일 5분 단위로, 따릉이 대여소별 출발지 도착지 승객수를 확인할 수 있습니다.
따릉이 대여소에 대한 ID 값 정보는 연관데이터인 서울시 따릉이 대여소 마스터 정보를 참고하시기 바랍니다.
* 데이터 적재는 매일 5일전 데이터를 경신합니다.
* 2023년 2월 26일자 데이터부터 철림(집계_기준, 시작_대여소명, 종료_대여소명)이 추가되었습니다.

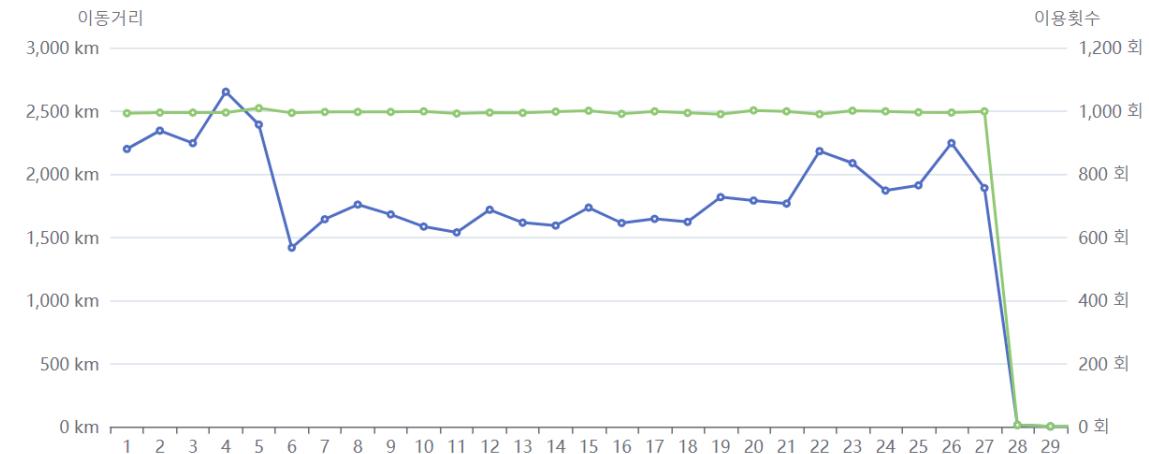
파일내려받기

* 파일에 이상이 있는 경우 '오류신고'를 통해 운영자에게 알려주세요. [오류신고](#)

NO	항목	파일명	용량 (MB)	수정일	내려받기
1	데이터	tpss_bcycl_od_statnhm_20231209.zip	2.1	2023.12.14.	
2	데이터	tpss_bcycl_od_statnhm_20231208.zip	2.67	2023.12.13.	
3	데이터	tpss_bcycl_od_statnhm_20231207.zip	2.44	2023.12.12.	
4	데이터	tpss_bcycl_od_statnhm_20231206.zip	1.78	2023.12.11.	
5	데이터	tpss_bcycl_od_statnhm_20231205.zip	2.51	2023.12.10.	
6	데이터	tpss_bcycl_od_statnhm_20231204.zip	2.22	2023.12.09.	
7	데이터	tpss_bcycl_od_statnhm_20231203.zip	1.34	2023.12.08.	
8	데이터	tpss_bcycl_od_statnhm_20231202.zip	1.52	2023.12.07.	
9	데이터	tpss_bcycl_od_statnhm_20231201.zip	1.93	2023.12.06.	
10	데이터	tpss_bcycl_od_statnhm_20221207.zip	66	2022.08.30.	
11	데이터	tpss_bcycl_od_statnhm_20221206.zip	68.91	2022.08.30.	
12	데이터	tpss_bcycl_od_statnhm_20221205.zip	80.31	2022.08.30.	

월별 이용자 거리 및 횟수

일자별 이동거리 및 횟수





데이터 시각화를 위하여 관리자 대시보드를 만들고, 월별 이용자 이동거리 및 횟수, 요일별 이용자 이동거리 및 횟수를 시각화하였다. 데이터는 서울시 공공자전거 이용 현황 데이터베이스 일부를 프로젝트의 데이터베이스로 들여와 작업하였으며, 특정 기간 별 공공자전거 이용 기록을 가져올 수 있는 쿼리와 API를 작성하였다.

그래프로 시각화하기 위해서 EChart를 이용하였으며, Line graph 두개를 그려 각각 이동 거리, 사용 횟수를 시각화하여 월별 통계량을 확인하고, 각 요일, 어느 날짜에 더 많이 사용하는지 시각화하도록 표현하였다.

5-2. 랭킹 확인 기능

랭킹			
이용시간 행정 이용횟수 행정 이용거리 행정			
등수	이름	이메일	사용시간
1	명준강	kang@naver.com	281
2	방지민	jimin@naver.com	86
3	홍길동	hong@gmail.com	8
4	마이클	mike@naver.com	4
5	하지민	but@gmail.com	1
6	하지현	jihyen@naver.com	1
7	김병준	lucid_dawn@naver.com	1
8	푸바오	fubao@naver.com	1
9	김국민	apple@gmail.com	0

랭킹			
이용시간 행정 이용횟수 행정 이용거리 행정			
등수	이름	이메일	사용횟수
1	방지민	jimin@naver.com	5
2	하지민	but@gmail.com	3
3	홍길동	hong@gmail.com	3
4	마이클	mike@naver.com	2
5	김지민	apple@gmail.com	2
6	병준강	kang@naver.com	2
7	김병준	lucid_dawn@naver.com	1
8	하지현	jihyen@naver.com	1
9	푸바오	fubao@naver.com	1

랭킹			
이용시간 행정 이용횟수 행정 이용거리 행정			
등수	이름	이메일	거리
1	하지현	jihyen@naver.com	14441
2	명준강	kang@naver.com	12078
3	방지민	jimin@naver.com	6944
4	홍길동	hong@gmail.com	4533
5	하지민	but@gmail.com	1866
6	마이클	mike@naver.com	1362
7	김지민	apple@gmail.com	1342
8	김병준	lucid_dawn@naver.com	776
9	푸바오	fubao@naver.com	392

따릉이의 각 사용자별 이용시간 랭킹, 이용횟수 랭킹, 이용거리 랭킹을 확인하는 시각화를 제공하였다. 백엔드에서 각 유저별 랭킹을 가져오는 API를 작성하고, 이를 프론트엔드에서 가져와 Table 형태로 제공하였다. 쉬운 탐색을 위하여 10개씩 페이지 단위로 제공하는

Pagination을 제공하였고, 해당 부분은 상단 Navbar에서 들어갈 수 있도록 링크를 제공해 비회원 및 일반 사용자도 접근 가능하도록 구현하였다.

5-3, 5-4. 지도 내 대여소 정보, 자전거 도로 시각화 기능

데이터 정보

공개일자	2022.01.28.	최신수정일자	2023.12.12.	
갱신주기	수시	분류	교통	
원본시스템	서울시 교통정보 시스템(TOPIS)	비교하기	저작권자	서울특별시
제공기관	서울특별시	제공부서	도시교통실 교통기획관 미래첨단교통과	
담당자	임재영 (02-2133-4959)			
원본형태	DB	제3저작권자	없음	
라이선스	저작권자표시(BY) 이용이나 변경 및 2차적 저작물의 저작권을 포함한 자유이용을 허락합니다.			
관련태그	띠링이, 대여소, 공공, 자전거, 마스터			

미리보기

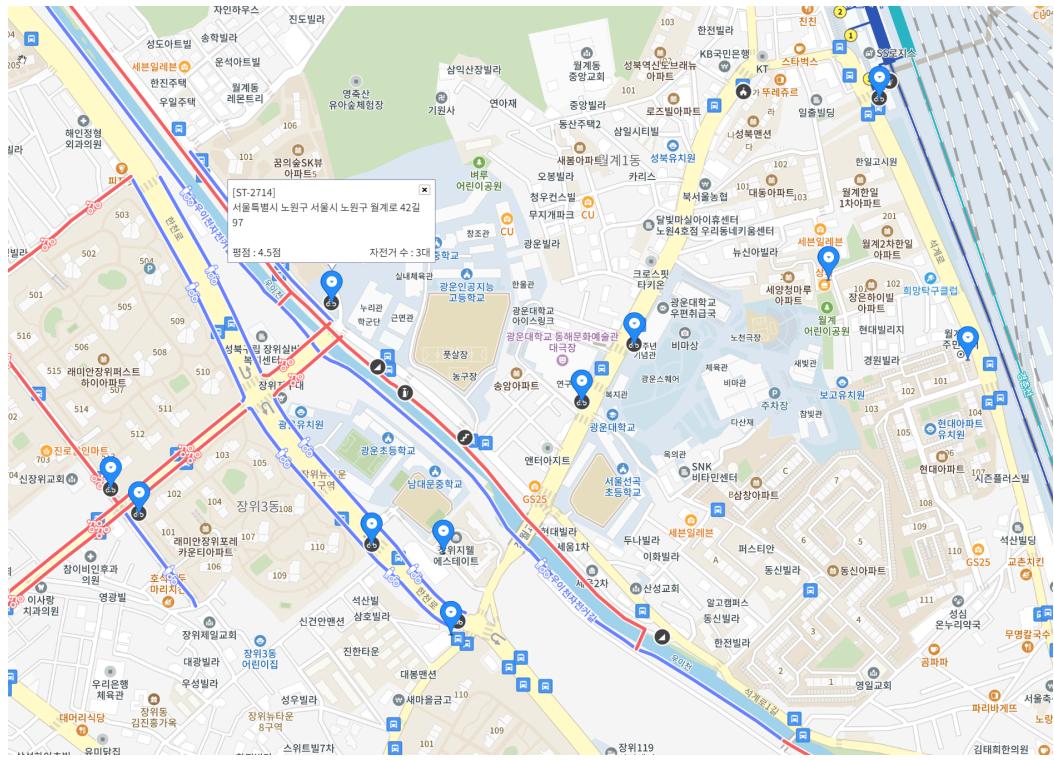
Sheet
Open API
닫힘 -

필드명

검색명

내려받기(CSV)
내려받기(JSON)
조회

대여소_ID	주소1	주소2	위도	경도
ST-999	서울특별시 양천구 목동서로 280	목동아파트 8단지 상가동	0.000000	0.000000
ST-998	서울특별시 양천구 목동서로 130	목동아파트 4단지 상가동	0.000000	0.000000
ST-997	서울특별시 양천구 목동중앙로 49	목동3단지 시내버스정류장	37.534390	126.869598
ST-996	서울특별시 양천구 남부순환로88...	양강중학교앞 교차로	37.524334	126.850548
ST-995	서울특별시 양천구 중앙로 153	금... SBS방송국	37.510597	126.857323
ST-994	서울특별시 양천구 목동서로161	SBS방송국	37.529163	126.872749
ST-993	서울특별시 양천구 신월로 342-1	...	37.521511	126.857384
ST-992	서울특별시 미포구 미포대로 163	서울신용보증재단	37.549061	126.954178
ST-991	서울특별시 마포구 성암로330	DMC첨단산업센터	37.584503	126.885597
ST-990	서울특별시 상암동 1715-29	롯데하이마트 (상암월드컵점)	37.573620	126.898048
ST-99	서울특별시 광진구 능동로 10	독성유원지역 1번출구 앞	37.531860	127.067192
ST-989	서울특별시 월드컵로5길 11	합정동 주민센터	0.000000	0.000000
ST-988	서울특별시 광진구 능동로 42	신양초교앞 교차로	37.535221	127.068398
ST-987	서울특별시 광진구 아차산로 304	원일교회	37.538052	127.076599
ST-986	서울특별시 광진구 아차산로78길 7	광진유진스웨	37.550488	127.108955
ST-985	서울특별시 광진구 능동로 지하 417	중곡역 1번출구 앞	37.565659	127.084297



따릉이를 대여 및 반납을 진행할 때 카카오 맵을 통하여 따릉이 대여소를 선택하게 된다. 이때, 카카오맵 내 마커를 이용해 따릉이 대여소의 위치를 실시간으로 확인할 수 있도록 시각화하였으며, 드래그를 이용해 맵을 이동할 때마다 새롭게 마커가 생성되어 쉽게 따릉이 대여소의 위치를 파악할 수 있다. 이 때 대여소 정보는 실제 따릉이 대여소 정보를 Database에 저장한 뒤, 서버에서 받아 활용하였다.

또한, 자전거 도로를 카카오 맵 내에서 표시함으로써 시각화를 진행하였고, 사용자는 자전거 도로를 확인해 이동할 수 있다.

6. YouTube 및 GitHub 링크

※ 성적 마감시까지 공개로 설정

- Youtube: <https://www.youtube.com/watch?v=9LMbMik5Zag>
 - Github:
 - Frontend : https://github.com/2023-KW-DB/2023_DB_FE
 - Backend : https://github.com/2023-KW-DB/2023_DB_BE

7. 고찰

강병준

데이터베이스 및 데이터시각화 프로젝트를 통해 웹 환경에서의 git&github 협업 경험과 더불어 수업 시간에 배웠던 데이터베이스 관련 이론 지식을 바탕으로 SQL 쿼리문을 직접 사용해보며 데이터베이스 관련 로직들을 수행하였다.

프로젝트가 진행될수록 초기의 데이터베이스 모델링에 의해 데이터베이스 구조를 바꾸는 것이 애매한 상황이 찾아오는 등 초기의 모델링에 대한 기반을 확실하게 다져야 프로젝트 유지 보수 측면에서 뛰어난 생산성을 가져올 수 있으며, 데이터베이스 모델링의 중요성에 대해서 확실하게 깨달을 수 있었다.

또한 백엔드로서 과제의 요구 사항이었던 ORM 사용 대신 SQL 쿼리를 직접 사용하기 위해 채택한 기술인 JDBC template을 선택하였다. JDBC template을 통해 작성한 SQL 쿼리문이 사실 수업 시간에 배웠던 내용이었던 SQL injection 공격에 대해 방지하는 방법인 prepared statement였다는 사실 또한 알 수 있었다.

김성진

본 프로젝트에서 프론트엔드 개발 및 데이터베이스 구조 설계를 진행하였다. 프로젝트를 진행하면서 프로젝트의 이용 명세를 정확히 하고, 이를 통해 데이터베이스 구조를 설계하고, SQL문을 직접 작성해보면서 어떤 상황에 어떤 구조를 갖는 것이 유리할지, 어떤 쿼리문을 사용하여 데이터를 가져오고, 삽입하고, 수정하고, 삭제하는 것이 유리할지 고려하며 프로그램을 작성할 수 있었다. 또한, 데이터베이스를 구조를 작성하며 생길 수 있는 데이터베이스의 Anomaly나, Primary Key-Foreign Key를 계속 고려해가며 데이터베이스를 설계해 데이터베이스 지식을 습득할 수 있을 뿐 아니라 안정적인 소프트웨어를 작성할 수 있었다.

그 외에도, 이를 웹 프로젝트로 확장하여 웹으로 시각화하기 위한 React, 차트 라이브러리, UI작성 프레임워크들을 습득할 수 있었고 이를 활용하여 프로덕션에서 사용 가능할 정도의 완성도를 이끌어내었다. 추후 비슷한 프로젝트로 인해 웹을 통한 시각화가 필요하다면, 안정적인 웹 소프트웨어를 작성할 수 있을 것이다.

하지현

이전에는 이렇게 데이터베이스와 웹에 대하여 접해볼 기회가 많지 않아 이번 데이터 베이스 프로젝트를 진행하면서 처음으로 **react, spring, java, javascript**등을 사용해 보았던 것 같다. 학기중에 처음 접하게 되어 사용법이나 문법, 기능이 어색하여 프로젝트를 진행하면서 공부를 함께 진행하는 부분에서 어려움을 느꼈지만 여러가지 오류가 발생하여 오류에 대해 찾아보고 해결하는 과정에서 과제를 위한 프로젝트를 떠나서 배우는게 많았다. 이번 프로젝트를 계기로 데이터베이스에 대하여 관심을 갖게 되어 프로젝트를 진행하면서 부족했던 부분은 이후에 프로젝트가 끝나고 다시 공부해 보고 싶다. 프론트 파트를 하면서 결과가 바로바로 눈에 보여서 오류를 고치는것이 수월했다. 부족한 부분이 많았지만 팀원들과 상의하고 도움받으며 해결할 수 있었다.

또한 2차에서 **SQL**문을 작성할 때 서로 다른 종류의 **SQL**문을 사용하여 작성하라고 명시되어 있어 강의자료에 나와있던 모든 종류를 이용하자고 했다. 작성한 **SQL**문을 실제 프로젝트에 사용하지 않아도 된다고 나와있었지만 최대한 많이 실제 프로젝트에 적용할 수 있도록 작성하는게 좋다고 생각되어 구현할 기능에 맞게 작성했다. 몇몇 **SQL**문은 어디에 적용할 수 있을까 생각하는게 어렵고 오래걸리기도 했지만 직접 작성해보는게 결과적으로 수업시간에 배운 내용을 이해하는데 정말 많은 도움이 됐다.

방지민

프로젝트를 수행하면서 어플리케이션 기능 정의, 데이터베이스 구조 설계 및 개발 과정을 모두 수행하게 되었다. 먼저 기능을 정의하고 데이터베이스 구조를 직접 설계해가면서 거기서 알 수 있는 **key**의 관계들에 대해 개념적으로만 보던 것보다 더 정확히 이해할 수 있었다. 이 과정에서 데이터베이스 정규화, 관계 설정 및 제약 조건의 중요성에 대해 실질적인 경험을 통해 배울 수 있었고, 데이터 무결성과 성능에 기여하는 요소임을 다시 한 번 깨달을 수 있었다.

또한, **Springboot**에서 기존에 사용해왔던 **JPA** 대신 쿼리를 직접 사용하기 위해 **JDBC Template**을 이용하면서, 직접 **SQL** 쿼리를 작성하고 실행하는 과정에서 **SQL**에 대한 심도있는 이해를 할 수 있었으며, 쿼리만을 사용하는 로직과 서비스 코드에서 쿼리를 분리하여 사용하는 로직의 차이점에 대해 학습할 수 있었다. 또한,

이 프로젝트에서는 수업에서 배웠던 이론들을 실제로 적용하였기 때문에 이론과 실제를 연결할 수 있었다.

추가로, 이 프로젝트를 수행하면서 효율적인 팀원간 협업을 통한 성취를 얻을 수 있었다.

8. 2차 제안서 대비 변경 사항

2차 제안서에서 제안한 특별 기능인 즐겨찾기, 따릉이 ticket, 이용권 내역 조회, 대여소 평점, 지도 관련 표시, 따릉이 대여 및 반납, 관리자의 자전거 관리, 대여반납이력 외에도 카카오 소셜 로그인, FCM 알림 기능, 이메일 인증 기능, 뉴스 등 새로운 특별 기능이 추가된 것이 변경 사항이다.