VIBEMASTER

나만의 음악, 너와의 공유.

목차

01. 프로젝트 소개

개요

Vibe Master는 음악을 찾아 듣는 걸 즐겨하는 사용자들을 겨냥한 음악 플레이리스트 공유 사이트 입니다.

기존 스트리밍 사이트에서 제공하는 플레이리스트 공유 기능과의 차별점을 두기 위해 플레이리스트 별 태그 기반 검색 기능에 주안점을 두어 개발을 진행했습니다.

사용 언어 / 기술

Front-End

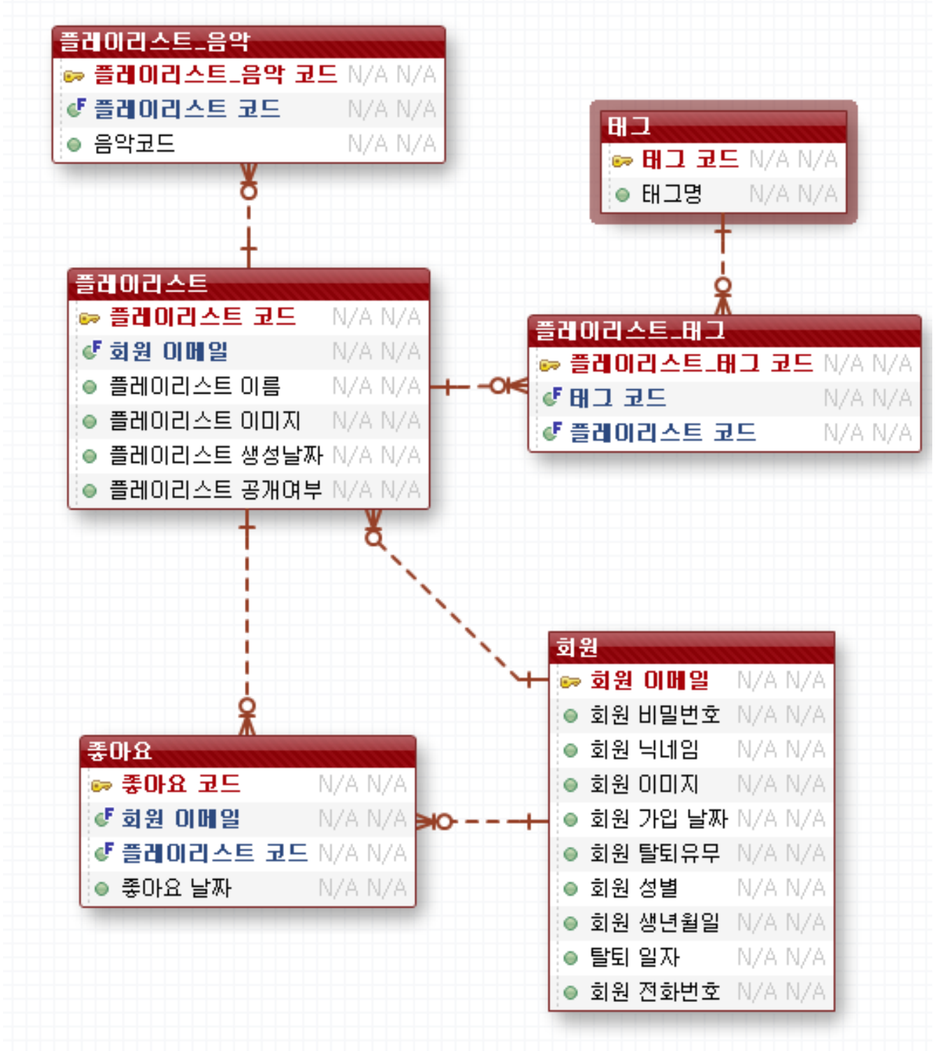
Back-End

IDE & Tools

Open API

DB Modeling

각 음악에 관한 정보들은 Open API에서 제공하는 정보들을 활용했기 때문에 DataBase를 보다 간소화 하여 구성하였습니다.



요구사항 목록

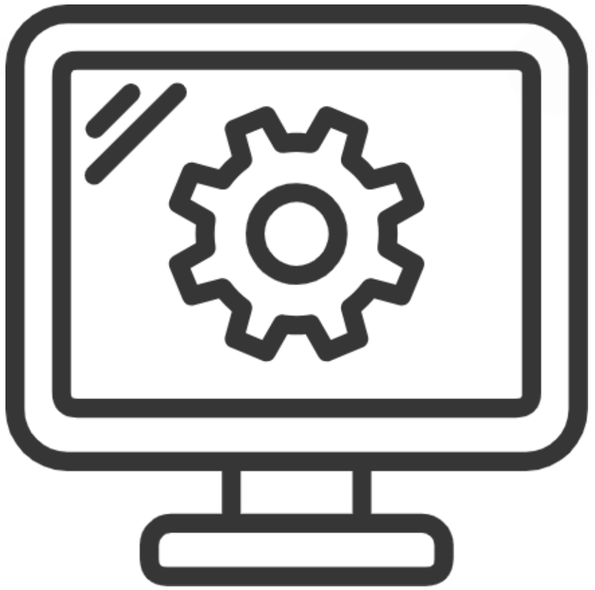
각 기능 별 요구사항을 정리 및 분배 한 뒤

우선순위를 결정하여 개발을 진행하였습니다.

분류	요구사항 명	요구사항 설명	우선순위	구현 中	완전구현
로그인	회원가입	계정이 없는 사용자는 새로운 계정으로 회원가입	상	☑	☑
	로그인	계정이 있는 사용자는 로그인하여 접속	상	☑	☑
	로그아웃	로그인 되어 있는 사용자가 사이트에서 로그아웃	상	☑	☑
	아이디 찾기	이메일 잊어버린 사용자가 이메일 조회	중	☑	☑
	비밀번호 찾기	비밀번호 잊어버린 사용자가 비밀번호 재설정	중	☑	☑
마이페이지	회원정보수정	회원 본인의 정보를 수정	중	☑	☑
	회원탈퇴	회원이 해당 서비스 탈퇴	하	☑	☑
	좋아요한 플레이 리스트 조회	회원 본인이 좋아요 표시한 플레이리스트 목록 조회	중	☑	☑
	내가 만든 플레이리스트 조회	회원 본인이 작성한 플레이리스트 목록 조회	중	☑	☑
	내가 좋아하는 태그 조회	회원 본인이 좋아하는 태그 조회	중	☑	☑
검색 / 조회	태그 검색	태그 검색을 통해 원하는 플레이리스트 목록 조회	상	☑	☑
	플레이리스트 이름 검색	플레이리스트 이름 검색을 통해 원하는 플레이리스트 조회	상	☑	☑
	랭킹 순 조회	좋아요가 많은 플레이리스트 순서 대로 조회	상	☑	☑
	태그 별 랭킹 순 조회	사용자가 찾고자하는 태그를 가진 플레이리스트를 랭킹순으로 조회	중	☑	☑
	최근 좋아요 순 조회	짧은 기간내에 좋아요를 많이 받은 플레이리스트 순으로 조회	중	☑	☑
	성별 별 좋아요 순 조회	해당 성별에게 좋아요를 많이 받은 플레이리스트 조회	하	☑	☑
	연령대 별 좋아요 순 조회	해당 연령대에게 좋아요를 많이 받은 플레이리스트 조회	하	☑	☑
	플레이리스트 페이징	플레이 리스트 페이징	중	☑	☑
	플레이리스트 랜덤 조회	플레이리스트를 랜덤으로 조회	하	☑	☑
좋아요	플레이리스트 좋아요	플레이리스트 좋아요 기능	상	☑	☑
플레이리스트	플레이리스트 생성	사용자가 직접 플레이리스트 생성	상	☑	☑
		생성한 플레이리스트에 태그 추가	상	☑	☑
		플레이리스트 생성 시 이미지 설정	상	☑	☑
	플레이리스트 수정	플레이리스트 태그 수정	상	☑	☑
		플레이리스트 태그 삭제	중	☑	☑
		플레이리스트 제목 수정	중	☑	☑
		플레이리스트 이미지 수정	상	☑	☑
		플레이리스트 공개/비공개 기능 수정	상	☑	☑
	플레이리스트 곡 추가	자신의 플레이리스트에 곡 추가	상	☑	☑
	플레이리스트 곡 삭제	플레이리스트에 있는 곡 삭제	중	☑	☑
	플레이리스트 삭제	자신의 플레이리스트를 삭제	중	☑	☑
	플레이리스트 내 곡 조회	플레이리스트 내 곡 조회	상	☑	☑
스트리밍	음악재생	플레이리스트 내에 해당 음악 클릭 시 재생	하	☑	☑
레이아웃	메인페이지	메인페이지 제작	상	☑	☑
	음악 검색페이지	음악 검색페이지 제작	상	☑	☑
	마이페이지	마이페이지 제작	상	☑	☑
	로그인페이지	로그인페이지 제작	상	☑	☑
	회원가입페이지	회원가입페이지 제작	상	☑	☑
	플레이리스트 검색 페이지	플레이리스트 검색 페이지 제작	상	☑	☑
	좋아요 순 랭킹 페이지	전체 좋아요 순 랭킹 페이지 제작	상	☑	☑
	좋아한 플레이리스트 페이지	회원이 좋아한 플레이리스트 페이지 제작	상	☑	☑
	내가 만든 플레이리스트 페이지	회원이 만든 플레이리스트 조회 페이지 제작	상	☑	☑
	회원수정 페이지	회원 수정 페이지 제작	상	☑	☑
	계정찾기 페이지	계정 찾기 페이지 제작	상	☑	☑
	연령대 별 랭킹 페이지	연령대 별 랭킹 페이지 제작	상	☑	☑
	성별 별 랭킹 페이지	성별 별 랭킹 페이지 제작	상	☑	☑

02. 담당 업무

- 회원 관련 기능 개발
  - mvc 패턴에 맞춰 회원가입 / 로그인 / 회원수정 / 회원 탈퇴 기능 개발
  - Spring-security 적용 하여 개발
- 플레이리스트 좋아요 기능 개발
  - Toggle 방식으로 좋아요 및 취소가 가능하도록 개발
- Open API 활용 하여 음악 검색 / 조회 기능 개발
  - Server에서 Token을 발급 받아 정보를 가져오는 방식으로 개발
- 각 페이지 레이아웃에 맞춰서 퍼블리싱 작업 수행
- 프로젝트 취합 및 총괄 역할 수행



회원 관련 기능

회원 가입

```
$("#userEmail").keyup(() => {
    emailCheck = false;
    const regExp = /^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$/;
    $("#emailCheck").val("Check")
        .css({"color": "#787775",
            "font-size": "0.8rem"});
    if(regExp.test($("#userEmail").val())){
        regEmailCheck = true;
        $("#userEmail").css("border-bottom", "1px solid green");
    }else{
        regEmailCheck = false;
        $("#userEmail").css("border-bottom", "1px solid red");
    }

});
```

사용자가 값 입력 시 정규표현식에 맞게 입력했는지 체크하기 위해

Client에서 유효성 검사 시행

```
<form action="registerUser" method="post" onsubmit="return validate()">

function validate(){
    if(!emailCheck) alert("이메일 체크를 진행해주세요.");
    else if(!nicknameCheck) alert("닉네임 체크를 진행해주세요.");
    else if(!regEmailCheck) alert("이메일형식을 확인해주세요.");
    else if(!regPasswordCheck) alert("패스워드 형식을 확인해주세요. (영어/숫자/특수문자 포함 8~14자)");
    else if(!passwordSameCheck) alert("패스워드가 다릅니다.");
    else if(!phoneCheck) alert("전화번호 형식을 확인해주세요.");
    else if(!nicknameCheck) alert("닉네임 형식을 확인해주세요. (영어/한글/숫자)");
    return emailCheck && nicknameCheck && regEmailCheck && nicknameCheck
    && regPasswordCheck && passwordSameCheck && phoneCheck;
}
```

Form 태그의 onsubmit 속성에 validate 함수를 호출 함으로써

보기 중의 하나라도 유효성 검사에 실패한다면 Client 자체에서 값이

Server로 넘어가지 않도록 막아준다.

회원 관련 기능

회원 가입

```
$("#emailCheck").click(() => {
    $.ajax({
        type: "post",
        url: "/emailCheck",
        data: {
            userEmail: $("#userEmail").val()
        },
        success: function(checkOk){
            if(!checkOk) {
                alert("중복된 이메일 입니다.");
                $("#userEmail").css("border-bottom", "1px solid red");
                emailCheck = false;
            }else if($("#userEmail").val() === ""){
                alert("이메일을 입력해주세요.");
                $("#userEmail").css("border-bottom", "1px solid red");
                emailCheck = false;
            }else if(!regEmailCheck){
                alert("이메일 형식으로 입력해주세요.");
                $("#userEmail").css("border-bottom", "1px solid red");
                emailCheck = false;
            }else{
                alert("사용가능한 이메일 입니다.");
                emailCheck = true;
                $("#userEmail").css("border-bottom", "1px solid green");
                $("#emailCheck").val("✓")
                    .css({"color": "black", "font-size": "1rem"});
            }
        }
    });
});
```

1

6

```
// ajax - 회원가입시 아이디 중복 조회
@ResponseBody
@PostMapping("/emailCheck")
public boolean emailCheck(String userEmail) {
    return userService.emailCheck(userEmail);
}
```

2

5

```
// 회원가입 시 이메일 중복 체크
public boolean emailCheck(String userEmail) {
    if(userMapper.emailCheck(userEmail) == null) return true;
    return false;
}
```

3

4

```
User emailCheck(String userEmail); // 회원가입 시 이메일 중복 체크
```

```
<!-- 회원가입 시 이메일 중복 확인 -->
<select id="emailCheck" parameterType="String" resultMap="userMap">
    SELECT * FROM user
    WHERE user_email = #{userEmail}
</select>
```

Email과 Nick name은 Database에 존재 하는 다른 값들과
중복되면 안되기 때문에 ajax 통신을 활용해 비동기 방식으로 Server와 값을 주고
받는다.

회원 관련 기능

회원 가입

유효성 검사에 모두 통과 했다면 Server로 값을 전달하여 회원가입 처리

```
@Autowired
private PasswordEncoder bcpe;

// 회원가입
public int register(User user) {
    user.setUserPassword(bcpe.encode(user.getUserPassword()));
    return userMapper.register(user);
}
```

비밀번호는 암호화를 위해 spring-security에서 제공하는

PasswordEncoder Interface 내의 encode 메서드를 활용

```
<!-- 회원가입 -->
<insert id="register" parameterType="User">
    INSERT INTO user(user_email, user_password, user_nickname, user_gender, user_birth, user_phone)
    VALUES(#{userEmail}, #{userPassword}, #{userNickname}, #{userGender}, #{userBirth}, #{userPhone})
</insert>
```

회원가입 시 사용되는 Query 문

회원 관련 기능

로그인

```
.formLogin(login ->
    login.loginPage("/login")
    .defaultSuccessUrl("/", true)
    .failureHandler(new DomainFailureHandler())
    .permitAll())
```

Spring Security에서 제공하는 formLogin 설정을 통해 로그인 인증 처리

로그인 실패 시 DomainFailureHandler 호출

```
@Override
public void onAuthenticationFailure(HttpServletRequest request, HttpServletResponse response,
        AuthenticationException exception) throws IOException, ServletException {
    // 실패로직 핸들링
    String errorMsg = "";
    String userEmail = request.getParameter("username");

    if (exception instanceof BadCredentialsException) {
        errorMsg = "비밀번호를 잘못 입력하셨습니다.";
    } else if (exception instanceof UsernameNotFoundException) {
        errorMsg = "아이디를 잘못입력하셨습니다.";
    } else if (exception instanceof AccountExpiredException) { // 해당 X
        errorMsg = "계정만료";
    } else if (exception instanceof CredentialsExpiredException) { // 해당 X
        errorMsg = "비밀번호만료";
    } else if (exception instanceof DisabledException) {
        errorMsg = "탈퇴회원";
    } else if (exception instanceof LockedException) { // 해당 X
        errorMsg = "계정잠김";
    } else {
        errorMsg = "아이디 혹은 비밀번호를 잘못 입력하셨습니다.";
    }

    errorMsg = URLEncoder.encode(errorMsg, "UTF-8");
    setDefaultFailureUrl("/loginError?error=" + errorMsg + "&username=" + userEmail);
    super.onAuthenticationFailure(request, response, exception);
}
```

```
// 로그인 에러 시 리턴할 메세지
@GetMapping("/loginError")
public String loginError(Model model, String error, String username) {
    if (error.equals("탈퇴회원"))
        error = "재가입까지 " + userService.rejoinDate(username) + "일 남았습니다.";

    model.addAttribute("msg", error);
    return "user/login";
}
```

전달 받은 값을 model로 담아 client로 전달

(탈퇴회원의 경우 재가입 남용 방지를 위해 7일간 데이터 보관 후 삭제)

해당하는 예외처리 후 값을 Controller로 전달

회원 관련 기능

회원 탈퇴

```
// 회원탈퇴
@ResponseBody
@PostMapping("/deleteUser")
public boolean deleteUser(String userPassword, Model model, HttpServletRequest request) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User user = (User) authentication.getPrincipal();

    if (userService.deleteUser(user, userPassword)) {
        HttpSession session = request.getSession(false);
        session.invalidate();
        SecurityContextHolder.getContext().setAuthentication(null);
        return true;
    }
    return false;
}
```

Client에서 비밀번호를 입력 받아 현재 접속중인 유저와 일치하면 비즈니스 로직 실행 후 Session 값 만료 후 true 값을 리턴, 일치하지 않으면 false 값을 리턴 하도록 Controller를 구성

```
// 회원탈퇴
public boolean deleteUser(User user, String userPassword) {
    if (bcpe.matches(userPassword, user.getUserPassword())) {
        userMapper.deleteUser(user.getUserEmail());
        // 회원탈퇴 user의 playlist를 playlistManager 계정으로 옮기기
        playlistMapper.movePlaylist(user.getUserEmail());
        return true;
    }
    return false;
}
```

접속중인 유저의 password는 암호화 되어 있기 때문에 PasswordEncoder Interface 내의 matches 메서드를 활용하여, 입력 받은 비밀번호와 일치한지 확인 후 mapper를 통해 해당 query문 실행

회원 관련 기능

회원 탈퇴

```
<!-- 회원탈퇴 -->
<update id="deleteUser" parameterType="String">
    UPDATE user
    SET user_ent_yn = 'Y',
        user_enroll_date = current_date()
    WHERE user_email = #{userEmail}
</update>
```

회원 탈퇴 시 Query 문

유저의 탈퇴 유무와 탈퇴 날짜에 해당하는 컬럼의 값을 변경해 주기 위해 Update 문 사용

(회원 탈퇴 시 7일간 정보 보관 뒤 자동 삭제 : 무분별한 재가입 방지)

```
-- 이벤트 스케줄러 생성
DELIMITER //
CREATE EVENT user_delete
ON SCHEDULE EVERY 1 DAY
STARTS '2024-08-06 12:00:00'
DO
BEGIN
    DELETE FROM user
    WHERE user_email IN(
        SELECT user_email
        FROM (
            SELECT user_email
            FROM user
            WHERE DATE_ADD(user_enroll_date, INTERVAL 7 DAY) <= CURDATE()
        ) AS temp
    );
END//
DELIMITER ;
```

회원 정보 자동 삭제를 위해 MySql에서 사용할 수 있는 Event Scheduler 기능 사용

하루에 한번씩 12시 기준으로 조회 하여 탈퇴회원이 현재날짜 기준 7일이 지났을 경우 해당 회원의 데이터를 자동으로 삭제

플레이리스트 좋아요 기능

좋아요

```
function clickLike(event) {
    const plLike = event.currentTarget;
    $.ajax({
        type: 'post',
        url: '/userLike',
        data: {
            plCode: plLike.getAttribute("data-code")
        },
        success: function(data){
            const count = plLike.querySelector('.likeCount').innerHTML;
            if(data){
                plLike.querySelector('i').style.color = 'red';
                plLike.querySelector('i').setAttribute('class', 'fa-solid fa-heart');
                plLike.querySelector('.likeCount').innerHTML = Number(count) + 1;
            }else{
                plLike.querySelector('i').style.color = 'white';
                plLike.querySelector('i').setAttribute('class', 'fa-regular fa-heart');
                plLike.querySelector('.likeCount').innerHTML = Number(count) - 1;
            }
        },
        error: function(){
            alert("로그인 후 이용해 주세요.");
        }
    });
}
```

좋아요 관련 버튼을 클릭 시 발생하는 Javascript 함수

Ajax 방식으로 해당 플레이리스트의 코드(고유 값)을 Server로 요청 및 응답 받은 값에 따라

이미지와 텍스트 내용 수정

```
@ResponseBody
@PostMapping("/userLike")
public boolean userLike(Model model, int plCode) {
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
    User user = (User) authentication.getPrincipal();

    PlaylistLikeDTO dto = new PlaylistLikeDTO();
    dto.setPlCode(plCode);
    dto.setUserEmail(user.getUserEmail());

    return playlistLikeService.userLike(dto);
}
```

요청을 보낸 유저의 이메일과 좋아요를 요청한 해당 플레이리스트의 고유값을 DTO로 담은 뒤

Service로 전달

플레이리스트 좋아요 기능

좋아요

```java
// 좋아요
public boolean userLike(PlaylistLikeDTO dto) {
    // 조회해서 없으면 추가
    if(playlistLikeMapper.userLikePlaylistCheck(dto) == null) {
        playlistLikeMapper.userLike(dto);
        return true;
    }else {
        // 있으면 삭제
        playlistLikeMapper.userUnLike(dto);
        return false;
    }
}
```

좋아요에 대한 정보 table에 존재하지 않다면 새로운 값을 추가하고(좋아요),

이미 정보가 table에 존재한다면 그 값을 삭제(좋아요 취소)

```xml
<!-- 좋아요 했는지 확인 -->
<select id="userLikePlaylistCheck" parameterType="PlaylistLikeDTO" resultMap="playlistLikeMap">
    SELECT *
    FROM playlist_like
    WHERE pl_code = #{plCode}
        AND user_email = #{userEmail}
</select>
<!-- 좋아요 -->
<insert id="userLike" parameterType="PlaylistLikeDTO">
    INSERT INTO playlist_like(pl_code, user_email)
    VALUES(#{plCode}, #{userEmail})
</insert>
<!-- 좋아요 취소 -->
<delete id="userUnLike" parameterType="PlaylistLikeDTO">
    DELETE FROM playlist_like
    WHERE pl_code = #{plCode}
        AND user_email = #{userEmail}
</delete>
```

좋아요에 기능에 해당하는 Query 문 (SELECT, INSERT, DELETE)

Opne API 활용한 음악 정보 추출 기능

Token 발급

```
// token 발급 받는 메서드
private String getAccessToken() {

    String clientId = [redacted];
    String clientSecret = [redacted];
    String url = "https://accounts.spotify.com/api/token";

    String credentials = clientId + ":" + clientSecret;
    String base64Credentials = new String(Base64.getEncoder().encode(credentials.getBytes()));

    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.APPLICATION_FORM_URLENCODED);
    headers.set("Authorization", "Basic " + base64Credentials);

    MultiValueMap<String, String> body = new LinkedMultiValueMap<String, String>();
    body.add("grant_type", "client_credentials");

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

    ResponseEntity<Map> response = restTemplate.postForEntity(url, request, Map.class);
    return response.getBody().get("access_token").toString();
}
```

Open API 사이트에서 발급받은 id와 secret Key를 가공한 뒤 Token 발급 주소로 요청

응답받은 데이터 중 “access_token”이라는 key 값의 value를 추출

(추출한 token을 사용해서 음악정보를 추출)

Opne API 활용한 음악 정보 추출 기능

음악 정보 추출

```java
// 검색한 음악정보 요청해서 받아오는 메서드
public ArrayList<Music> getMusicInfoForMusicName(String musicName, int offset) {
    String accessToken = getAccessToken();

    // 기능마다 바뀌는 구문
    String url = "https://api.spotify.com/v1/search?q=" + musicName + "&type=track&limit=10&offset=" + offset;

    HttpHeaders headers = new HttpHeaders();
    headers.set("Authorization", "Bearer " + accessToken);

    HttpEntity<String> request = new HttpEntity<String>(headers);

    ResponseEntity<JsonNode> response = restTemplate.exchange(url, HttpMethod.GET, request, JsonNode.class);
```

원하는 정보 추출을 위해 api 문서 참고하여 해당하는 url 주소와 값들을 입력.

Token은 header에 세팅해준 상태에서 정보 추출에 필요한 url과 함께

해당하는 http 메서드 방식으로 요청

Opne API 활용한 음악 정보 추출 기능

음악 정보 추출

```
JsonNode musicData = response.getBody();

// api를 통해 response 받는 구문
ArrayList<Music> musicInfo = new ArrayList<>();
JsonNode trackInfo = musicData.get("tracks").get("items");

for(JsonNode track : trackInfo) {

    JsonNode albumInfo = track.get("album");
    String id = track.get("id").asText();
    String albumUrl = albumInfo.get("images").get(0).get("url").asText();
    String albumName = albumInfo.get("name").asText();
    String artistName = track.get("artists").get(0).get("name").asText();
    String musicTitle = track.get("name").asText();

    Music music = new Music();
    music.setId(id);
    music.setAlbumName(albumName);
    music.setAlbumUrl(albumUrl);
    music.setArtistName(artistName);
    music.setMusicTitle(musicTitle);

    musicInfo.add(music);
}
return musicInfo;
```

응답 받은 데이터를 확인한 뒤 필요한 값들만 추출한 뒤 List에 담아 Controller로 리턴

03. 트러블 슈팅

- 문제 발생
  - 회원 수정 기능 사용 중 수정할 내용이 database 수정 전에 session에 적용되어 버리는 현상
- 사실 수집
  - Client에서 받아온 값들을 Server상에 어떻게 넘어오는지 확인 후 문제가 생기는 구간 확인
- 원인추론
  - 현재 접속해 있는 유저 정보를 복사한 뒤 Client에서 받아온 값들을 복사한 객체의 데이터를 수정하면서 발생
  - 얕은 복사로 생긴 오류
- 문제해결
  - User 클래스에 Cloneable Interface 상속 후 Clone 메서드를 사용하여 해결

트러블 슈팅

```java
@NoArgsConstructor @AllArgsConstructor
@Data @Builder
public class User implements UserDetails, Cloneable{
    private String userEmail;
    private String userPassword;
    private String userNickname;
    private String userImg;
    private Date userDate;
    private char userEntYn;
    private char userSpotifyYn;
    private String userGender;
    private Date userBirth;
    private char userManager;
    private Date userEnrollDate;
    private String userPhone;
    private String ageGroup;
```

1. Cloneable 인터페이스 상속

```java
@Override
public User clone() throws CloneNotSupportedException{
    return (User)super.clone();
}
```

2. clone 메서드를 재정의

```java
Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
User user = (User) authentication.getPrincipal();
User changeUser = null;
try {
    changeUser = user.clone();
} catch (CloneNotSupportedException e) {
}
```

3. 접속한 유저 정보를 재정의한 clone 메서드 활용하여 객체 내용을 복사

04. 개선 점

현재 프로젝트에서의 음악 플레이어 기능은 그 해당 페이지를 벗어나는 순간 사라지게 되는데 이 문제는 Spotify에서 제공해주는 Web Playback SDK를 사용해야 한다는 것을 뒤늦게 알게 되었습니다.
다음에 기회가 된다면 Web Playback SDK를 사용하면서 Client 또한 React 라이브러리를 사용하는 것에 초점을 맞추고 개선하여 더욱 완성도 높은 프로젝트가 되었으면 좋겠습니다.

Thanks for watching!