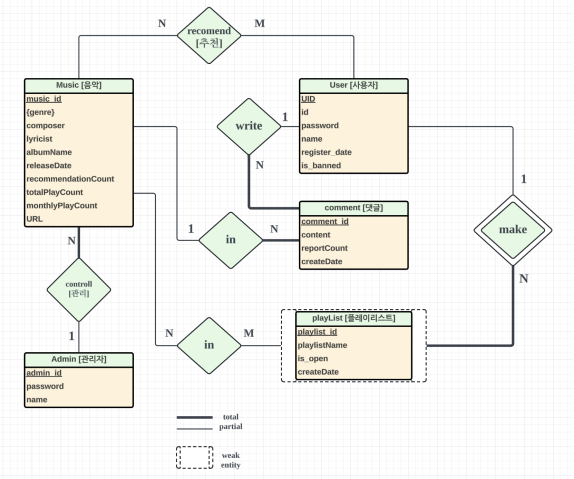
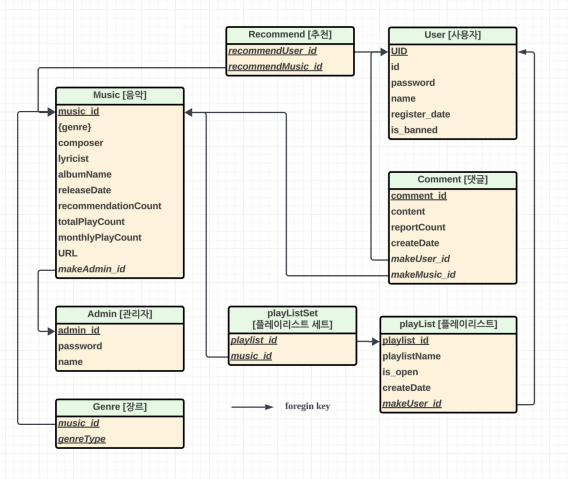
Final 프로젝트: DBMS 프로그램 개발

2020057265 컴퓨터소프트웨어학부 김진태

0. 목차

1. ER diagram / Relational diagram 변경사항





기존에 제출한 ER diagram 과 Relational diagram 입니다.

final 프로젝트를 진행하면서 발생한 문제점에 따라 스키마 변경이 불가피했습니다.

1.1 문제점

일반 사용자(Users), 관리자(Admin) 의 구별하기 어려움

초기 설계에서 User Table과 Admin Table 간의 구분이 쉽지 않았습니다.

- passport 를 사용하여 로그인을 구현하였습니다. 사용자 인증 후 사용자정보를 세션에 저장하지만, 저장되는 정보는 한정적입니다.

- serializerUser 를 이용해서 세션 데이터를 수동으로 확장해야했기에 차라리 User 와 Admin table 을 합쳐서 상태 하나로 관리했습니다.

1.2 결론

- ER diagram 및 Relational diagram 에서 Admin Table 은 삭제되었습니다.
- User Table 에 is_admin attribute 가 추가되었습니다.

2. 사용 언어 및 프레임워크

2.1 사용언어

- Node.js: JavaScript 실행환경 입니다.

2.2 프레임워크

- Express: 라우팅, 미들웨어, API 구현에 다양한 기능을 제공합니다.

2.3 사용자 인증 및 회원가입

- Passport: 인증미들웨어 입니다.
- 저는 Local 전략을 사용해서 이메일/비밀번호 기반으로 한 인증을 구현했습니다.

2.4 DBMS

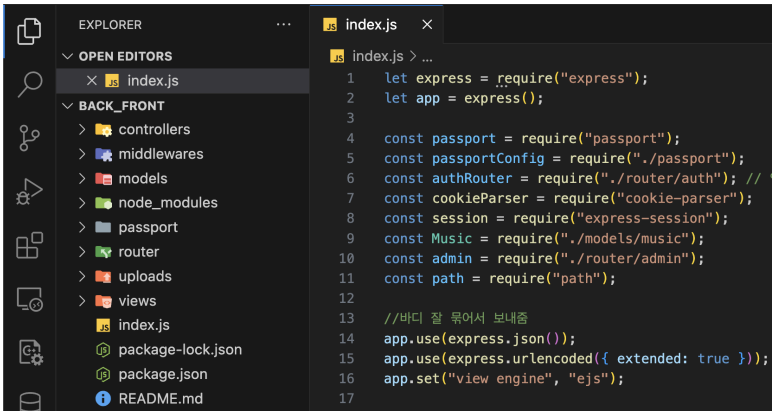
- Mysql: 관계형 데이터베이스 관리 시스템 입니다.

2.5 통합개발환경

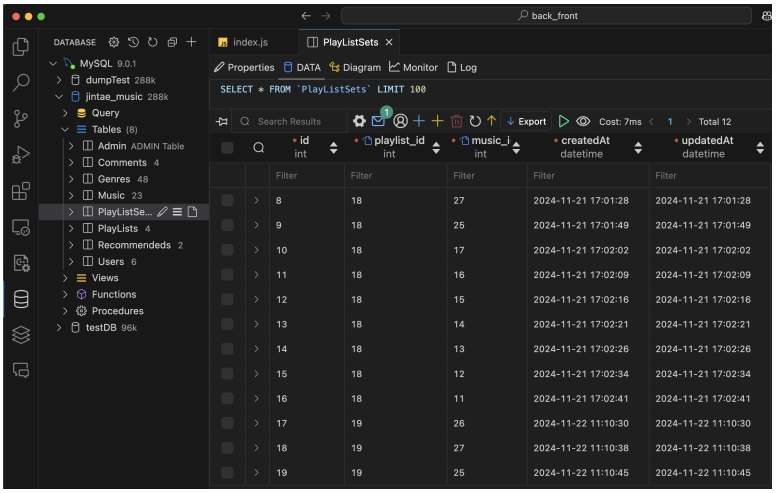
- VSCode
  - 내장 터미널에서 node.js 를 실행합니다.

```
➜  back_front git:(main) ✗ nodemon index.js
[nodemon] 3.1.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
express-session deprecated req.secret; provide secret option index.js:21:3
```

  - 코드 작성 및 관리



- 데이터베이스 접근 익스텐션

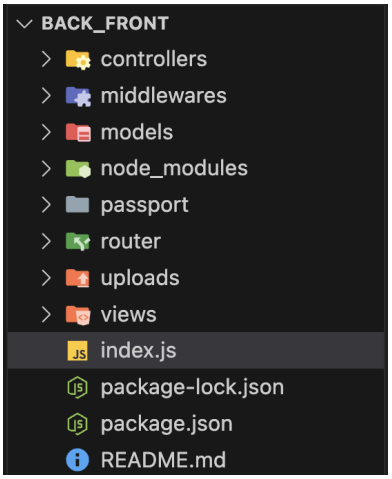


- git 관리

```
→  back_front git:(main) ✗ git add .
→  back_front git:(main) ✗ git commit -m "passport index.js fixed"
[main 1ce5553] passport index.js fixed
 3 files changed, 11 insertions(+), 8 deletions(-)
 delete mode 160000 back_front/jintae_music
→  back_front git:(main) git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 667 bytes | 667.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To https://github.com/KimJinTae1/jintae_music.git
   c32e52b..1ce5553  main -> main
```

3. 폴더구조

- MVC 패턴을 따르도록 노력했습니다.
- uploads 폴더는 음악파일을 저장합니다.
- views 폴더는 ejs 파일을 저장합니다.
- middlewares 폴더는 관리자를 판단하는 is_admin.js 함수와 업로드에 필요한 upload.js 가 들어있습니다.



4. 코드 설명

4.1 index.js

해당 코드는 기본적인 구성요소를 초기화하고 경로설정을 합니다.

또한 라우팅의 엔트리포인트를 설정해줍니다.

4.2 models 폴더

4.2.1 Sequelize와 DataTypes 임포트

```
const { Sequelize, DataTypes } = require("sequelize");
```

- node.js 환경에서 SQL 데이터베이스와 연결을 도와주는 라이브러리를 사용합니다.

4.2.2 데이터베이스 연결

```
const sequelize = new Sequelize("jintae_music", "root", "1234
  host: "localhost",
  dialect: "mysql",
});
```

- sequelize 를 이용하여 mysql 데이터베이스에 연결합니다. DB이름, 사용자이름, 비밀번호, 및 기타 설정을 인자로 입력받습니다.

4.2.3 모델 정의

```
const PlayListSet = sequelize.define("PlayListSet", {
  playlist_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    foreignKey: true,
  },
  music_id: {
    type: DataTypes.INTEGER,
    allowNull: false,
    foreignKey: true,
  },
});
```

- 각 모델별로 정보를 담기 위한 테이블을 정의합니다.
- 해당 정보는 각 모델별로 필요한 attribute 마다 다르게 정의됩니다.

4.2.4 모델 동기화

```
(async () => {
  try {
    await sequelize.sync();
    console.log("PlayListSet table has been created.");
  } catch (error) {
    console.error("Error creating table:", error);
  }
})();
```

- await sequelize.sync() 코드는 해당 테이블이 없다면 새로 생성하고, 존재한다면 업데이트합니다.
- 옵션으로 force : true 를 주면 기존 테이블을 덮어씁니다.

4.2.5 모델 export 하여 다른 파일에서 해당 모델을 데이터베이스에서 불러와 사용할 수 있습니다.

4.2.6 관계 설정

```
PlayListSet.associate = (models) => {
  PlayListSet.belongsTo(models.PlayList, {
    foreignKey: "playlist_id",
    targetKey: "playlist_id", // PlayList 모델의 primary key
    onDelete: "CASCADE",
  });

  PlayListSet.belongsTo(models.Music, {
    foreignKey: "music_id",
    targetKey: "music_id", // Music 모델의 primary key
    onDelete: "CASCADE",
  });
};
```

- foreignKey 설정이 필요한 모델들에 대해서 명시적으로 표시해주었습니다.
- 동시에 삭제 되었을때 옵션도 설정해주었습니다.

4.2.7 문제점

자동으로 Sequelize 모델을 사용하여 테이블을 생성할 때 Sequelize에서 default 로 모델 이름 뒤에 (s) 를 추가하여 테이블을 생성한다는것을 발견했습니다.

ex) music 테이블 생성 시 실제 DB 테이블 이름 → musics

이로 인해 Mysql 에서 Sql 문을 사용하며 만든 테이블과 별개의 테이블이 추가로 생성된 문제가 발생했습니다.

4.2.8 해결방법

freezeTableName: true 옵션이 있다는 사실을 나중에 발견하긴 했습니다. 테이블명 뒤에 (s) 를 제거할 수 있었습니다.

개발 당시에는 DB table 을 전부 밀어버리고 새로 sequelize 를 이용해 다시 만들어주었습니다.

4.3 router 폴더

4.3.1 코드 설명

```
router.get("/make_playlist", async (req, res) => {
  res.render("make_playlist.ejs");
});

router.get("/playlist", async (req, res) => {
  res.render("playlist.ejs");
});
router.post("/delete_comment", async (req, res) => {
  const { comment_id } = req.body;
  try {
    await Comment.destroy({
      where: {
        comment_id,
      },
    });
    res.redirect("/music/detail?music_id=" + req.body.music_i
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Server Error" });
  }
});

router.get("/search", musicController.searchMusic);
router.get("/searchByGerne", musicController.searchMusicByGen
router.post("/add_comment", commentController.createComment);
router.post("/report_comment", commentController.reportCommen
router.get("/getAllMusic", musicController.getAllMusic);
router.post("/create", upload.single("sing"), musicController
router.get("/getAllMusicByDate", musicController.getAllMusicB
router.get("/getAllMusicByPlayCount", musicController.getAllM
```

- express.js 의 라우더 설정 입니다. URL 요청에 의해 어떤 행동을 수행할지 정해줍니다.
- 실제 코드 수행은 Controller 에서 제공해줍니다.

4.3.2 문제점

해당 코드에서 컨트롤러에서 수행하는 작업과 라우터 파일 자체에서 직접 처리하는 작업들이 혼합되어 있습니다.

이는 추후 유지보수 및 확장에 어려움이 있을 수 있습니다.

4.3.3 이유

대부분 router 폴더의 전반적으로 초반에 작성한 코드에서 나타나는 문제점들입니다.

이는 개발 초기 MVC 패턴의 이해 부족으로 인해 일어난 문제입니다. 컨트롤러의 역할을 이해하지 못하고 해당 방식으로 코드를 짜도 프로그램은 잘 돌아갔습니다. 그렇기에 수정을 하지 않다가 점점 늘어나는 코드와 더불어 관리가 어려워지면서 추후 더 공부를 하며 MVC 패턴으로 수정을 했습니다.

현재까지 바꾸지 않은 부분은 향후 보고서 작성을 위해 그대로 남겨두었으며, 이를 통해 초기 코드 작성 시의 문제점을 더 잘 설명할 수 있도록 할 계획입니다.

4.3.4 해결 방법

비즈니스로직은 전부 controller 로 옮겨주고, router 에서는 단순히 요청을 처리하고 컨트롤러를 호출하는 역할만 할당해주면 됩니다.

4.4 controller 폴더

4.4.1 코드 설명

```
const music = require("../models/music");
const recommended = require("../models/recommended");
const Sequelize = require("sequelize");
const passport = require("passport");
const Genre = require("../models/genre");
```

```
const Music = require("../models/music");
const upload = require("../middlewares/upload");
```

- 필요한 모듈을 불러와줍니다.

```
exports.getAllMusic = async (req, res) => {
  try {
    const musicInfo = await music.findAll();
    res.status(200).json(musicInfo);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: "Server Error" });
  }
};
```

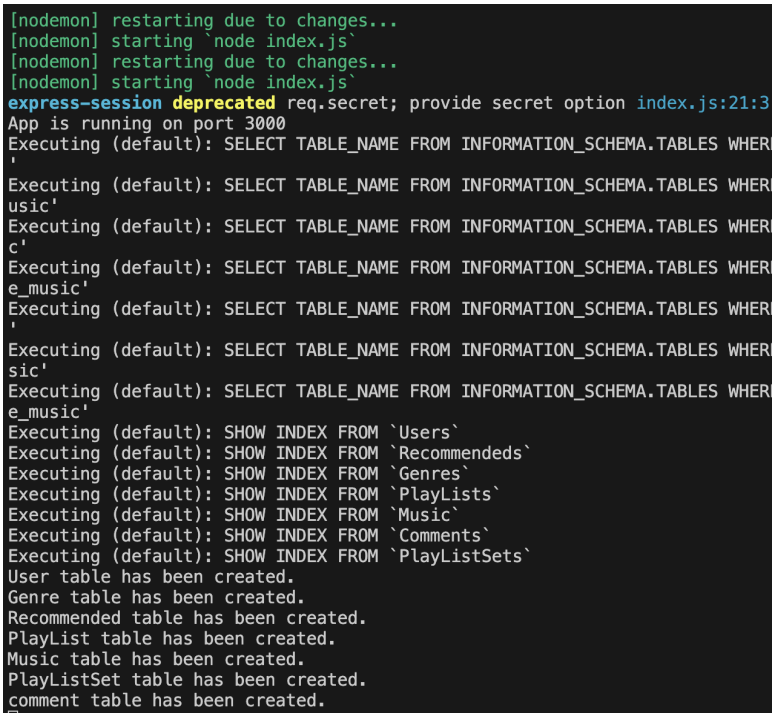
- router 폴더에서 필요하다고 생각하는 기능들을 만들어주었습니다.
- 가령 모든 음악정보를 불러오는 getAllMusic 함수에는 Sequelize 의 findAll() 을 이용합니다.
- 이외 SQL 문들은 추후 서술될 SQL 설명 파트에서 더 자세하게 설명하겠습니다.

이와 같이 각 controller 에서 로직을 담당하여 데이터베이스와 통신하며 데이터를 create, update, delete 등의 조작을 합니다.

5. 동작 화면 스크린샷

5.0 실행 화면

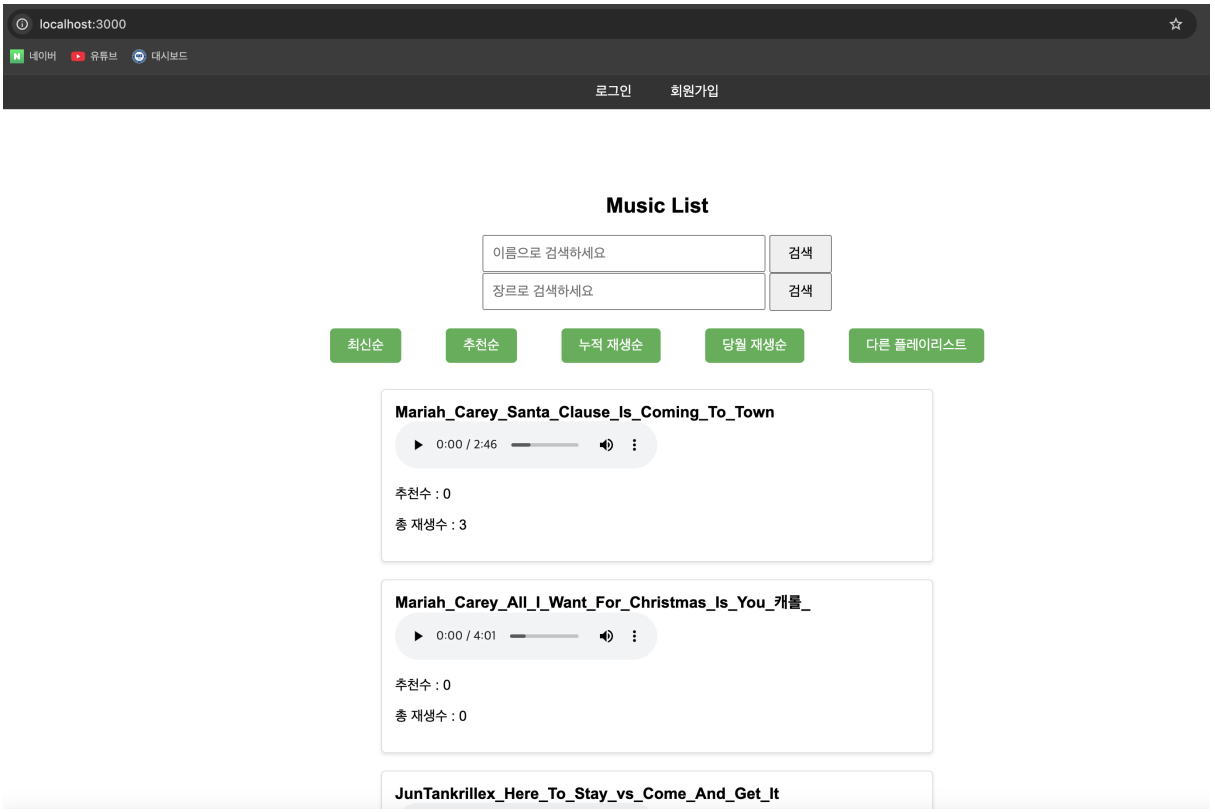
index.js 를 실행시킨 모습입니다.



5.1 기본 화면

5.1.1 로그인 되지 않은 상태

인터넷 브라우저에서 localhost:3000 으로 접속하면 아래 화면을 볼 수 있습니다.

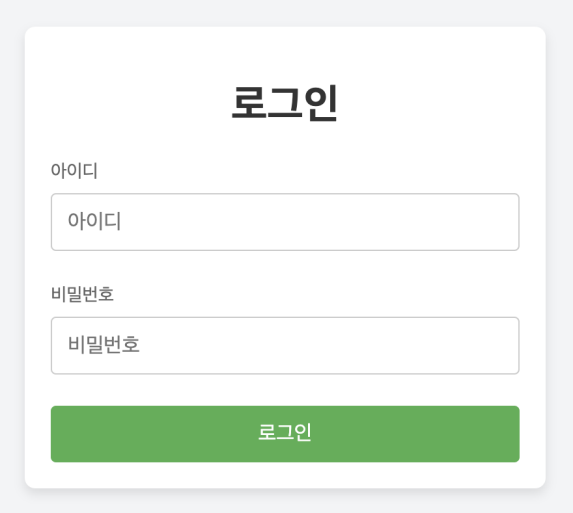


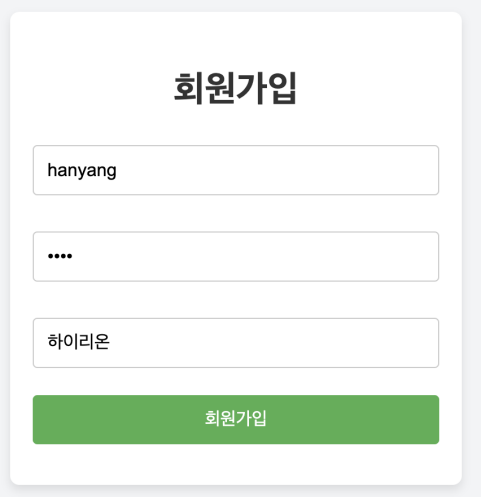
해당 상태에서 수행 가능한 기능입니다.

- 음악 감상
- 정렬 상태 변경
- 이름 및 장르로 노래 검색

(해당 음악은 미리 준비해놓은 더미데이터 입니다.)

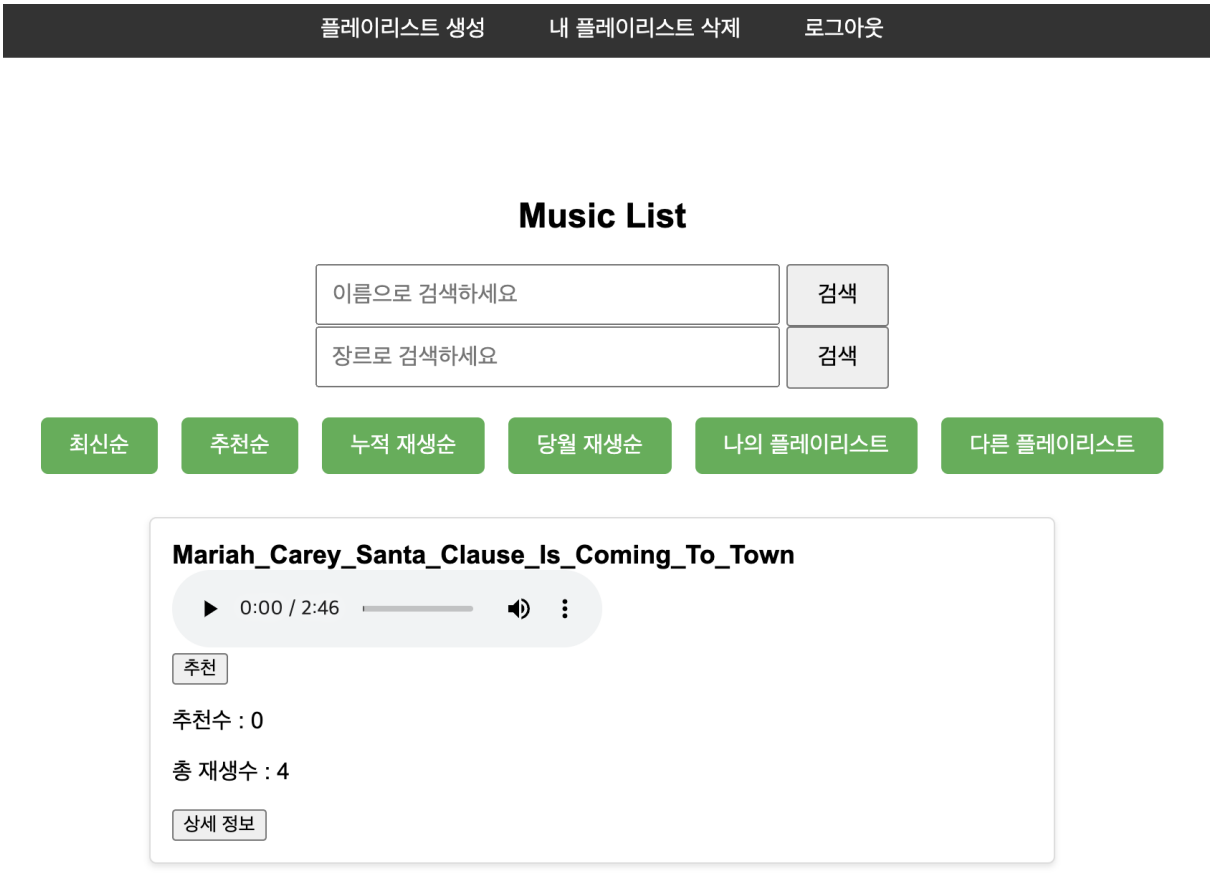
다른 사람의 플레이리스트나 세부 정보를 보기 위해서는 로그인을 해야하며, 접근시 middleware 로 등록한 is_loggined 가 실행되며 로그인을 하지 않았을 경우 로그인 화면으로 가게 됩니다.





회원가입시 DB에 유저의 정보가 저장되며, 비밀번호는 암호화되어 저장됩니다.

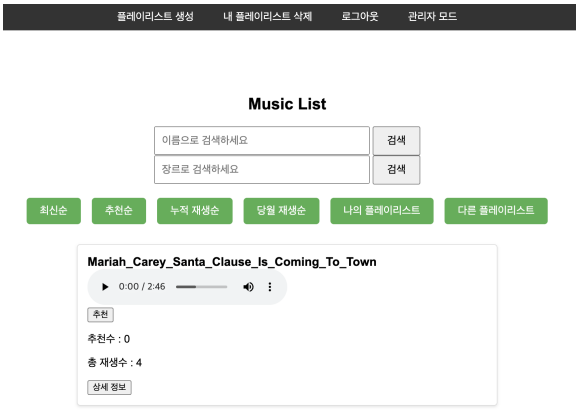
5.1.2 로그인 된 일반 유저



추가된 기능입니다.

- 음악 추천 기능
- 상세정보 열람 권한
  - 댓글 작성
  - 댓글 시청
- 플레이리스트 생성
- 플레이리스트 삭제
- 플레이리스트 확인
- 플레이리스트에 음악 추가

5.1.3 로그인 된 관리자 유저



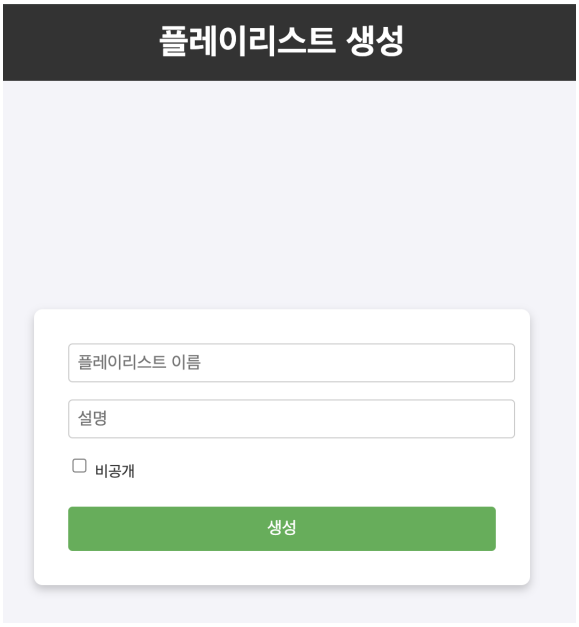
기존의 일반 유저의 모든 기능을 사용할 수 있으며, 관리자 모드 버튼이 추가되었습니다.

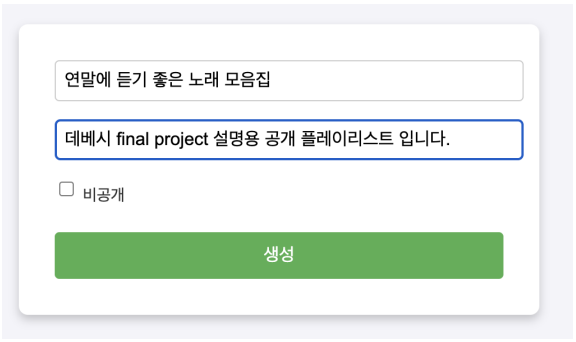
- 음악 추가
- 음악 삭제
- 댓글 관리
- 사용자 관리
- 조회수 초기화

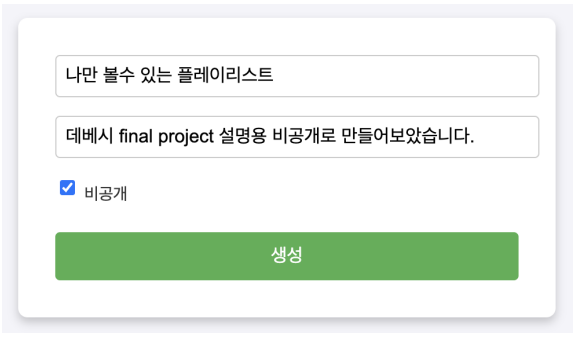
5.2 플레이 리스트 관련 화면

5.2.1 플레이리스트 생성

유저는 직접 플레이리스트를 생성할 수 있습니다.





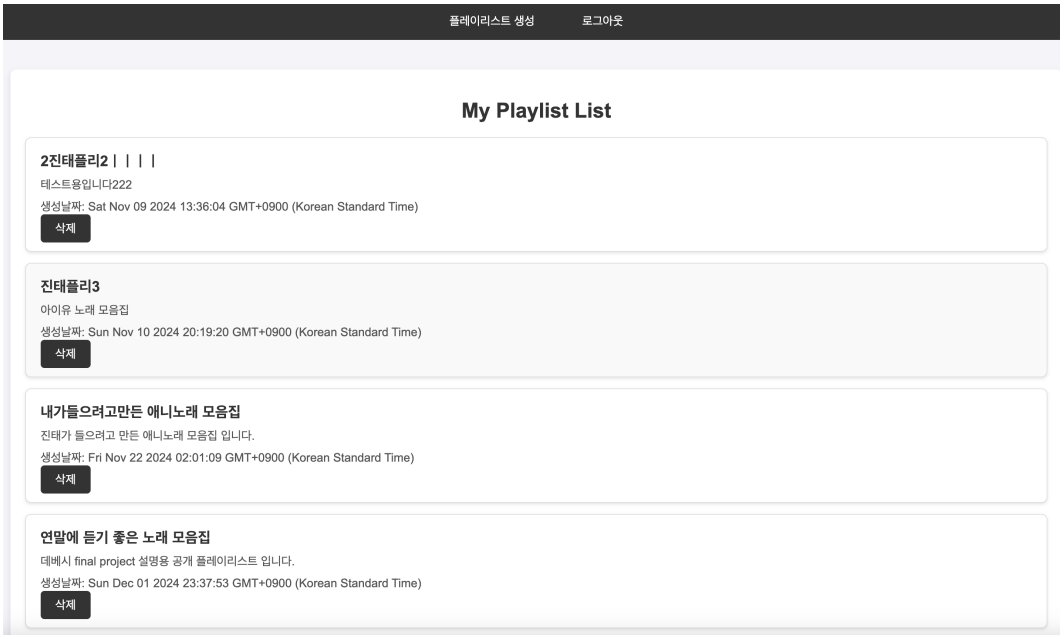


옵션을 걸어 비공개 플레이리스트도 생성할 수 있습니다.

5.2.2 플레이리스트 제거

혹은 이미 만들 플레이리스트를 제거할 수 있습니다.

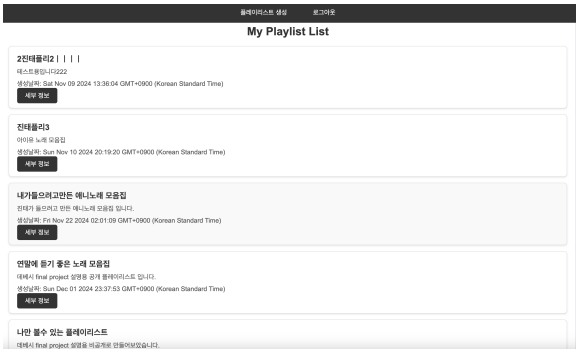
삭제 정책에 따라 제거된 플레이리스트 내부 정보들은 자동 삭제됩니다.



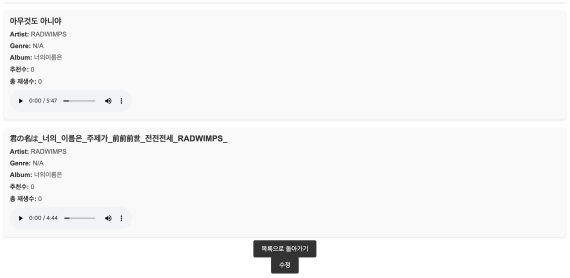
5.2.3 플레이리스트 확인

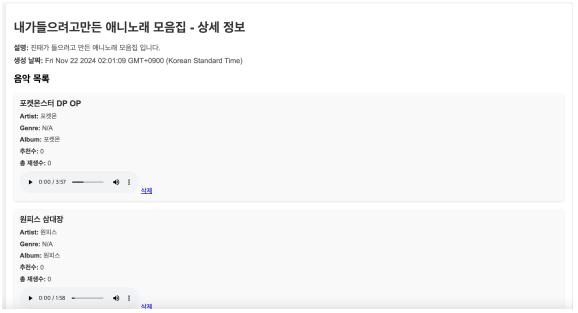
본인이 만들어놓은 플레이리스트를 확인할 수 있습니다.

세부정보를 누르면 플레이리스트 안에 추가한 노래 목록을 볼 수 있습니다.



또한 하단에 수정 버튼을 눌러 노래를 삭제할 수도 있습니다.

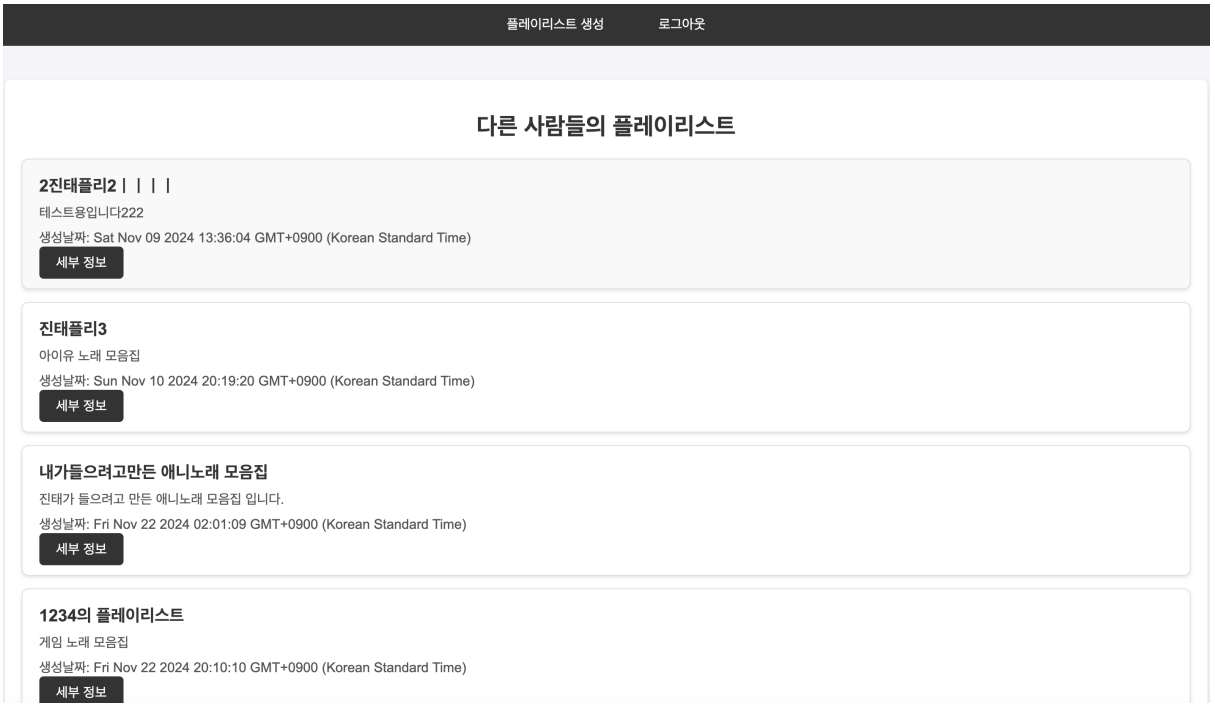




5.2.4 다른 사람들의 플레이리스트 확인

다른 사람들의 플레이리스트도 확인할 수 있습니다.

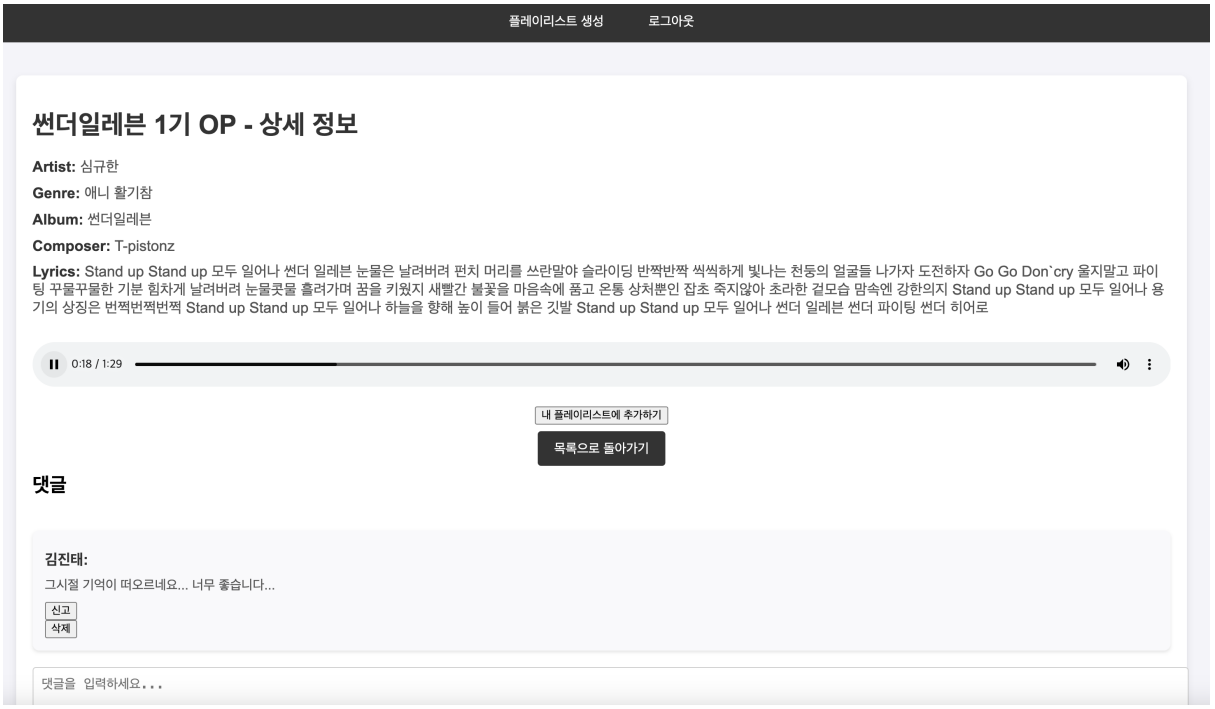
이 경우 비공개로 설정한 플레이리스트는 보여지지 않습니다.



5.2 음악 상세 정보

5.2.1 상세정보 확인

각 음악 별로 관리자가 설정한 음악 정보를 확인할 수 있습니다.



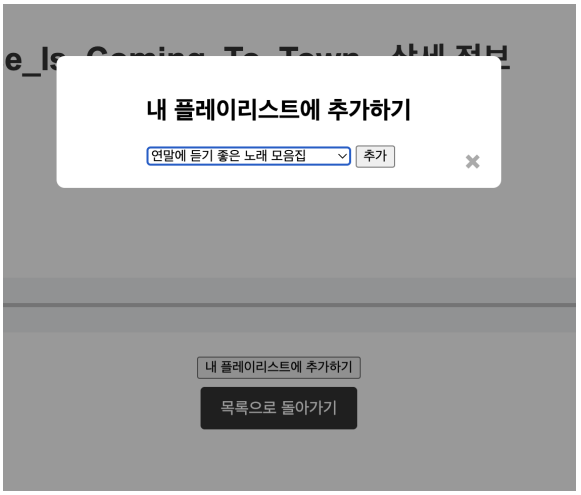
5.2.2 댓글 기능

다른사람의 댓글을 볼 수 있습니다.

본인의 댓글은 본인이 삭제할 수 있습니다.

5.2.3 플레이리스트에 추가 기능

본인의 플레이리스트 목록을 선택 후 추가할 수 있습니다.

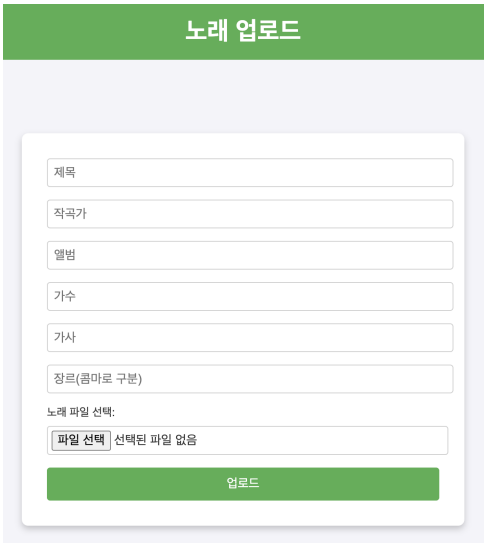


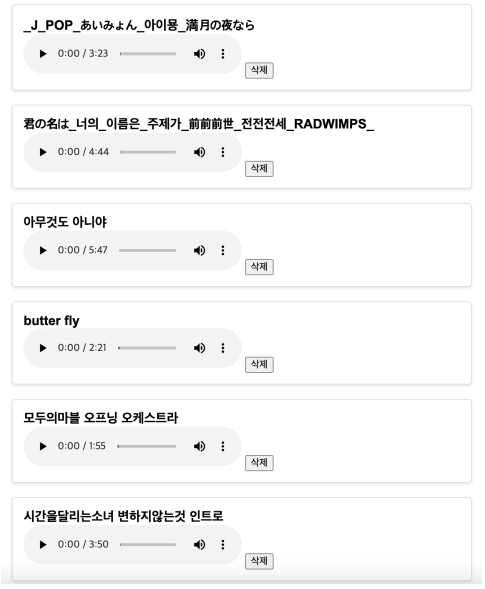
5.3 관리자 모드

5.3.1 음악추가 및 삭제

음악을 추가하거나 삭제할 수 있습니다.

음악 삭제시 삭제 정책에 따라 제거된 음악 내부 댓글, 플레이리스트, 등의 정보들은 자동 삭제됩니다.



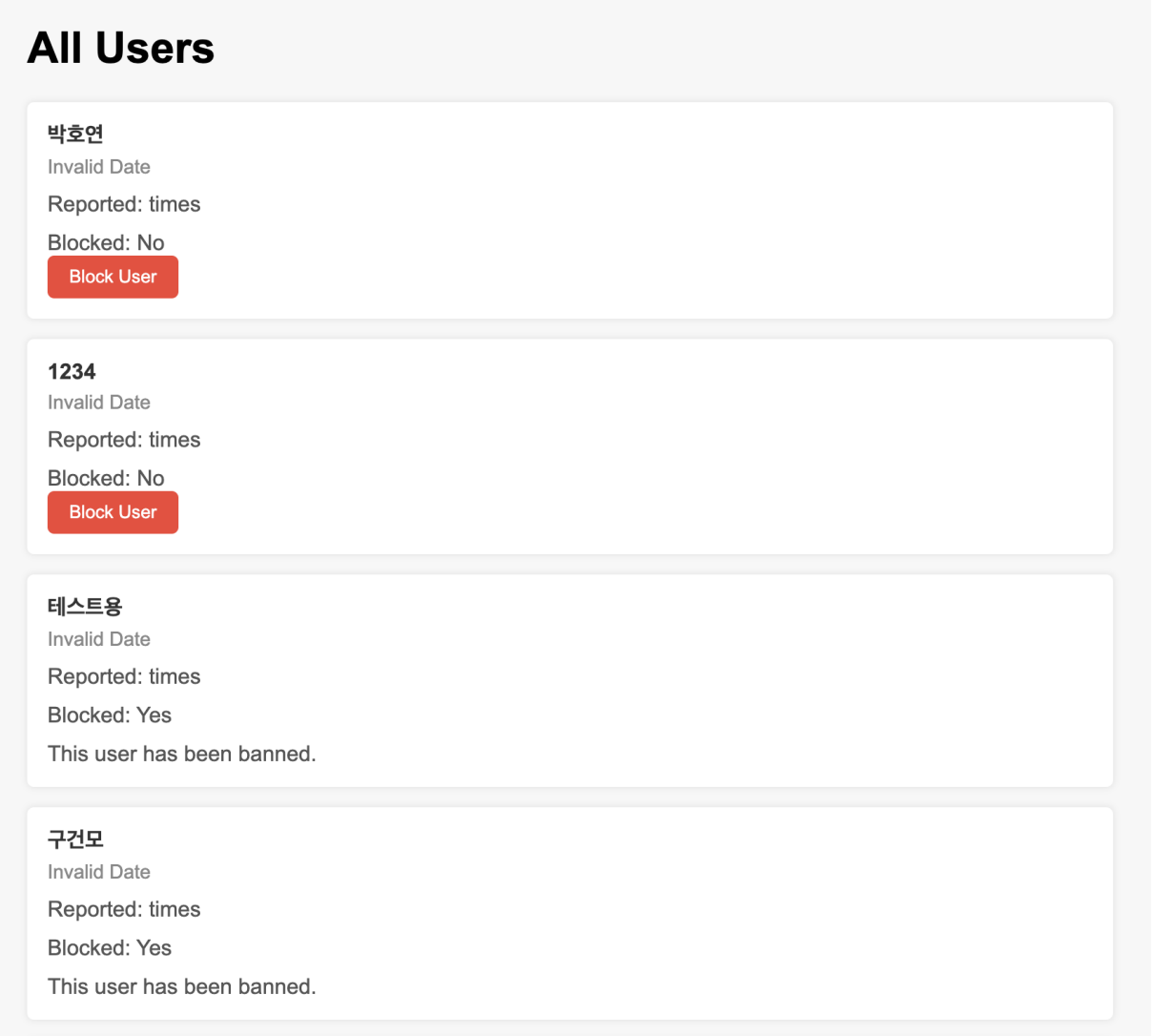


5.3.2 댓글 관리

신고수 내림차순으로 모든 댓글이 정렬이 됩니다. 불건전한 댓글을 삭제할 수 있습니다.

5.3.3 사용자 관리

악성 사용자들에 대해서 제제를 가할 수 있습니다. 해당 사용자들은 로그인이 불가능합니다.



5.3.4 조회수 초기화

클릭하면 monthly_play_count 가 전부 0으로 세팅됩니다.

```
router.get("/update_pc", async (req, res) => {
  const musicList = await Music.findAll();
  //musicList 의 모든 monthly_play_count를 0으로 초기화
  musicList.forEach(async (music) => {
    await Music.update(
      {
        monthly_play_count: 0,
      },
      { where: { music_id: music.music_id } }
    );
  });
  res.redirect("/");
});
```

* total_play_count int	* monthly_play_count int
Filter	Filter
0	0
0	0
0	0
0	0
0	0
0	0
6	0
1	0
0	0
0	0

6. 사용되는 SQL문 명세

6.1 commentController.js

- getAllComments : 모든 댓글을 return 합니다. 관리자가 댓글을 볼때 사용됩니다.

```
const commentInfo = await comment.findAll({ order: orderby })
```

- getCommentByMusicId : music_id 에 맞는 댓글을 return 합니다. 세부항목을 볼때 사용됩니다.

```
const commentInfo = await comment.findAll({
      where: { music_id: musicId },
    });
    return commentInfo;
```

- getCommentByUserId : params 로 전달된 id에 맞는 댓글을 return 합니다.

```
const commentInfo = await comment.findAll({
      where: { user_id: req.params.user_id },
    });
    return commentInfo;
```

- createComment : 댓글을 생성할때 호출됩니다.

```
const { music_id } = req.body; // 댓글을 달 때 music_id는 req.bo
    await comment.create(req.body);
```

- reportComment : 이전에 신고한 기록이 있는지 먼저 확인합니다. 이후 reported 가 1 증가합니다.

```
const commentInfo = await comment.findOne({ where: { comment_
...
await comment.update(
      { reported: updatedReportCount },
      { where: { comment_id } }
    );
```

- blockComment : 관리자가 댓글을 삭제할때 사용됩니다. 완전 삭제가 아닌 block 을 1 로 설정합니다.

```
const commentInfo = await comment.findOne({ where: { comment_
...
await comment.update({ is_blocked: true }, { where: { comment
```

6.2 musicController.js

- **getAllMusic : 모든 음악을 return 합니다.**

```
const musicInfo = await music.findAll();
```

- **createMusic : 음악 생성시에 호출됩니다. 실제로 파일을 보내는 로직은 미들웨어에 등록되어있습니다.**

```
const musicInfo = await music.create({
  ...req.body, // 폼 데이터
  file_path: filePath, // 파일 경로 추가
});
```

- **getAllMusicByDate, getAllMusicByPlayCount, getAllMusicByRecommendation**
- 특정 기준(날짜, 재생 수, 추천 수)에 따라 음악 목록을 정렬하는 기능을 제공합니다.
- 필요하다고 생각했지만 데이터를 백엔드에서 처리한 후 프론트엔드로 넘겨주는 방식으로 변경하면서 필요성이 줄어들어 해당 기능을 프론트엔드에서 구현하도록 수정했습니다.

```
const musicInfo = await music.findAll({
  order: [["release_date", "DESC"]],
});

const musicInfo = await music.findAll({
  order: [["total_play_count", "DESC"]],
});
res.status(200).json(musicInfo);


const musicInfo = await music.findAll({
  order: [["recommendation", "DESC"]],
});
res.status(200).json(musicInfo);
```

- **getMusicById : 음악 세부 정보를 볼 때 사용됩니다. 파일 경로를 알아야 하기 때문입니다.**

```
const musicInfo = await music.findOne({
  where: { music_id: musicId },
});
return musicInfo;
```

- **recommendMusic : 음악을 추천할 때 사용됩니다. 이미 추천한 기록이 있는지 먼저 확인합니다. 이후 추천수를 1 증가 시킵니다.**

```
const recommendInfo = await recommended.findOne({
  where: { music_id, UID },
});
...

const musicInfo = await music.findOne({
  where: { music_id },
});
await music.update(
  { recommendation: musicInfo.recommendation + 1 },
  { where: { music_id } }
);
await recommended.create({ music_id, UID });
```

- **endMusic : 음악이 종료되었을때 호출됩니다.** total_play_count, monthly_play_count 가 각 1씩 증가합니다.

```
const musicInfo = await music.findOne({
  where: { music_id },
});
await music.update(
  {
    total_play_count: musicInfo.total_play_count + 1,
    monthly_play_count: musicInfo.monthly_play_count + 1,
  },
  { where: { music_id } }
);
res.status(200).json({ message: "재생수 증가" });
```

- **searchMusic : 제목으로 음악을 검색할때 사용됩니다. OP.like 는 SQL 에서 LIKE 문법과 동일한 역할을 수행합니다.**

```
const searchQuery = req.query.q || "";
```

```
let musicList = [];
if (searchQuery.trim()) {
  musicList = await music.findAll({
    where: {
      title: {
        [Sequelize.Op.like]: `%${searchQuery}%`,
      },
    },
  });
}
```

- **searchMusicByGenre : 장르로 음악을 검색할 때 사용됩니다. 위와 동일합니다.**

```
const genre = req.query.q || "";
let musicList = [];
if (genre.trim()) {
  const genreData = await Genre.findAll({
    where: { genre_name: genre },
    include: [
      {
        model: Music,
        as: "music",
      },
    ],
  });

  musicList = genreData.map((genreItem) => genreItem.musi
c);
}
```

6.3 playlistController.js

- **getAllPlaylist : 모든 플레이리스트를 return 합니다.**
- 실제로 사용되지는 않았습니다.

```
const playlistInfo = await playlist.findAll();
```

- **createPlaylist : 플레이리스트를 생성할때 사용됩니다.**

```
const playlistInfo = await playlist.create(req.body);
```

- **getPlaylistByUserId : 특정 사람의 플레이리스트를 확인할때 사용됩니다.**
- 디버깅할때 사용되었습니다.

```
const playlistInfo = await playlist.findAll({
  where: { maker_id: UID },
});
return playlistInfo;
```

- **getPlaylistByMyId**
- 본인의 플레이리스트를 확인할때 사용됩니다.
- 본인의 플레이리스트를 확인하기 때문에 is_open 을 조건으로 걸지는 않았습니다.

```
const playlistInfo = await playlist.findAll({
  where: { maker_id: UID },
});
return playlistInfo;
```

- **addMusic : 플레이리스트에 음악을 추가할때 사용됩니다.**

```
const playlistSetInfo = await playlistSet.findOne({
  where: { music_id: req.body.music_id, playlist_id: req.bo
dy.playlist_id },
});

if (playlistSetInfo) {
```

```
  // 이미 추가한 음악이라면 바로 return 합니다.
  return;
}

await playlistSet.create(req.body);
```

- **getPlaylistByPlaylistId : 플레이리스트ID 로 플레이리스트를 찾을때 사용됩니다.**
- 디버깅할때 유용하게 사용했습니다.

```
const playlistInfo = await playlist.findOne({
  where: { playlist_id },
});
return playlistInfo;
```

6.4 이외 router 에서 controller 로 옮기지 않은 SQL문입니다

- music 의 detail 을 가져올때 사용됩니다.
- musicID 를 인자로 받으며 해당 id를 조건으로 걸고 검색을 합니다.
- 장르 역시 전부 찾아서 string 으로 합쳐 렌더링 인자로 넘겨줍니다.

```
    const music = await musicController.getMusicById(musicId,
    // console.log("music:", music);
    console.log("user:", req.user);
    const playlistInfo = await playlistController.getPlaylist
      res,
      req.user.UID
    );
    user_id = req.user.UID;
    const comments = await Comment.findAll({
      where: { music_id: musicId },
      include: [
        {
          model: User,
```

```
          as: "user",
          attributes: ["user_name"],
        },
      ],
    });

    const genres = await Gerne.findAll({
      where: {
        music_id: musicId,
      },
    });
```

- delete comment
- 본인 댓글을 삭제할때 사용됩니다.
- 관리자가 삭제하는것과 별개로 DB에서 아예 삭제가 됩니다.

```
const { comment_id } = req.body;
try {
  await Comment.destroy({
    where: {
      comment_id,
    },
  });
```

- 다른 사람의 플레이리스트를 확인할때는 is_open 을 조건으로 걸어줍니다.

```
    const playlistInfo = await Playlist.findAll({
      where: { is_open: 1 },
    });
```

- delete playlist

- 삭제하려는 playlist 의 maker_id 와 passport.session() 으로 세션을 확인해 현재 본인의 정보와 비교해서 같은지 판단한 후, 같다면 삭제하는 코드입니다.

```
passport.session();
playlistInfo = await Playlist.findAll({
  where: { maker_id: req.user.UID },
});
if (playlistInfo[0].maker_id != req.user.UID) {
  return res.redirect("/");
}
res.render("delete_playlist.ejs", {
  user: req.user,
  playlistInfo,
  isLoggedIn: req.isAuthenticated(),
});
```