

프로젝트 결과 보고서

음악 앨범 공유 사이트 제작

컴퓨터과학부

2016920004

권은진

클라우드 컴퓨팅

2020.06.20

목 차

1. 개요.....	1
1.1. 주제	1
1.2. 개발 환경	1
1.3. 수행 방안	1
2. 시스템 설계	2
2.1. 사용 사례 목록.....	2
2.2. 데이터베이스 설계	3
2.3. 시스템 구조.....	4
3. 구현 결과.....	5
3.1. 회원가입 및 로그인.....	5
3.2. 메인 화면	6
3.3. 앨범 조회	7
3.4. 프로필 조회.....	8
3.5. 데모	10
4. 결론.....	10
4.1. 문제점 및 해결방법.....	10
4.2. 느낀 점	10
5. 참고 자료.....	10

1. 개요

1.1. 주제

본 프로젝트에서는 아티스트가 제작한 음악 앨범을 웹에 업로드를 하거나, 다른 아티스트가 제작한 앨범을 조회하거나 아티스트의 정보를 조회할 수 있는 음악 앨범 공유 사이트를 제작한다.

1.2. 개발 환경

구분	이름 및 버전
운영체제	Ubuntu 18.04
프로그래밍 언어	Python 3
프레임워크	Flask
데이터베이스	Redis

[표 1-1] 개발 환경

1.3. 수행 방안

구현할 핵심 기능은 다음과 같다.

핵심 기능	설명
회원제	아티스트는 회원 가입 후, 앨범등록과 협업하기 기능을 이용할 수 있다.
앨범 등록	아티스트는 제작한 앨범을 웹에 업로드하거나 이미 올라온 정보를 수정 및 삭제할 수 있다.
앨범 조회	웹에 올라온 모든 앨범 또는 특정 앨범을 조회할 수 있다.
협업하기	다른 아티스트와 협업할 수 있다.
프로필 조회	아티스트의 정보를 조회할 수 있다.

[표 2-2] 핵심 기능

위 기능을 구현하기 위해 데이터베이스에는 앨범과 아티스트의 정보가 저장되며, 각각 앨범 식별자와 아티스트 식별자를 키 값으로 가진다. Redis는 NoSQL 데이터베이스이기 때문에 식별자를 통해서만 원하는 데이터에 접근할 수 있다.

원활한 테스트를 위해 Kaggle의 [Music Label Dataset](#)을 참고하여 데이터를 생성하였으며, 해당 데이터셋은 아티스트 정보 10,000건과 앨범 정보 5,000건을 포함한다.

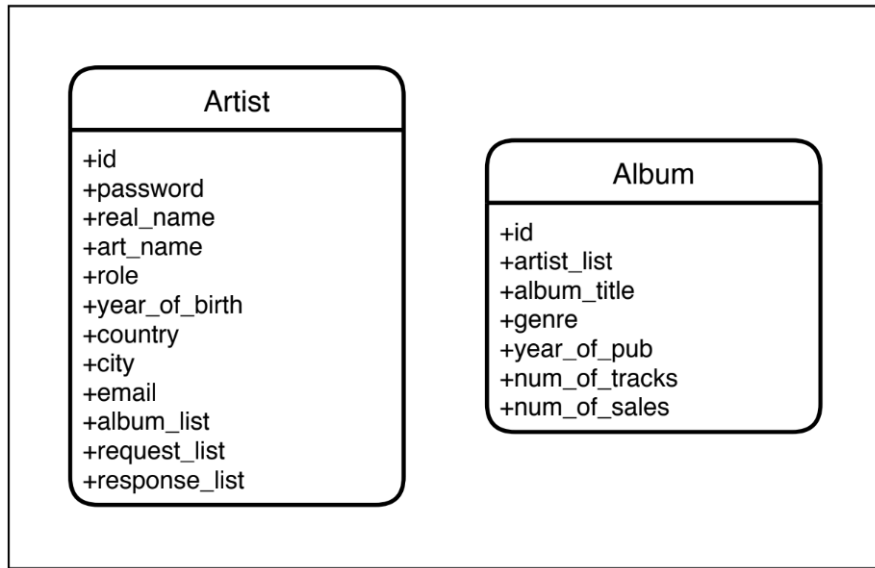
2. 시스템 설계

2.1. 사용 사례 목록

기능 구분	요구사항 식별자	사용 사례 구분	설명
회원제	REQ-SW-01	회원가입	아티스트를 등록하는 기능
	REQ-SW-02	로그인	사용자가 입력한 정보와 등록된 아티스트가 맞는지 검사한 후 회원제를 허용하는 기능
앨범 등록	REQ-SW-03	앨범 올리기	등록된 아티스트가 자신이 제작한 앨범정보를 웹에 올리는 기능
	REQ-SW-04	앨범 정보 수정	자신이 제작한 앨범 정보를 수정하는 기능
	REQ-SW-05	앨범 정보 삭제	자신이 제작한 앨범 정보를 삭제하는 기능
앨범 조회	REQ-SW-06	모든 앨범 조회	사용자가 메인 화면에서 모든 앨범을 조회하는 기능. 처음에는 30개만 보여주고 “더 보기” 버튼 누르면 30개를 추가적으로 보여준다.
	REQ-SW-07	앨범 장르별 조회	사용자가 메인 화면에서 장르별로 앨범을 볼 수 있는 기능. 처음에는 30개만 보여주고 “더 보기” 버튼 누르면 30개를 추가적으로 보여준다.
	REQ-SW-08	앨범 세부 정보 확인	특정 앨범에 대한 세부 정보를 조회하는 기능
협업하기	REQ-SW-09	협업 요청	등록된 아티스트는 다른 아티스트의 프로필 조회 시 협업을 요청할 수 있다.
	REQ-SW-10	협업 수락	자신에게 온 협업 요청을 수락하는 기능. 앨범 등록 시 제작자로 함께 아티스트 정보를 추가할 수 있다. 협업이 유효한 기간은 일주일이다.
	REQ-SW-11	협업 거절	자신에게 온 협업 요청을 거절하는 기능. 거절당한 상대는 하루 동안은 다시 요청할 수 없다.
프로필 조회	REQ-SW-12	아티스트 정보 조회	특정 아티스트의 프로필을 조회하는 기능. 해당 아티스트의 앨범과 협업 결과물을 확인할 수 있다.
	REQ-SW-13	프로필 수정	본인 프로필 조회 시 정보를 수정하는 기능
	REQ-SW-14	협업 결과 조회	협업 요청 시 상대의 수락 및 거절 유무를 파악할 수 있고, 다른 아티스트로부터 온 협업 요청 결과를 확인하는 기능. 본인 프로필에서만 확인 가능
	REQ-SW-15	본인 앨범 조회	해당 아티스트가 제작한 모든 앨범을 조회하는 기능

[표 2-1] 사용 사례 목록

2.2. 데이터베이스 설계



[그림 2-1] 데이터베이스 설계도

구분	설명
id	아티스트 아이디 (Key)
artist_list	계정 비밀번호 해쉬
real_name	실제 이름
art_name	아티스트 활동명
role	역할 (예: Singer, Rapper, Rocker, ... 등)
year_of_birth	출생년도
country	현재 거주 국가
city	현재 거주 도시
email	이메일
album_list	등록한 앨범 식별자 목록
request_list	협업 요청 목록
response_list	협업 응답 목록

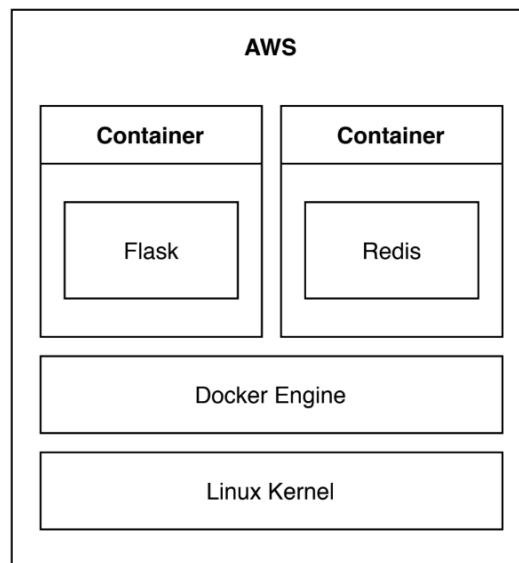
[표 2-2] 아티스트 데이터 구조

구분	설명
id	앨범 식별자 (Key)
artist_list	프로듀싱에 참여한 아티스트 목록
album_title	앨범 제목
genre	장르
year_of_pub	출간년도
num_of_tracks	앨범에 포함된 노래 개수
num_of_sales	앨범 판매량

[표 2-3] 앨범 데이터 구조

Artist와 Album은 각각 아티스트 및 앨범 정보를 의미하는 하나의 Key-Value Entry이며, id를 키 값으로 가진다. Artist 의 password 는 실제 비밀번호의 해쉬 값을 저장하고 있고, album_list 는 해당 아티스트가 제작한 앨범에 해당되는 Album ID 들을 저장하는 리스트이다. 또한, request_list 는 해당 아티스트가 협업 요청을 했을 때 저장되는 데이터 목록이며 요청 대상인 Artist ID 와 요청 결과인 status, 요청 날짜인 datetime 이 저장된다. 마찬가지로 response_list 는 해당 아티스트에게 다른 아티스트가 협업 요청을 했을 때 저장되는 데이터 목록이며 응답해야하는 대상인 Artist ID 와 요청 결과인 status, 요청 날짜인 datetime 이 저장된다. 이전에 설명한 album_list 는 단순히 식별자를 저장하는 리스트 자료구조이지만, request_list 와 response_list 는 모두 Artist ID 를 키 값으로 하는 Key-Value 자료구조를 요소로 가지는 리스트 자료구조이다. Album 의 artist_list 는 해당 앨범을 제작(produce)한 아티스트들의 식별자를 저장하는 리스트 자료구조이다. 협업 요청을 하지 않았다면 1 명의 아티스트 식별자만 저장하게 된다.

2.3. 시스템 구조

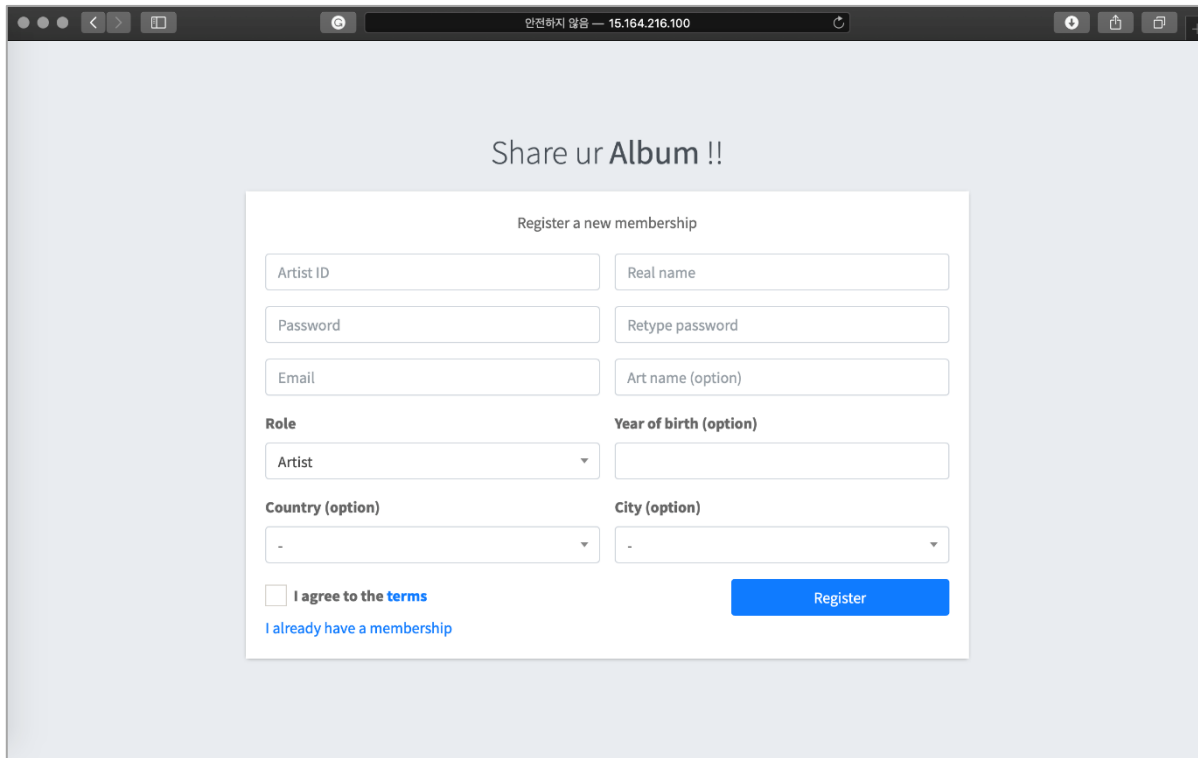


[그림 2-2] 시스템 설계도

수업 시간에 배운 도커를 활용해서 개발 환경과 유사하게 서버의 배포 환경을 구축하였으며, Flask와 Redis 컨테이너가 하나의 네트워크로 연결되도록 docker-compose로 도커를 실행하였다.

3. 구현 결과

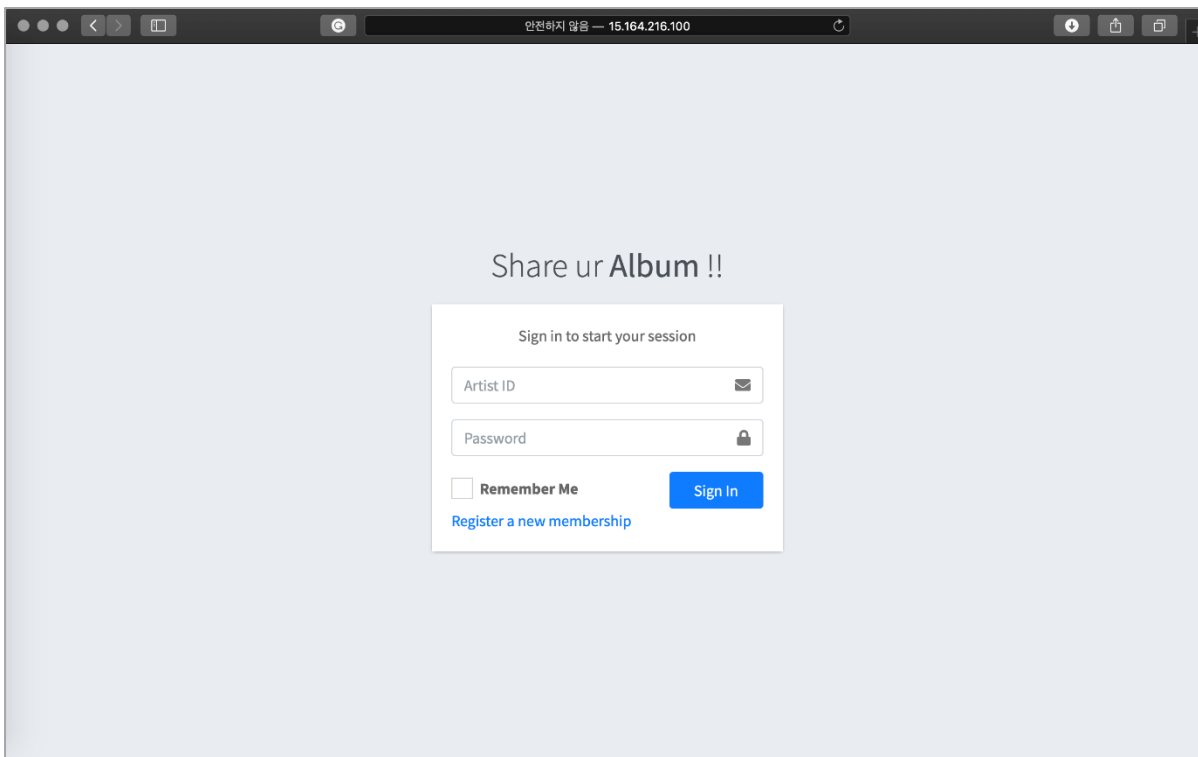
3.1. 회원가입 및 로그인



The screenshot displays a web browser window with the address bar showing '안전하지 않음 — 15.164.216.100'. The page title is 'Share ur Album !!'. The main content is a registration form titled 'Register a new membership'. The form includes the following fields and options:

- Artist ID (text input)
- Real name (text input)
- Password (text input)
- Retype password (text input)
- Email (text input)
- Art name (option) (text input)
- Role (dropdown menu, currently set to 'Artist')
- Year of birth (option) (text input)
- Country (option) (dropdown menu, currently set to '-')
- City (option) (dropdown menu, currently set to '-')
- ☐ I agree to the [terms](#)
- [I already have a membership](#)
- [Register](#) (blue button)

[그림 3-1] 회원가입 페이지



The screenshot displays the same web browser window as the previous image, but the main content is a login form titled 'Sign in to start your session'. The form includes the following fields and options:

- Artist ID (text input with an eye icon for visibility toggle)
- Password (text input with a lock icon for visibility toggle)
- ☐ Remember Me
- [Sign In](#) (blue button)
- [Register a new membership](#)

[그림 3-2] 로그인 페이지

3.2. 메인 화면

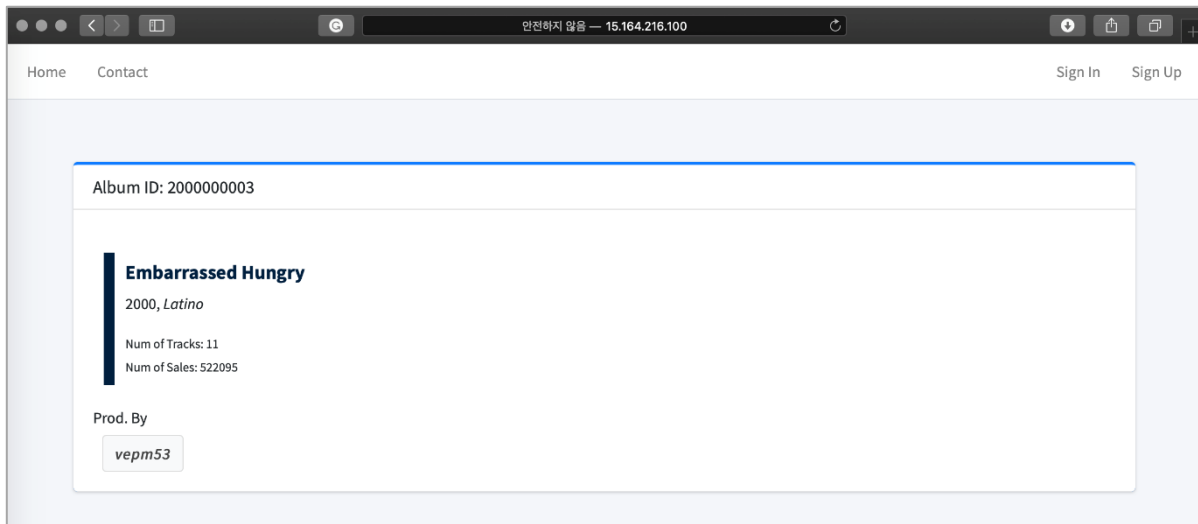


[그림 3-3] 메인 페이지

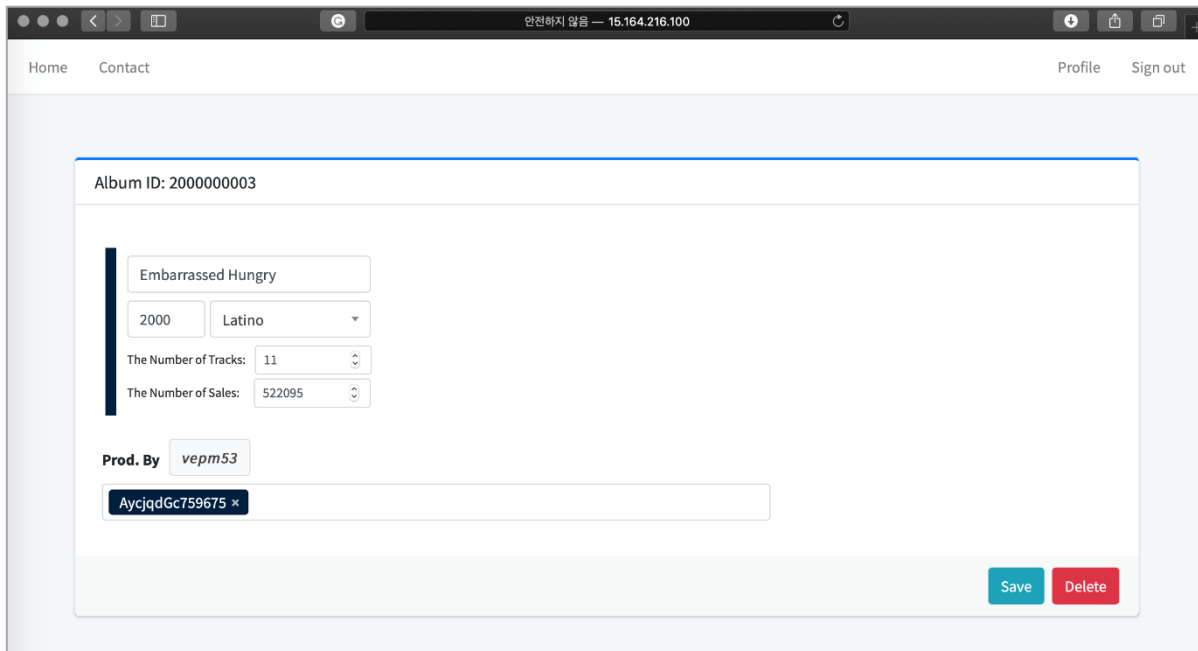
위에서부터 이벤트, 공지사항, 현재 멤버 수와 등록된 앨범 수를 알려주고 있고, 그 밑에 무작위로 추천된 3 명의 아티스트를 보여준다. 추천된 아티스트의 아이디와 활동 명 및 앨범 정보가 포함되어 있다. 마지막으로 모든 앨범을 보여주는데, 장르별로 모아서 볼 수도 있다. 장르는 총 38 개이며, 처음에 30 개씩 가져와서 보여주고 더 보기 버튼을 누르면 추가로 서버로부터 요청해서 데이터를 보여준다.

메인 화면에서 보여지는 앨범 정보는 앨범 제목만 있으며, 클릭 시 아래의 앨범 조회 페이지로 넘어간다.

3.3. 앨범 조회

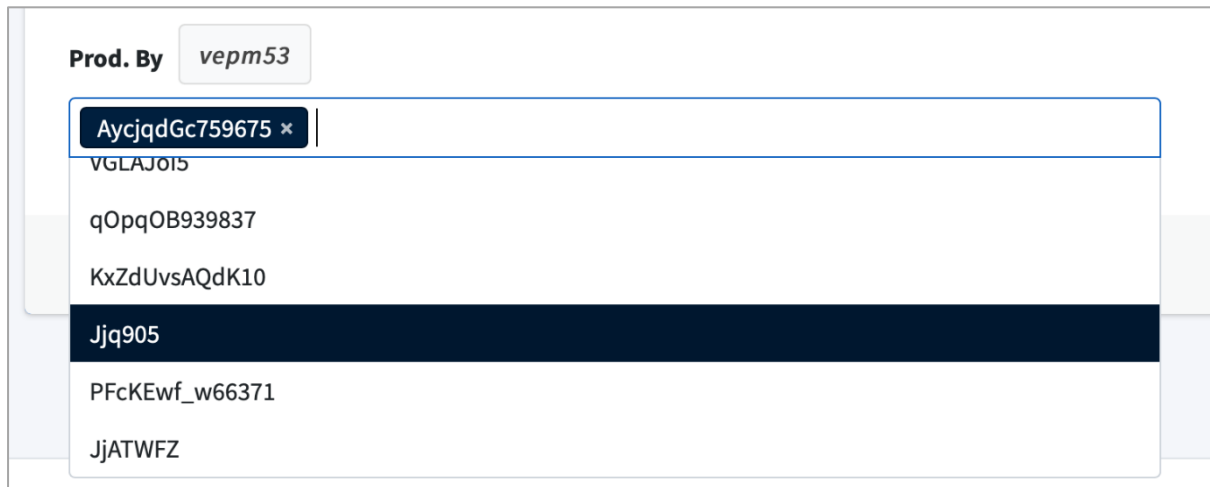


[그림 3-4] 앨범 조회



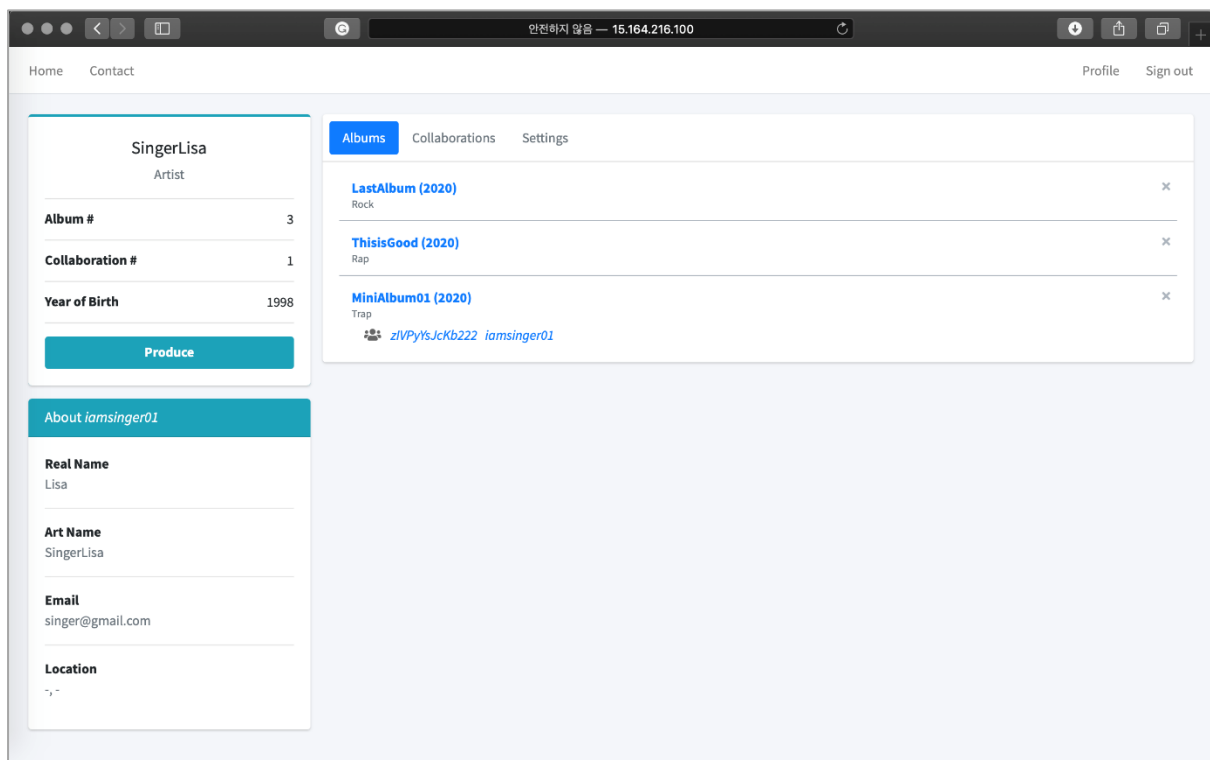
[그림 3-5] 앨범 등록 및 수정 페이지

앨범 수정 페이지의 아래에 위치한 프로듀서 정보에 협업 요청이 수락된 다른 아티스트의 정보들을 추가할 수 있다.

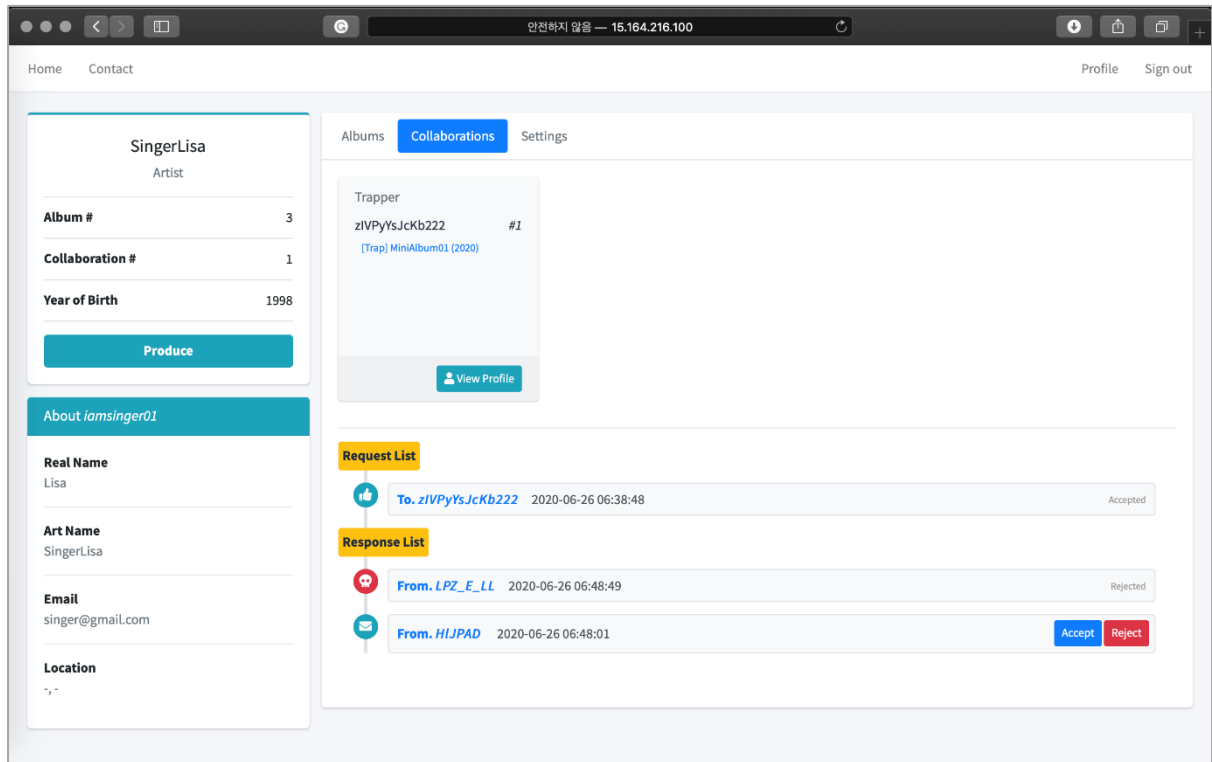


[그림 3-6] 협업 요청 결과: 앨범 등록 페이지의 프로듀서 목록

3.4. 프로필 조회

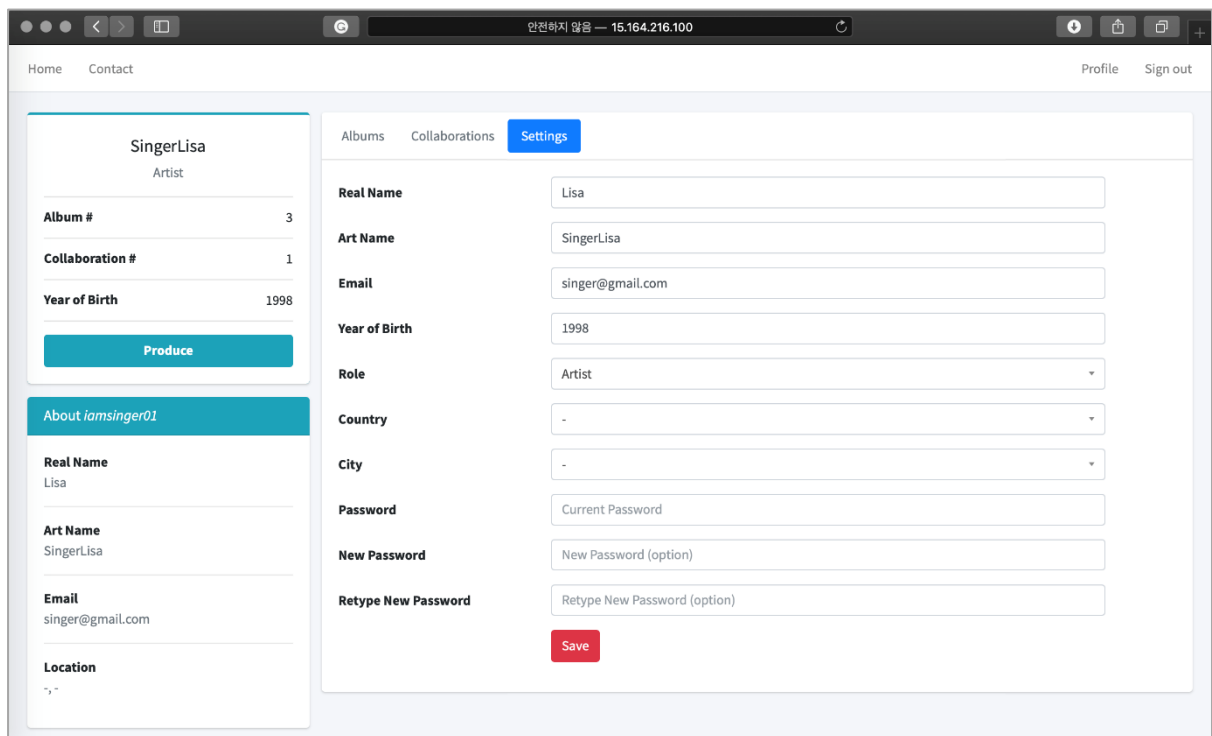


[그림 3-7] 프로필 화면: 앨범



[그림 3-8] 프로필 화면: 협업

[그림 3-8]에서 보이는 협업 요청 결과 및 응답 결과는 본인의 프로필 페이지에서만 확인가능하다.



[그림 3-9] 프로필 화면: 개인 정보 설정

3.5. 데모

영상: <https://drive.google.com/file/d/1U9wndKZQueEjXwzIL0qqOMjtO8QwXcD7/view?usp=sharing>

사이트: <http://ec2-15-164-216-100.ap-northeast-2.compute.amazonaws.com/>

4. 결론

4.1. 문제점 및 해결방법

데이터가 많아서 모든 앨범 정보를 메인 페이지에 렌더링하는데 어려움이 있었다. 따라서, 나눠서 보여줘야 했는데 데이터베이스에서는 키 값만으로만 데이터에 접근할 수 있어서 하나하나 키를 확인해서 보여줘야 하는 상황이었다. 굉장히 성능이 좋지 않았기 때문에 개선할 필요가 있었고, 앨범 데이터의 식별자를 숫자로 지정해서 데이터가 등록되는 순서대로 숫자를 하나씩 늘려 저장하기로 하였다. 그 이유는 렌더링 된 마지막 앨범 데이터의 식별자만 알고 있으면 다음 30개의 데이터를 받아오는데 시간 복잡도가 $O(n)$ 이 되었기 때문이었다.

4.2. 느낀 점

(키, 값) 쌍으로 저장하는 NoSQL 데이터베이스는 처음 사용해봤는데, 무작정 데이터를 저장하기 보다 어떤 목적으로 데이터를 저장할 지 많은 고민을 해야 했다. 또한, RDBMS 만 사용해왔기 때문에 외래 키 개념이 없는 상태에서 정렬된 데이터를 가져오는 경우를 최대한 제외하려고 하였다. 그러나, 앨범을 장르별로 보여줄 필요가 있는 등 없으면 아쉬운 기능들에 대해서는 시간 복잡도가 $O(n^2)$ 이더라도 그대로 구현할 수밖에 없었다. 성능 측면에서는 아쉬운 점이 많이 있었다. 그래도 앨범 조회나 프로필 조회, 그리고 로그인 시 검사하는 등 키 값으로 데이터를 조회할 때는 $O(1)$ 의 시간 복잡도를 갖기 때문에 필드 별 정렬이 필요하지 않은 경우에는 앞으로도 유용하게 사용할 수 있을 것 같다.

사용자 세션을 관리하기 위해 세션키를 Redis 에 저장하여 로그인 검사를 했는데 이 부분이 가장 Redis 의 장점이 부각된 것이라고 생각을 했다. 지금은 5000 건의 아티스트 정보를 저장하고 있지만 더 많은 데이터를 가지고 있었다면 각각의 세션을 관리하는데 $O(n)$ 의 시간 복잡도만 되어도 성능이 늦어졌기 때문이다. 그 외에 협업하기 기능을 구현하기 위해 사용자별로 협업 요청/응답 결과 필드를 생성해서 각 사용자별 식별자를 저장했는데 식별자만 저장하면 되기 때문에 조회 속도 측면에서 빠를 뿐만 아니라 디스크 공간도 절약할 수 있어 NoSQL 의 특징이 돋보였다. 또한, Redis 의 명령어 인터페이스는 굉장히 단순해서 쉽고 빠르게 데이터베이스를 생성하고 수정할 수 있었다는 점에서 다음에도 Redis 를 활용하고 싶다는 생각이 들었다.

5. 참고 자료

- [1] Kaggle, "Music Label Dataset", <https://www.kaggle.com/revilrosa/music-label-dataset>
- [2] Tistory blog, "Flask + Redis를 이용한 로그인 서비스 구현", <https://jinseyou.tistory.com/6>
- [3] Redis Documentation, <https://redis-py.readthedocs.io/en/stable/>
- [4] Flask Documentation, <https://flask.palletsprojects.com/en/1.1.x/>