

Database Final Project - 音樂猜歌遊戲系統

Group 9

111550040	曾紹樟
111550080	曾煥宗
111550083	黃永恩
111550100	邱振源
111550143	林彥佑

● 簡介

An introduction of your application, including why you want to develop the application and the main functions of your application.

demo影片及code連結:

https://drive.google.com/drive/u/1/folders/1Cs_gckU57026lC11mB8agx6Nj-0-Ekco?fbclid=IwAR1fWpkO2nuSPYdQH0hqlDaMAbO7_hMGQ6kvFi_gZyR2Ayk7Kiq3iOOHD_A

● 動機:

和朋友出去玩的時候發現，猜歌遊戲是一個可以快速炒熱氣氛的方式。然而，在上網搜尋有關猜歌遊戲的網站之後，發現現在網路上的猜歌遊戲寥寥可數，就算有也是侷限在狹小的範圍。因此，如果想要玩猜歌遊戲，勢必得要有一個人出題，這也影響了大家的遊戲體驗。

Spotify現今社會許多人使用的音樂串流平台之一，亦是許多大學生選用的平台，其背後也擁有龐大的資料量，所以，我們選用Spotify設計資料庫，讓使用者可以透過設定限制來進行猜歌遊戲，如果聽到喜歡的歌也可以即時透過網站的查詢功能知道這首歌的詳細資料，並添加自己的歌單。

● 功能介紹:

- 首頁
- 每周排行:顯示Spotify每日前50名歌曲
- 查詢歌曲:使用者可以提供歌曲或藝人或專輯名稱來查詢歌曲，我們會先從資料庫搜尋歌曲並顯示，當資料庫發現沒有該歌曲時會自動將該歌曲更新置資料庫
- 製作歌單
- 猜歌小遊戲:使用者可以隨意輸入想要的猜歌條件，如歌手、專輯或歌曲名稱(可以猜是哪種版本增加樂趣)，程式會隨機播放片段並提供選項給使用者作猜歌，答對答錯都可以選擇下議題或結束猜題

● 資料庫設計-Schema

Database design - describe the schema of all your tables in the database, including keys and index, if applicable (why you need the keys or why you think adding an index is or is not helpful).

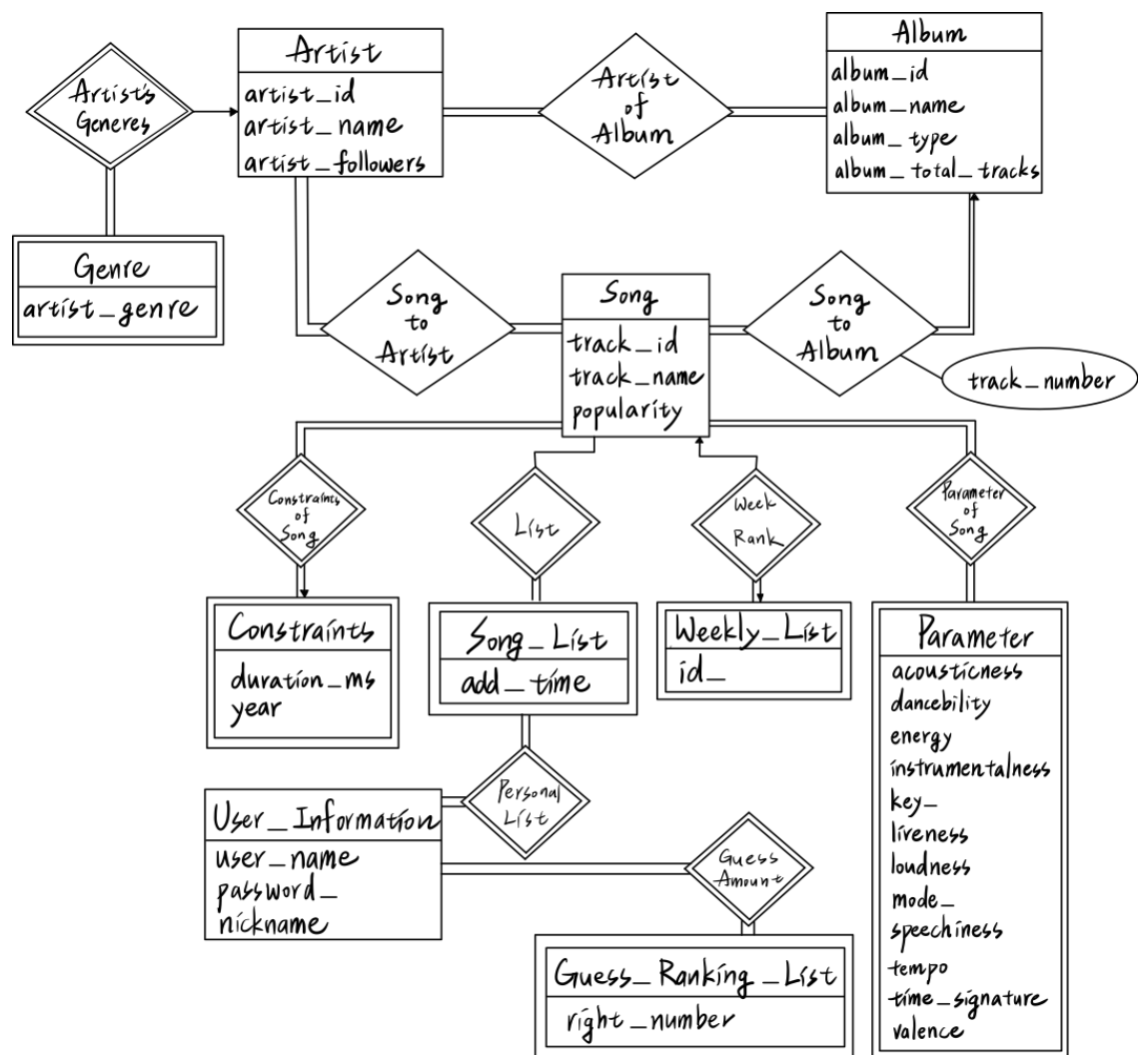
● Schema:

在歌曲相關資料的部分，我們使用kaggle中的Spotify Complete 1986 to 2023來當作原始資料庫，再針對缺乏的資料進行更新，因此，再歌曲相關資料的部分，我們參考上述資料庫的attribute，刪掉一些不需要的attribute。

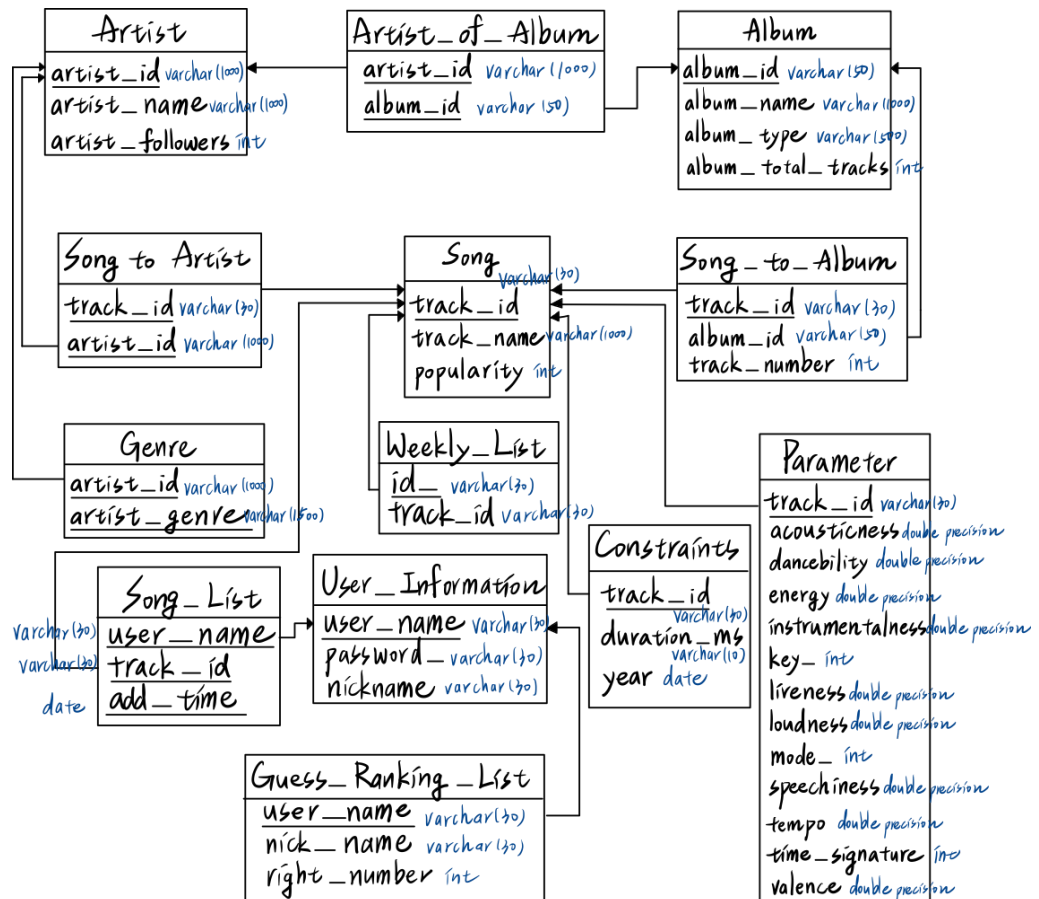
再來，我們針對系統上的一些功能，添加需要的attributes，與原先的部分一同設計，ER Model再將其轉換為Schema，有關ER Model和最後Schema的設計可以參考下圖。

在歌曲資訊的部分，PK主要都是track_id, artist_id, album_id用來記錄針對每首歌曲、歌手、專輯之間的相關性質，這三個也是主要各個表FK reference的attribute，而有些表的PK是由兩個以上的attribute構成，主要是我們在處理multivalue的時候必須把他們拆出來了；而使用者資訊的部分，我們主要則使用user_name作為PK，有關每個表仔細的PK與FK可以看到下圖schma的部分(_: PK ; →: FK)。

● ER Model



● Schema



- Index

我們也針對每個表create index, 主要是針對其primary key來進行生成, 例:

```
create index Genre_index on Genre(artist_id, artist_genre);
```

如此一來在透過query尋找資料的時候可以更加快速, 從而提升query的效率, 有關每個表詳細的index可以看檔案裏面的程式碼, 這裡就不一一列出來。

- 資料庫設計-Normal Form

Database design - describe the normal form of all your tables. If the tables are not in BCNF, please include the reason (performance trade-off, etc.).

我們的表都是BCNF, 詳細的functional dependency可以看下面的部分, 由於 { Primary Key we choose } is a candidate key for R, 所以 for all non-trivial FD in $\alpha \rightarrow \beta$ in F^+ , α are super key, 我們可以知道所有的表皆為 BCNF.

- Functional Dependency:

Artist

- (artist_id) \rightarrow (*all attributes in Artist)

Song

- (track_id) \rightarrow (*all attributes in Song)

Genre

- (artist_id, artist_genre) → (artist_id)
- (artist_id, artist_genre) → (artist_genre)

Song_to_Artist

- (track_id, artist_id) → (track_id)
- (track_id, artist_id) → (artist_id)

Artist_to_Album

- (artist_id, album_id) → (artist_id)
- (artist_id, album_id) → (album_id)

Album

- (album_id) → (*all attributes in Album)

Constraints

- (track_id) → (*all attributes in Constraints)

Song_to_Album

- (track_id) → (*all attributes in Song to Album)

Parameter

- (track_id) → (*all attributes in Parameter)

Weekly_List

- (id_) → (track_id)
- (track_id) → (id_)

User_Information

- (user_name) → (*all attributes in User Information)

Song_List

- (user_name, track_id, add_time) → (*all attributes in Song List)

Guess_Ranking_List

- (user_name) → (*all attributes in Guess Ranking List)

● Data Source and the Original Format

From the data sources to the database - describe the data source and the original format.

在歌曲資料的部分，我們使用kaggle上的資料集: Spotify Complete 1986 to 2023 (<https://www.kaggle.com/datasets/nicolasfierro/spotify-1986-2023>), 裡面紀錄了每首歌在Spotify的id、歌曲名字、人氣值、可以發行的市場、disc_number、長度、有沒有歌詞、是專輯裡的第幾首歌、URL連結、專輯id、專輯名稱、專輯發行日期、專輯型態、專輯裡有幾首歌、歌手名字、歌手id、主要的歌手id、主要的歌手名字、歌手類型、主要的個歌手追蹤者、分析URL、和一些參數 (acousticness, danceability, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, time_signature, valence)、年分。

● 資料匯入與更新

From the data sources to the database - describe the methods of importing the original data to your database and strategies for updating the data, if you have one.

- 資料匯入：

我們先將kaggle上的.csv檔下載下來，並藉由\copy public.spotify_dataset(track_id, track_name, popularity, available_markets, duration_ms, track_number, album_id, album_name, album_type, album_total_tracks, artists_names, artists_ids, principal_artist_id, principal_artist_name, artist_genres, principal_artist_followers, acousticness, danceability, energy, instrumentalness, key_, liveness, loudness, mode_, speechiness, tempo, time_signature, valence, year, duration_min) FROM 'C:\Program Files\PostgreSQL\16\bin\archive\spotify_dataset.csv' DELIMITER ',' CSV HEADER;來將其匯入table public.spotify_dataset (大表格)，輸進我們的資料庫。

- 資料更新：

使用AWS lambda function更新資料庫，主要功能為：

- 使用node-postgres連線至資料庫

```
1 import axios from 'axios';
2 import pkg from 'pg';
3 const { Pool, Client } = pkg;
4 const pool = new Pool({
5   user: 'postgres',
6   host: 'database-2.c8dhylstxkf5.us-east-1.rds.amazonaws.com',
7   database: 'postgres',
8   password: '',
9   port: 5432,
10  ssl: {
11    rejectUnauthorized: false
12  }
13 });
```













- 利用Spotify Web API 並使用axios發request獲取音樂相關資料

```

93 const GetTracksData_Limit = async(token, track_id_list_limit) => {
94   let track_data = [];
95   var url = 'https://api.spotify.com/v1/tracks?ids='+track_id_list_limit[0];
96   for(var i=1;i<track_id_list_limit.length;i++){
97     url+='%2c'+track_id_list_limit[i];
98   }
99   try{
100     // console.log(url);
101     const response = await axios.get(url, {
102       headers: {
103         'Authorization': `Bearer ${token}`
104       }
105     });
106     response.data.tracks.forEach((track) => {
107       var temp = [];
108       var t_genre = [];
109       var t_artist = [];
110       var t_song_to_artist = [];
111       // var t_market = [];
112       var t_artist_of_album = [];
113       var t_song = [];
114       var t_constraints = [];
115       var t_album = [];
116       var t_song_to_album = [];
117       var t_parameter = [];
118       // console.log(track.artists[0]);
119       t_genre.push(track.artists[0].id, track.artists[0].genres[0]||'null');
120       t_artist.push(track.artists[0].id, track.artists[0].name, track.artists[0].followers[0]||null);
121       t_song_to_artist.push(track.id, track.artists[0].id);
122       t_artist_of_album.push(track.artists[0].id, track.album.id);
123       t_song.push(track.id, track.name, track.popularity);
124       //console.log(typeof track.album.release_date);
125       var t_date = (track.album.release_date.length==10) ? track.album.release_date:null;
126       t_constraints.push(track.id, track.duration_ms, t_date);
127       t_album.push(track.album.id, track.album.name, track.album.album_type, track.album.total_tracks);
128       t_song_to_album.push(track.id, track.album.id, track.track_number);
129       t_parameter.push(track.id);
130       // console.log(t_album);
131       cache_genre.push(t_genre);
132       cache_artist.push(t_artist);
133       cache_song_to_artist.push(t_song_to_artist);
134       cache_artist_of_album.push(t_artist_of_album);
135       cache_song.push(t_song);
136       cache_constraints.push(t_constraints);
137       cache_album.push(t_album);
138       cache_song_to_album.push(t_song_to_album);
139       cache_parameter.push(t_parameter);
140       temp.push(track.album.album_type, track.album.total_tracks, track.album.id, track.album.name,
141         track.artists[0].genres, track.artists[0].id, track.artists[0].name, track.artists[0].followers,
142         track.avaliabile_markets, track.duration_ms, track.id, track.name, track.popularity, track.track_number,
143         track.album.release_date);
144       track_data.push(temp);
145       // console.log(track_data);
146     });
147   }
148   catch(error){
149     console.log(error);
150   }
151   return track_data;
152 }
153 };

```

```

14 var cache_genre = [];
15 var cache_artist = [];
16 var cache_song_to_artist = [];
17 // var cache_market = [];
18 var cache_artist_of_album = [];
19 var cache_song = [];
20 var cache_constraints = [];
21 var cache_album = [];
22 var cache_song_to_album = [];
23 var cache_parameter = [];
24
25 const GetToken = async() => {};
38
39 const GetAlbumsFromArtist = async(token, artist_id) => {};
60
61 const GetTracksFromAlbum = async(token, album_id) => {};
81
82 const GetTracksFromAlbums = async(token, album_id_list) => {};
92
93 const GetTracksData_Limit = async(token, track_id_list_limit) => {};
154
155 const GetTracksData = async(token, track_id_list) => {};
172
173 const GetTracksAudioFeatures_limit = async(token, track_id_list_limit) => {};
199
200 const GetTracksAudioFeatures = async(token, track_id_list) => {};
209
210 const GetArtistFromTrack = async(token, track_id) => {};
226
227 const GetArtistFromAlbum = async(token, album_id) => {};
243
244 const WriteCache = async() => {}
284
285 const GetTopPlaylist = async(token, play_list_id) => {};

```

- 將cache update到資料庫中

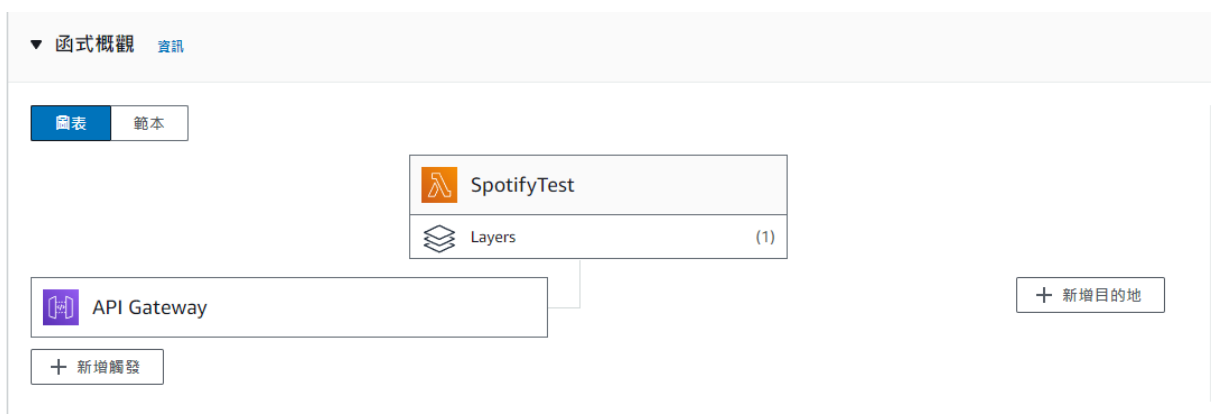
```

const WriteCache = async() => {
  for(let i=0;i<cache_album.length;i++){
    console.log(await pool.query(`insert into album values($1, $2, $3, $4) ON CONFLICT (album_id) DO NOTHING;`,
      [cache_album[i][0],cache_album[i][1],cache_album[i][2],cache_album[i][3]]));
  }
  for(let i=0;i<cache_song.length;i++){
    console.log(await pool.query(`insert into song values($1, $2, $3) ON CONFLICT (track_id) DO UPDATE SET popularity = EXCLUDED.popularity;`,
      [cache_song[i][0],cache_song[i][1],cache_song[i][2]]));
  }
  for(let i=0;i<cache_artist.length;i++){
    console.log(await pool.query(`insert into artist(artist_id, artist_name, artist_followers) values($1, $2, $3) ON CONFLICT (artist_id) DO UPDATE SET artist_name = EXCLUDED.artist_name, artist_followers = EXCLUDED.artist_followers;`,
      [cache_artist[i][0],cache_artist[i][1], cache_artist[i][2]]));
  }
  for(let i=0;i<cache_artist_of_album.length;i++){
    console.log(await pool.query(`insert into artist_of_album values($1, $2) ON CONFLICT (artist_id, album_id) DO NOTHING;`,
      [cache_artist_of_album[i][0],cache_artist_of_album[i][1]]));
  }
  for(let i=0;i<cache_song_to_album.length;i++){
    console.log(await pool.query(`insert into song_to_album values($1, $2, $3) ON CONFLICT (track_id) DO NOTHING;`,
      [cache_song_to_album[i][0],cache_song_to_album[i][1],cache_song_to_album[i][2]]));
  }
  for(let i=0;i<cache_genre.length;i++){
    console.log(await pool.query(`insert into genre values($1, $2) ON CONFLICT (artist_id, artist_genre) DO NOTHING;`,
      [cache_genre[i][0],cache_genre[i][1]]));
  }
  for(let i=0;i<cache_song_to_artist.length;i++){
    console.log(await pool.query(`insert into song_to_artist values($1, $2) ON CONFLICT (track_id, artist_id) DO NOTHING;`,
      [cache_song_to_artist[i][0],cache_song_to_artist[i][1]]));
  }
  for(let i=0;i<cache_constraints.length;i++){
    /*console.log(await pool.query(`insert into constraints values($1, $2, $3) ON CONFLICT (track_id) DO NOTHING;`,
      [cache_constraints[i][0],cache_constraints[i][1],cache_constraints[i][2]]));*/
  }
  for(let i=0;i<cache_parameter.length;i++){
    console.log(await pool.query(`insert into parameter values($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12, $13) ON CONFLICT (track_id) DO NOTHING;`,
      [cache_parameter[i][0],cache_parameter[i][1],cache_parameter[i][2],cache_parameter[i][3],cache_parameter[i][4],
        cache_parameter[i][5],cache_parameter[i][6],cache_parameter[i][7],cache_parameter[i][8],cache_parameter[i][9],
        cache_parameter[i][10],cache_parameter[i][11],cache_parameter[i][12]]));
  }
}

```

- 利用AWS API Gateway並實作REST API 提供網頁端trigger lambda function, 使用 post method在lambda function中調用spotify api更新cache

```
switch(operation){
  case 'UpdateByTrack':
    await GetTracksData(token, id_from_post);
    await WriteCache();
    break;
  case 'UpdateByAlbum':
    var track_list_ByAlbum = await GetTracksFromAlbums(token, id_from_post);
    await GetTracksData(token, track_list_ByAlbum);
    await WriteCache();
    break;
  case 'UpdateByArtist':
    var album_list_ByArtist = await GetAlbumsFromArtist(token, id_from_post);
    var track_list_ByArtist = await GetTracksFromAlbums(token, album_list_ByArtist);
    await GetTracksData(token, track_list_ByArtist);
    await WriteCache();
    break;
  case 'AddUser':
    console.log(await pool.query('insert into user_information values($1, $2, $3)',[event.user_name, event.user_password, event.user_nickname]));
    break;
  case 'UpdateSongList':
    let ret = await pool.query('select count(*) from song where track_id=$1',[event.track_id])
    if(ret.rows[0].count!=0){
      console.log(await pool.query('insert into song_list values($1, $2, $3)',[event.user_name, event.track_id, current_date]));
    }
    else{
      await GetTracksData(token, event.track_id);
      await WriteCache();
      console.log(await pool.query('insert into song_list values($1, $2, $3)',[event.user_name, event.track_id, current_date]));
    }
    break;
  case 'UpdateGuessRanking':
    await pool.query('insert into guess_ranking_list values($1, $2, $3) ON CONFLICT (user_name) DO UPDATE SET right_number = guess_ranking_list.r');
    break;
  case 'GetToken':
    console.log(token);
    return {
      ...metaData,
      statusCode: 200,
      body: token
    };
    break;
}
```



● Application with Database - 資料庫的必要性

Application with database - explain why your application needs a database.

由於我們需要記錄使用者資訊, 同時記錄歌單的資料, 查詢歌曲, 與隨機選擇指定主題的歌曲, 因此我們需要資料庫來儲存更新, 記錄各種不同的資料。且使用資料庫可以更快的

進行 query 功能, 也能使在不同環境下部屬的後端拿到透過連到相同的資料庫拿到相同的資料。

● Application with Database

Application with database - includes the queries that are performed by your application, how your application performed these queries (connections between application and database), and what the cooperating functions for your application.

前端(react js):

我們前端使用react js、並部屬在github page上, 藉由axios、router、antd來製作網站功能

後端(node js):

我們後端使用node js來部屬server進行aws rds postgres與react js的api串接

大致上會先在pool的地方設定資料庫屬性

之後每一個query function都分配一個api, 讓react前端做取得資料或更新db的動作

程式碼如下:

```
const express = require('express');
const { Pool } = require('pg');
const cors = require('cors');

const app = express();
const port = 8080;

app.use(express.json());
app.use(cors());

const pool = new Pool({
  user: 'postgres',
  host: 'database-2.c8dhylstxkf5.us-east-1.rds.amazonaws.com',
  database: 'postgres',
  password: '1qaz2wsx',
  port: 5432,
  ssl: true,
  connectionString:
'postgresql://postgres:1qaz2wsx@database-2.c8dhylstxkf5.us-east-1.rds.a
mazonaws.com:5432/postgres',
  ssl: { rejectUnauthorized: false }
});

console.log('Database pool connected successfully!');

//search
```

```

app.get('/api/data/query_search_song', async (req, res) => {
  const {song, artist, album} = req.query;
  console.log(song);
  console.log(artist);
  console.log(album);
  let query = '';
  let queryParams = [];
  query =
    `SELECT Song.track_id, Song.track_name, Artist.artist_name,
Album.album_name, Song_to_Album.track_number, Song.popularity,
Constraints.duration_ms, Constraints.year
FROM public.Song
JOIN public.Song_to_Artist ON Song.track_id =
Song_to_Artist.track_id
JOIN public.Artist ON Song_to_Artist.artist_id = Artist.artist_id
JOIN public.Song_to_Album ON Song.track_id = Song_to_Album.track_id
JOIN public.Album ON Song_to_Album.album_id = Album.album_id LEFT
JOIN public.Constraints ON Song.track_id = Constraints.track_id
LEFT
JOIN public.Parameter ON Song.track_id = Parameter.track_id
WHERE Song.track_name ILIKE COALESCE(NULLIF($1, ''), '%%') AND
Artist.artist_name ILIKE COALESCE(NULLIF($2, ''), '%%') AND
Album.album_name ILIKE COALESCE(NULLIF($3, ''), '%%')`;
  queryParams = [song, artist, album];
  try {
    const result = await pool.query(query, queryParams);
    console.log('Query result:', result.rows);
    res.json(result.rows);
  } catch (error) {
    console.error('Error executing query', error);
    res.status(500).json({ error: 'Internal Server Error', details:
error.message});
  }
});

//guess
app.get('/api/data/query_guess_song', async (req, res) => {
  const {song, artist, album} = req.query;
  console.log(song);
  console.log(artist);
  console.log(album);

  let query = '';

```

```

let queryParams = [];
query = `WITH FilteredTracks AS
(
  SELECT s.track_id, s.track_name, a.artist_name, al.album_name,
s.popularity,
  ROW_NUMBER() OVER (ORDER BY RANDOM()) AS rn
  FROM public.Song s
  JOIN public.Song_to_Artist sta ON s.track_id = sta.track_id
  JOIN public.Artist a ON sta.artist_id = a.artist_id
  JOIN public.Song_to_Album sta2a ON s.track_id = sta2a.track_id
  JOIN public.Album al ON sta2a.album_id = al.album_id
  WHERE s.popularity >= 20
  AND s.track_name ILIKE COALESCE(NULLIF($1, ''), '%') AND
a.artist_name ILIKE COALESCE(NULLIF($2, ''), '%') AND al.album_name
ILIKE COALESCE(NULLIF($3, ''), '%')

  SELECT track_id, track_name, artist_name, album_name, popularity
  FROM FilteredTracks WHERE rn <= 4;`;
queryParams = [song, artist, album];
try {
  const result = await pool.query(query, queryParams);
  console.log('Query result:', result.rows);
  res.json(result.rows);
} catch (error) {
  console.error('Error executing query', error);
  res.status(500).json({ error: 'Internal Server Error', details:
error.message});
}
});

//song list
app.get('/api/data/query_gain_sonplist', async (req, res) => {
  const {current_name} = req.query;

  let query = '';
  let queryParams = [];
  query = 'SELECT track_id, add_time FROM Song_List WHERE user_name =
$1;';
  queryParams = [current_name];
  try {
    const result = await pool.query(query, queryParams);
    console.log('Query result:', result.rows);
    res.json(result.rows);
  }
});

```

```

    } catch (error) {
      console.error('Error executing query', error);
      res.status(500).json({ error: 'Internal Server Error', details:
error.message});
    }
  });

//user table
app.get('/api/data/query_gain_passwordnickname', async (req, res) => {
  const {current_name} = req.query;

  let query = '';
  let queryParams = [];
  query = 'SELECT password_, nickname FROM User_Information WHERE
user_name = $1;';
  queryParams = [current_name];
  try {
    const result = await pool.query(query, queryParams);
    console.log('Query result:', result.rows);
    res.json(result.rows);
  } catch (error) {
    console.error('Error executing query', error);
    res.status(500).json({ error: 'Internal Server Error', details:
error.message});
  }
});

//guess ranking list
app.get('/api/data/query_guess_ranking', async (req, res) => {
  const {} = req.query;

  let query = '';
  let queryParams = [];
  query = 'SELECT user_name, nickname, right_number FROM
Guess_Ranking_List;';
  queryParams = [];
  try {
    const result = await pool.query(query, queryParams);
    console.log('Query result:', result.rows);
    res.json(result.rows);
  } catch (error) {
    console.error('Error executing query', error);
  }
});

```

```
res.status(500).json({ error: 'Internal Server Error', details:
error.message});
    }
});

app.listen(port, () => {
  console.log(`Server is running on port ${port}`);
});
```

● Future Works

1. 以動畫為主題的猜歌體驗：

我們原本計畫利用 json API (<https://api-docs.animethemes.moe/jsonapi/>) 抓取每部動畫的所有歌曲，打造一個以動畫為主題的猜歌遊戲。但由於時間問題，目前還沒有實現。

2. 推薦歌曲

利用 table Parameter 裡面的資料，以及分析使用者過去聆聽的歌曲，向使用者推薦可能符合他們喜好的歌曲，就像 Spotify 內建的自動推播功能一樣。這將為每位使用者帶來更加個人化且豐富的音樂體驗。