CIS5500 Spring 2025

Routes/API Specification

(50 Points)

Here, all the backend routes, their API specifications, and related tasks are listed. These are tasks 1-12 and consist of half the points in the assignment.

IMPORTANT NOTE: Parameters, unless specified, are required. Optional parameters are marked with an asterisk (*). Default values are indicated when necessary. You will find these links helpful in understanding route and query parameters.

NAMING NOTE: in the provided template code and as reflected in this writeup, route names, route parameters, and query parameters are snake cased (e.g. album_songs or song_id) but Javascript variables will be camel cased. You may therefore see code like `const pageSize = req.query.page_size ? req.query.page_size : 10`; using both camel casing (for the variable name) and snake casing (for the query parameter). Make sure to abide by the same convention to prevent variable naming issues.

Route 1

Route: `/author/:type`
Description: returns the author of the app as a JSON object.
Route Parameter(s): `type` (string)
Query Parameter(s): *None*
Route Handler: `author(req, res)`
Return Type: JSON Object
Return Parameters: { `data` (string) }
Expected (Output) Behavior:

- Case 1: If the route parameter (`type`)='name'
  - Return a JSON object with a single field `data` containing the author's name.
- Case 2: If the route parameter(`type`)= 'pennkey'
  - Return a JSON object with a single field `data` containing the author's pennkey.
- Case 3: If the route parameter is defined but does not match cases 1 or 2:
  - Return with a 400 status an empty JSON object.

Tasks:

- Task 1 (1 pts): replace the values of name and pennkey variables with your own
- Task 2 (2 pts): edit the else if condition to check if the request parameter is 'pennkey' and if so, send back a JSON response with the pennkey

---

Route 2

Route: `/random`
Description: Returns a random song
Route Parameter(s): *None*
Query Parameter(s): `explicit` (string)*
Route Handler: `random(req, res)`
Return Type: JSON Object
Return Parameters: { `song_id` (string), `title` (string) }
Expected (Output) Behavior:

- Case 1: If the query parameter **`explicit`** is defined and equals “true”
  - Return a random song that may or may not be explicit
- Case 2: If the query parameter **`explicit`** is not defined or does not equal “true”
  - Return a random song that definitely is not explicit
- Hint: Consider using the order by random() function in PostgreSQL.

Tasks:

- Task 3 (2 pts): also return the song title in the response

Route 3

Route: `/song/:song_id`
Description: Returns all information about a song
Route Parameter(s): **`song_id`** (string)
Query Parameter(s): *None*
Route Handler: **`song(req, res)`**
Return Type: JSON Object
Return Parameters: { **`song_id`** (string), **`album_id`** (string), **`title`** (string), **`number`** (int), **`duration`** (int), **`plays`** (int), **`danceability`** (float), **`energy`** (float), **`valence`** (float), **`tempo`** (int), **`key_mode`** (string), **`explicit`** (int) }
Expected (Output) Behavior:

- If a valid **`song_id`** is provided, return the relevant information from the database as specified in return parameters, otherwise the behavior can be undefined (although it is best to just return an empty object)

Tasks:

- Task 4 (4 pts): implement a route that given a song_id, returns all information about the song

---

Route 4

Route: `/album/:album_id`
Description: Returns all information about an album
Route Parameter(s): **`album_id`** (string)
Query Parameter(s): *None*
Route Handler: **`album(req, res)`**
Return Type: JSON Object
Return Parameters: { **`album_id`** (string), **`title`** (string), **`release_date`** (string), **`thumbnail_url`** (string) }
Expected (Output) Behavior:

- If a valid **`album_id`** is provided, return the relevant information from the database as specified in return parameters, otherwise the behavior can be undefined (although it is best to just return an empty object)

Tasks:

- Task 5 (4 pts): implement a route that given a album_id, returns all information about the album

---

Route 5

Route: `/albums`
Description: Returns all albums ordered by release date
Route Parameter(s): *None*
Query Parameter(s): *None*
Route Handler: `albums(req, res)`
Return Type: JSON Array
Return Parameters: [{ `album_id` (string), `title` (string), `release_date` (string), `thumbnail_url` (string) }]
Expected (Output) Behavior:

- Return an array of objects sorted by release date descending

Tasks:

- Task 6 (4 pts): implement a route that returns all albums ordered by release date (descending)

Route 6

Route: `/album_songs/:album_id`
Description: Returns all songs on a given album (with some but not all information about each song) ordered by track number ascending
Route Parameter(s): `album_id` (string)
Query Parameter(s): *None*
Route Handler: `album_songs(req, res)`
Return Type: JSON Array
Return Parameters: [{ `song_id` (string), `title` (string), `number` (int), `duration` (int), `plays` (int) }]
Expected (Output) Behavior:

- Return an array of objects sorted by track number ascending

Tasks:

- Task 7 (4 pts): implement a route that given an album_id, returns all songs on that album ordered by track number (ascending)

Route 7

Route: `/top_songs`
Description: Returns songs with some relevant information in order of number of plays, optionally paginated
Route Parameter(s): *None*
Query Parameter(s): `page` (int)*, `page_size` (int)* (default: `10`)
Route Handler: `top_songs(req, res)`
Return Type: JSON Array
Return Parameters: [{ `song_id` (string), `title` (string), `album_id` (string), `album` (string), `plays` (int) }]
Expected (Output) Behavior:

- Case 1: If the page parameter (`page`) is defined

- Return songs with all the above return parameters for that page number by considering the **`page`** and **`page_size`** parameters. For example, page 1 and page 7 for a page size 10 should have entries 1 through 10 and 61 through 70 respectively.

- Case 2: If the page parameter (**`page`**) is not defined
  - Return all songs with all the above return parameters

Tasks:

- Task 8 (2 pts): use the ternary (or nullish) operator to set the pageSize based on the query or default to 10
- Task 9 (4 pts): query the database and return all songs ordered by number of plays (descending)
  - Make sure you only return the attributes specified in the return parameters and use AS to make sure the schema matches.
- Task 10 (4 pts): reimplement TASK 9 with pagination
  - LIMIT/OFFSET: https://www.geeksforgeeks.org/postgresql-limit-with-offset-clause/

Route 8

Route: `/top_albums`
Description: Returns albums with some relevant information in order of aggregate number of plays (across all songs in the album), optionally paginated
Route Parameter(s): *None*
Query Parameter(s): **`page`** (int)*, **`page_size`** (int)* (default: `10`)
Route Handler: **`top_albums(req, res)`**
Return Type: JSON Array
Return Parameters: [{ **`album_id`** (string), **`title`** (string), **`plays`** (int) }]
Expected (Output) Behavior:

- Case 1: If the page parameter (**`page`**) is defined
  - Return albums with all the above return parameters for that page number by considering the **`page`** and **`page_size`** parameters. For example, page 1 and page 3 for a page size 3 should have entries 1 through 3 and 7 through 9 respectively.
- Case 2: If the page parameter (**`page`**) is not defined
  - Return all albums with all the above return parameters

Tasks:

- Task 11 (8 pts): return the top albums ordered by aggregate number of plays of all songs on the album (descending), with optional pagination (as in route 7)
  - You will need to use SQL aggregation with a GROUP BY

Route 9

Route: `/search_songs`
Description: Returns an array of song with all their properties matching the search query ordered by title (ascending)
Route Parameter(s): *None*
Query Parameter(s): **`title`** (string)*, **`duration_low`** (int)* (default: `60`), **`duration_high`** (int)* (default: `660`), **`plays_low`** (int)* (default: `0`), **`plays_high`** (int)* (default: `1100000000`), **`danceability_low`** (int)* (default: `0`), **`danceability_high`** (int)* (default: `1`), **`energy_low`**

(int)* (default: `0`), **`energy_high`** (int)* (default: `1`), **`valence_low`** (int)* (default: `0`), **`valence_high`** (int)* (default: `1`), explicit* (string)

Route Handler: **`search_songs(req, res)`**

Return Type: JSON Array

Return Parameters: [{ **`song_id`** (string), **`album_id`** (string), **`title`** (string), **`number`** (int), **`duration`** (int), **`plays`** (int), **`danceability`** (float), **`energy`** (float), **`valence`** (float), **`tempo`** (int), **`key_mode`** (string), **`explicit`** (int) }]

Expected (Output) Behavior:

- Return an array with all songs that match the constraints ordered by song title (ascending). If no song satisfies the constraints, return an empty array without causing an error
- If **`title`** is specified, match all songs with titles that contain the **`title`** query parameter as a substring. If **`title`** is undefined, no filter should be applied (return all songs matching the other conditions).
    - Hint: use LIKE to perform substring matching. In Postgres, LIKE is case sensitive–this is fine and the intended behavior in the autograder.
    - Hint: although no default value of **`title`** is given, consider what default value would cause an undefined **`title`** to not apply any filter to the search.
- If **`explicit`** is not defined or does not equal “true” then filter out any songs that are labeled as explicit. If **`explicit`** does equal “true” then all songs are included.
    - Hint: refer back to our implementation of route 2
- x_high and x_low are the upper and lower bound filters for an attribute x. Entries that match the ends of the bounds should be included in the match. For example, if **`energy_low`** were 0.2 and **`energy_high`** were 0.8, then all players whose **`energy`** attribute was >=0.2 and <=0.8 would be included.

Tasks:

- Task 12 (10 pts): return all songs that match the given search query with parameters defaulted to those specified in API spec ordered by title (ascending)