Question - 1
Springboot: Music Streaming

This project is a practical assessment designed to evaluate understanding and skills in working with RESTful APIs using Spring Boot.

Below are examples of the JSON data for this project.:

Example of Artist data JSON Object

```
{
  "id": 1,
  "artistName": "Alex Doe",
  "bio": "Alex Doe is a popular artist...",
  "genre": "Pop",
  "origin": "USA",
  "formedYear": "2000",
  "socialLink": "https://www.alexdoe.com",
  "image": "https://www.alexdoe.com/image.jpg",
  "tracksProduced": [
    {
      "id": 1,
      "title": "Track 1",
      "albumName": "Album 1",
      "releaseDate": "2022-01-01",
      "duration": "3:30",
      "genre": "Pop",
      "description": "This is a description of Track 1",
      "playCount": 1000,
      "fileUrl": "https://www.alexdoe.com/track1.mp3",
      "coverImage": "https://www.alexdoe.com/track1.jpg"
    }
  ]
}
```

Example of Playlist data JSON Object

```
{
  "id": 1,
  "name": "My Playlist",
  "description": "This is my favorite playlist",
  "tracks": [
    {
      "id": 1,
      "name": "Track 1",
      "duration": "3:45",
      "artist": {
        "id": 1,
        "name": "Artist 1"
      }
    },
    {
      "id": 2,
      "name": "Track 2",
      "duration": "4:30",
      "artist": {
        "id": 2,
        "name": "Artist 2"
```

```
      }
    }
  ]
}
```

Example of Track data JSON Object

```
{
  "id": 1,
  "title": "Track 1",
  "albumName": "Album 1",
  "releaseDate": "2022-01-01",
  "duration": "3:30",
  "genre": "Pop",
  "description": "This is a description of Track 1",
  "playCount": 1000,
  "fileUrl": "https://www.alexdoe.com/track1.mp3",
  "coverImage": "https://www.alexdoe.com/track1.jpg",
  "artist": {
    "id": 1,
    "artistName": "Alex Doe",
    "bio": "Alex Doe is a popular artist...",
    "genre": "Pop",
    "origin": "USA",
    "formedYear": "2000",
    "socialLink": "https://www.alexdoe.com",
    "image": "https://www.alexdoe.com/image.jpg"
  }
}
```

Implement the following APIs:

ArtistController

- *POST /music/platform/v1/artists* - Create a new artist.
- *GET /music/platform/v1/artists* - Get all artists.
- *GET /music/platform/v1/artists/{artistId}* - Get an artist by ID.
- *PUT /music/platform/v1/artists/{artistId}* - Update an artist by ID.
- *DELETE /music/platform/v1/artists/{artistId}* - Delete an artist by ID.

PlayListController

- *GET /music/platform/v1/playlists/{playlistId}* - Get a playlist by ID.
- *POST /music/platform/v1/playlists* - Create a new playlist.
- *DELETE /music/platform/v1/playlists/{playlistId}* - Delete a playlist by ID.

TrackController

- *GET /music/platform/v1/tracks* - Get all tracks.
- *POST /music/platform/v1/tracks* - Create a new track.
- *GET /music/platform/v1/tracks/{trackId}* - Get a track by ID.
- *PUT /music/platform/v1/tracks/{trackId}* - Update a track by ID.
- *DELETE /music/platform/v1/tracks/{trackId}* - Delete a track by ID.

Complete the project so that it passes all the test cases when running the provided unit tests. The project, by default, supports the use of the H2 database.

▼ Example Requests and Responses

POST request to /music/platform/v1/artists

Request data JSON Object:

```
{
  "artistName": "Alex Doe",
```

```
  "bio": "Alex Doe is a popular artist...",
  "genre": "Pop",
  "origin": "USA",
  "formedYear": "2000",
  "socialLink": "https://www.alexdoe.com",
  "image": "https://www.alexdoe.com/image.jpg"
}
```

Response JSON Object:

```
{
  "id": 1,
  "artistName": "Alex Doe",
  "bio": "Alex Doe is a popular artist...",
  "genre": "Pop",
  "origin": "USA",
  "formedYear": "2000",
  "socialLink": "https://www.alexdoe.com",
  "image": "https://www.alexdoe.com/image.jpg"
}
```

GET request to /music/platform/v1/artists

Response JSON Object:

```
[
  {
    "id": 1,
    "artistName": "Alex Doe",
    "bio": "Alex Doe is a popular artist...",
    "genre": "Pop",
    "origin": "USA",
    "formedYear": "2000",
    "socialLink": "https://www.alexdoe.com",
    "image": "https://www.alexdoe.com/image.jpg"
  }
]
```

GET request to /music/platform/v1/artists/{artistId}

Response JSON Object:

```
{
  "id": 1,
  "artistName": "Alex Doe",
  "bio": "Alex Doe is a popular artist...",
  "genre": "Pop",
  "origin": "USA",
  "formedYear": "2000",
  "socialLink": "https://www.alexdoe.com",
  "image": "https://www.alexdoe.com/image.jpg"
}
```

PUT request to /music/platform/v1/artists/{artistId}

Request data JSON Object:

```
{
  "artistName": "Alex Doe",
  "bio": "Alex Doe is a popular artist...",
  "genre": "Pop",
  "origin": "USA",
  "formedYear": "2000",
  "socialLink": "https://www.alexdoe.com",
  "image": "https://www.alexdoe.com/image.jpg"
}
```

Response JSON Object:

```
{
  "id": 1,
  "artistName": "Alex Doe",
  "bio": "Alex Doe is a popular artist...",
  "genre": "Pop",
  "origin": "USA",
  "formedYear": "2000",
  "socialLink": "https://www.alexdoe.com",
  "image": "https://www.alexdoe.com/image.jpg"
}
```

DELETE request to /music/platform/v1/artists/{artistId}

No response data. Returns a 204 No Content status.

POST request to /music/platform/v1/playlists:

Request data JSON Object:

```
{
  "name": "My Playlist",
  "description": "This is my favorite playlist"
}
```

Response JSON Object:

```
{
  "id": 1,
  "name": "My Playlist",
  "description": "This is my favorite playlist"
}
```

GET request to /music/platform/v1/playlists/{playlistId}

Response JSON Object:

```
{
  "id": 1,
  "name": "My Playlist",
  "description": "This is my favorite playlist"
}
```

DELETE request to /music/platform/v1/playlists/{playlistId}

No response data. Returns a 204 No Content status.

POST request to /music/platform/v1/tracks

Request data JSON Object:

```
{
  "title": "Track 1",
  "albumName": "Album 1",
  "releaseDate": "2022-01-01",
  "duration": "3:30",
  "genre": "Pop",
  "description": "This is a description of Track 1",
  "playCount": 1000,
  "fileUrl": "https://www.alexdoe.com/track1.mp3",
  "coverImage": "https://www.alexdoe.com/track1.jpg"
}
```

Response JSON Object:

```
{
  "id": 1,
  "title": "Track 1",
  "albumName": "Album 1",
  "releaseDate": "2022-01-01",
  "duration": "3:30",
```

```
  "genre": "Pop",
  "description": "This is a description of Track 1",
  "playCount": 1000,
  "fileUrl": "https://www.alexdoe.com/track1.mp3",
  "coverImage": "https://www.alexdoe.com/track1.jpg"
}
```

GET request to /music/platform/v1/tracks

Response JSON Object:

```
[
  {
    "id": 1,
    "title": "Track 1",
    "albumName": "Album 1",
    "releaseDate": "2022-01-01",
    "duration": "3:30",
    "genre": "Pop",
    "description": "This is a description of Track 1",
    "playCount": 1000,
    "fileUrl": "https://www.alexdoe.com/track1.mp3",
    "coverImage": "https://www.alexdoe.com/track1.jpg"
  }
]
```

GET request to /music/platform/v1/tracks/{trackId}

Response JSON Object:

```
{
  "id": 1,
  "title": "Track 1",
  "albumName": "Album 1",
  "releaseDate": "2022-01-01",
  "duration": "3:30",
  "genre": "Pop",
  "description": "This is a description of Track 1",
  "playCount": 1000,
  "fileUrl": "https://www.alexdoe.com/track1.mp3",
  "coverImage": "https://www.alexdoe.com/track1.jpg"
}
```

PUT request to /music/platform/v1/tracks/{trackId}

Request data JSON Object:

```
{
  "title": "Track 1",
  "albumName": "Album 1",
  "releaseDate": "2022-01-01",
  "duration": "3:30",
  "genre": "Pop",
  "description": "This is a description of Track 1",
  "playCount": 1000,
  "fileUrl": "https://www.alexdoe.com/track1.mp3",
  "coverImage": "https://www.alexdoe.com/track1.jpg"
}
```

Response JSON Object:

```
{
  "id": 1,
  "title": "Track 1",
  "albumName": "Album 1",
  "releaseDate": "2022-01-01",
  "duration": "3:30",
  "genre": "Pop",
  "description": "This is a description of Track 1",
  "playCount": 1000,
```

```
  "fileUrl": "https://www.alexdoe.com/track1.mp3",
  "coverImage": "https://www.alexdoe.com/track1.jpg"
}
```

DELETE request to /music/platform/v1/tracks/{trackId}

No response data. Returns a 204 No Content status.

Question - 2

Springboot: Sports Health Coaching

This project is a practical assessment designed to evaluate understanding and skills in working with RESTful APIs using Spring Boot.

Here is an example of a customer JSON object:

```
{
  "id": 1,
  "height": 180,
  "weight": 80,
  "coach": {
    "id": 1,
    "name": "Alex Doe"
  }
}
```

Here is an example of a coach JSON object:

```
{
  "id": 1,
  "name": "Alex Doe"
}
```

The project consists of two empty controller classes, *CustomerController* and *CoachController*, that must be implemented. The controllers should have the following endpoints:

CustomerController

POST request to /api/customer:

- Create a new customer.
- Accepts a JSON body with height, weight, and coach_id fields.
- Return the created record with a 201 Created status.

GET request to /api/customer:

- Return a list of all customer records in the database sorted by id in ascending order.

GET request to /api/customer/{id}:

- Return a customer record with the given id.
- If the customer record exists, return the record with a 200 OK status.
- If the customer record does not exist, return a 404 Not Found status.

CoachController

POST request to /api/coach:

- Create a new coach.
- Accepts a JSON body with name field.
- Return the created record with a 201 Created status.

GET request to /api/coach:

- Return a list of all coach records in the database sorted by id in ascending order.
- Return the list with a 200 OK status.

GET request to /api/coach/{id}:

- Return a coach record with the given id.
- If the coach record exists, return the record with a 200 OK status.
- If the coach record does not exist, return a 404 Not Found status.

Please ensure that all endpoints handle errors appropriately and return the correct HTTP status codes so that all unit tests pass.

▼ **Example Requests and Responses**

POST request to /api/customer:

Request data JSON Object:

```
{
  "height": 180,
  "weight": 80,
  "coach_id": 1
}
```

Response JSON Object:

```
{
  "id": 1,
  "height": 180,
  "weight": 80,
  "coach": {
    "id": 1,
    "name": "Alex Doe"
  }
}
```

GET request to /api/customer:

Response JSON Object:

```
[
  {
    "id": 1,
    "height": 180,
    "weight": 80,
    "coach": {
      "id": 1,
      "name": "Alex Doe"
    }
  }
]
```

GET request to /api/customer/{id}:

Response JSON Object:

```
{
  "id": 1,
  "height": 180,
  "weight": 80,
  "coach": {
    "id": 1,
    "name": "Alex Doe"
  }
}
```

POST request to /api/coach:

Request data JSON Object:

```
{
  "name": "Alex Doe"
```

```
}
```

Response JSON Object:

```
{
  "id": 1,
  "name": "Alex Doe"
}
```

GET request to /api/coach:

Response JSON Object:

```
[
  {
    "id": 1,
    "name": "Alex Doe"
  },
  {
    "id": 2,
    "name": "Sam"
  }
]
```

GET request to /api/coach/{id}:

Response JSON Object:

```
{
  "id": 1,
  "name": "Alex Doe"
}
```

Question - 3

Springboot: Artist Management APIs

In this challenge, implement a simple REST API to manage a collection of Artist data.

Each artist is a JSON entry with the following keys:

- id: the unique ID of the event (Long).
- firstName: first name of the artist (String).
- lastName: last name of the artist (String).

Here is an example of an artist JSON object:

```
{
      "firstName": "Dasun",
      "lastName": "Anushka"
}
```

An implementation of the Artist model is provided. Implement a REST service that exposes the /v1/artists endpoints, which allows for managing the collection of artists' records in the following way:

POST request to /v1/artists:

- CCreates a new artist data record.
- The response code is 201, and the response body is the created record, including its unique id.

GET request to /v1/artists:

- The response code is 200.
- The response body is a list of matching records, ordered by their ids in increasing order.

GET request to /v1/artists/{artistId}:

- Returns a record with the given id and status code 200.
- If there is no record with the given id, the response code is null.

DELETE request to /v1/artists/{artistId}:

- Delete the record with the given id and return status code 204.

Complete the project so that it passes all the test cases when running the provided unit tests. The project, by default, supports the use of the H2 database.

▼ Example requests and responses

`POST` request to /v1/artists

Request body:

```
{
          "firstName": "Dasun",
          "lastName": "Anushka"
}
```

The response code is 201, and when converted to JSON, the response body is:

```
{
        "id" : 1,
        "firstName": "Dasun",
        "lastName": "Anushka"
}
```

This adds a new object to the collection with the given properties and id 1.

`GET` request to /v1/artists

The response code is 200, and when converted to JSON, the response body (assuming that the objects are in the collection) is as follows:

```
[
        {
          "id" : 1,
          "firstName": "Dasun",
          "lastName": "Anushka"
        },
        {
          "id": 2,
          "firstName": "Anushka",
          "lastName": "Lakmal"
        },
        {
          "id": 3,
          "firstName": "Era",
          "lastName": "Jaye"
        }
  ]
```

`GET` request to /v1/artists/1

The response code is 200, and when converted to JSON, the response body is as follows:

```
[
        {
          "id" : 1,
          "firstName": "Dasun",
          "lastName": "Anushka"
        }
  ]
```

If an object with id 1 does not exist, the response is null.

DELETE request to /v1/artists/1
Assuming that an object with id 1 exists, the response code is 204, and the response body is empty.

Question - 4
Springboot: Music Track APIs

In this project, you'll be working with a team to create a music streaming platform. Your main task is to develop a REST API that shares details about music tracks. You need to ensure this API can receive, show, and query data like track title, album, description, and play count. Your work will be judged on how accurately and efficiently the application handles and processes this data.

Music track data is represented as a JSON object, each describing various properties of a music track. The following properties are included:

- id: The unique integer ID of the object(Long).
- title: The title of the track(String).
- albumName: The corresponding album name of the track(String).
- releaseDate: The release date of the track, formatted as YYYY-MM-DD(Date).
- playCount: An integer value denoting how many times a user has played this track(Integer).

Example of a music track data JSON object:

```
{
  "title": "Lost in Echoes",
  "albumName": "Echoes of the Unknown",
  "releaseDate": "2021-07-15",
  "playCount": 5000
}
```

The REST service must expose the endpoint /music/platform/v1/tracks, enabling the management of music track records as follows:

POST request to /music/platform/v1/tracks:

- Creates a new music track record.
- It expects a valid music track data object as its body payload, excluding the id property.
- The service assigns a unique long id to the added object.
- The response includes the created record with its unique id, and the response code is 201.

GET request to /music/platform/v1/tracks:

- Responds with a list of all music track records and a response code of 200.

GET request to /music/platform/v1/tracks/search:

- Responds with music track records filtered by title, Consider track title is unique. (`Eg - /music/platform/v1/tracks/search?title=Henry`)
- The response code is 200. It accepts query string parameter title.
- Records are returned based on matching title.

DELETE request to /music/platform/v1/tracks/{trackId}:

- Deletes the record with the specified track id if it exists in the database with the status code of 204.

Your task is to ensure the project meets all test case criteria when running the provided rspec tests. The project uses H2 database by default. Start by implementing the POST request to music/platform/v1/tracks, as testing other methods requires the POST functionality to work correctly.

▼ Example requests and responses

`POST` request to /music/platform/v1/tracks

Request body:

```
{
   "title":"Lost in Echoes",
   "albumName":"Echoes of the Unknown",
   "releaseDate":"2021-07-15",
   "playCount":5000
}
```

The response code is 201, and when converted to JSON, the response body is:

```
{
   "id":1,
   "title":"Lost in Echoes",
   "albumName":"Echoes of the Unknown",
   "releaseDate":"2021-07-15",
   "playCount":5000
}
```

This adds a new object to the collection with the given properties and id 1.

`GET` request to /music/platform/v1/tracks

The response code is 200, and when converted to JSON, the response body (assuming that the below objects are all objects in the collection) is as follows:

```
[
   {
      "id":1,
      "title":"Lost in Echoes",
      "albumName":"Echoes of the Unknown",
      "releaseDate":"2021-07-15",
      "playCount":5000
   },
   {
      "id":2,
      "title":"Bravos",
      "albumName":"Unknown",
      "releaseDate":"2021-07-16",
      "playCount":100
   }
]
```

`GET` request to /music/platform/v1/tracks/search

The response code is 200, and when converted to JSON, the response body (assuming that the below objects are all objects with title is "Lost in Echoes" also unique) is as follows:

```
{
   "id":1,
   "title":"Lost in Echoes",
   "albumName":"Echoes of the Unknown",
   "releaseDate":"2021-07-15",
   "playCount":5000
}
```

DELETE request to music/platform/v1/tracks/1

Assuming that the object with id 1 exists, then the response code is 204 and the response body is empty.

Question - 5

Springboot: Playlist Management APIs

Implement a simple REST API to manage a collection of music playlist data.

Each PlayList is a JSON entry with the following keys:

- id: the unique ID of the event (Long).
- name: name of the play list (String).
- tracksCount: initial number of tracks in the play list (Integer).

Here is an example of a PlayList JSON object:

```
{
      "name": "Henry1s list",
      "tracksCount": 10
}
```

An implementation of the PlayList model is provided. Implement a REST service that exposes the v1/playlists endpoints, which allows for managing the collection of PlayList records in the following way:

POST request to /v1/playlists:

- Create a new playlist data record.
- The response code is 201, and the response body is the created record, including its unique *id*.

GET request to /v1/playlists:

- The response code is 200.
- The response body is a list of matching records, ordered by their ids in increasing order.

GET request to /v1/playlists/{playListId}:

- Return a record with the given *id* and status code 200.
- If there is no record with the given *id*, the response code is null.

DELETE request to /v1/playlists/{playListId}:

- Delete the record with the given *id* and return status code 204.

Complete the project so that it passes all the test cases when running the provided unit tests. The project, by default, supports the use of the H2 database.

▼ Example requests and responses

`POST` request to /v1/playlists

Request body:

```
{
      "name": "Henry1s list",
      "tracksCount": 10
}
```

The response code is 201, and when converted to JSON, the response body is:

```
{
      "id": 1,
      "name": "Henry1s list",
      "tracksCount": 10
}
```

This adds a new object to the collection with the given properties and id 1.

`GET` request to /v1/playlists

The response code is 200, and when converted to JSON, the response body (assuming that the objects are all in the collection) is as follows:

```
[
   {
      "id":1,
      "name":"Henry1s list",
      "tracksCount":10
   },
   {
      "id":2,
      "name":"Bera",
      "tracksCount":10
   },
   {
      "id":3,
      "name":"Namal",
      "tracksCount":10
   }
]
```


`GET` request to /v1/playlists/1

The response code is 200, and when converted to JSON, the response body is as follows:


```
{
   "id":1,
   "name":"Henry1s list",
   "tracksCount":10
}
```


DELETE request to /v1/playlists/1

Assuming that the object with id 1 exists, the response code is 204, and the response body is empty.