CPSC 304 Project Cover Page

Milestone #4

Date: 08-06-2023

Group Number: 32

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Jiawei Liu	55362669	b7j6x	1943743535@qq.com
Flora Deng	14085211	d1i2t	floraa817@gmail.com
Tammie Liang	52445806	c1g1c	tammieliang@hotmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

Project Description

Music is gradually becoming an integral part of people's lives. People with headphones can be seen everywhere. Various styles of music are played in shopping malls or stores, and every day there are different concerts or lives happening around the world. At the same time, artists and the record labels behind them are becoming more and more known and a topic of conversation. Some of the loved artists or songs are known all over the world.

Data plays a huge role in the dissemination of information about records, artists and the labels associated with them. Most music apps can see the current most searched artists, songs or albums, but their main functions are still focused on the listening function rather than songs or artists' information browsing. The music information contained is not complete, even some niche music information is completely blank.

Through this project, our team hopes to create a more concise and clear platform for searching music information through the establishment of a database.

Our Melodex is a comprehensive web platform tailored for music lovers. Built using cutting-edge technologies such as React (frontend design), SpringBoot (backend design) and MySQL (database). It stores music artists and discographies, as well as being able to find them easily. It is useful for, say, a music wikipedia staff that needs to store and be able to find information about all the music artists and discographies in the world. The platform aims to make it easy for users to access, manage, and find music-related information.

Our database includes navigation bars that can navigate users to artists, music, records, labels, and the home page. The platform's administrative features allow the user to add information into appropriate pages, or make editing changes, as well as delete invalid information.

In addition to this, our platform provides users with a number of additional functions for interacting with the data:

Selection operation —— find a discography that meets a certain condition.

Join operation —— find albums with the same name as the discography.

Projection operation —— find the name and release date of all albums.

Nested Aggregation —— find the artist with the lowest average number of songs per album among the average number of songs per album.

Division operation —— find listeners who have attended all live events.

The above queries take into account most of the lookup requirements and allow the user to observe the overall data comparison.

Final Schema

Our database has been focused on the acquisition of information content about songs, records, artists and music companies from the beginning of the project, and the platform's primary focus has stayed the same. However, during the implementation of the project, we removed some of the entities and attributes.

First of all, at the beginning of the design, we designed different account interfaces for artists and listeners. However, considering that the restriction that users need an account to use the platform would affect user experience and would not be conducive to quick and easy instant searches, we decided to remove this feature. Therefore, we removed 'email' and 'userPassword' from 'Artist_ContractedWith.'

In addition, our platform focuses on managing, collecting, and querying music information. The list of songs that users listen to changes many times a day, and the 'ListenTo" entity requires users to have a personal account and does not contribute to the collection of music information, as well as placing an unnecessary burden on the platform regarding information storage. Therefore, we removed the 'ListensTo' schema.

We also removed 'numOfLikes' and 'ranking' attributes from 'Song' and 'Album' entities because most users need an official source of the number of favorites when adding information, which they may not have, and the number changes too quickly to rely solely on user updates. The lack of favorites will lack a unified reference for ranking.

(mainly implemented means that’s where you’ll find the actual query, but the logistics always begin from tracing the Controller funtion that uses it)

INSERTION QUERY

MAINLY implemented in Artist_ContractedWithRepository, insertNewArtist function (backend)

Frontend call:

```
INSERT INTO Artist_ContractedWith (artistName, age, country, biography,
numOfMembers, labelID)
VALUES (:artistName, :age, :country, :biography, :numOfMembers, :labelID);
```

DELETE QUERY

MAINLY implemented in Artist_ContractedWithRepository, deleteArtist function (backend)

Frontend call:

```
DELETE FROM Artist_ContractedWith
WHERE artistID = :artistID
```

UPDATE QUERY

MAINLY implemented in Artist_ContractedWithRepository, updateArtist function(backend)

Frontend call:

```
UPDATE Artist
SET artistName = :artistName, age=:age, biography=:biography,
numOfMembers=:numOfMembers-- user should specify which attribute to update
WHERE artistID = :artistID;
```

SELECTION QUERY

MAINLY implemented in GeneralController, getDynamicSelection function (backend)

Front end call:

The user is able to select from either Artist_ContractedWith, or Discography_Main joined with Album table (because it is meaningless to get Discography_Main on its own–Discography_Main is either an Album or Song according to ER diagram concept).

They are then allowed to select **any number of attributes** regardless of table choice (so as long as the attributes exist for the respective table, of course)

- If the user chooses Artist_ContractedWith…
  - they can pick Field1 to be either “artistName” or “country”. Val1 can be anything
  - They can pick Field2 to be either “numOfMembers” or “age”. Val2 can be anything

  For example,

  ```
  SELECT artistID, age
  FROM Artist_ContractedWith
  WHERE country='South Korea'  where 'South Korea' is an example value for
  :val1 AND numOfMembers>=1  where 1 is an example value for :val2
  ```

  For example,

```
SELECT artistName, age, biography
FROM Artist_ContractedWith
WHERE artistName='(G)I-DLE' AND age>=1
```

For example,

```
SELECT *
FROM Artist_ContractedWith
WHERE country='Canada' AND age>=20
```

- If the user chooses Discography_Main joined with Album…
  - They can pick Field1 to be either “genre” or “discoName”. Val1 can be anything
  - They can pick Field2 to be either “dID” or “numOfSongs”. Val2 can be anything

For example,

```
SELECT discoName, dID
FROM Discography_Main, Album
WHERE genre='Kpop' where 'Kpop' is an example value for {:val1} AND
dID>=1  where 1 is an example value for {:val2}
```

For example,

```
SELECT discoName, dID, genre
FROM Discography_Main, Album
WHERE discoName='GITTU RAP' AND numOfSongs>=1
```

For example,

```
SELECT dID, genre, numOfSongs
FROM Discography_Main, Album
WHERE genre='Kpop' AND numOfSongs>=2
```

PROJECTION QUERY

MAINLY implemented in Artist_ContractedWithRepository, getDynamicSelection function (backend)

Frontend call:

```
SELECT ATTRIBUTES_FROM_USER_INPUT
FROM Artist_ContractedWith;
```

JOIN QUERY

MAINLY implemented in AlbumRepositroy, getAllAlbumsWithName function (backend)

Frontend call:

```
SELECT *
FROM Album A, Discography_Main D
WHERE A.albumID = D.dID AND D.discoName = :discoName
```

AGGREGATION WITH GROUP BY QUERY

MAINLY implemented in Artist_ContractedWithRepository, aggregationGroupBy function (backend)
Frontend call:

```
SELECT R.artistID, COUNT(dID) AS numOfDiscography
FROM Releases R
GROUP BY R.artistID;
```

AGGREGATION WITH HAVING QUERY
MAINLY implemented in Artist_ContractedWithRepository, aggregationHaving function (backend)
Frontend call:

```
SELECT A.artistName, MIN(D.releaseDate) as earliestReleaseDate
FROM Artist_ContractedWith A, Discography_Main D, Releases R
GROUP BY A.artistName
HAVING COUNT(D.dID) > 1;
```

NESTED AGGREGATION WITH GROUP BY QUERY
MAINLY implemented in Artist_ContractedWithRepository, nestedAggregation function (backend)
Frontend call:

```
SELECT artistID
FROM Artist_ContractedWith A
WHERE (
    SELECT AVG(songCount)
    FROM (
        SELECT albumID,
               (SELECT COUNT(*) FROM Song S WHERE S.songID = Album.albumID) AS
songCount
        FROM Album
        WHERE A.artistID = Album.albumID
    ) AS inner_query
) =
(
    SELECT MIN(avgSongs)
    FROM (
        SELECT artistID,
               AVG(
                    (SELECT COUNT(*) FROM Song S WHERE S.songID = Album.albumID)
               ) AS avgSongs
        FROM Album
        GROUP BY artistID
    ) AS overall_avg_query
);
```

DIVISION QUERY

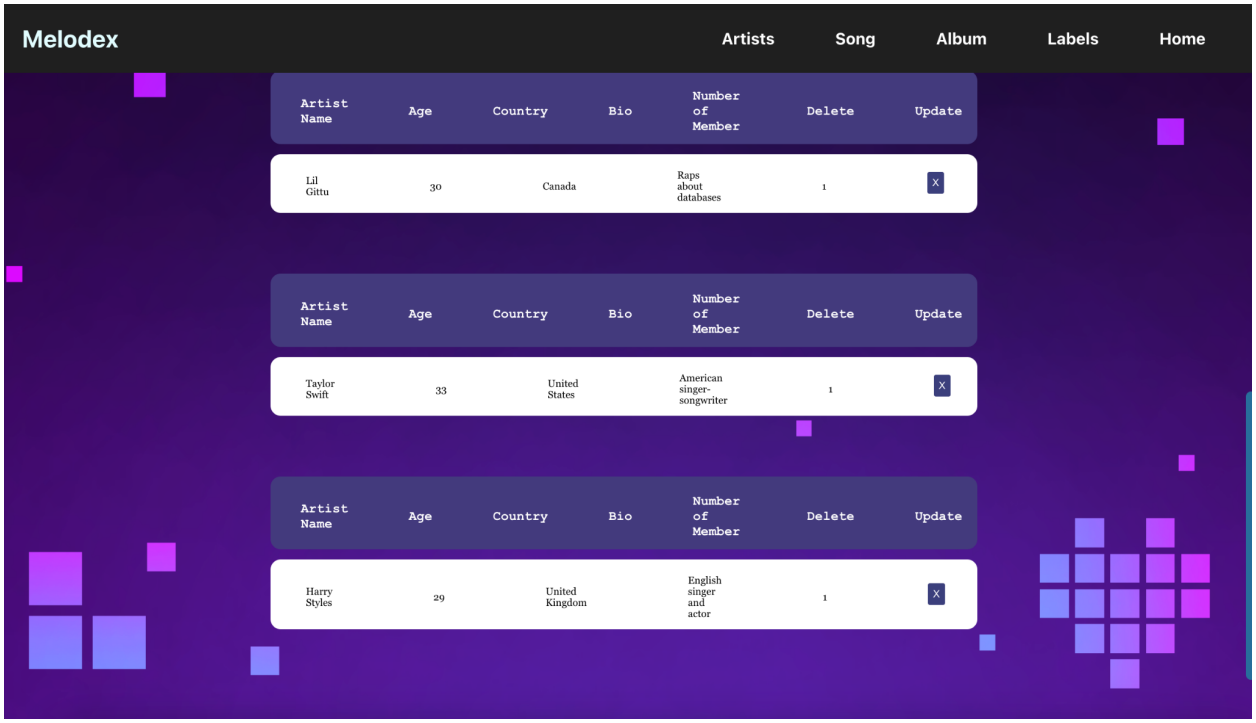
MAINLY implemented in Artist_ContractedWithRepository, division function (backend)
Frontend call:

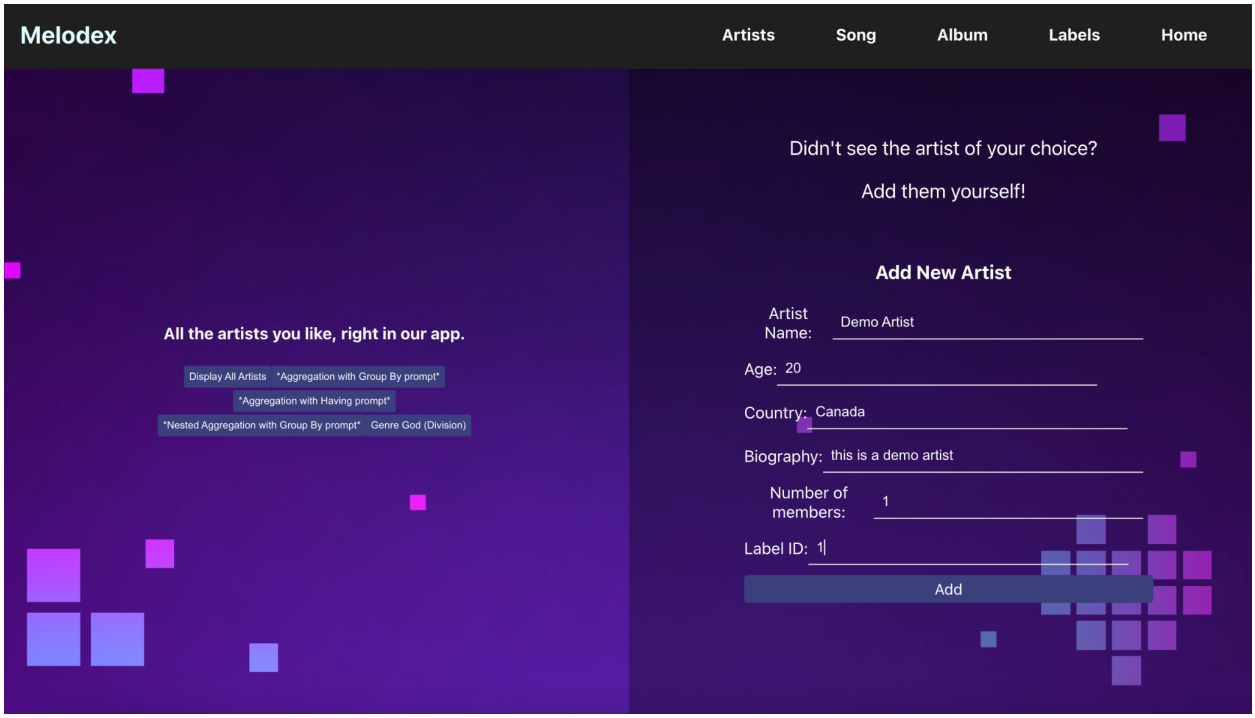
```
SELECT artistID
FROM Artist_ContractedWith A
WHERE NOT EXISTS (
    SELECT DISTINCT genre
    FROM Discography_Main D
    WHERE D.genre NOT IN (
        SELECT DISTINCT Dm.genre
        FROM Discography_Main Dm, Song S, Releases R
        WHERE Dm.dID = R.dID AND S.songID = R.dID AND R.artistID = A.artistID
    )
);
```

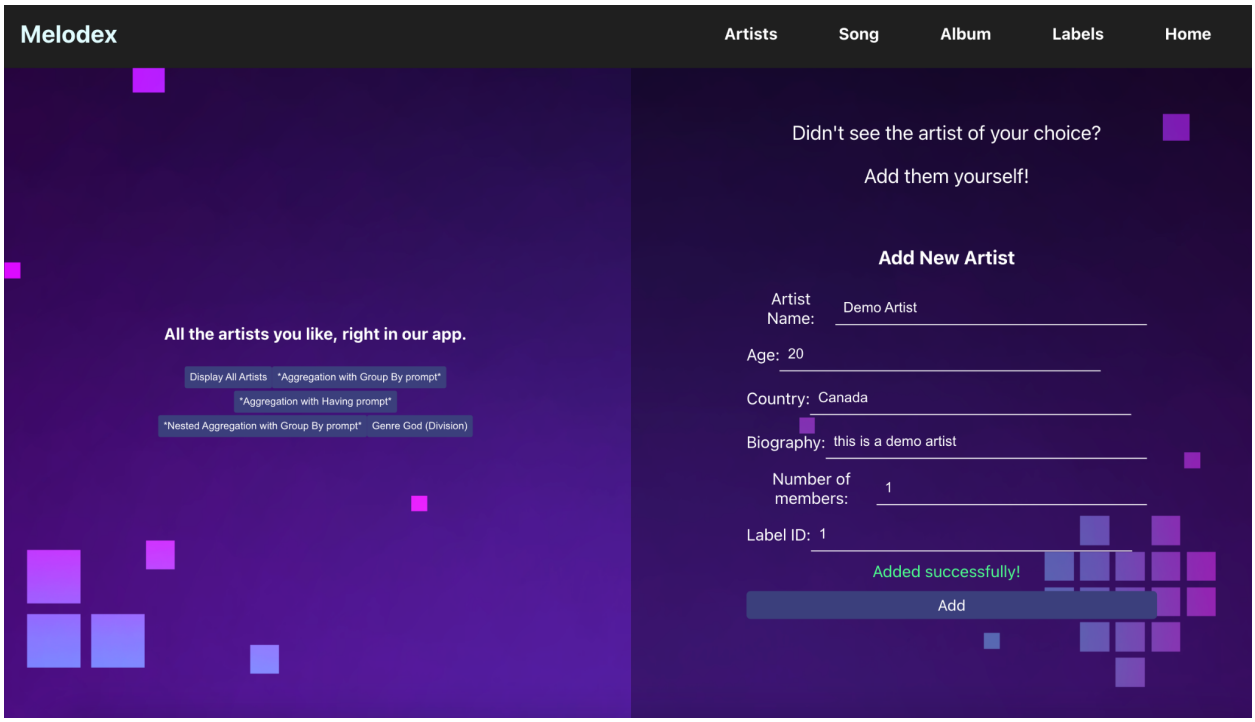
Screenshots of the sample output of the queries using the GUI

INSERTION QUERY:

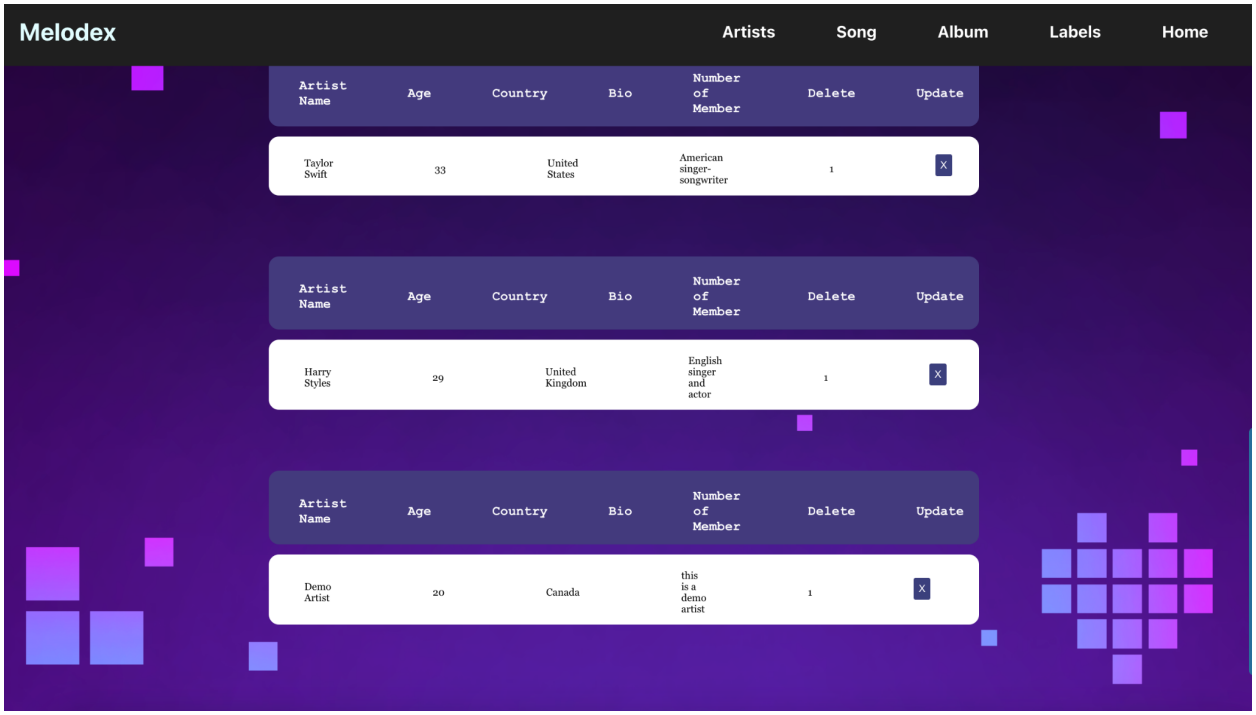
Before insertion:




Melodex
Artists
Song
Album
Labels
Home
All the artists you like, right in our app.
Display All Artists
Aggregation with Group By prompt
Aggregation with Having prompt
Nested Aggregation with Group By prompt
Genre God (Division)
Didn't see the artist of your choice?
Add them yourself!
Add New Artist
Artist Name:
Demo Artist
Age:
20
Country:
Canada
Biography:
this is a demo artist
Number of members:
1
Label ID:
1
Add

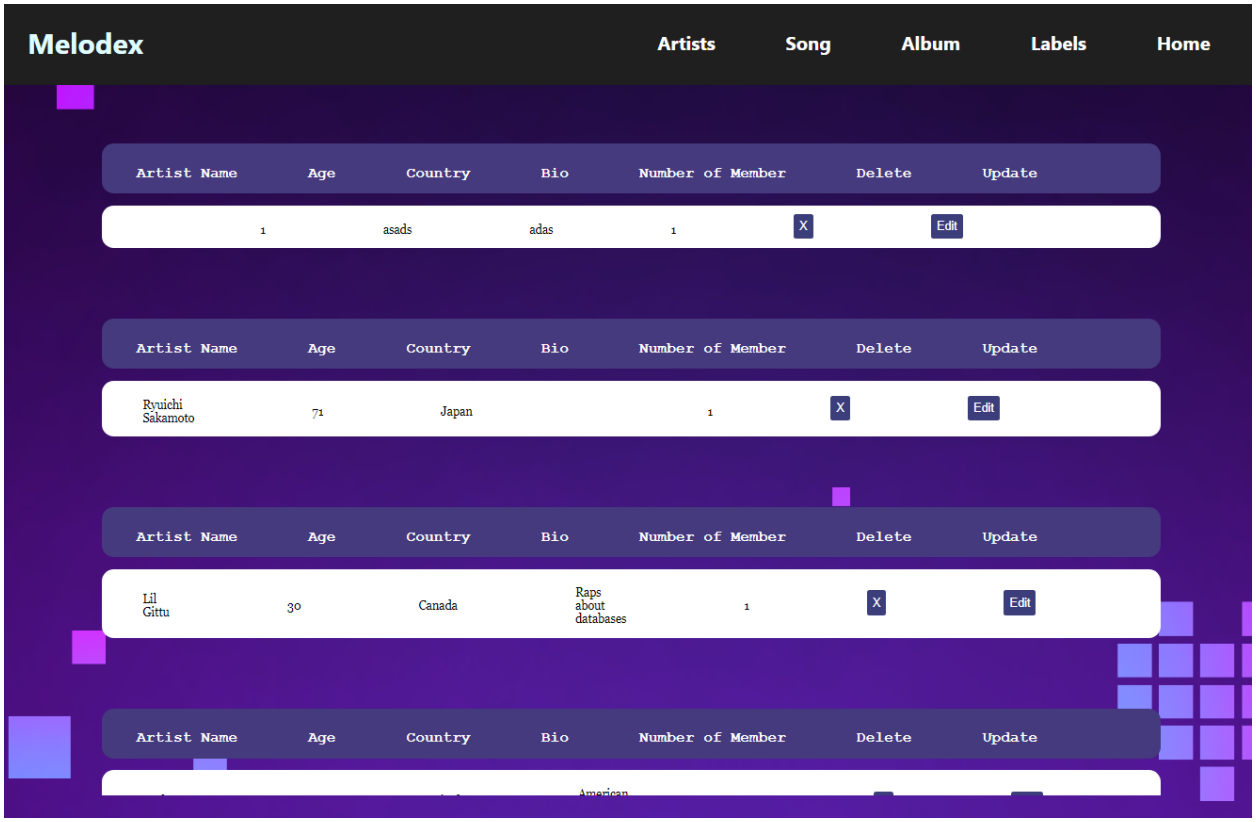

Melodex
Artists
Song
Album
Labels
Home
All the artists you like, right in our app.
Display All Artists
Aggregation with Group By prompt
Aggregation with Having prompt
Nested Aggregation with Group By prompt
Genre God (Division)
Didn't see the artist of your choice?
Add them yourself!
Add New Artist
Artist Name:
Demo Artist
Age:
20
Country:
Canada
Biography:
this is a demo artist
Number of members:
1
Label ID:
1
Added successfully!
Add

After insertion:

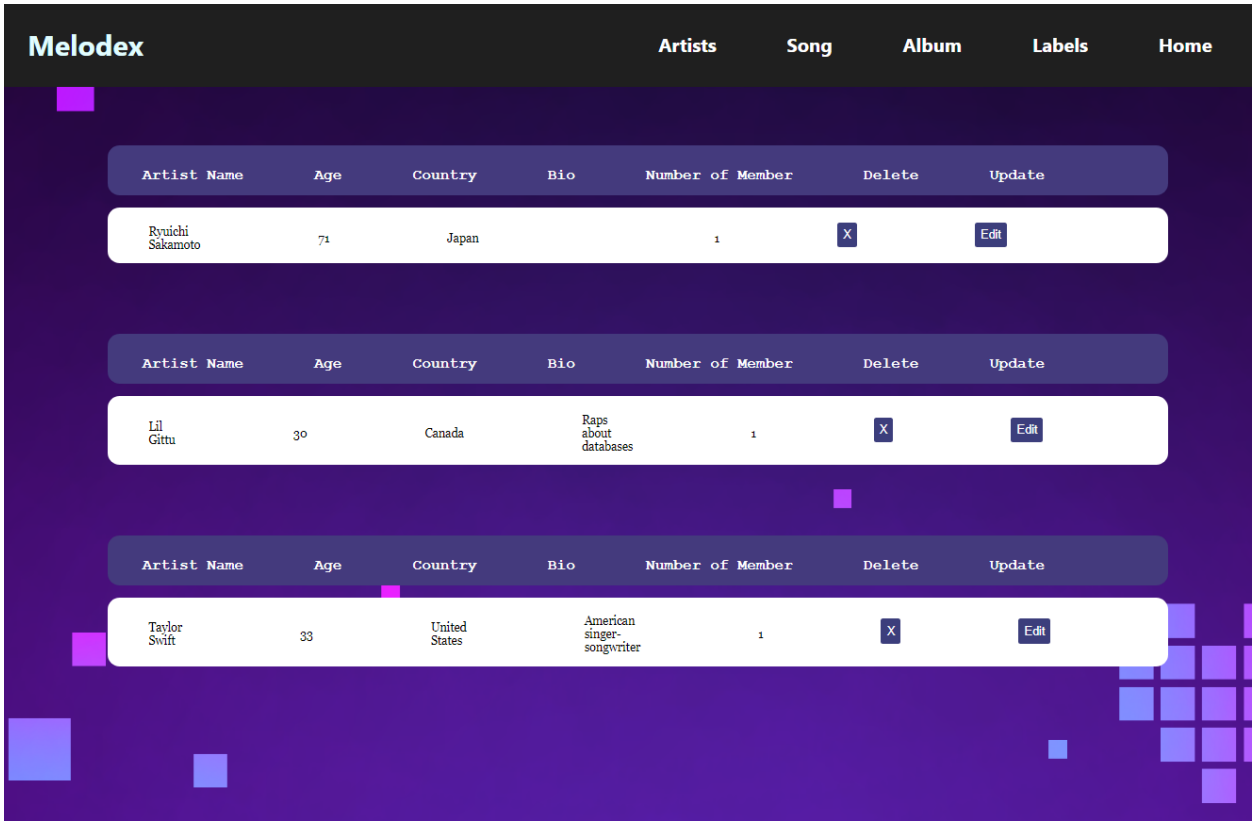


DELETION QUERY:

Before deletion:

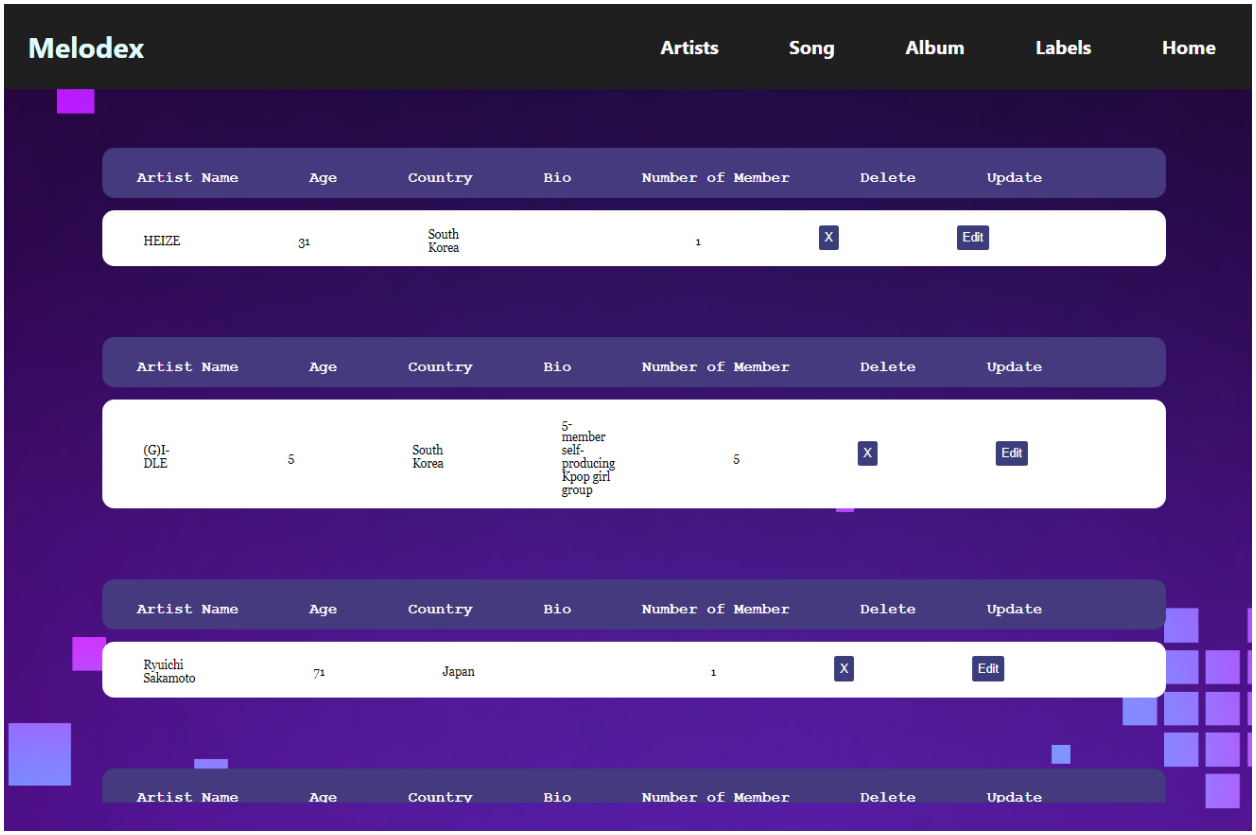


After deletion:

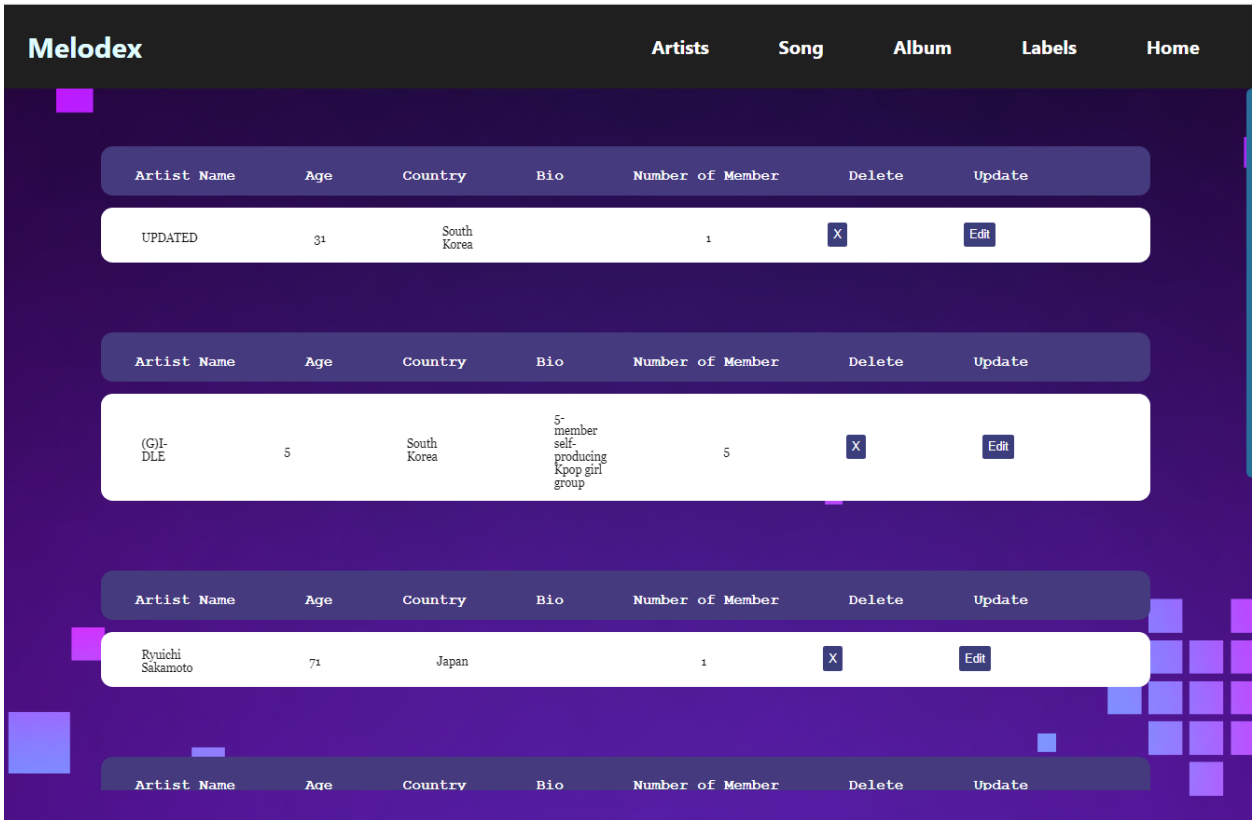


UPDATE QUERY

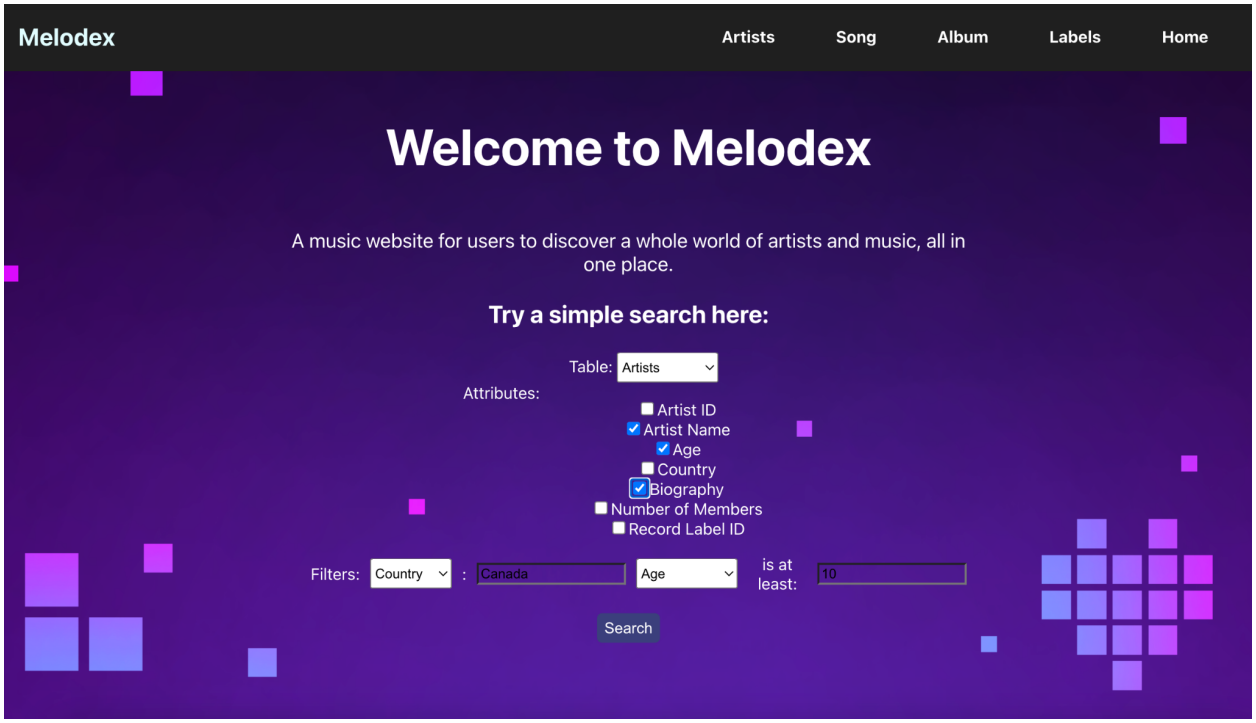
Before update:



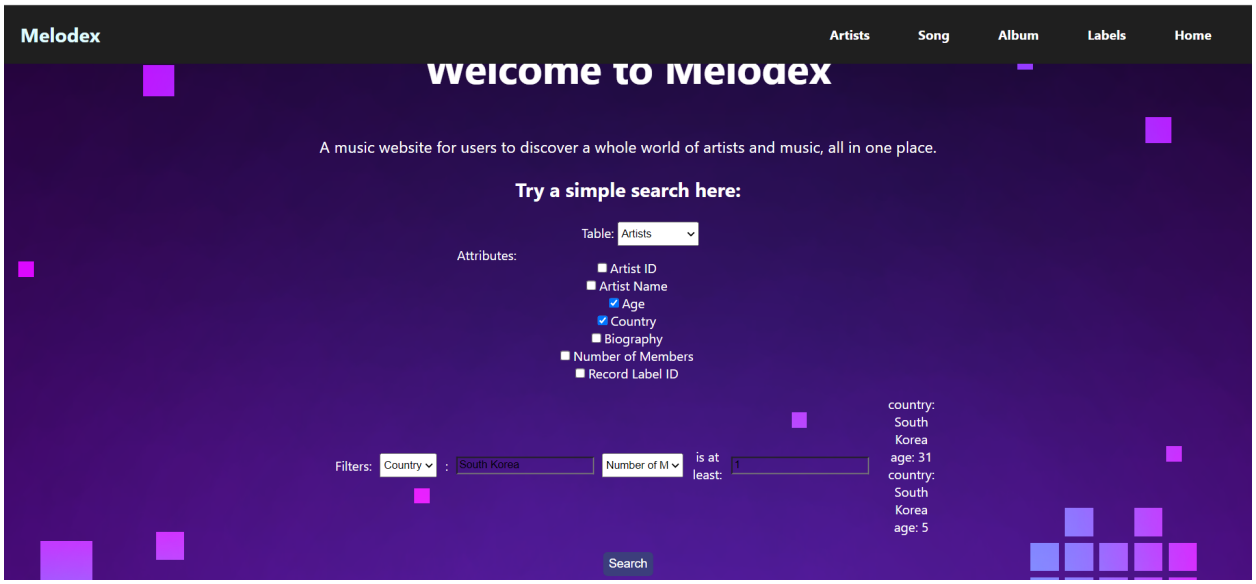
After update:

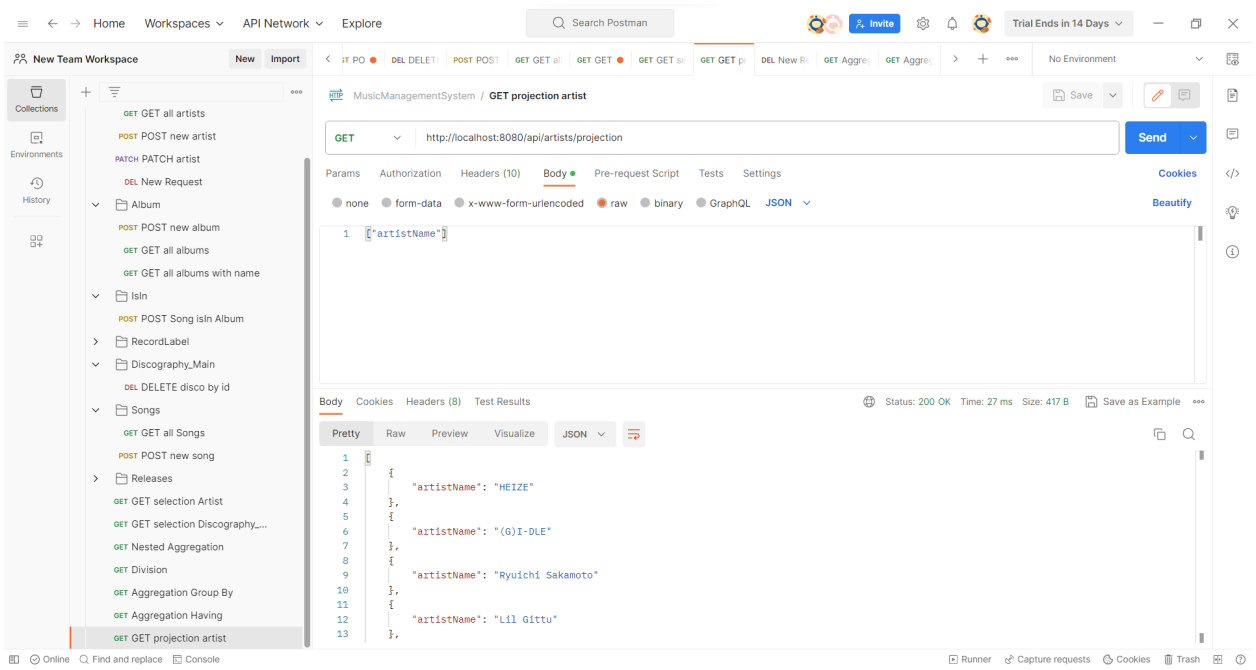


SELECTION QUERY:

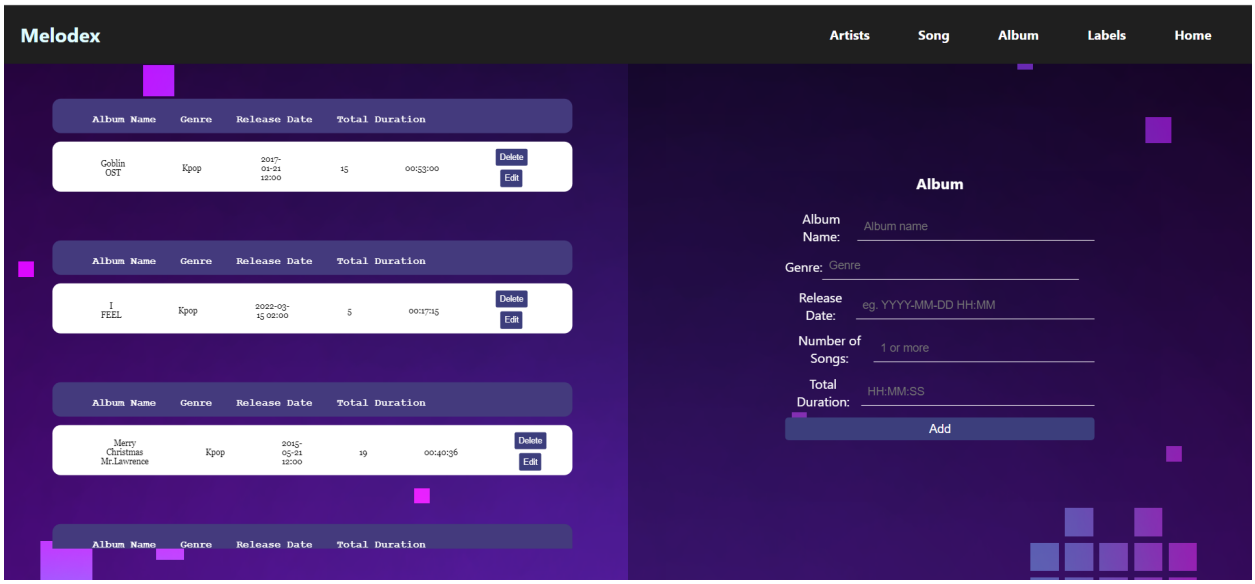


PROJECTION QUERY


Melodex
Artists
Song
Album
Labels
Home
Welcome to Melodex
A music website for users to discover a whole world of artists and music, all in one place.
Try a simple search here:
Table: Artists
Attributes:
Artist ID
Artist Name
Age
Country
Biography
Number of Members
Record Label ID
Filters: Country : South Korea
Number of M
is at least: 1
country: South Korea age: 31
country: South Korea age: 5
Search


Home
Workspaces
API Network
Explore
Search Postman
Invite
Trial Ends in 14 Days
New Team Workspace
New
Import
No Environment
Collections
Environments
History
GET GET all artists
POST POST new artist
PATCH PATCH artist
DEL New Request
Album
POST POST new album
GET GET all albums
GET GET all albums with name
IsIn
POST POST Song isIn Album
RecordLabel
Discography_Main
DEL DELETE disco by id
Songs
GET GET all Songs
POST POST new song
Releases
GET GET selection Artist
GET GET selection Discography_...
GET Nested Aggregation
GET Division
GET Aggregation Group By
GET Aggregation Having
GET GET projection artist
MusicManagementSystem / GET projection artist
Save
GET
http://localhost:8080/api/artists/projection
Send
Params
Authorization
Headers (10)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{"artistName"}
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 27 ms
Size: 417 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
[
{
"artistName": "HEIZE"
},
{
"artistName": "(G)I-DLE"
},
{
"artistName": "Ryuichi Sakamoto"
},
{
"artistName": "Lil Gittu"
},
Online
Find and replace
Console
Runner
Capture requests
Cookies
Trash

JOIN QUERY



AGGREGATION WITH HAVING

Melodex

All the artists you like, right in our app.

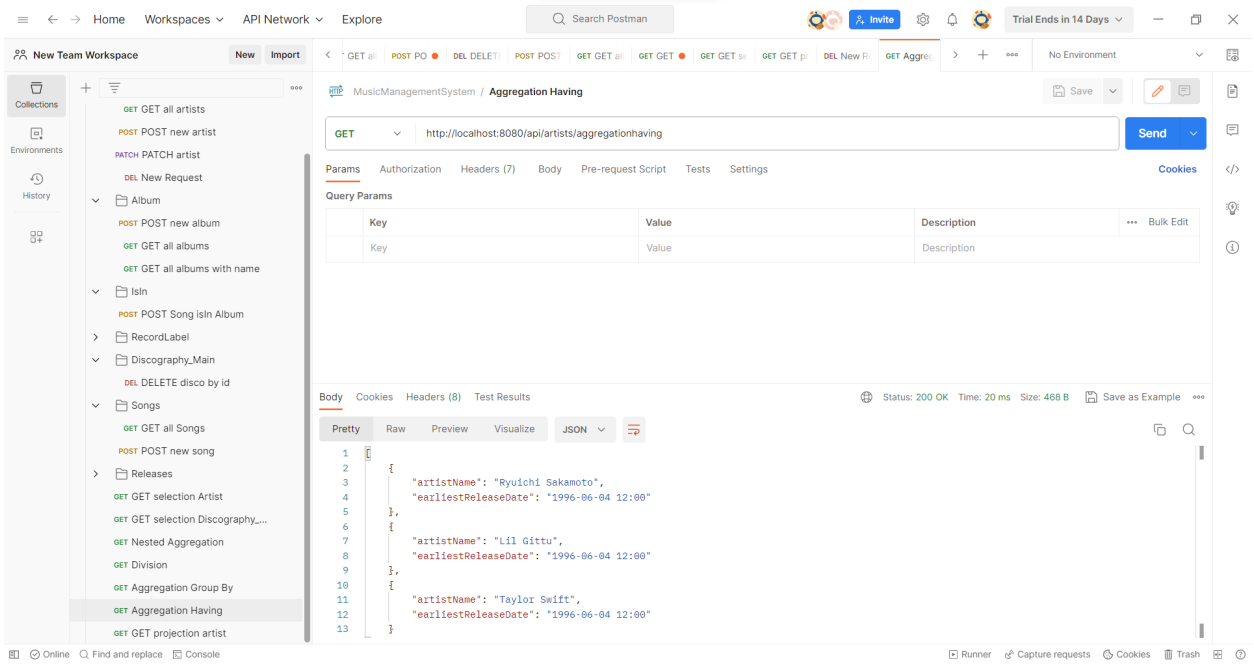
Display All Artists | *Aggregation with Group By prompt*

Aggregation with Having prompt | *Nested Aggregation with Group By prompt*

Genre God (Division)

Result:

[{"artistName":"UPDATED","earliestReleaseDate":"1996-06-04 12:00"},{"artistName":"(G)I-DLE","earliestReleaseDate":"1996-06-04 12:00"},{"artistName":"Ryuichi Sakamoto","earliestReleaseDate":"1996-06-04 12:00"},{"artistName":"Lil Gittu","earliestReleaseDate":"1996-06-04 12:00"},{"artistName":"Taylor Swift","earliestReleaseDate":"1996-06-04 12:00"},{"artistName":"Harry Styles","earliestReleaseDate":"1996-06-04 12:00"}]



AGGREGATION WITH GROUP BY

Melodex

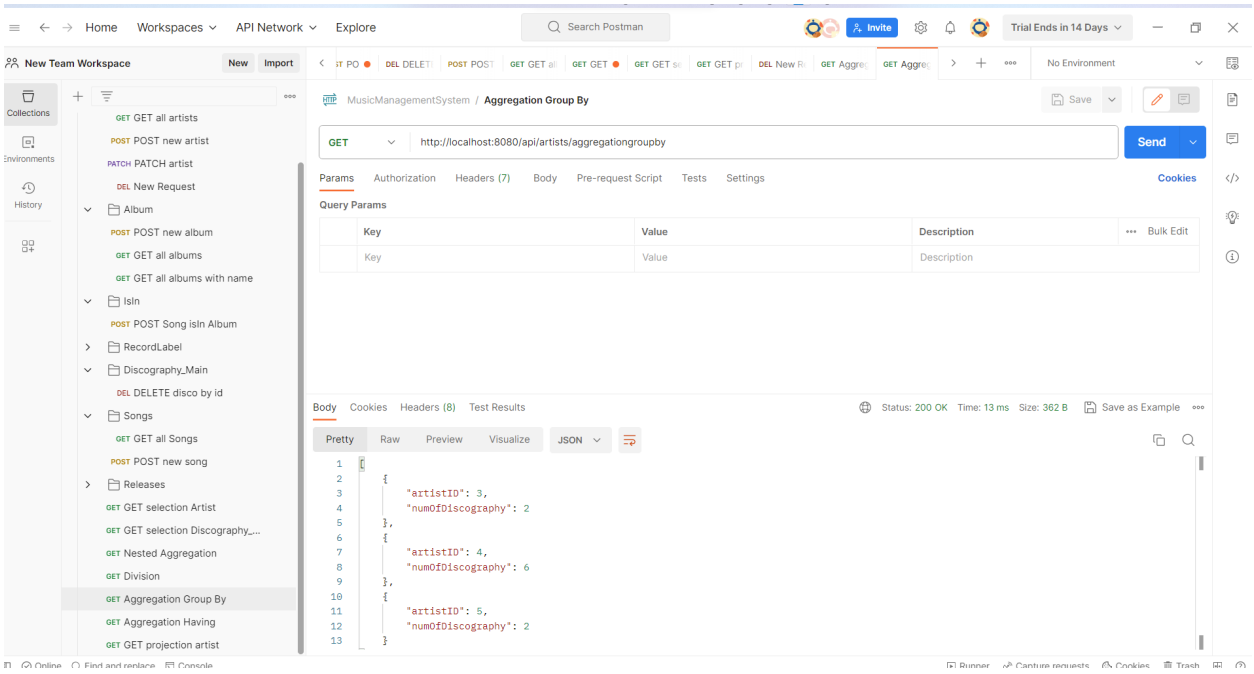
All the artists you like, right in our app.

Display All Artists | *Aggregation with Group By prompt*

Aggregation with Having prompt | *Nested Aggregation with Group By prompt*

Genre God (Division)

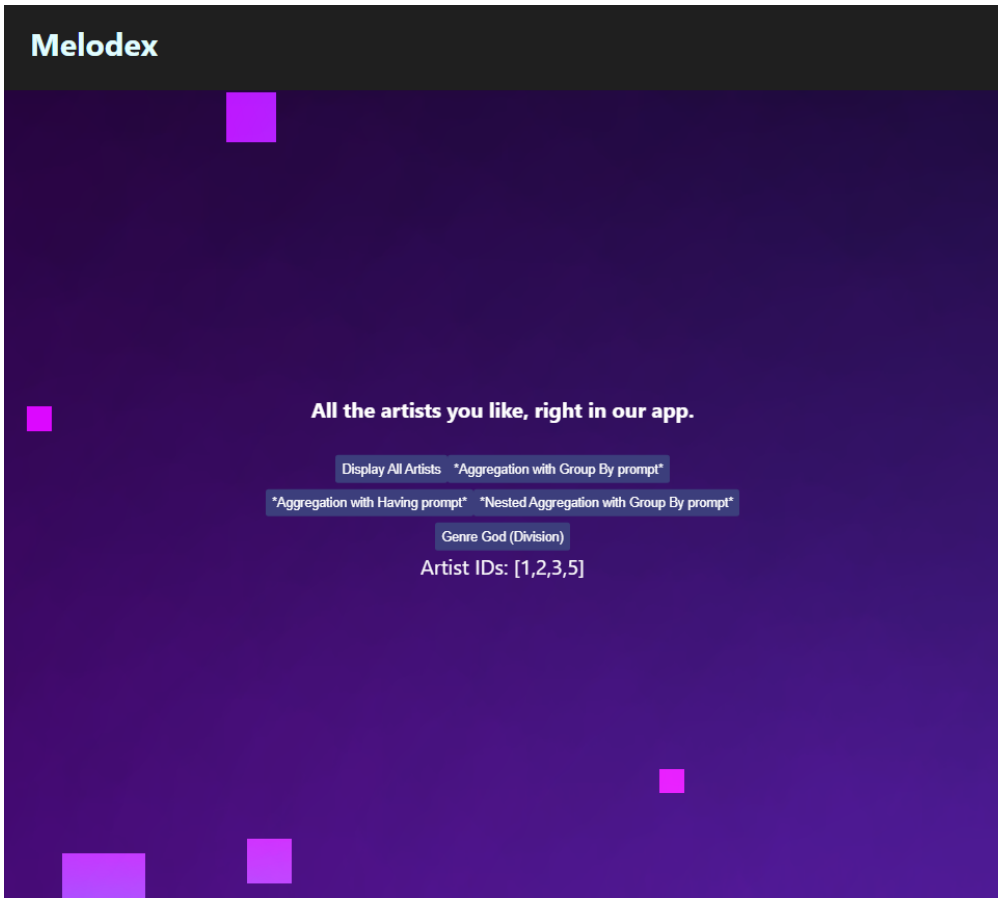
Result: [{"artistID":1,"numOfDiscography":2},
{"artistID":2,"numOfDiscography":2},
{"artistID":3,"numOfDiscography":2},
{"artistID":4,"numOfDiscography":6},
{"artistID":5,"numOfDiscography":2}]

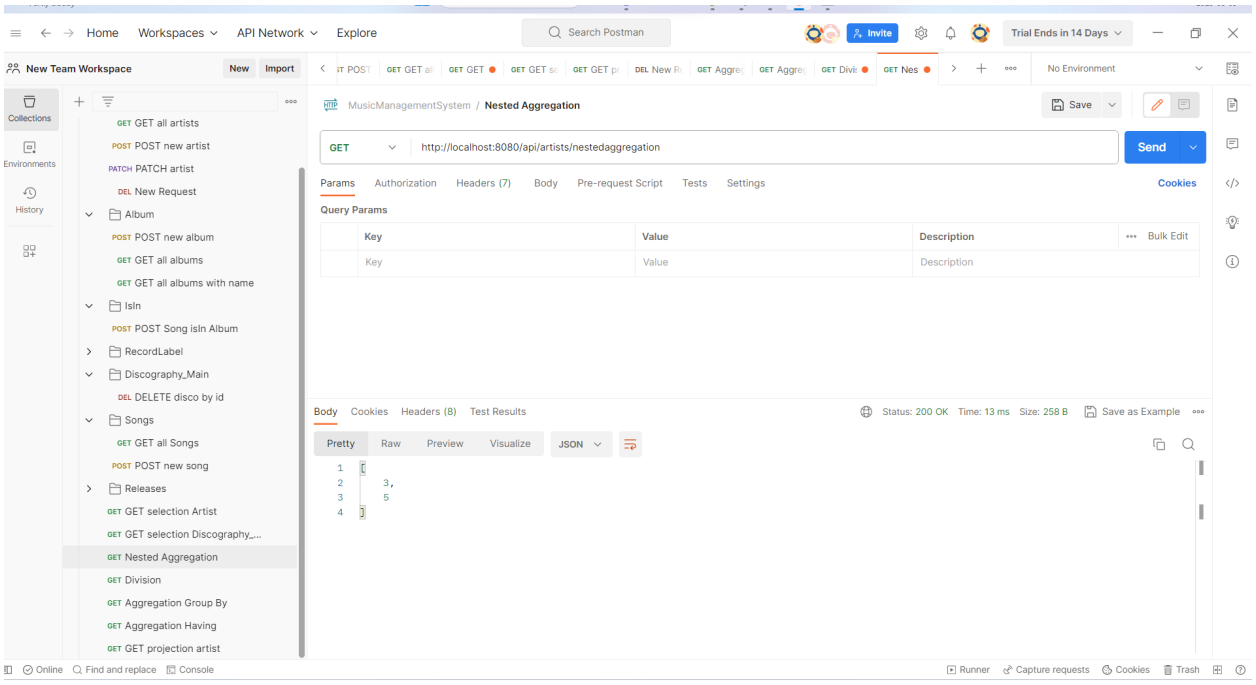


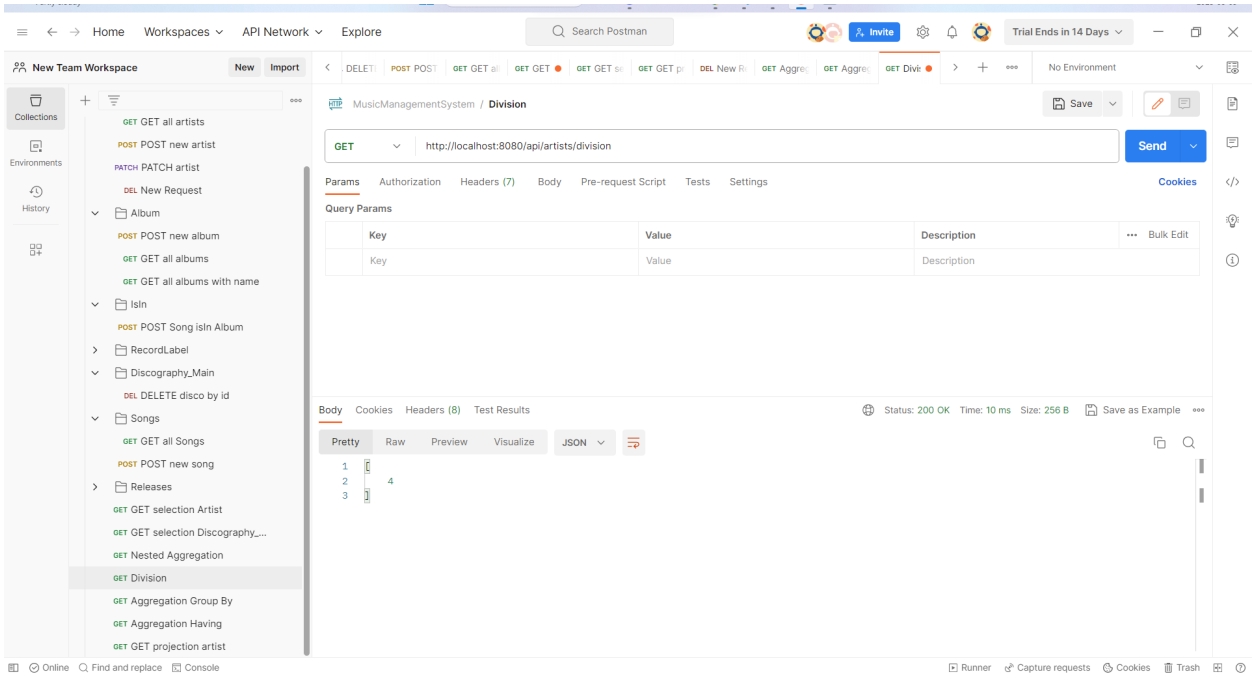
Relevant frontend code:

```
const aggGroupFetch = async (e) => {
  const response = await fetch('http://localhost:8080/api/artists/aggregationgroupby', {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setAggGroup(json);
    console.log("Fetched aggrogroup! ", json)
  } else {
    console.log("error");
  }
}

const aggHavingFetch = async (e) => {
  const response = await fetch('http://localhost:8080/api/artists/aggregationhaving', {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setAggGroup(json);
    console.log("Fetched aggHaving! ", json)
  } else {
    console.log("error");
  }
}
```

NESTED AGGREGATION WITH GROUP BY

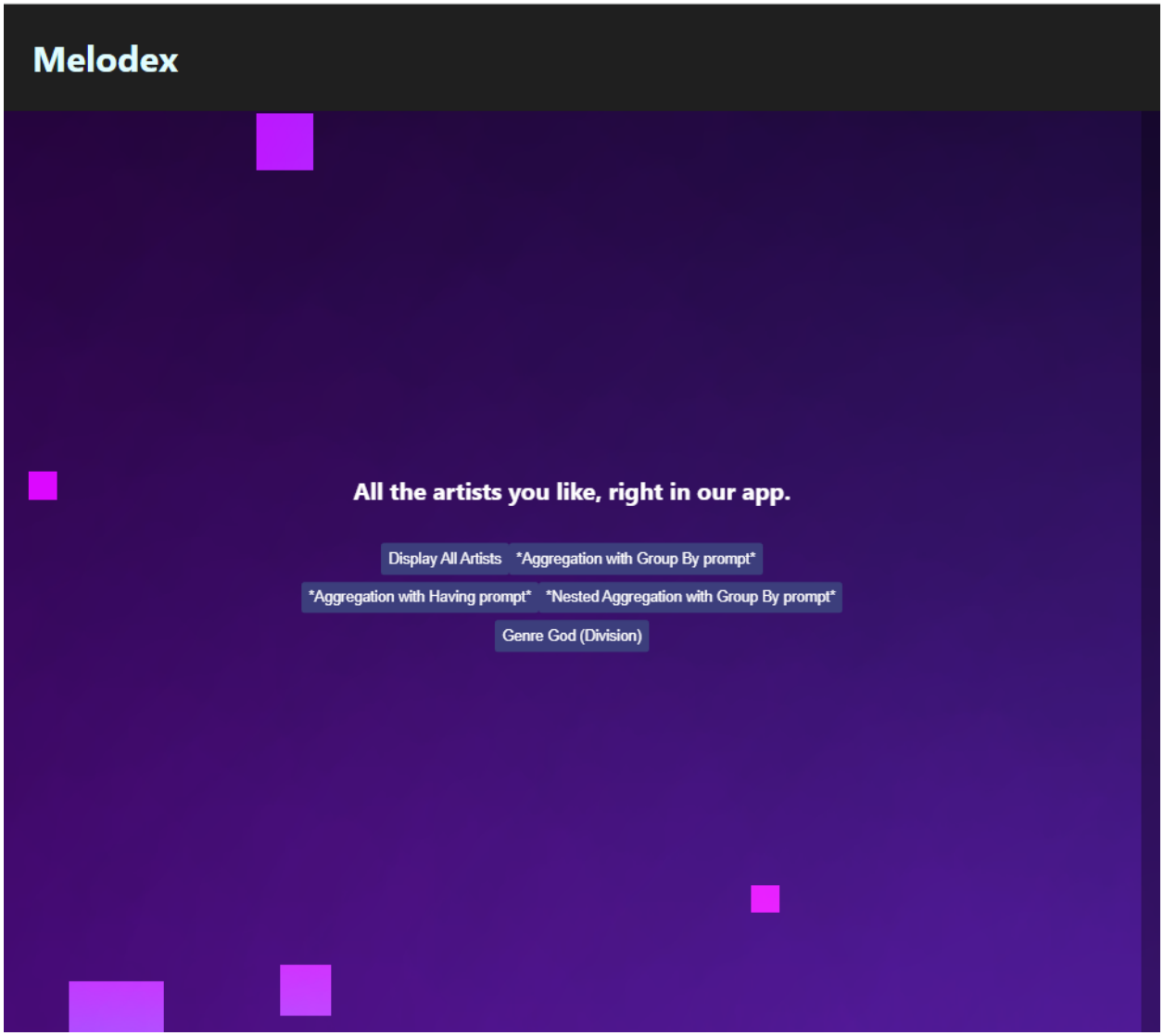

Melodex
All the artists you like, right in our app.
Display All Artists
Aggregation with Group By prompt
Aggregation with Having prompt
Nested Aggregation with Group By prompt
Genre God (Division)
Artist IDs: [1,2,3,5]


Home
Workspaces
API Network
Explore
Search Postman
Invite
Trial Ends in 14 Days
New Team Workspace
New
Import
No Environment
Collections
Environments
History
MusicManagementSystem / Nested Aggregation
Save
GET
http://localhost:8080/api/artists/nestedaggregation
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (8)
Test Results
Status: 200 OK
Time: 13 ms
Size: 258 B
Save as Example
Pretty
Raw
Preview
Visualize
JSON
[
3,
5
]
GET GET all artists
POST POST new artist
PATCH PATCH artist
DEL New Request
Album
POST POST new album
GET GET all albums
GET GET all albums with name
IsIn
POST POST Song isIn Album
RecordLabel
Discography_Main
DEL DELETE disco by id
Songs
GET GET all Songs
POST POST new song
Releases
GET GET selection Artist
GET GET selection Discography_...
GET Nested Aggregation
GET Division
GET Aggregation Group By
GET Aggregation Having
GET GET projection artist
Online
Find and replace
Console
Runner
Capture requests
Cookies
Trash

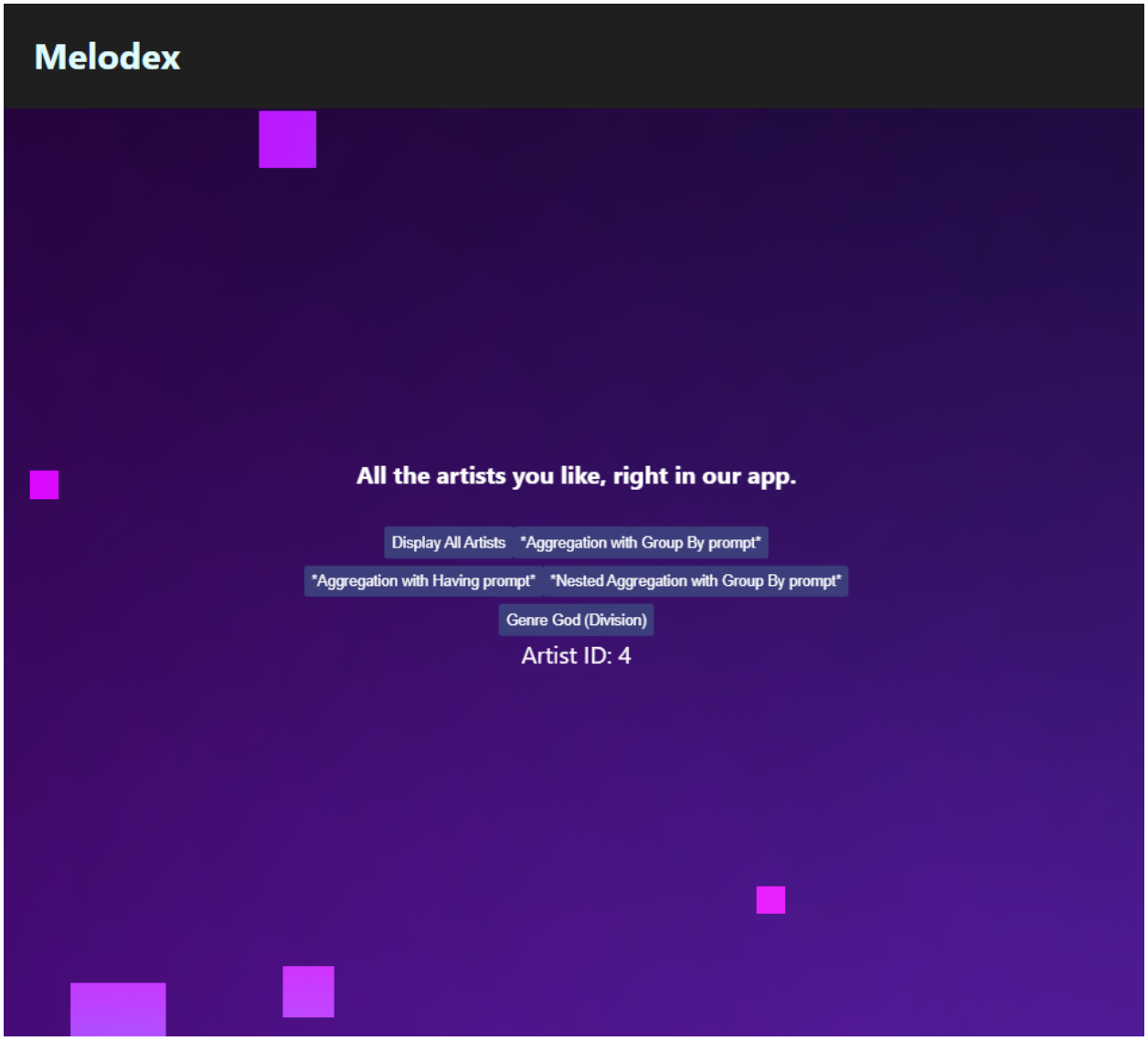


DIVISION QUERY

Before division:



After division:



Relevant frontend code:

```
const [aggGroupClicked, setAggGroupClicked] =useState(false);
const [aggHavingClicked, setAggHavingClicked] =useState(false);
const [nestedAggClicked, setNestedAggClicked] =useState(false);
const [divisionClicked, setDivisionClicked] =useState(false);

const [aggGroup, setAggGroup] =useState('');
const [aggHaving, setAggHaving] =useState('');
const [nestedAgg, setNestedAgg] =useState('');
const [divisionObj, setDivisionObj] =useState('');


const aggGroupFetch = async (e) => {
  setAggGroupClicked(true);
  const response = await fetch('http://localhost:8080/api/artists/aggregationgroupby', {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setAggGroup(json);
    console.log("Fetched aggrogroup! ", json)
  } else {
    console.log("error");
  }
}
```

```
const aggHavingFetch = async (e) => {
  setAggHavingClicked(true);
  const response = await fetch('http://localhost:8080/api/artists/aggregationhaving', {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setAggHaving(json);
    console.log("Fetched aggHaving! ", json)
  } else {
    console.log("error");
  }
}

const divisionFetch = async (e) => {
  setDivisionClicked(true);
  const response = await fetch('http://localhost:8080/api/artists/division', {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setDivisionObj(json);
    console.log("Fetched division! ", json)
  } else {
    console.log("error");
  }
}
```

```
const nestedFetch = async (e) => {
  setNestedAggClicked(true);
  const response = await fetch('http://localhost:8080/api/artists/nestedaggregation', {
    method: 'GET'
  }).catch((err) => {
    console.log(err);
  })

  if (response.ok) {
    const json = await response.json();
    setNestedAgg(json);
    console.log("Fetched nested! ", json)
  } else {
    console.log("error");
  }
}
```

```
return (
  <div className={styles.regContainer}>
    <div className={styles.leftBody}>
      <div className={styles.centered}>
        <div className={styles.title}>
          <h4>All the artists you like, right in our app.</h4>
          <Link to="/display-artists">
            <button>Display All Artists</button>
          </Link>
          <button onClick={aggGroupFetch}>*Aggregation with Group By prompt*</button>
          <button onClick={aggHavingFetch}>*Aggregation with Having prompt*</button>
          <button onClick={nestedFetch}>*Nested Aggregation with Group By prompt*</button>
          <button onClick={divisionFetch}>Genre God (Division)</button>
        </div>

        {divisionClicked && <div>Artist ID: {divisionObj}</div>}
        {aggHavingClicked && <div>Result: {JSON.stringify(aggHaving)}</div>}
        {aggGroupClicked && <div>Result: {JSON.stringify(aggGroup)}</div>}
        {nestedAggClicked && <div>Artist IDs: {JSON.stringify(nestedAgg)}</div>}
      </div>
    </div>
    <div className={styles.rightBody}>
      <div className={styles.centered}>
        <div className={styles.addForm}>
          <p>Didn't see the artist of your choice?</p>
          <p>Add them yourself!</p>
          <ArtistForm />
        </div>
      </div>
    </div>
  </div>
);
```