Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

Obtain the Chinook sample database:

```
student@tuffix-vm:~$ wget https://www.sqlitetutorial.net/wp-content/uploads/20
18/03/chinook.zip
--2023-09-22 21:13:45--  https://www.sqlitetutorial.net/wp-content/uploads/201
8/03/chinook.zip
Resolving www.sqlitetutorial.net (www.sqlitetutorial.net)... 172.67.172.250, 1
04.21.30.141, 2606:4700:3037::ac43:acfa, ...
Connecting to www.sqlitetutorial.net (www.sqlitetutorial.net)|172.67.172.250|:
443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 305596 (298K) [application/zip]
Saving to: ‘chinook.zip’

chinook.zip         100%[===================>] 298.43K  --.-KB/s    in 0.06s


2023-09-22 21:13:45 (4.78 MB/s) - ‘chinook.zip’ saved [305596/305596]

student@tuffix-vm:~$ unzip chinook.zip
Archive:  chinook.zip
  inflating: chinook.db
student@tuffix-vm:~$
```

Update system, Install sqlite3 and dump the sample database:

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```
student@tuffix-vm:~$ sudo apt update
[sudo] password for student:
Get:1 http://packages.microsoft.com/repos/code stable InRelease [3,569 B]
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:3 https://storage.googleapis.com/bazel-apt stable InRelease [2,262 B]
Get:4 https://cli.github.com/packages stable InRelease [3,917 B]
Hit:5 http://gb.archive.ubuntu.com/ubuntu jammy InRelease
Get:6 http://gb.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:7 http://packages.microsoft.com/repos/code stable/main armhf Packages [80.
7 kB]
Get:8 http://gb.archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
```

```
Get:70 http://gb.archive.ubuntu.com/ubuntu jammy-backports/universe DEP-11 64x
64 Icons [25.6 kB]
Get:71 http://gb.archive.ubuntu.com/ubuntu jammy-backports/universe amd64 c-n-
f Metadata [640 B]
Fetched 10.7 MB in 19s (552 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
421 packages can be upgraded. Run 'apt list --upgradable' to see them.
student@tuffix-vm:~$
```

```
The following NEW packages will be installed:
  sqlite3
0 upgraded, 1 newly installed, 0 to remove and 421 not upgraded.
Need to get 768 kB of archives.
After this operation, 1,873 kB of additional disk space will be used.
Get:1 http://gb.archive.ubuntu.com/ubuntu jammy-updates/main amd64 sqlite3 amd
64 3.37.2-2ubuntu0.1 [768 kB]
Fetched 768 kB in 2s (472 kB/s)
Selecting previously unselected package sqlite3.
(Reading database ... 266904 files and directories currently installed.)
Preparing to unpack .../sqlite3_3.37.2-2ubuntu0.1_amd64.deb ...
Unpacking sqlite3 (3.37.2-2ubuntu0.1) ...
Setting up sqlite3 (3.37.2-2ubuntu0.1) ...
Processing triggers for man-db (2.10.2-1) ...
student@tuffix-vm:~$
```

Database dump was too long to post, so just a screen shot from the end.

```
CREATE INDEX [IFK_InvoiceCustomerId] ON "invoices" ([CustomerId]);
CREATE INDEX [IFK_InvoiceLineInvoiceId] ON "invoice_items" ([InvoiceId]);
CREATE INDEX [IFK_InvoiceLineTrackId] ON "invoice_items" ([TrackId]);
CREATE INDEX [IFK_PlaylistTrackTrackId] ON "playlist_track" ([TrackId]);
CREATE INDEX [IFK_TrackAlbumId] ON "tracks" ([AlbumId]);
CREATE INDEX [IFK_TrackGenreId] ON "tracks" ([GenreId]);
CREATE INDEX [IFK_TrackMediaTypeId] ON "tracks" ([MediaTypeId]);
COMMIT;
student@tuffix-vm:~$
```

<u>Configure a python virtual environment:</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```
student@tuffix-vm: ~
Setting up libjs-jquery (3.6.0+dfsg+~3.5.13-1) ...
Setting up libjs-underscore (1.13.2~dfsg-2) ...
Setting up python3.10-minimal (3.10.12-1~22.04.2) ...
Setting up libpython3.10-stdlib:amd64 (3.10.12-1~22.04.2) ...
Setting up libjs-sphinxdoc (4.3.2-1) ...
Setting up libpython3.10:amd64 (3.10.12-1~22.04.2) ...
Setting up python3.10 (3.10.12-1~22.04.2) ...
Setting up libpython3.10-dev:amd64 (3.10.12-1~22.04.2) ...
Setting up python3.10-dev (3.10.12-1~22.04.2) ...
Setting up libpython3-dev:amd64 (3.10.6-1~22.04) ...
Setting up python3.10-venv (3.10.12-1~22.04.2) ...
Setting up python3-venv (3.10.6-1~22.04) ...
Setting up python3-dev (3.10.6-1~22.04) ...
Processing triggers for desktop-file-utils (0.26-1ubuntu3) ...
Processing triggers for gnome-menus (3.36.0-1ubuntu3) ...
Processing triggers for libc-bin (2.35-0ubuntu3.1) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for mailcap (3.70+nmu1ubuntu1) ...
student@tuffix-vm:~$ python3.10 -m venv $HOME/.venv
student@tuffix-vm:~$ echo 'source $HOME/.venv/bin/activate' | tee -a $HOME/.ba
shrc
source $HOME/.venv/bin/activate
student@tuffix-vm:~$ . $HOME/.venv/bin/activate
(.venv) student@tuffix-vm:~$
```

Install n version manager for Node.js and update node package manager to run the API's:

```
######################################################################## 100.0
     copying : node/18.18.0
   installed : v18.18.0 (with npm 9.8.1)
=== n successfully installed.
  The active Node.js version is: v18.18.0

  Run `n -h` for help.
  To update n later, run `n-update`.
  To uninstall, run `n-uninstall`.

  IMPORTANT: OPEN A NEW TERMINAL TAB/WINDOW or run `. /home/student/.bashrc`
             before using n and Node.js.
===
(.venv) student@tuffix-vm:~$ . $HOME/.bashrc
(.venv) student@tuffix-vm:~$ npm update --global

removed 2 packages, and changed 60 packages in 5s

28 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New major version of npm available! 9.8.1 -> 10.1.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.1.0
npm notice Run npm install -g npm@10.1.0 to update!
npm notice
(.venv) student@tuffix-vm:~$
```

Install the Soul server to expose a REST API for an existing database:

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```
(.venv) student@tuffix-vm:~$ npm update --global

removed 2 packages, and changed 60 packages in 5s

28 packages are looking for funding
  run `npm fund` for details
npm notice
npm notice New major version of npm available! 9.8.1 -> 10.1.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.1.0
npm notice Run npm install -g npm@10.1.0 to update!
npm notice
(.venv) student@tuffix-vm:~$ npm install --global soul-cli

added 161 packages in 17s

20 packages are looking for funding
  run `npm fund` for details
(.venv) student@tuffix-vm:~$
```

Install the tuql server to expose the GraphQL API for an existing database:

```
(.venv) student@tuffix-vm:~$ npm install --global tuql
npm WARN ERESOLVE overriding peer dependency
npm WARN While resolving: graphql-sequelize@9.5.1
npm WARN Found: graphql-relay@0.6.0
npm WARN node_modules/tuql/node_modules/graphql-relay
npm WARN   graphql-relay@"^0.6.0" from tuql@1.7.0
npm WARN   node_modules/tuql
npm WARN     tuql@"*" from the root project
npm WARN
npm WARN Could not resolve dependency:
npm WARN peer graphql-relay@"^0.4.2 || ^0.5.0 || ^0.7.0 || ^0.8.0 || ^0.9.0 ||
 ^0.10.0" from graphql-sequelize@9.5.1
npm WARN node_modules/tuql/node_modules/graphql-sequelize
npm WARN   graphql-sequelize@"^9.3.6" from tuql@1.7.0
npm WARN   node_modules/tuql
npm WARN
npm WARN Conflicting peer dependency: graphql-relay@0.10.0
npm WARN node_modules/graphql-relay
npm WARN   peer graphql-relay@"^0.4.2 || ^0.5.0 || ^0.7.0 || ^0.8.0 || ^0.9.0
|| ^0.10.0" from graphql-sequelize@9.5.1
npm WARN   node_modules/tuql/node_modules/graphql-sequelize
npm WARN     graphql-sequelize@"^9.3.6" from tuql@1.7.0
npm WARN     node_modules/tuql
npm WARN deprecated node-pre-gyp@0.11.0: Please upgrade to @mapbox/node-pre-gy
p: the non-scoped node-pre-gyp package is deprecated and only the @mapbox scop
ed package will recieve updates in the future
npm WARN deprecated express-graphql@0.9.0: This package is no longer maintaine
d. We recommend using `graphql-http` instead. Please consult the migration doc
ument https://github.com/graphql/graphql-http#migrating-express-grpahql.
npm WARN deprecated sequelize@5.22.5: Please update to v6 or higher! A migrati
on guide can be found here: https://sequelize.org/v6/manual/upgrade-to-v6.html

added 182 packages in 2m

10 packages are looking for funding
  run `npm fund` for details
(.venv) student@tuffix-vm:~$
```

Start the API servers:

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```
student@tuffix-vm: ~                    student@tuffix
(.venv) student@tuffix-vm:~$ tuql --db chinook.db --graphiql

 > Reading schema from /home/student/chinook.db
 > Running at http://localhost:4000/graphql
```

```
student@tuffix-vm: ~                    student@tuffix-vm: ~
(.venv) student@tuffix-vm:~$ soul --database chinook.db --studio
No extensions directory provided
Soul is running...
 > Core API at http://localhost:8000/api/
 > Studio at http://localhost:8000/studio/
```

Run test query in GraphQL to display artistId and name from the artist table:

localhost:4000/graphql?query=query {%0A%20

GraphiQL ▶ Prettify Merge Copy History

```
query {
  artist(where: {artistId: 1}) {
    artistId,
    name
  }
}
```

```
{
  "data": {
    "artist": {
      "artistId": 1,
      "name": "AC/DC"
    }
  }
}
```

Albums by the artist “Red Hot Chili Peppers.” (GraphiQL)

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#


query {
   artist(where: {name: "Red Hot Chili Peppers"}) {
    albums{
      title
    }
  }
}
```

```
{
  "data": {
    "artist": {
      "albums": [
        {
          "title": "Blood Sugar Sex Magik"
        },
        {
          "title": "By The Way"
        },
        {
          "title": "Californication"
        }
      ]
    }
  }
}
```

For the code above, we want information about an artist. The “where” keyword is to filter the artist. The “name” keyword is used to find an artist whose name is Red Hot Chili Peppers. Inside there is a field called “album” which signifies we are grabbing information about the albums that are associated with Red Hot Chili Peppers. The “title” field allows us to retrieve the titles of the albums associated with Red Hot Chili Peppers. This query allows us to request the titles of the albums associated with Red Hot Chili Peppers.

<u>Genres associated with the artist “U2.” (GraphiQL)</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

GraphiQL ▶ Prettify Merge Copy History

```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#


query {
   artist(where: {name: "U2"}) {
    albums{
      tracks {
        genre {
          name
        }
      }
    }
  }
}
```

QUERY VARIABLES


```
{
  "data": {
    "artist": {
      "albums": [
        {
          "tracks": [
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            },
            {
              "genre": {
                "name": "Rock"
              }
            }
```


The “artist” signifies we want information about an artist. The “where” and the “name” is the filter of data and we want to find an artist by the name of “U2”. “Albums” indicates we want to retrieve information about the albums associated with “U2”. We then grab the “tracks” by implementing the field right after. Next “genre” and “name” are implemented to get the genre of the track. This query allows us to get the names of the genres for the tracks associated with the albums of the artist U2.

<u>Names of tracks on the playlist Grunge and their associated artists and albums(GraphiQL)</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

GraphiQL ▶ Prettify Merge Copy History

```
# Welcome to GraphiQL
#
# GraphiQL is an in-browser tool for writing, validating, and
# testing GraphQL queries.
#
# Type queries into this side of the screen, and you will see intelligent
# typeaheads aware of the current GraphQL type schema and live syntax and
# validation errors highlighted within the text.
#
# GraphQL queries typically start with a "{" character. Lines that starts
# with a # are ignored.
#
# An example GraphQL query might look like:
#
#     {
#       field(arg: "value") {
#         subField
#       }
#     }
#
# Keyboard shortcuts:
#
#  Prettify Query:  Shift-Ctrl-P (or press the prettify button above)
#
#     Merge Query:  Shift-Ctrl-M (or press the merge button above)
#
#       Run Query:  Ctrl-Enter (or press the play button above)
#
#   Auto Complete:  Ctrl-Space (or just start typing)
#


query {
   playlist(where: {name: "Grunge"}) {
    tracks{
      name
      album {
        title
        artist {
          name
        }
      }
    }
  }
}
```

QUERY VARIABLES

```
{
  "data": {
    "playlist": {
      "tracks": [
        {
          "name": "Man In The Box",
          "album": {
            "title": "Facelift",
            "artist": {
              "name": "Alice In Chains"
            }
          }
        },
        {
          "name": "Smells Like Teen Spirit",
          "album": {
            "title": "Nevermind",
            "artist": {
              "name": "Nirvana"
            }
          }
        },
        {
          "name": "In Bloom",
          "album": {
            "title": "Nevermind",
            "artist": {
              "name": "Nirvana"
            }
          }
        },
        {
          "name": "Come As You Are",
          "album": {
            "title": "Nevermind",
            "artist": {
              "name": "Nirvana"
            }
          }
        },
        {
          "name": "Lithium",
          "album": {
            "title": "Nevermind",
            "artist": {
              "name": "Nirvana"
            }
          }
        },
        {
          "name": "Drain You",
          "album": {
            "title": "Nevermind",
            "artist": {
              "name": "Nirvana"
            }
          }
        },
        {
          "name": "On A Plain",
```

The “playlist” field indicates we want information about a playlist. The “where” and “name” is to filter the data and match a playlist that is Grunge. We then have “tracks” with several nested fields that include “name”, “album”, “artist”. All these fields help us to get the names of the in the Grunge Playlist, the associated albums of each track, and the associated artist. We then added an “album” field followed by “title” which helps to get the title of each track followed by an “artist” field. The “artist” field has a “name” that gets the name of the artist of each track. This query allows us to look into the playlist Grunge.

<u>Albums by the artist “Red Hot Chili Peppers.” (REST):</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

home > student > api_calls.py > get_track_info

```python
import requests

BASE_URL = "http://localhost:8000/api"
GRAPHQL_URL = "http://localhost:4000/graphql"

# ================ Rest API Calls =====================
def get_artist_id(artist_name): # Albums by red hot chili peppers
    endpoint = f"/tables/artists/rows?_search={artist_name.replace(' ', '%20')}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()

    if response.status_code == 200:
        return data['data'][0]['ArtistId']
    return response.status_code

artist_id = get_artist_id("Red Hot Chili Peppers")

def get_albums_by_artist(artist_id):
    endpoint = f"/tables/albums/rows?_filters=ArtistId:{artist_id}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()

    album_list = []
    if response.status_code == 200:
        parse = data['data']
        for albums in parse:
            album_list.append(albums['Title'])
        return album_list
    else:
        return response.status_code

print(f"Albums by Red Hot Chili Pepper(REST):")
albums = get_albums_by_artist(artist_id)
for album in albums:
    print(album)
```

Def_get_artist id(artist name) is a function that takes the id based on an artist name. It allows us to query the REST api and it builds a url that then sends a GET request. If the status code is 200, the HTTP request is successful. It then would extract and return the ArtistId.

get_albums_by_artist (artist id) builds a URL endpoint to get the albums associated with the artist. It then sends a get request to the REST API. If the status code is 200, it returns the list of album titles. If the status code is not 200 then it returns the resulting status code.

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

Genres associated with the artist “U2.” (REST)

```python

def get_artist_id(artist_name): # U2 Genres
    endpoint = f"/tables/artists/rows?_search={artist_name.replace(' ', '%20')}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()

    if response.status_code == 200 and data['data']:
        return data['data'][0]['ArtistId']
    else:
        return response.status_code

def get_albums_by_artist(artist_id):
    endpoint = f"/tables/albums/rows?_filters=ArtistId:{artist_id}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()
    if response.status_code == 200:
        return data['data']
    else:
        return response.status_code

def get_tracks_by_album(album_id):
    endpoint = f"/tables/tracks/rows?_filters=AlbumId:{album_id}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()
    if response.status_code == 200:
        return data['data']
    else:
        return response.status_code

def get_genre_by_id(genre_id):
    endpoint = f"/tables/genres/rows?_filters=GenreId:{genre_id}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()
    if response.status_code == 200 and data['data']:
        return data['data'][0]['Name']
    else:
        return response.status_code
```

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```python

def get_genres_by_album(album_id):
    tracks = get_tracks_by_album(album_id)
    genre_ids = set([track['GenreId'] for track in tracks if track['GenreId'] is not None])
    genres = [get_genre_by_id(genre_id) for genre_id in genre_ids]
    return genres

def get_all_genres_for_artist(artist_id):
    albums = get_albums_by_artist(artist_id)
    all_genres = []
    for album in albums:
        album_id = album['AlbumId']
        genres = get_genres_by_album(album_id)
        all_genres.extend(genres)
    return set(all_genres)

artist_name = "U2"
artist_id = get_artist_id(artist_name)
albums = get_albums_by_artist(artist_id)

if artist_id:
    unique_genres = get_all_genres_for_artist(artist_id)
    print(f"\nGenres by {artist_name}(REST):")
    for genre in unique_genres:
        print(genre)
```

get_artist_id(artist_name) function takes in the parameter artist name(U2) string. Using the REST URL get function we are able to search for the artist_id by artist name. The function then checks the response status, if it is 200 then it returns the artist_id. If not then it returns the resulting status code.

get_albums_by_artist function takes in the parameter of artist_id. It then runs a get request, if successful 200 then it returns the albums by U2. if not then it returns the resulting status code.

get_tracks_by_album(album_id) function takes in the parameter of album_id. Then runs a get request, if successful 200 then returns the tracks by U2.

get_genre_by_id(genre_id) takes in the parameter of genre_id. It then runs a get request, if successful 200 then it returns the genre_ids associated with U2. if not then it returns the resulting status code.

get_genres_album(album_id) takes in the parameter of album_id and returns a list of genres associated with U2.

get_all_genres_for_artist(artist_id) takes in the parameter of artist_id and returns a list of genres associated with U2. This function calls the function above in order to retrieve each album of that specific artist.

Once artist_name is U2, it calls the get_artist_id(artist_name) function in order to get the artist id. After artist id is called, get_albums_by_artist function is called to get the list of albums associated with U2. Then we extract the genres of each album and print the associated genres of U2.

<u>Names of tracks on the playlist “Grunge” and their associated artists and albums (REST):</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```python
def get_playlist_id(playlist_name): # tracks on playlist Grunge and artist/album names
    endpoint = f"/tables/playlists/rows?_search={playlist_name}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()

    if response.status_code == 200:
        return data['data'][0]['PlaylistId']
    else:
        return response.status_code

def get_track_ids(playlist_id):
    endpoint = f"/tables/playlist_track/rows?_page=1&_limit=15&_schema=TrackId&_filters=PlaylistId:{playlist_id}"
    response = requests.get(BASE_URL + endpoint)
    data = response.json()

    if response.status_code == 200:
        return data['data']
    else:
        return response.status_code

def get_track_titles(tracks_id):

    track_title_list = []
    for track_info in tracks_id:
        endpoint = f"/tables/tracks/rows?_schema=Name,AlbumId&_filters=TrackId:{track_info['TrackId']}"
        response = requests.get(BASE_URL + endpoint)
        data = response.json()

        if response.status_code == 200:
            track_title_list.append(data['data'])
        else:
            return response.status_code
    return track_title_list
```

```python
def get_track_info(track_title_list):
    result_list = []
    for track_info in track_title_list:
        album_id = track_info[0]['AlbumId']
        endpoint = f"/tables/albums/rows?_page=1&_limit=10&_schema=Title,ArtistId&_extend=ArtistId&_filters=AlbumId:{album_id}"
        response = requests.get(BASE_URL + endpoint)
        data = response.json()

        if response.status_code == 200:
            list_tuples = [(track_info[0]['Name']), (data['data'][0]['Title']), (data['data'][0]['ArtistId_data']['Name'])]
            result_list.append(list_tuples)
        else:
            return response.status_code
    return result_list

playlist_id = get_playlist_id("Grunge")
tracks_id = get_track_ids(playlist_id)
track_info = get_track_titles(tracks_id)
track_info = get_track_info(track_info)

print('\nNames of tracks on the playlist “Grunge” and their associated artists and albums (REST):')

for tracks in track_info:
    print(f"Track Name: {tracks[0]} Album Name: {tracks[1]} Artist Name: {tracks[2]}")
```

get_playlist_id(playlist_name) function takes in a string parameter playlist_name. It runs a get request which searches playlists for a playlist with the title “Grunge”. If successful 200 it returns the playlist_id. If not then it returns the resulting status code.

get_track_ids(playlist_id) function takes in a parameter of playlist_id. It runs a get request which searches playlist_track. If successful 200 it returns the TrackId's for all the tracks on the playlist Grunge. If not then it returns the resulting status code.

get_track_titles(tracks_id) function takes in a parameter of tracks_id which is a list of ids. It then runs a loop which loops through the list of track_id's. Each iteration of the loop it runs a get request which searches tracks. If successful 200 it adds the Name and AlbumId of the corresponding track_id to a list. If not successful it then returns the resulting status code. Once the loop is finished the function returns the list.

get_track_info(track_title_list) function takes in a list parameter. It then runs a loop which loops through the list. Every iteration of the loop it runs a get request which takes the AlbumId parameter of that current list index and searches the album table which is also extended to the artist table. If successful 200 then it makes a tuple list of Track Name, Album Title, and Artist Name which is appended to a result list. Once the loop is done it returns the result list. If not successful then it returns the resulting status code.

We are left with a resulting list which we can loop through and print out the track names, album names, and artist names of all the tracks in the Grunge playlist.

<u>Albums by the artist "Red Hot Chili Peppers." (GraphQL):</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```python
# ============ GraphQL Queries ================
def get_albums_graphql(artist_name): # albums by red  hot chili peppers
    query = f"""
    {{
      artists(where: {{name: "{artist_name}"}}) {{
        albums {{
          title
        }}
      }}
    }}
    """
    response = requests.post(GRAPHQL_URL, json={'query': query})
    if response.status_code == 200:
        data = response.json()
        return [album['title'] for album in data['data']['artists'][0]['albums']]
    return response.status_code

print(f"\nAlbums by Red Hot Chili Pepper(GraphQL):")

albums_graphql = get_albums_graphql("Red Hot Chili Peppers")
print('\n'.join(albums_graphql))
```

get_albums_graphql(artist_name), allows you to get album titles from a particular artist name. It contains a query and sends a Post request to Graphql endpoint. If the status code is 200, it returns the album's titles. Otherwise it returns the resulting status code. Then prints the results to the console.

<u>Genres associated with the artist “U2.” (GraphQL)</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```python
def get_genres_for_artist(artist_name): # Genres associated with U2
    query = f"""
    query {{
        artist(where: {{name: "{artist_name}"}}) {{
            albums{{
                tracks {{
                    genre {{
                        name
                    }}
                }}
            }}
        }}
    }}
    """
    response = requests.post(GRAPHQL_URL, json={'query': query})
    if response.status_code == 200:
        data = response.json()
        genres = []
        for album in data['data']['artist']['albums']:
            for track in album['tracks']:
                if track['genre']['name'] not in genres:
                    genres.append(track['genre']['name'])
        return genres
    return response.status_code

print("\nGenres associated with U2(GraphQL):")

genres = get_genres_for_artist("U2")
print('\n'.join(genres))
```

get_genres_for_artist(artist name) function retrieves genres associated with U2, it contains a query that sends a Post request to graphql api endpoint. If the request is successful 200 it gets the information of all the genres associated with U2. It then prints the results to the console.

<u>Names of tracks on the playlist “Grunge” and their associated artists and albums (GraphQL):</u>

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

```python
def get_tracks_on_playlist(playlist_name): #Names of tracks on the playlist Grunge and their associa
    query = f"""
    query {{
        playlist(where: {{name: "{playlist_name}"}}) {{
            tracks{{
                name
                album {{
                    title
                    artist {{
                        name
                    }}
                }}
            }}
        }}
    }}
    """
    response = requests.post(GRAPHQL_URL, json={'query': query})

    if response.status_code == 200:
        data = response.json()
        tracks = []
        for track in data['data']['playlist']['tracks']:
            track_name = track['name']
            album_title = track['album']['title']
            artist_name = track['album']['artist']['name']
            tracks.append((track_name, album_title, artist_name))
        return tracks
    return response.status_code

tracks_info = get_tracks_on_playlist("Grunge")

print("\nNames of tracks on the playlist Grunge and their associated artist and albums(GraphQL):")

for track_name, album_title, artist_name in tracks_info:
    print(f"Track: {track_name}, Album: {album_title}, Artist: {artist_name}")
```

Get track on playlist(playlist name) function has a query to get information in a playlist called Grunge. This query would involve all the tracks associated with the playlist Grunge. It then sends a Post request and if the status code is 200, it gets information of all the tracks in the playlist. It then gets the track name, album title, and artist name of each track with a track list of the associated tracks in the playlist. If the status code is not 200 then it returns the resulting status code. Then prints the results to the console.

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

Output for API calls for REST and GraphQL:

```
(.venv) student@tuffix-vm:~$ /bin/python3 /home/student/api_calls.py
Albums by Red Hot Chili Pepper(REST):
Blood Sugar Sex Magik
By The Way
Californication

Genres by U2(REST):
Pop
Rock

Names of tracks on the playlist “Grunge” and their associated artists and albums (REST):
Track Name: Man In The Box Album Name: Facelift Artist Name: Alice In Chains
Track Name: Smells Like Teen Spirit Album Name: Nevermind Artist Name: Nirvana
Track Name: In Bloom Album Name: Nevermind Artist Name: Nirvana
Track Name: Come As You Are Album Name: Nevermind Artist Name: Nirvana
Track Name: Lithium Album Name: Nevermind Artist Name: Nirvana
Track Name: Drain You Album Name: Nevermind Artist Name: Nirvana
Track Name: On A Plain Album Name: Nevermind Artist Name: Nirvana
Track Name: Evenflow Album Name: Ten Artist Name: Pearl Jam
Track Name: Alive Album Name: Ten Artist Name: Pearl Jam
Track Name: Jeremy Album Name: Ten Artist Name: Pearl Jam
Track Name: Daughter Album Name: Vs. Artist Name: Pearl Jam
Track Name: Outshined Album Name: A-Sides Artist Name: Soundgarden
Track Name: Black Hole Sun Album Name: A-Sides Artist Name: Soundgarden
Track Name: Plush Album Name: Core Artist Name: Stone Temple Pilots
Track Name: Hunger Strike Album Name: Temple of the Dog Artist Name: Temple of the Dog

Albums by Red Hot Chili Pepper(GraphQL):
Blood Sugar Sex Magik
By The Way
Californication

Genres associated with U2(GraphQL):
Rock
Pop
```

```
Genres associated with U2(GraphQL):
Rock
Pop

Names of tracks on the playlist Grunge and their associated artist and albums(GraphQL):
Track: Man In The Box, Album: Facelift, Artist: Alice In Chains
Track: Smells Like Teen Spirit, Album: Nevermind, Artist: Nirvana
Track: In Bloom, Album: Nevermind, Artist: Nirvana
Track: Come As You Are, Album: Nevermind, Artist: Nirvana
Track: Lithium, Album: Nevermind, Artist: Nirvana
Track: Drain You, Album: Nevermind, Artist: Nirvana
Track: On A Plain, Album: Nevermind, Artist: Nirvana
Track: Evenflow, Album: Ten, Artist: Pearl Jam
Track: Alive, Album: Ten, Artist: Pearl Jam
Track: Jeremy, Album: Ten, Artist: Pearl Jam
Track: Daughter, Album: Vs., Artist: Pearl Jam
Track: Outshined, Album: A-Sides, Artist: Soundgarden
Track: Black Hole Sun, Album: A-Sides, Artist: Soundgarden
Track: Plush, Album: Core, Artist: Stone Temple Pilots
Track: Hunger Strike, Album: Temple of the Dog, Artist: Temple of the Dog
(.venv) student@tuffix-vm:~$
```

Issues and Troubleshooting

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

*** We didn't know how to build a valid URL so we did research and found this website https://developers.google.com/maps/url-encoding. This website allowed us to discover the %20 encoded value to help build our URLs. Status Code 200 means everything is okay and the HTTP response status was successful. This was discovered in the following website https://www.dataquest.io/blog/python-api-tutorial/, it gave us a better understanding on how APIs work. The following video https://www.youtube.com/watch?v=PTfZcN20fro, shows the difference and the advantages between Graphql and Rest API. One thing I found interesting is that in Graphql one endpoint is required for all my queries while in Rest Api, there needs to be more than one endpoint. As a result, using graphql would make my life easier because it would take away the necessity of documenting numerous URLs if I was to use Rest Api. Once learning “Swagger” it helped a ton with testing and building our URL endpoints. I used Swagger and my groupmate chose to use Postman. Another issue that we ran into with the REST API, we noticed on some of our calls that we were missing data. From further research we discovered that this was a pagination issue, that the calls were defaulting to a limit of 10 entries per page. We were able to adjust the limits and get all the data we were requesting. Going into this exercise we were both fairly new to Python. This exercise definitely got us both more comfortable with Python.

References

Juan Uriarte
Fall 2023
Course Number: CPSC 449-01 13661
Exercise 1 Rest and GraphQL

https://www.dataquest.io/blog/python-api-tutorial/

https://developers.google.com/maps/url-encoding

https://www.youtube.com/watch?v=PTfZcN20fro

https://help.smartling.com/hc/en-us/articles/1260805176869-Pagination