**CPSC449**

**Project 2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Team Members:**

**Bony Roy ( Dev 1- owns the Sharding the tracks database)**

**Suramya Singh ( Dev 2- Creating XSPF playlists)**

**Brandon Tomich ( OPS)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

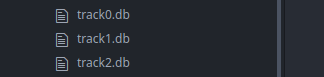
**Bony Roy: (Developer 1)**

I have worked on sharding of tracks database using GUID. Separating tracks from other 3 microservices(user, description, playlist). Also, change other part of the codes in other microservices as required. GUID is the key for Tracks database tracks table. 3 database created-

Database0= track0.db

Database1= track1.db

Database2= track2.db



Data is inserted into Tracks table in separate databases using json script. Function written to generate database number based on GUID.

**Commands to run my part of code: (please makesure .env file is present. Sometime it is removed when uploading)**

export FLASK\_APP=Tracks.py APP\_CONFIG=api.cfg

flask init

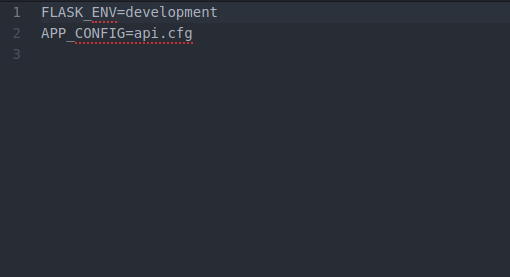
flask popdb

export FLASK\_APP=user.py APP\_CONFIG=api.cfg

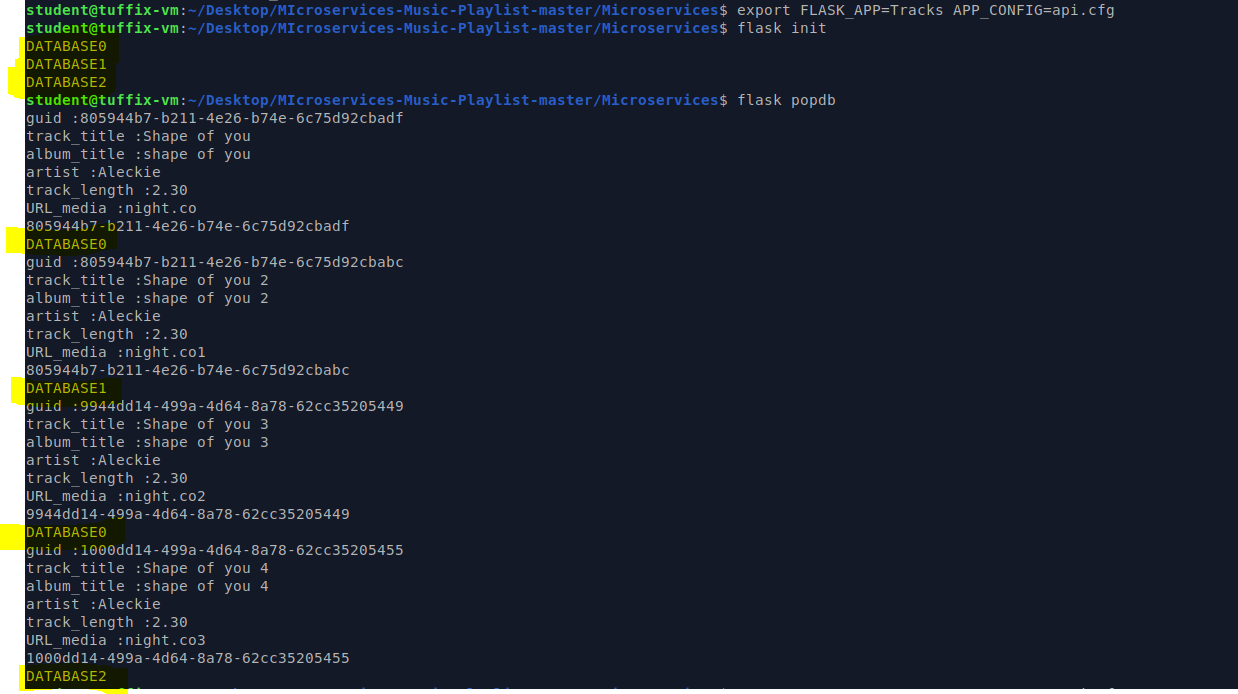
flask init

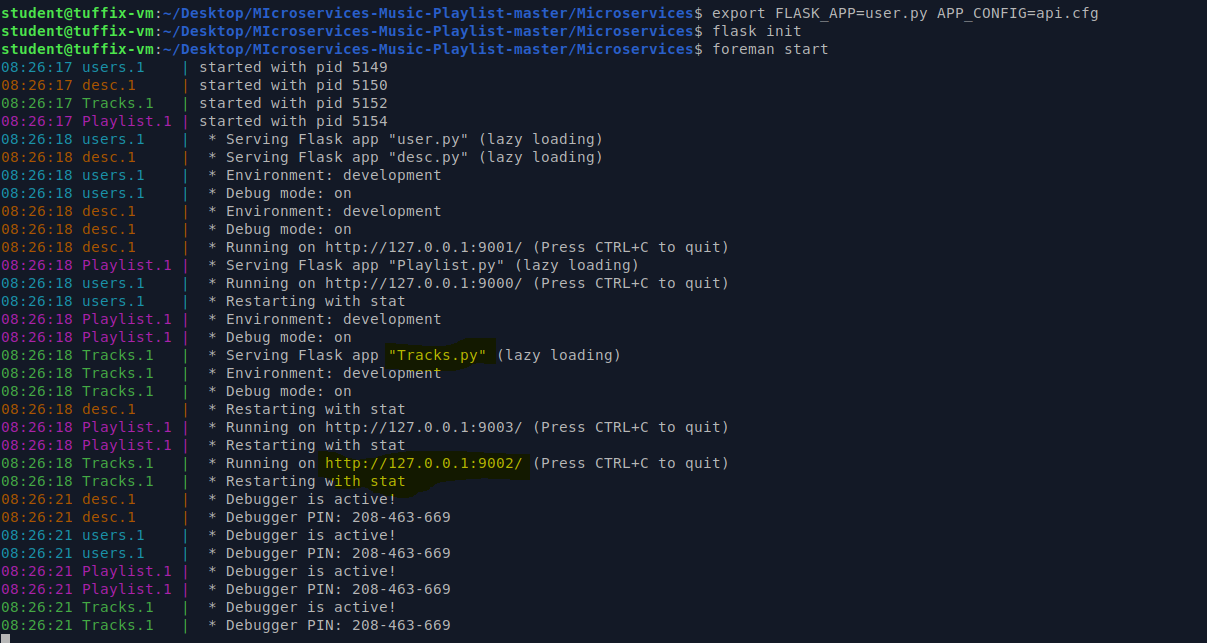
foreman start

**.end File if deleted from folder: (Not needed if you provide** APP\_CONFIG=api.cfg while running but best practice is to provide**)**

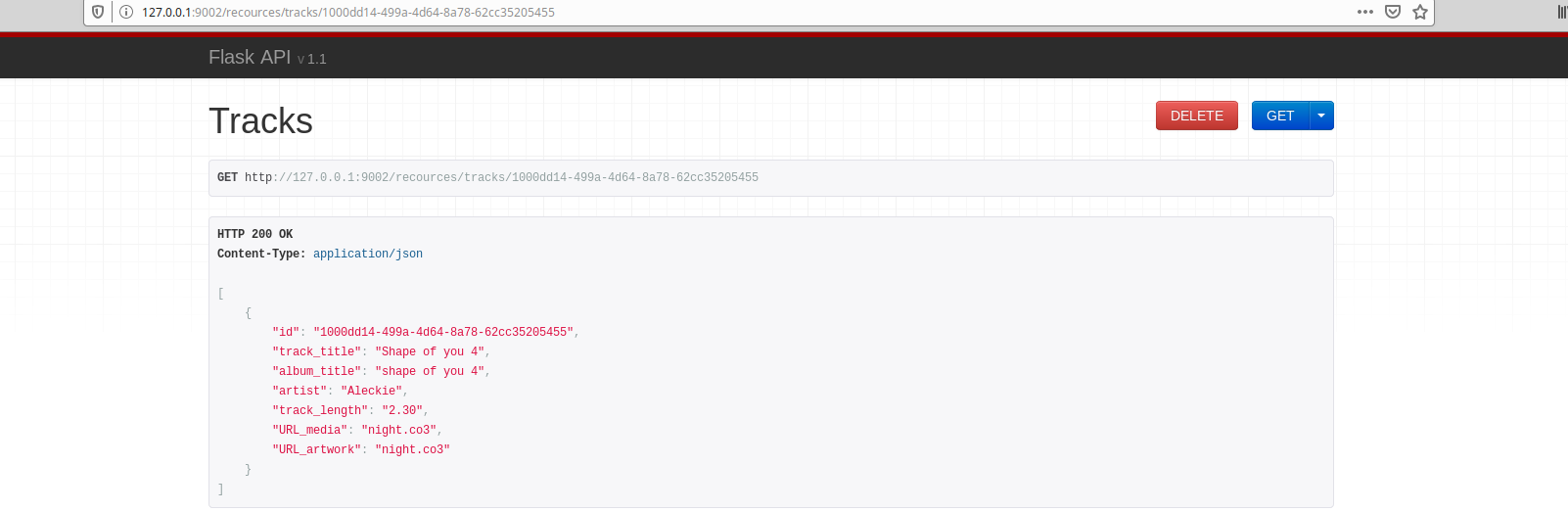


**Screenshots of sharded databases created.**





**Now copy any GUID created and check in API:**



**Similarly we can delete any Tracks or Add any new track. You do not need to provide any id GUID will be created.**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END OF DEV 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

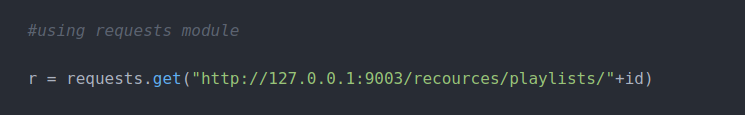
**Suramya Singh: (Developer 2)**

I created a new microservice that will generate XML playlist based on the Playlist ID. This service will be placed in front of the API gateway and will contact the other microservices through the API gateway as needed.

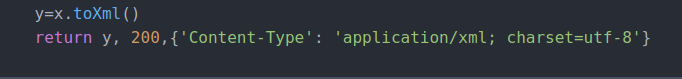
**For Requesting data from microservices:**

Install the requests module: **$ pip3 install --user requests**

* Used requests module in xspf file to fetch data from other microservices!



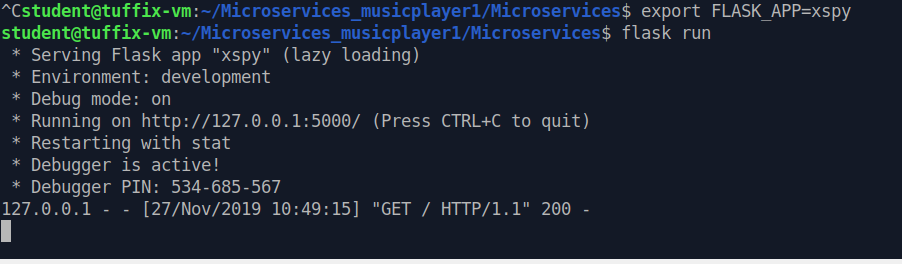
**Generating XSPF: (In XML format)**



* To Run the XSPF service: (Open a separate terminal) and run the below commands:

**export FLASK\_APP=xspy**

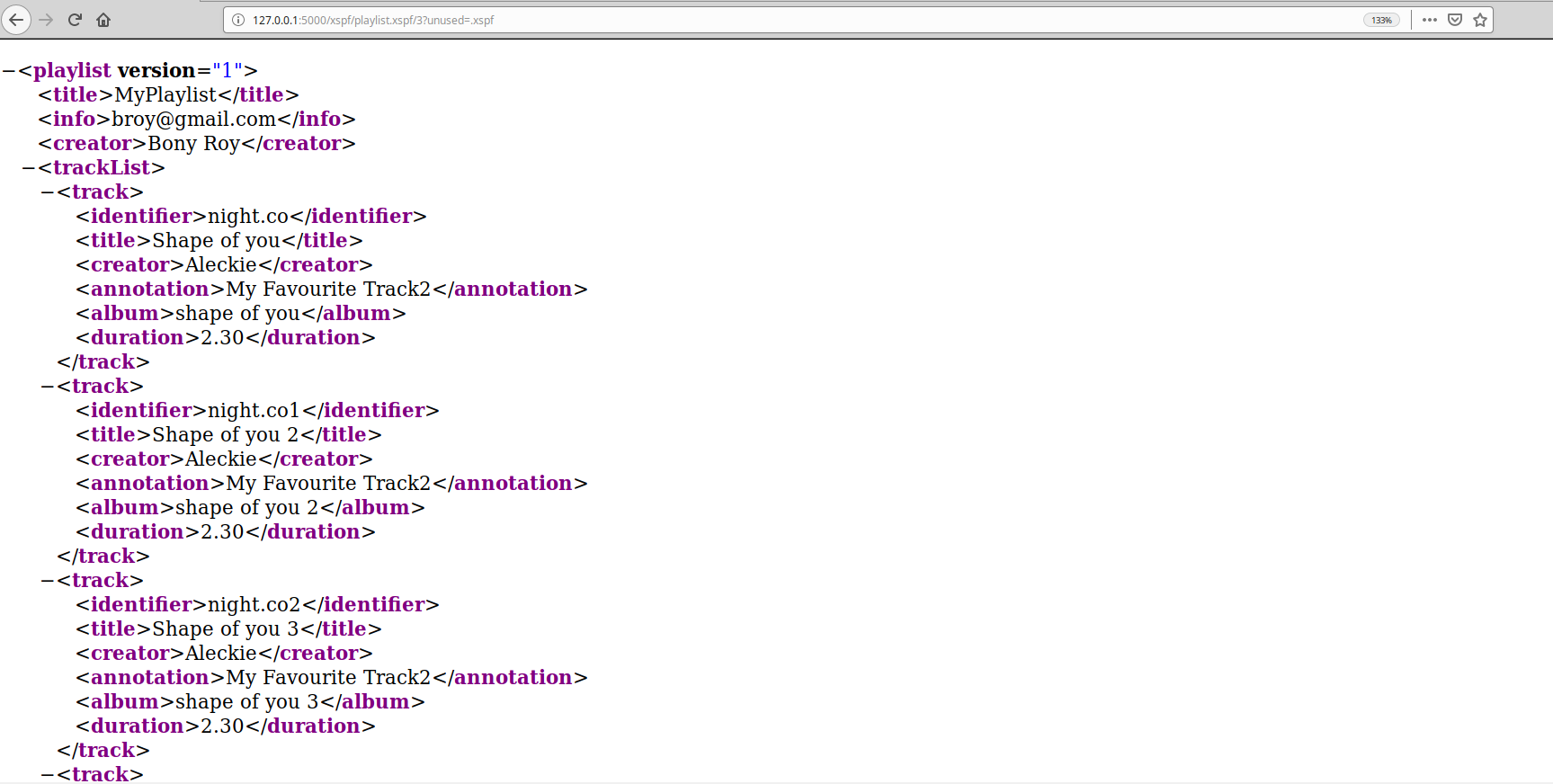
**flask run**



Please add playlist id out of the existing id's (1,2,3,4,5):

[http://127.0.0.1:5000/xspf/playlist.xspf/PLAYLIST\_ID?unused=.xspf](http://127.0.0.1:5000/xspf/playlist.xspf/PLAYLIST_ID?unused=.xspf%20) in order to get the details from user, description, Tracks, Playlist microservices

*The XML data(which is returned on the web page) should look like the below screenshot: (playlist ID=3)*



#### Testing playlists: I worked with my other teammate in the Operations role, to upload the media files into MinIO. I modified a chunk of my service, so that instead of directly using the other microservices, it’s connecting to Kong. For media files, I used [http://localhost:9000/media/](http://localhost:9000/media/%20) which would directly connect to MinIO. So that MinIO opens the original music files in VLC media using <http://127.0.0.1:5000/xspf/playlist.xspf/3?unused=.xspf>

#### 

#### Similarly, I made a few changes in the user.sql file, and included the media file name. Also, for MinIO, I updated a part of my xspf code to:

#### 

#### \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END OF DEV 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*