**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)**

**Note: After reading dev 1 and dev 2 only run the OPS section . Dev1 and Dev 2 sections are only here to document their work**

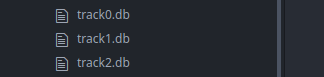
**Bony Roy: (Developer 1)**

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

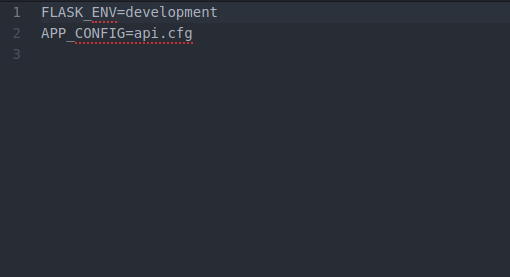
flask popdb

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

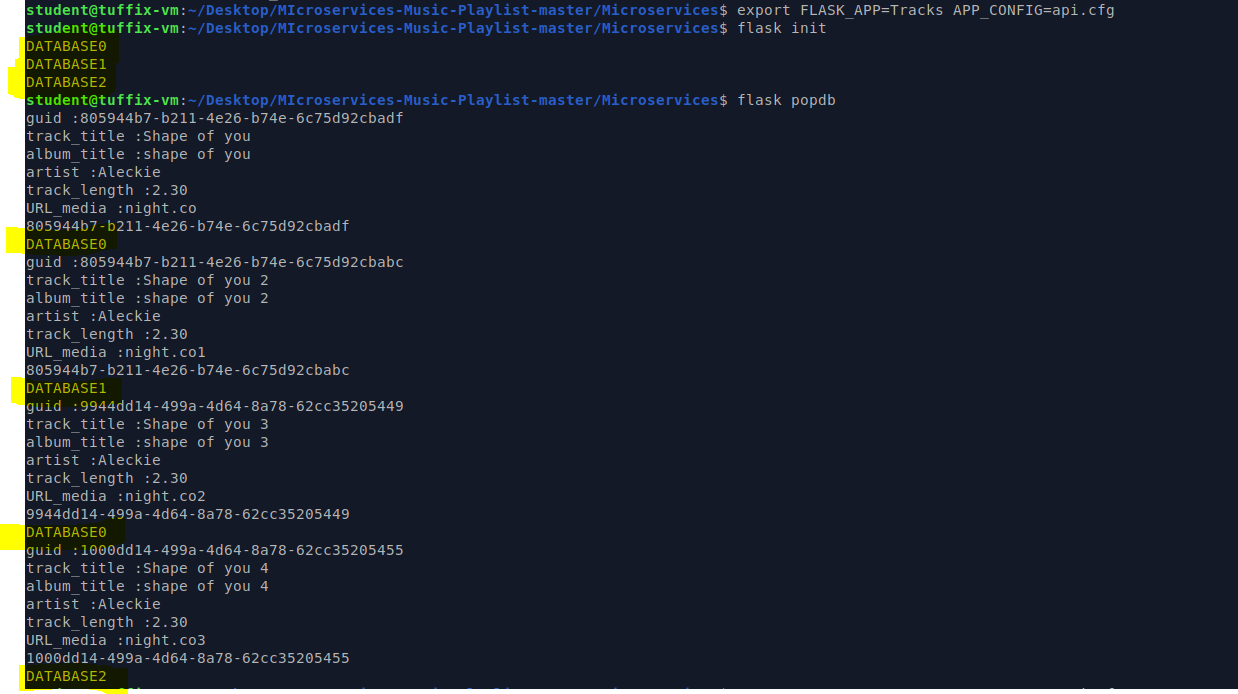
flask init

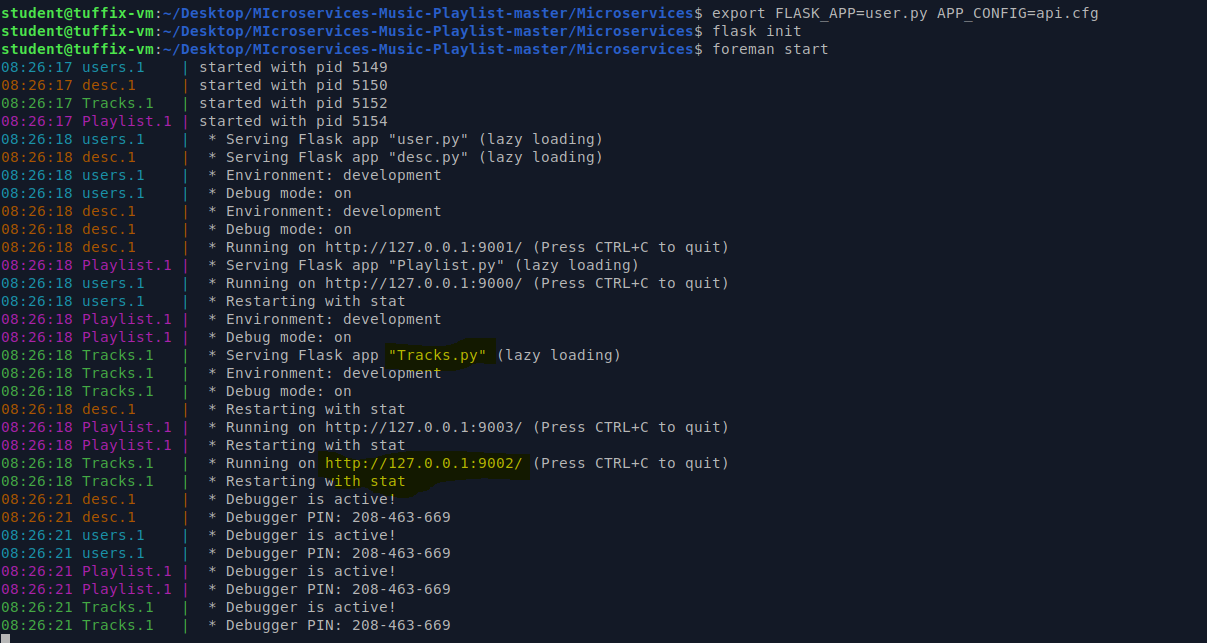
foreman start

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

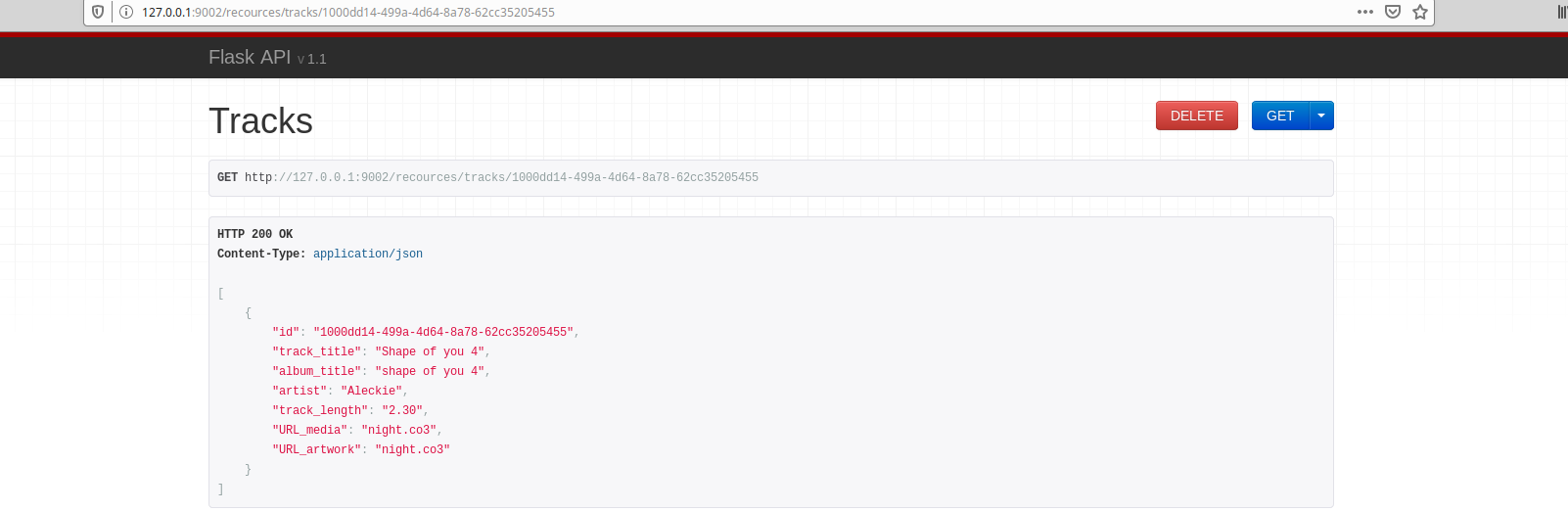


**Screenshots of shared databases created.**

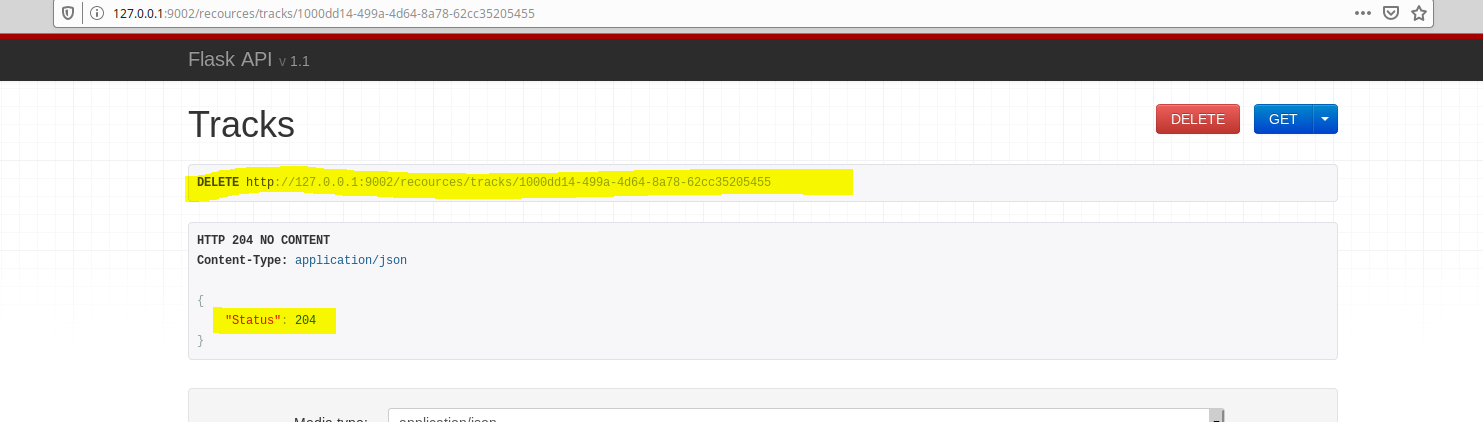




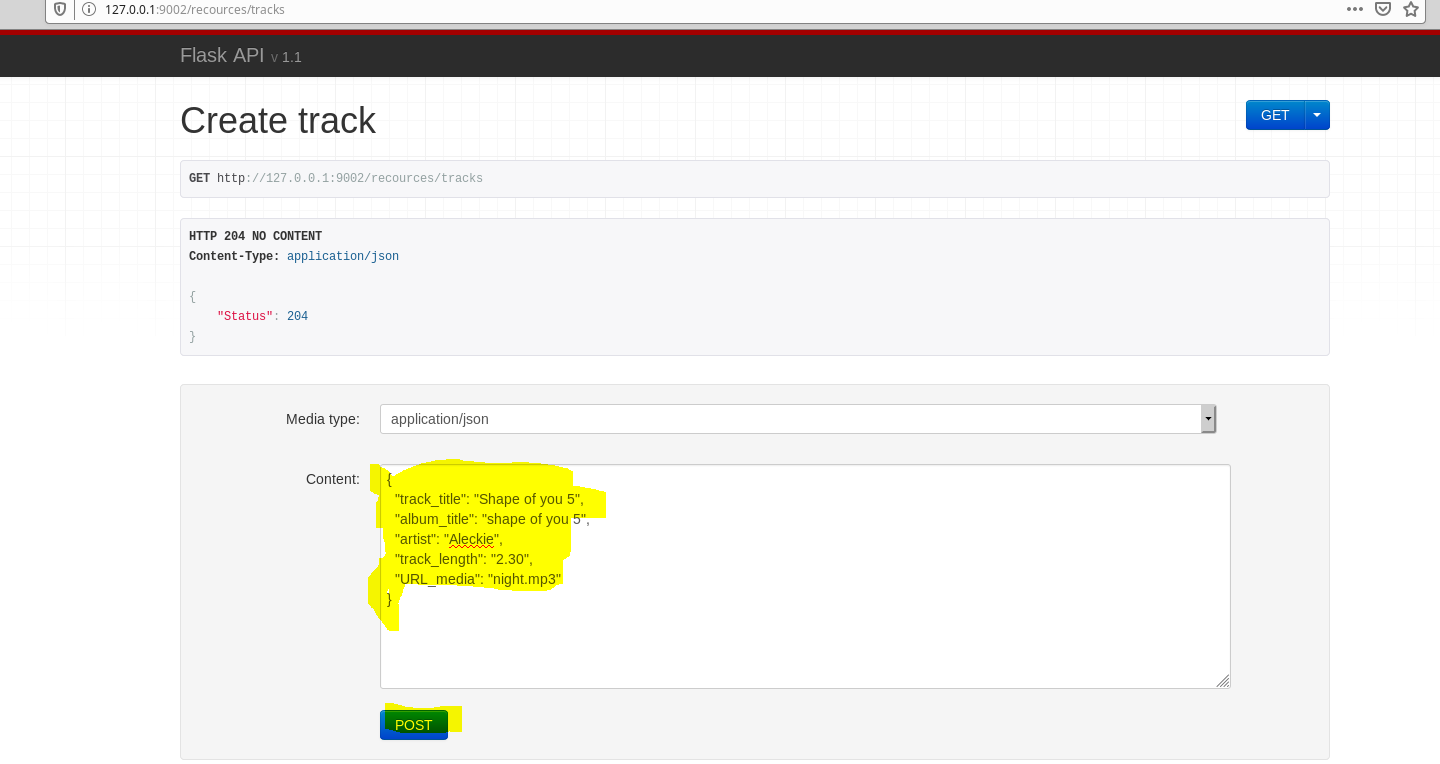
**Now copy any GUID created and check in API:**

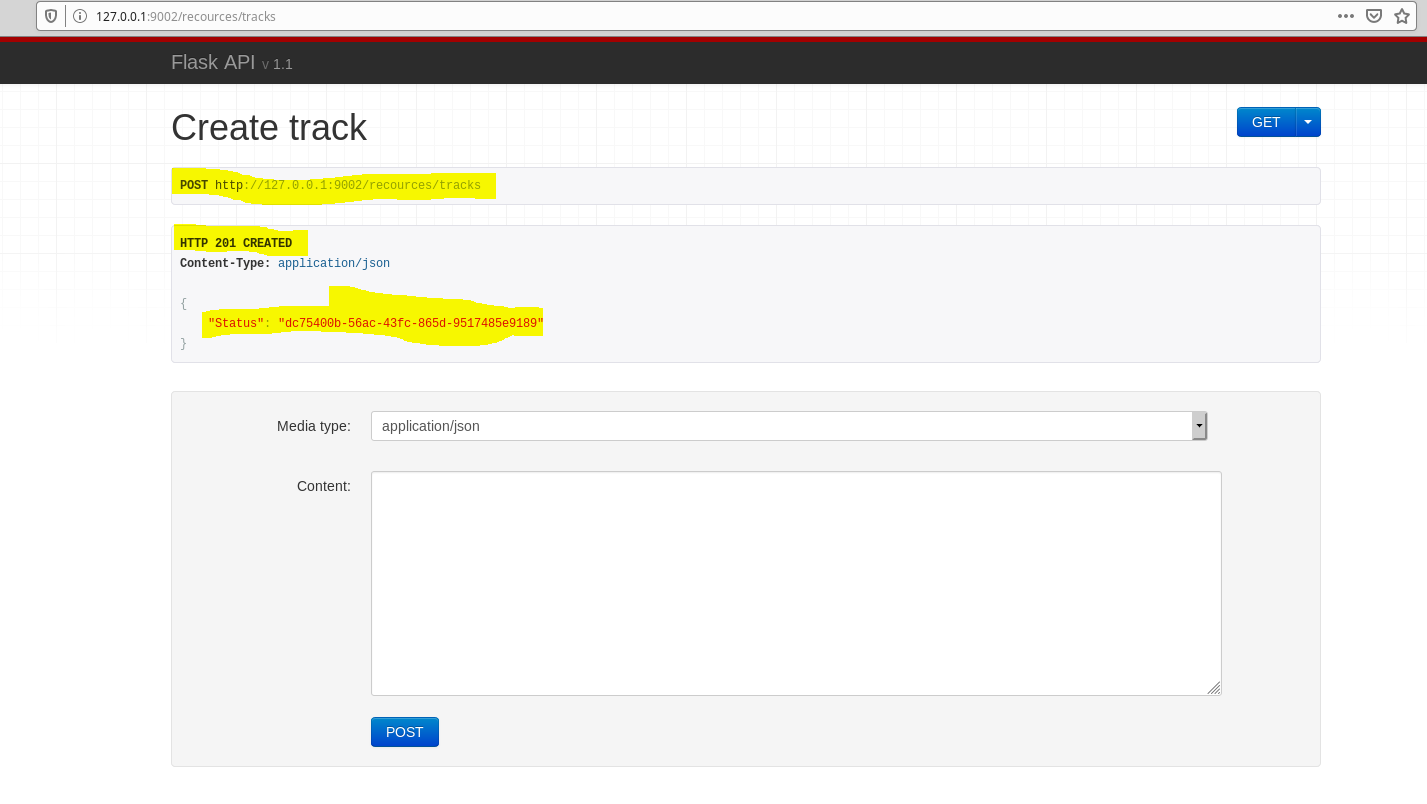


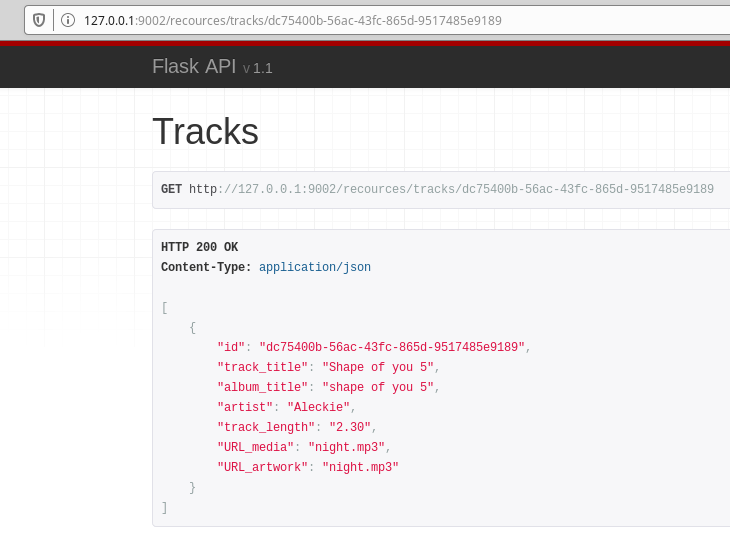
Deleting Track:



Adding Tracks:







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

For overall testing please look into OPS Part added by my team member.

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

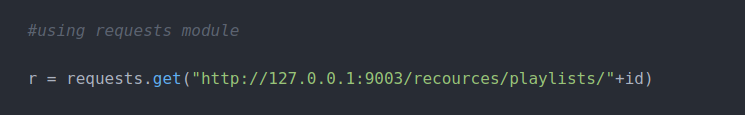
**Suramya Singh: (Developer 2)**

I created a new microservice(xspy.py) 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.

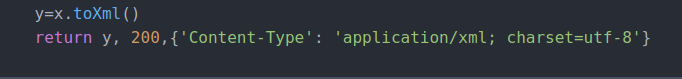
* **For Requesting data from microservices:**

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

* Used requests module in xspy.py file to fetch data from the other microservices!



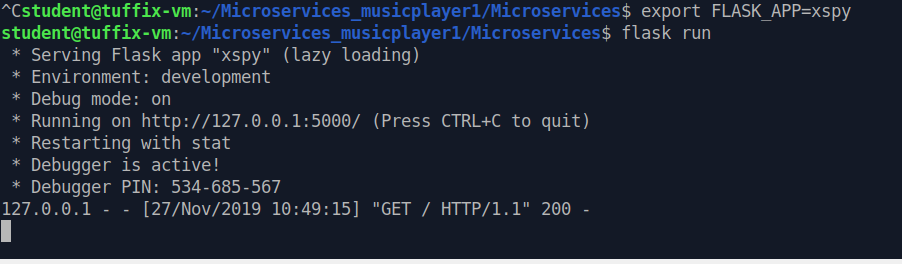
* **Generating XSPF: (In XML format)**



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

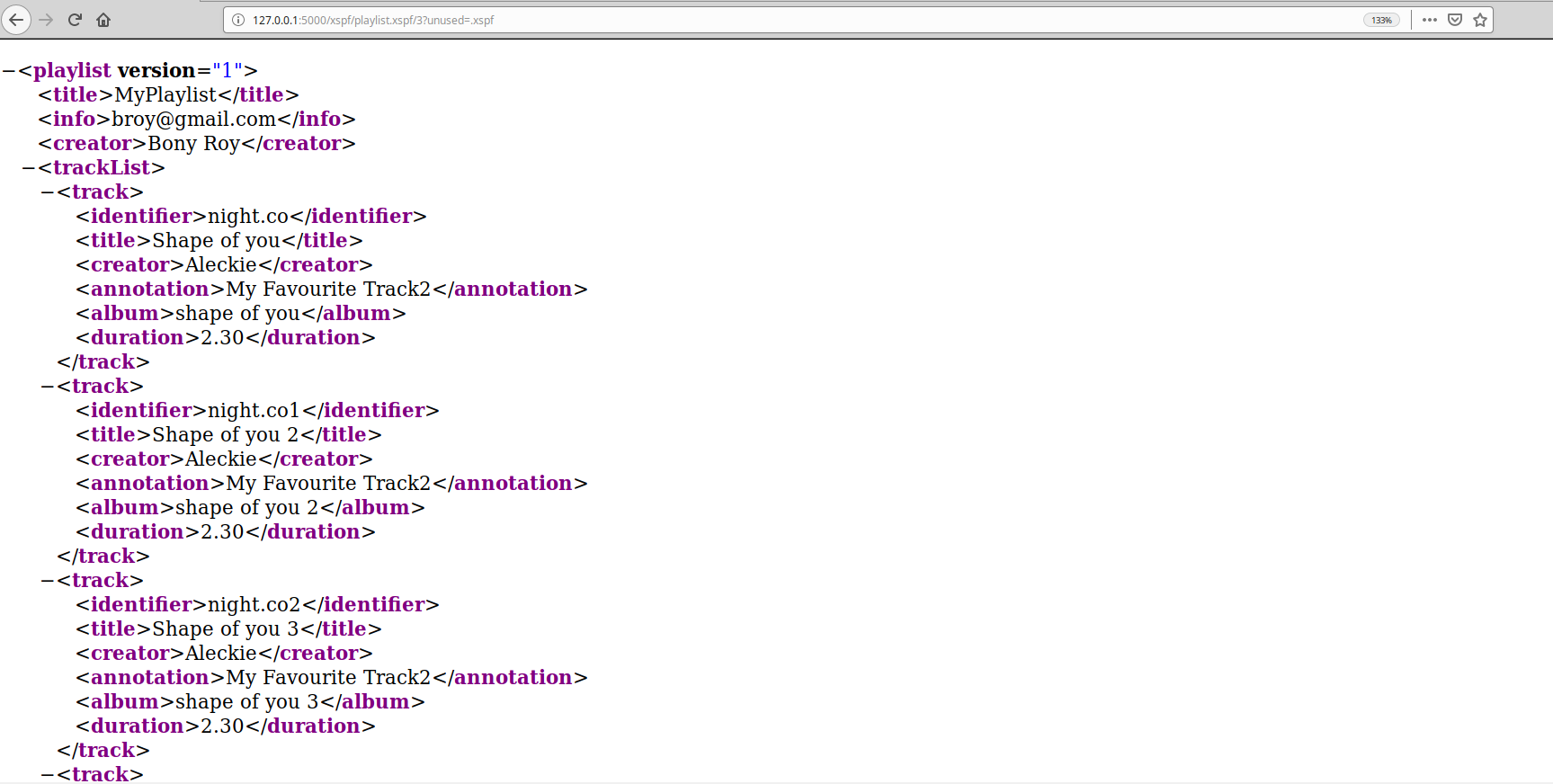
**export FLASK\_APP=xspy**

**flask run**



<http://localhost:6001/xspf/playlist.xspf/PLAYLIST_ID?unused=.xspf> 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)*



#### If the ID does not exist in the database (test case), below is the screenshot

#### 

#### 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://127.0.0.1:6001/xspf/playlist.xspf/1?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:

#### 

#### 

#### 

#### (print(x.toXml()))

#### After testing playlists, below is the final XSPF file:

#### 

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

**Brandon Tomich: (DevOPs)**

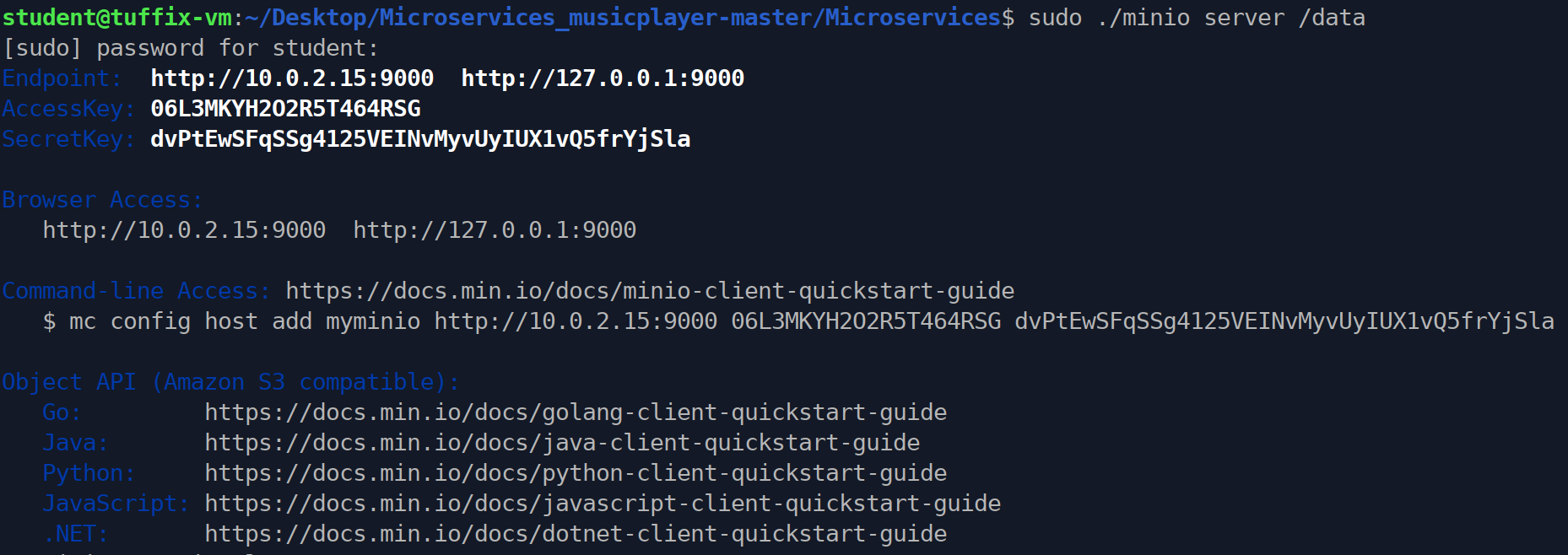
To put everything together we need to first start kong.

**The command to start Kong is ulimit -n 4096 && sudo kong start**

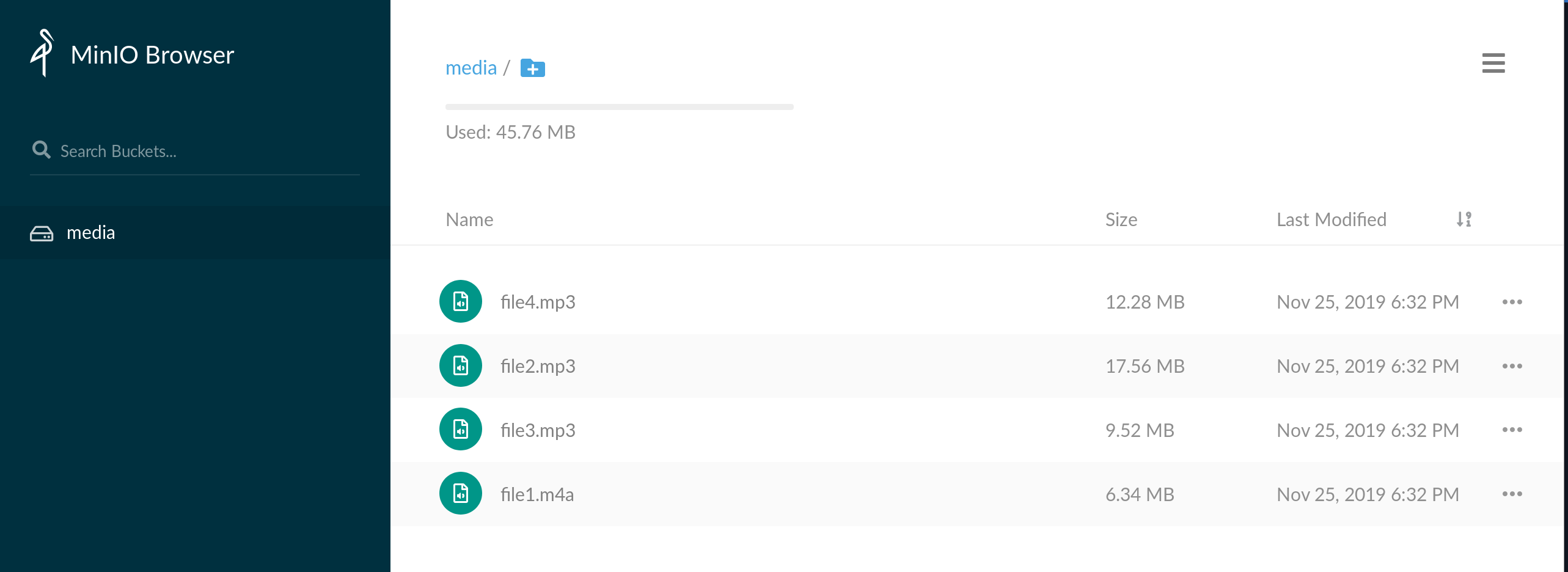
**Once Kong has started please run the Curl commands I saved in the file labeled “CurlCommands.txt”. Note that for each Curl command I left some comments prefixed by “//” to let you know what each command means. I also left a line of hyphens to help with readability between each section of upstreams that I defined for each ring balancer.**

**Next we need to start up minIO. (music is in a folder named “minIO\_music”)**

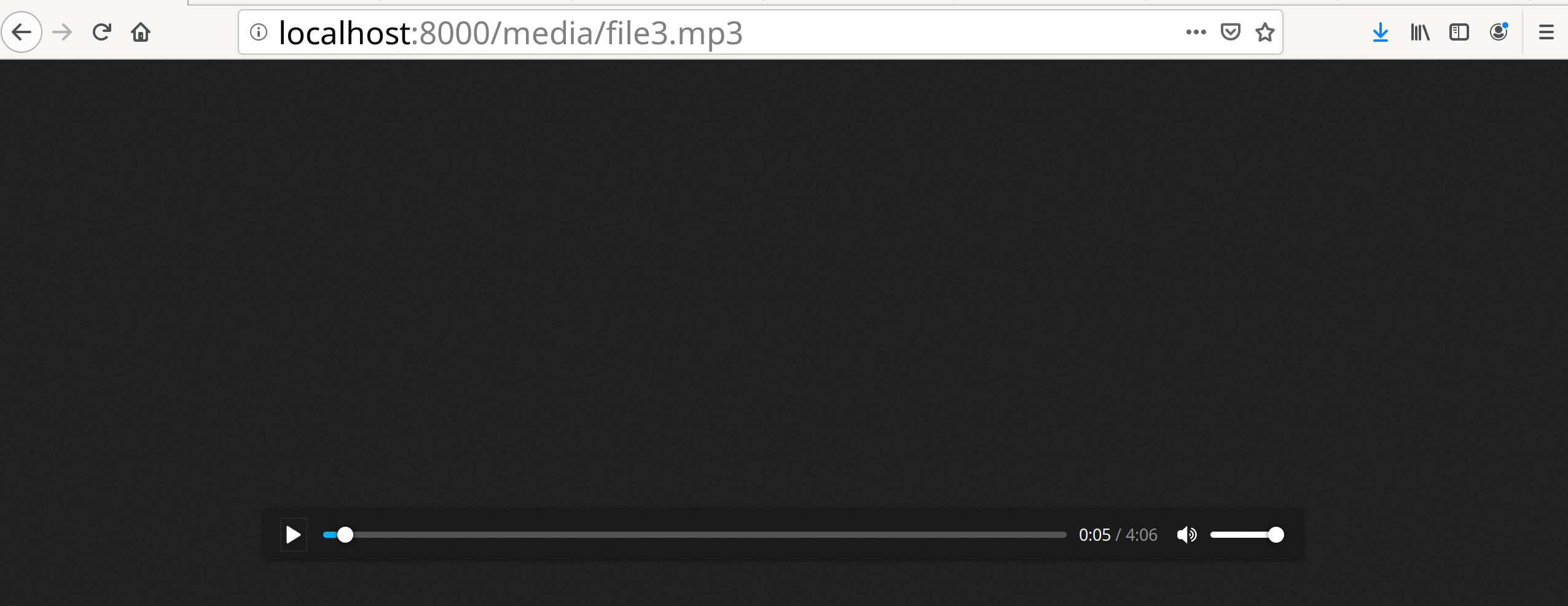
**Type the command “Sudo ./minio server/data” into a terminal.**



**After adding music into the minIO bucket the contents are displayed below.**



***Since Kong is configured to run minIO we can test it as seen in the image below.***



***Next (in a second terminal) we need to run our commands to start up all off our respective services other than minIO.***

**Note that this will be similar as what dev 1 did but will now reflect running three instances of each service rather than one.**

*Type the commands highlighted below*

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

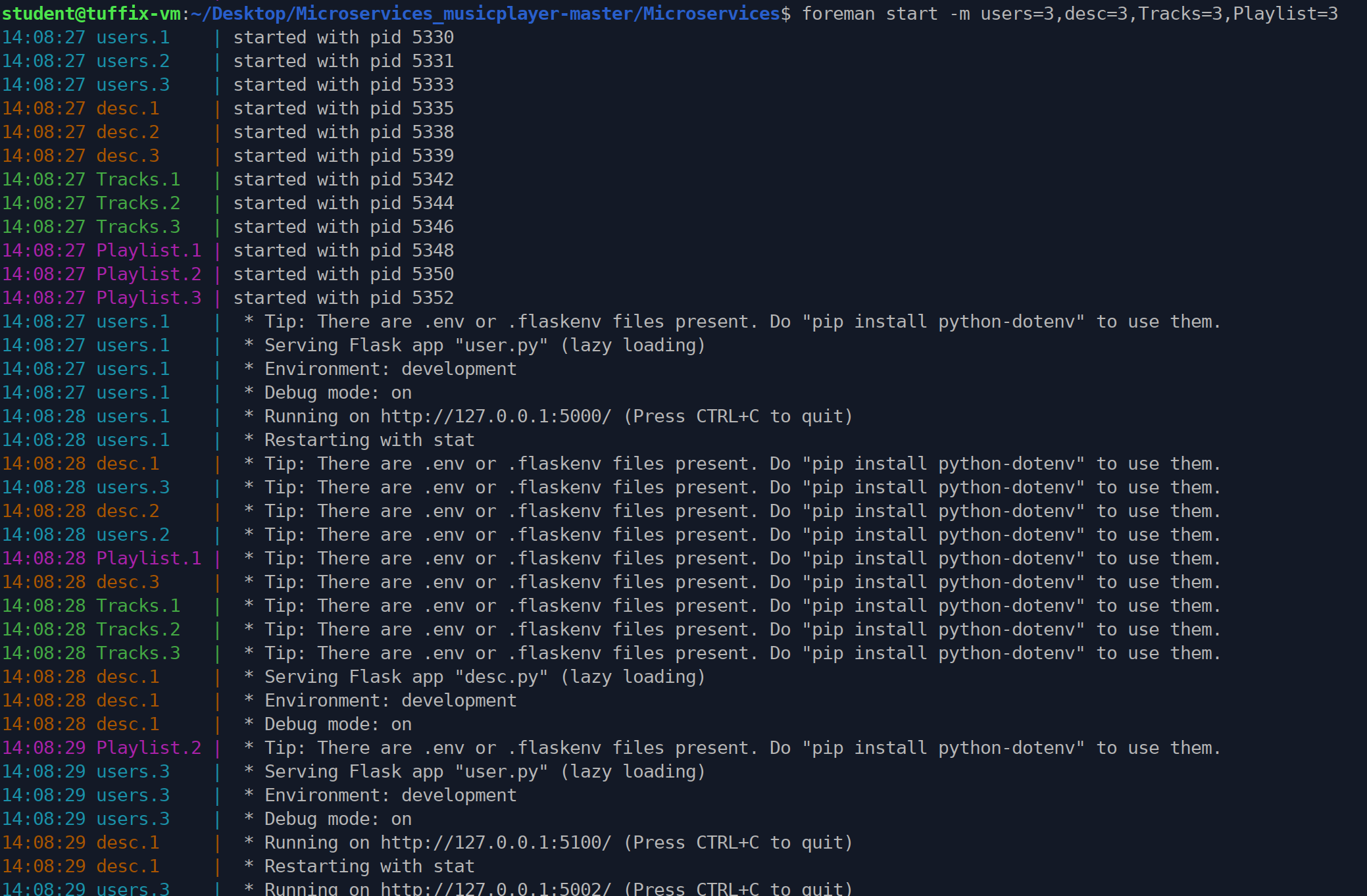
flask init

flask popdb

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

flask init

foreman start -m users=3,desc=3,Tracks=3,Playlist=3

Below is a snippet of the code after running the last line of commands in the terminal. Note that I have cut off most of it because it would span many pages if I put all of the contents. 

Each instance of the following services corresponds to these ports.

User.1 = port 5000

User.2 = port 5001

User.3 = port 5002

Desc.1 = port 5100

Desc.2 = port 5101

Desc.3 = port 5102

Tracks.1 = port 5200

Tracks.2 = port 5201

Tracks.3 = port 5202

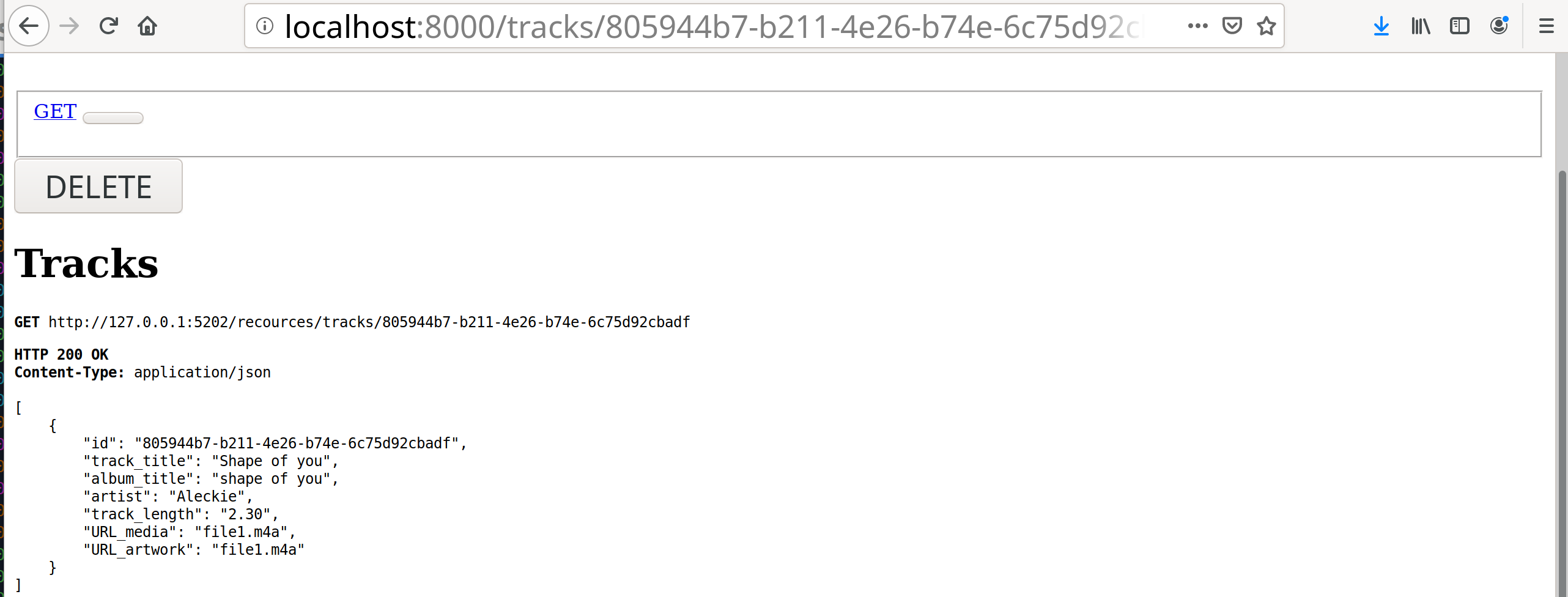
Playlist.1 = port 5300

Playlist.2 = port 5301

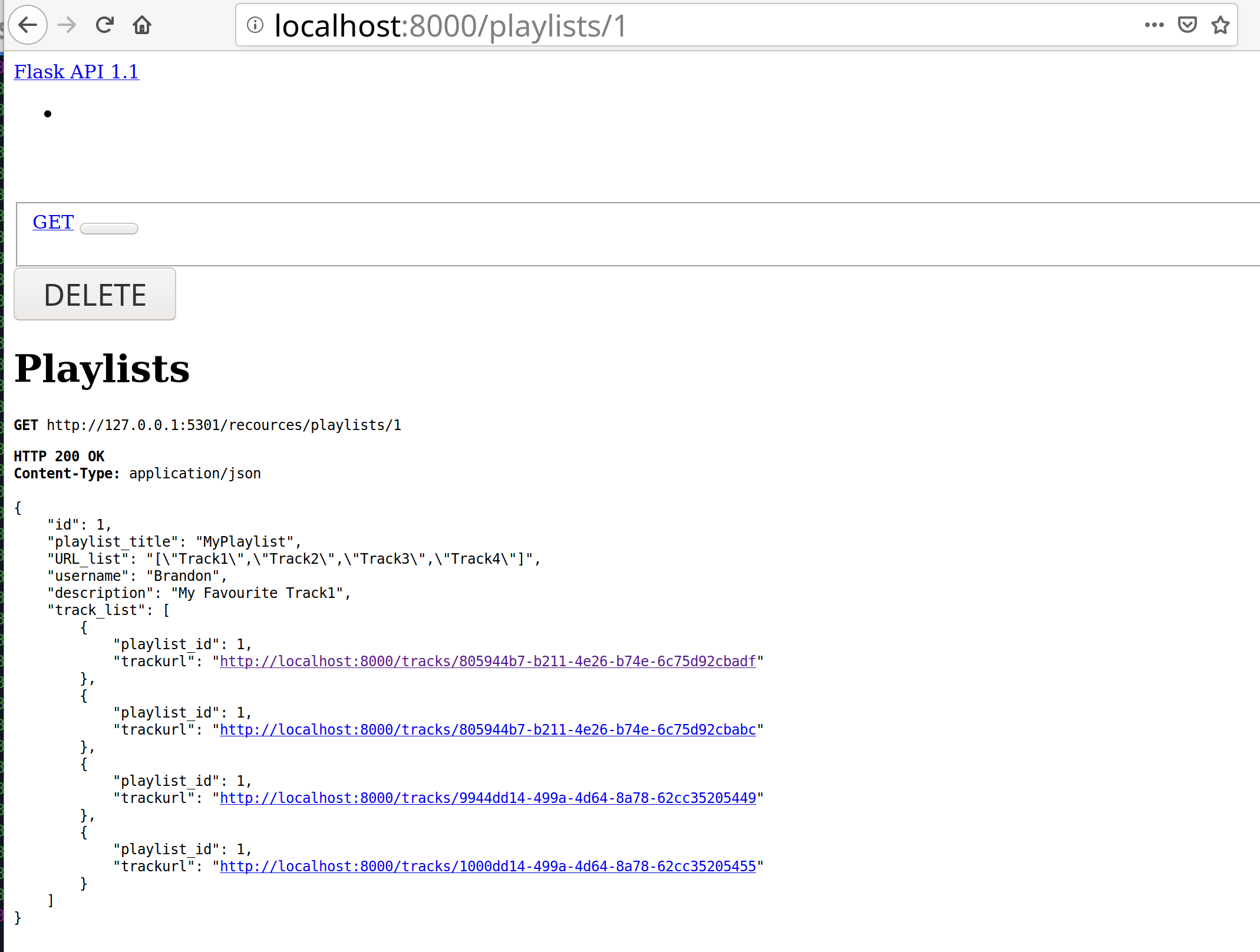
Playlist.3 = port 5302

**Here are some examples of Running Kong for each service**

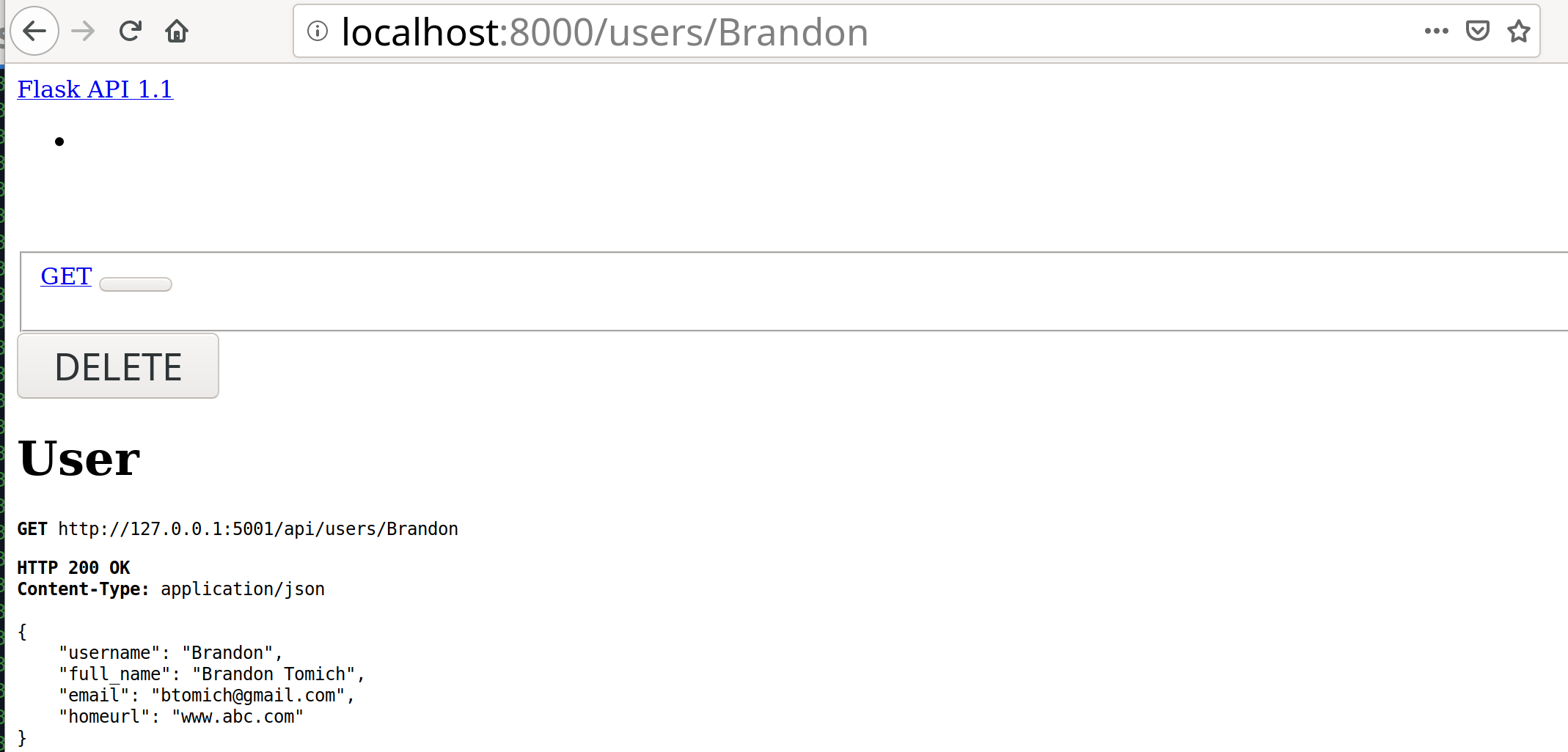
***Testing Kong through Tracks service***



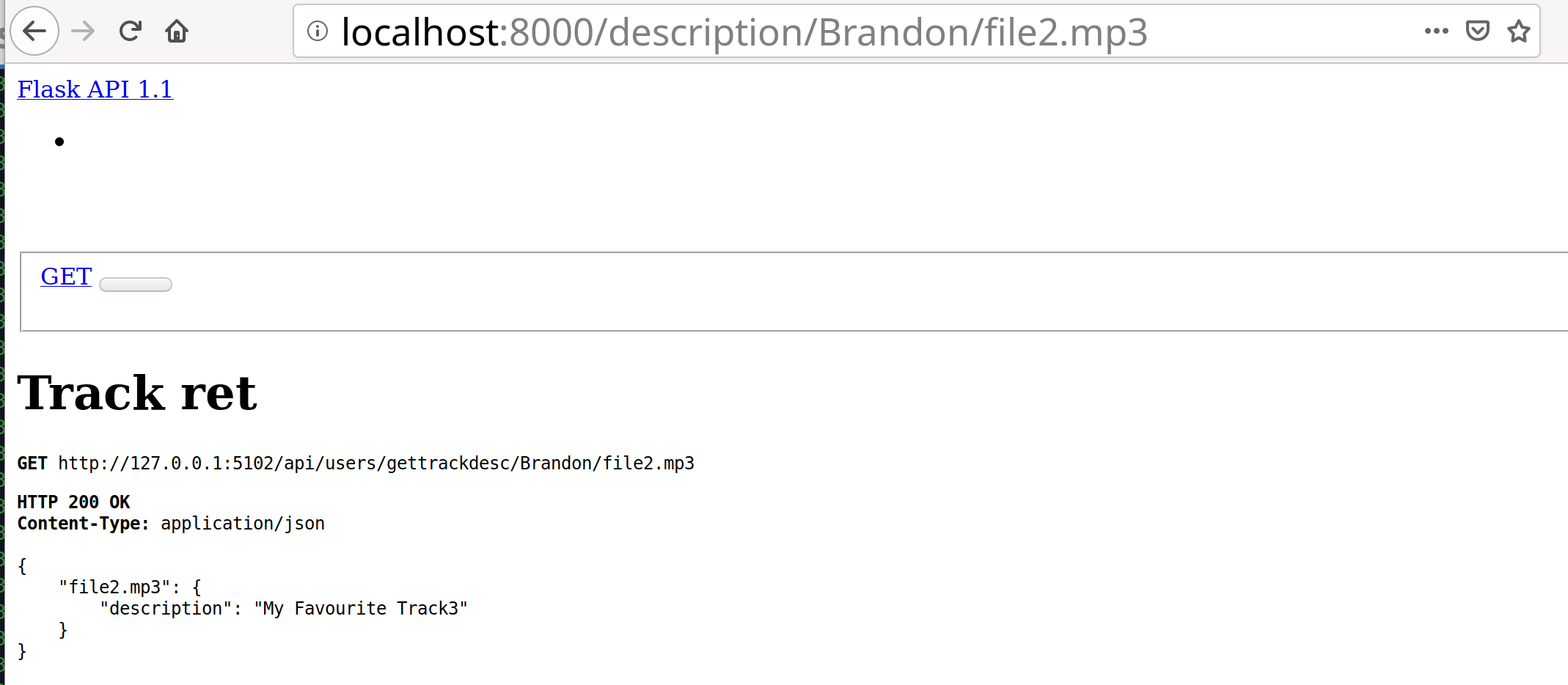
***Testing Kong through Playlists service***



***Testing Kong through users service***



***Testing Kong through desc service***



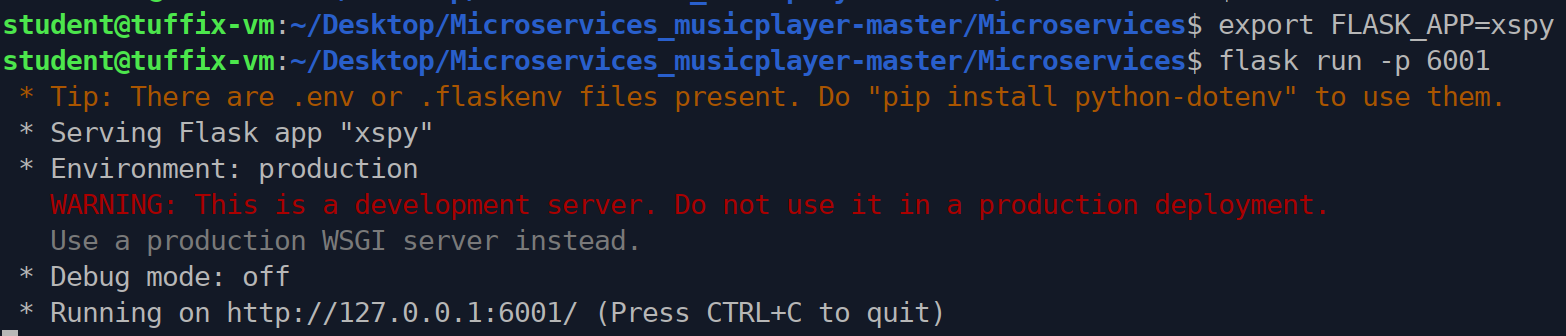
***Now in a third terminal we need to run the XSPY service which will connect to Kong.***

*Run the command* “export FLASK\_APP=xspy”

*Then run the command* “flask run -p 6001”

Note that xspy is running on port 6001 to avoid conflict with other ports.

**An example of the code is shown below.**



Now that the xspy service is running we can test it since its now calling Kong rather than the services directly.



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* END OF OPS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*