CPSC 449 - Web Back-End Engineering - Fall 2019

Project 1, due October 16

In this project, you will work in a team to build a set of microservices to allow users to create music playlists.

Microservices

Your team will build four microservices, hosted as four separate applications connected to a single database.

Given the service descriptions below, choose appropriate JSON data formats, HTTP methods, URL endpoints, and HTTP status codes for each operation.

Tracks microservice

Each track has a title, an album title, an artist, a length, a URL for the media file, and an optional URL for the album art.

(Note that files on your hard drive also have URLs, e.g. `file:///home/student/Music/1698%20Pachelbel%20%2C%20Canon%20in%20D.mp3`)

The following operations should be exposed:

- Create a new track
- Retrieve a track
- Edit a track
- Delete a track

Playlists microservice

A playlist has a title, a list of URLs for individual tracks, the username of its creator, and optionally a description. Track URLs should refer to the tracks microservice.

The following operations should be exposed:

- Create a new playlist
- Retrieve a playlist
- Delete a playlist
- List all playlists
- Lists playlists created by a particular user

Users microservice

Each user has a username, a hashed password, a display name shown to other users, an email address, and an optional URL for a homepage.

The following operations should be exposed:

- Create a new user
- Retrieve a user's profile, which includes all information except the hashed password
- Delete a user
- Change a user's password
- Authenticate a user, indicating whether the supplied username and password match the hashed password stored for that user

Descriptions microservice

Each user can post their own description for a track. As with the playlist microservice, individual tracks are referred to by their URLs.

The following operations should be exposed:

- Set a user's description of a track
- Retrieve a user's description of a track

API implementation

Implement your APIs in Python 3 using Flask. You are encouraged, but not required, to use Flask API to obtain additional functionality.

All data, including error messages, should be in JSON format with the `Content-Type` header field set to `application/json`.

HTTP status codes

Use appropriate HTTP status codes for each operation, with the following guidelines:

- In general, successful operations other than POST should return `HTTP 200 OK`.
- A successful POST should return `HTTP 201 Created`, with the URL of the newly-created object in the `Location` header field.
- Attempts to retrieve or modify an existing object should return `HTTP 404 Not Found` if the specified object does not exist (or no longer exists).
- Operations which result in a constraint violation such as attempting to `INSERT` a duplicate value into a column declared `UNIQUE` or attempting to `INSERT` a row with a

`FOREIGN KEY` referencing an item that does not exist in another table should return `HTTP 409 Conflict`.

Session state

Requests to each microservice must include all information necessary to complete the request; your APIs must not use the Flask `session` object to maintain state between requests.

Password hashing

The Flask framework is built on top of a WSGI library called Werkzeug, which includes security helper functions for working with hashed passwords. If you have Flask installed, you will also have access to this library.

Database

Use The Python Standard Library's `sqlite3` module as the database for your Flask application. You are encouraged, but not required, to use PugSQL as an alternative. Do not use an Object Relational Mapper such as SQLAlchemy.

Populating the microservices with data

Write a script to populate the microservices with tracks, descriptions, users, and playlists using your newly-defined APIs. Suitable approaches to scripting include:

- A shell script that calls `curl` commands
- A Python script using the requests library
- A YAML script using the Tavern plugin for pytest.

Managing processes

Write a `Procfile` and use foreman to start and manage the Flask process for each microservice.

Test platform

You may use any platform to develop services and tests, but note that per the Syllabus the test environment for projects in this course is a Tuffix VM with Python 3.6.7. It is the responsibility of the Operations role to ensure that your code runs on this platform.

Tips

- If you are not using PugSQL, read Using SQLite 3 with Flask carefully.
- Run your applications using the `flask` CLI.
- Use the Flask development environment.

- Enable foreign key support in SQLite
- Avoid Flask Blueprints and Python packages; implement each microservice in a single `.py` file.
- Get comfortable with the `curl` and `sqlite3` command-line interfaces.
- If you'd like some sample data, check out Luis Rocha's Chinook Database. (It is *not* recommended, however, to use the same database schema. These services can be much simpler.)

Teams

This project is written for teams of 3 students. Teams of 2 or 4 students must be pre-approved by the instructor, and only two teams of 2 (or one team of 4) will be approved. This project may not be submitted individually.

Team choices are not permanent; you may choose to work with different teams in later projects.

Roles

Teams must agree on a role for each member, and each project will specify a set of responsibilities for each role. There two *Development* roles and one *Operations* role for each project.

In order to receive full credit for the projects, each student must act in a Development role and an Operations role for at least one project.

Responsibilities

Dev 1 owns the Tracks and Playlists microservices.

Dev 2 owns the Users and Descriptions microservices.

Ops owns the `Procfile`, REST population script, and Tuffix deployment.

Each team member is responsible for documenting their work and assisting other team members with integrating their work together.

Students in two-member teams will each take a Dev role and split the Ops responsibilities. Students in four-member teams will take one microservice each and split the Ops responsibilities.

Submission

Submit your project by uploading your Python code, SQL schema, REST scripts, `Procfile`, documentation, and any other relevant artifacts to the `project1/` subdirectory of the folder that will be shared with you on Dropbox. Only one submission is required, but for safety consider uploading copies to each team member's submission folder. (Make certain, however, that the

copies are the same in each case; the instructor will not attempt to ascertain which is the “real” submission.)

A printed submission sheet will be provided on the due date. To finalize your submission, fill out the sheet with the requested information and hand it in to the professor by the end of class. Failure to follow any submission instructions exactly will incur a **10%** penalty on the project grade for all team members.