Spotify Playlist Generator

Project Goals

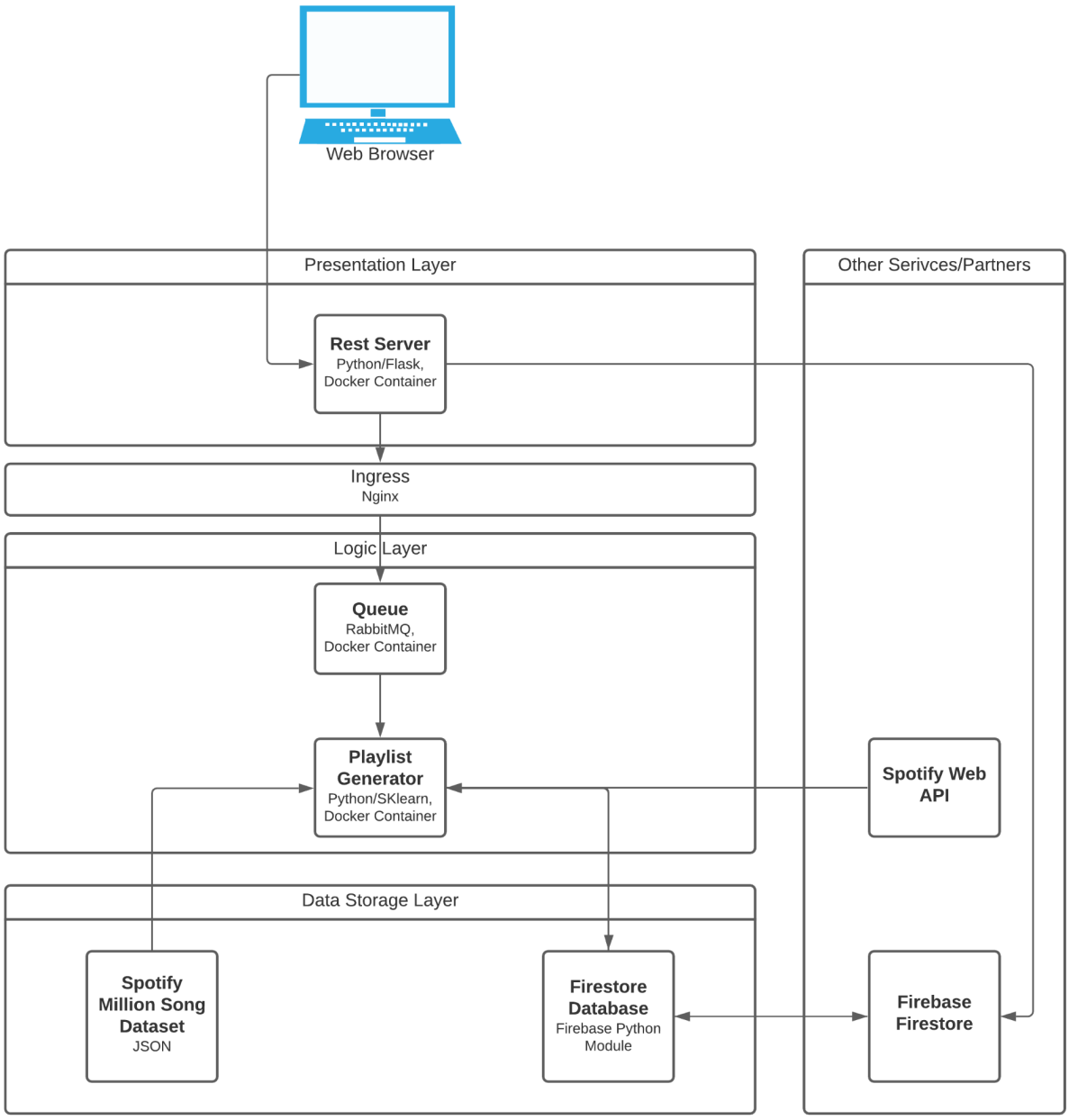
The service should be able to generate a new playlist for a user based on a song, or existing playlist. The generated playlist should be close to what the user expects. More relevant to the course, the service should keep an updated classification model for each user that is accessible at all times, the service should always be available while speed may be sacrificed when the service is in high demand. The classification model (add song to playlist or not) will be updated based on whether or not the user likes a given song in the generated playlist.

Software and Hardware Components

An object storage service like GCP Firestore will be used to store each user's trained, or pretrained (if the user has not liked a song from a generated playlist) model. The trained model will be a classification algorithm like K-Means, K-Nearest Neighbors, or Decision Ensembles that uses the audio features of a track to perform the classification. These audio features will be retrieved from Spotify's REST API. Algorithms will be trained using Spotify's million song dataset. The algorithm will be pulled from Python's SKlearn module. Models will be marshalled using Python's Pickle module, so that models can be stored in GCP Firestore. GCP Firestore will also be used to store each user's login credentials. RabbitMQ will be used to queue up requests to generate playlists and update the user's model. To bias the user's classification model to the user's music preferences, the playlist/song being used to generate a new playlist will be used to update/retrain the model. Kubernetes and Docker will be used to manage each service. Users will make playlist generation requests through a website supported by flask.

Document continues below

Architectural Diagram



Interaction of Software and Hardware Components

The service will be running on a GCP Kubernetes cluster, where each node will be an e2-medium. This is my best guess at a hardware setup that will be able to train, and update a model. I will be using the GCP API to automate the setup and teardown of the cluster. All of the software components will be running within Kubernetes.

Debugging and Testing

To debug my service, I will write a python script that will make REST API calls to the song generator service. To test my classification algorithm, I will split my dataset. This will most likely be an 80-20 split, where 80% of the data is used to train the algorithm, and 20% is used to test the algorithm. If time permits, I will test other data splits. Testing the output of the algorithm when inputting unseen data (data that is not in the million song dataset) will be qualitative: the performance of the algorithm will be based on the opinion of the listener.

How this will Meet the Project Requirements

This project will use the following datacenter software components:

1. REST API
    a. Spotify's REST API to retrieve information about a song
    b. REST API defined by myself to enable communication between the presentation, and logic layers
2. Google Firestore (storage service)
    a. Firestore will be used to store each user's classification model
    b. Firestore will also be used to store each user's credentials
3. Message Marshalling/Encoding
    a. To store classification models, each model will be marshalled using Python's Pickle module
    b. Though this doesn't fall under marshalling/encoding, passwords will also be hashed using SHA-256
4. Containers
    a. Docker containers and Kubernetes will be used to make sure each subcomponent is scalable, and independent of all other subcomponents.