JONGLEUR

INDEPENDENT WORK, FALL 2021

Joseph Rubin '22 (jmrubin)
Adviser: Dr. Jérémie Lumbroso

ABSTRACT

Jongleur is a web application that ingests audio practice recordings and returns useful information. Users can create an account and browse a list of supported pieces. They can select one and begin uploading recordings of your practice session. After processing, each practice session is saved and displayed on a timeline. Each practice recording is then displayed as a waveform, segmented by the different "recording attempts."

1 This Writeup

This writeup does not represent the submission of a single semester IW project. Jongleur is ongoing, continuing into the Spring semester as a two-semester / thesis project. As a progress report, I've included a brief explanation of the current project below:

1.1 Summary

Jongleur is a full-stack app built on AWS in several thousand lines of code. The majority is Typescript, and Python is used for audio analysis. The app consists of quite a lot of infrastructure.

On the front end, I'm using Remix, a new React framework (it came out just this Thanksgiving). Its focus is on nested routing and server-side generation of pages. Audio analysis uses librosa.

Although the analysis we're doing isn't extremely complicated (segmentation, tempo calculation), there were some small challenges, and one HUGE challenge: since we want to be able to handle large, multi-hour uploads (think 1 to 3 hours), I had to dispense with the assumption that the audio file could fit in memory all at once. We have to stream the audio files in chunks and process those chunks one at a time. This changes the kinds of analyses you can do.

But the payoff is great - Jongleur can process large audio files without eating up endless amounts of memory resources. Also, notably, this focus on large files was a consideration when designing the process by which users upload their files into Jongleur in the first place, but I won't get to that here. The rest of the app is the back end / middle end. We store data in DynamoDB and use S3 for audio file storage. We use GraphQL (AWS AppSync) to serve data to the front end.

Some front-end operations go through API Gateway instead, but most are through GraphQL. Our user authentication system is built on Cognito. Users authenticate to the GraphQL API using a Cognito accessToken JWT, so they can only access their own stuff.

The Remix app server is built into a Docker image and runs on AWS Fargate. The audio processing pipeline uses a few lambdas and is orchestrated by an AWS StepFunction. The primary one is also built into a Docker image so it can access all the requisite dependencies and run predictably. Audio files are served from CloudFront so they stream very quickly.

All of the infrastructure is written as code in CDK, which makes it reproducible (and also easy to deploy. You can launch the full stack with one command and a few small logistical tasks. See the README.md). In total, the CDK code that specifies this infrastructure is about nine hundred lines long; there's a lot of infrastructure (see infrastructure/lib/). But it's actually all serverless.

The code is here: https://github.com/josephrubin/jongleur

Statement of Academic Integrity

This report represents my own work in accordance with University regulations.
Signed, /Joseph Rubin/.