UC Santa Barbara

NEXT

CS 48 - Computer Science Project - Spring 19

Draft Project

Slack
GitHub
Pivotal
Travis-CI

Authors:
Cristian Gomez - c_k_c@ucsb.edu
Edward Zhong - ezhong@ucsb.edu
Kerem Celik - kerem@ucsb.edu
Ramiro Pinto - rpintoprieto@ucsb.edu
Steven Booke - stevenbooke@ucsb.edu

May 8, 2019

Contents

1 Project Background

Hiring a DJ to perform at a venue or a party is a cost-ineffective and inefficient way of playing music. The DJ must either play what the crowd requests or in most cases will only play music that the DJ enjoys. The crowd as an aggregate knows what they want to listen to and they should be responsible for choosing music. Entrusting the venue host with choosing a DJ that will know and then play what the crowd desires is simply outdated in the age of smartphones.

Solutions today include mobile apps like TouchTunes and Rockbot.
However, both of these apps only allow business like restaurants and stores to allow users to choose their songs and even require in-app purchases to actually select the next song. There does not exist a solution where users can host their own venues/events and allow other users to select songs.

Users should be able to utilize their phones as an impromptu jukebox and allow other nearby users to select and vote on the next songs for free.

2 Project Outcome

We propose a system where users will be able to create virtual jukeboxes (venues) and allow other users to connect to said venue and propose songs.

Venues will be location aware and will move with the host of the venue. Users can then find venues nearby them.

Venues will employ a voting-based system to determine the next song to play in order to enable the crowd to control what they listen to; the most popular song (desired by the most people) will always be played. Users can upvote or downvote songs and the song in the playlist with the highest score (`num_upvotes - num_downvotes`) will be selected. Additionally, a skip feature can enable users to vote to skip the current song being played. The threshold can be dynamic based on the average number of votes other songs in the venue have received, or the host of the venue can can configure a static threshold.

Users of the app, not only enterprise/business partners can host their own ad-hoc venues where other users can connect and vote on the next songs. Per-venue statistics exposed to the host after an event will allow the host to get insightful data about the music played and the engagement of the audience.

2.1 Assumptions

In designing this product we must make various assumptions:

1. at any venue the number of people with non-malicious intent outnumbers those with malicious intent (wanting to compromise the venue with ill-suited songs)
2. the voting algorithm will accurately represent the overall choices of the crowd on average
3. users will interact with the system consistently

3 Planning

3.1 Milestones

Sprint 1

1. Client wire mocks
2. Basic client integration with music player
3. Basic server integration with database layer

Sprint 2

1. Basic client and server integration
2. Location aware venue search

Sprint 3

1. Song voting and skipping support
2. Statistics system
3. Demo preparation

3.2 Components

Facilitator: Service that exposes a REST API to manage venues, tracks, votes, and voteskips. Interacts with DB to persist and fetch data.

Frontend: Service that exposes UI using React. Interacts with Facilitator to manage user actions and interacts with Napster API to query tracks, render tracks, and play tracks.

3.3 Architecture

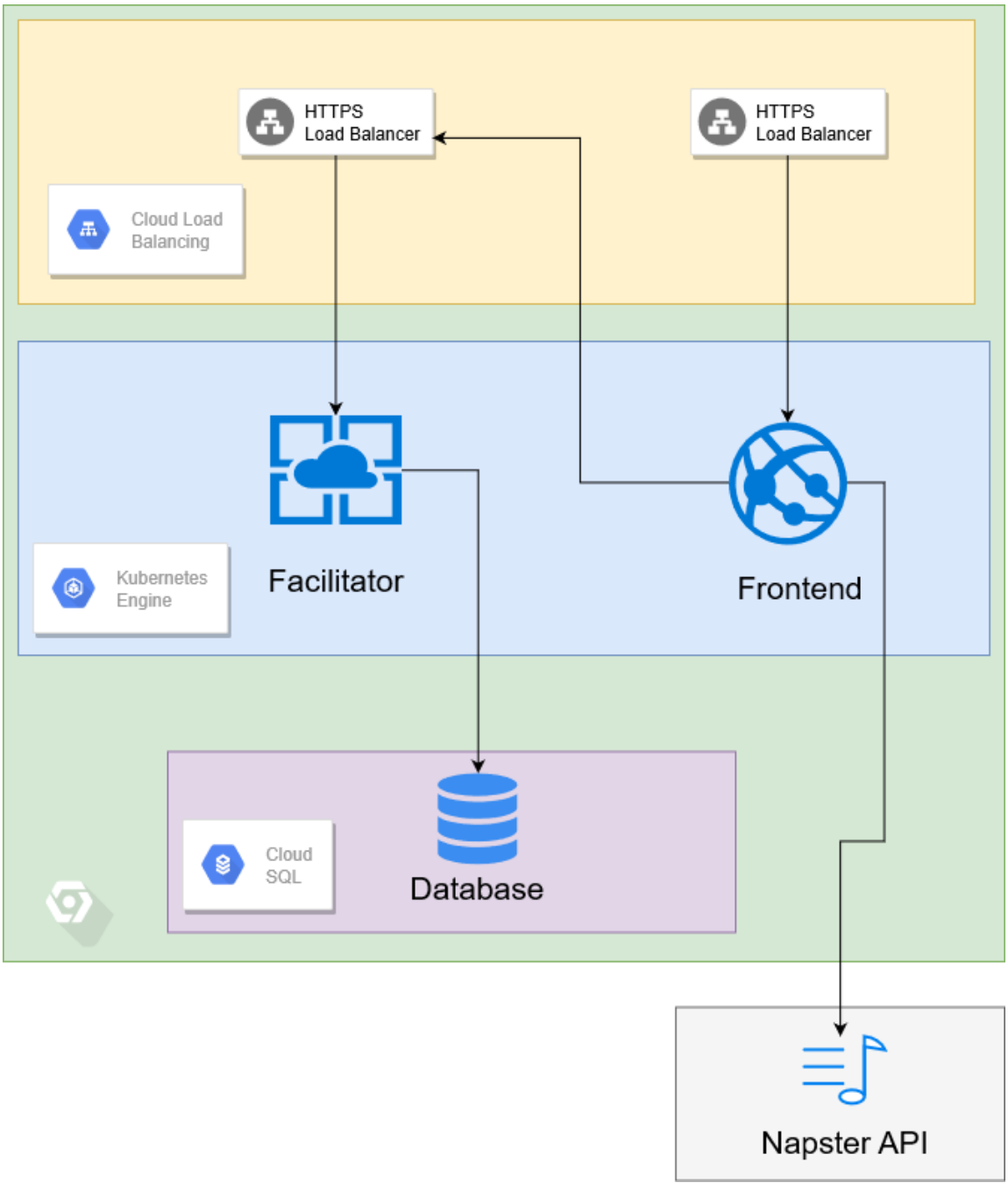

Figure 1: High-level system architecture

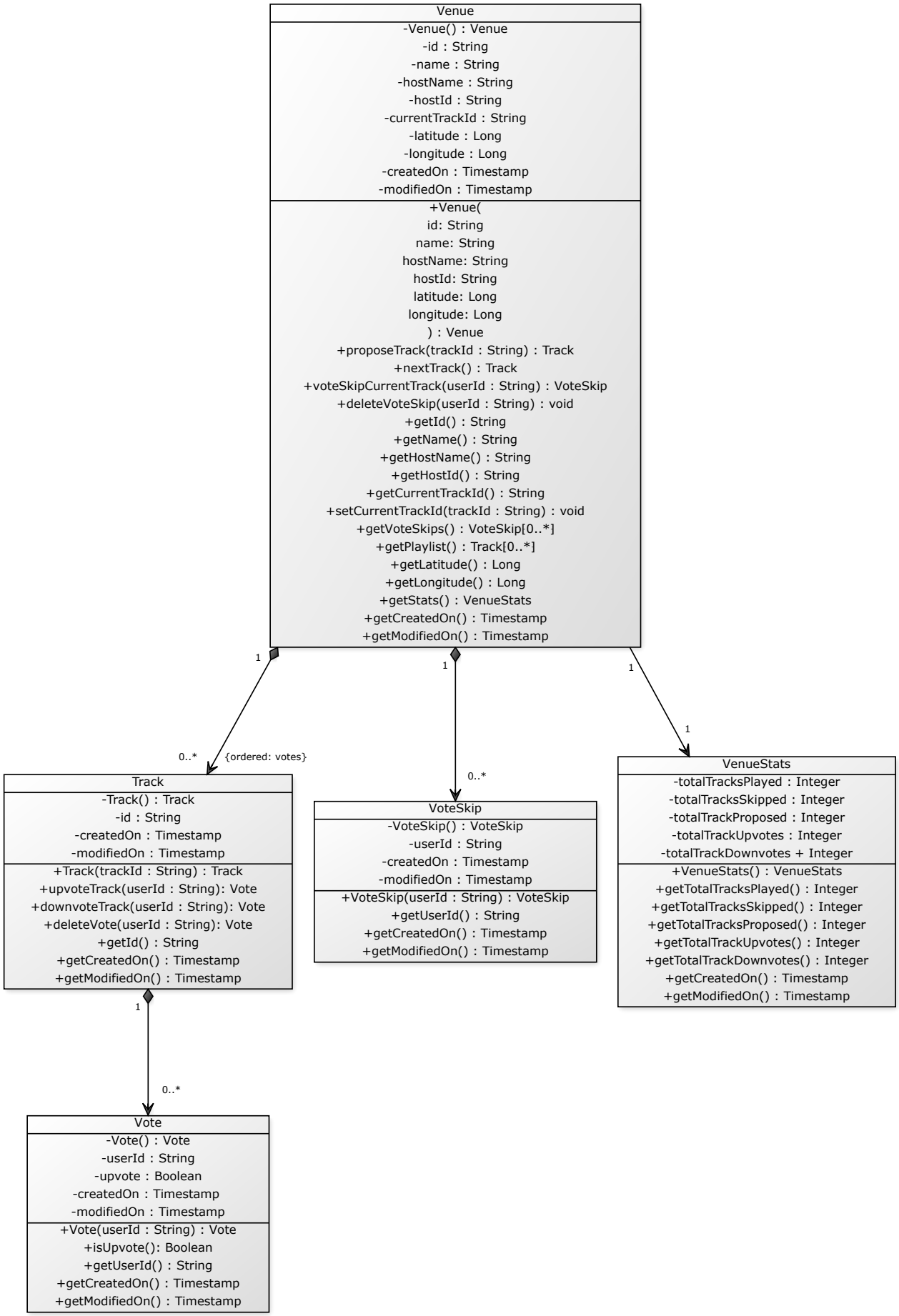

Figure 2: Entity classes UML diagram

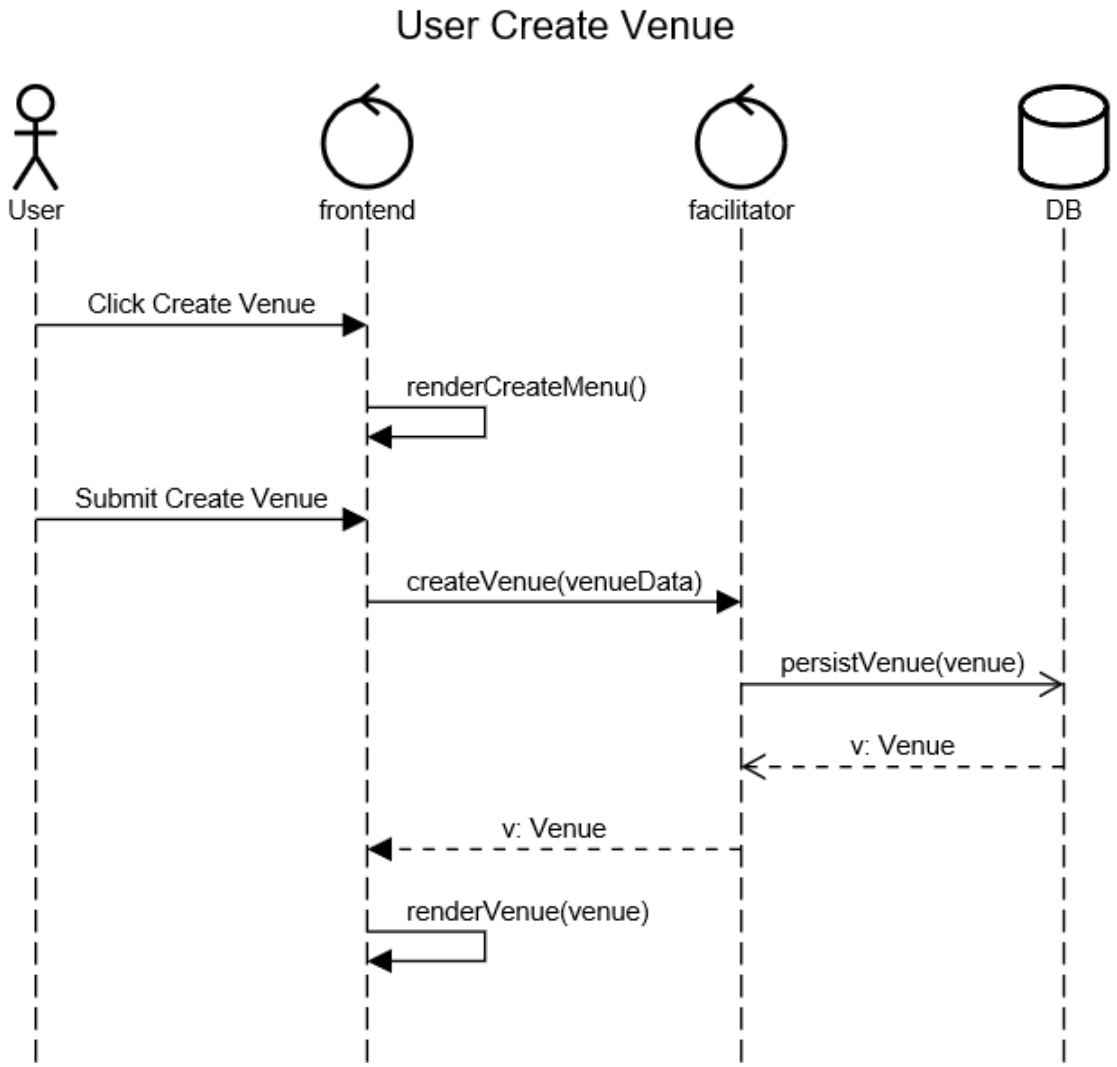

Figure 3: Venue creation sequence diagram (synchronous)

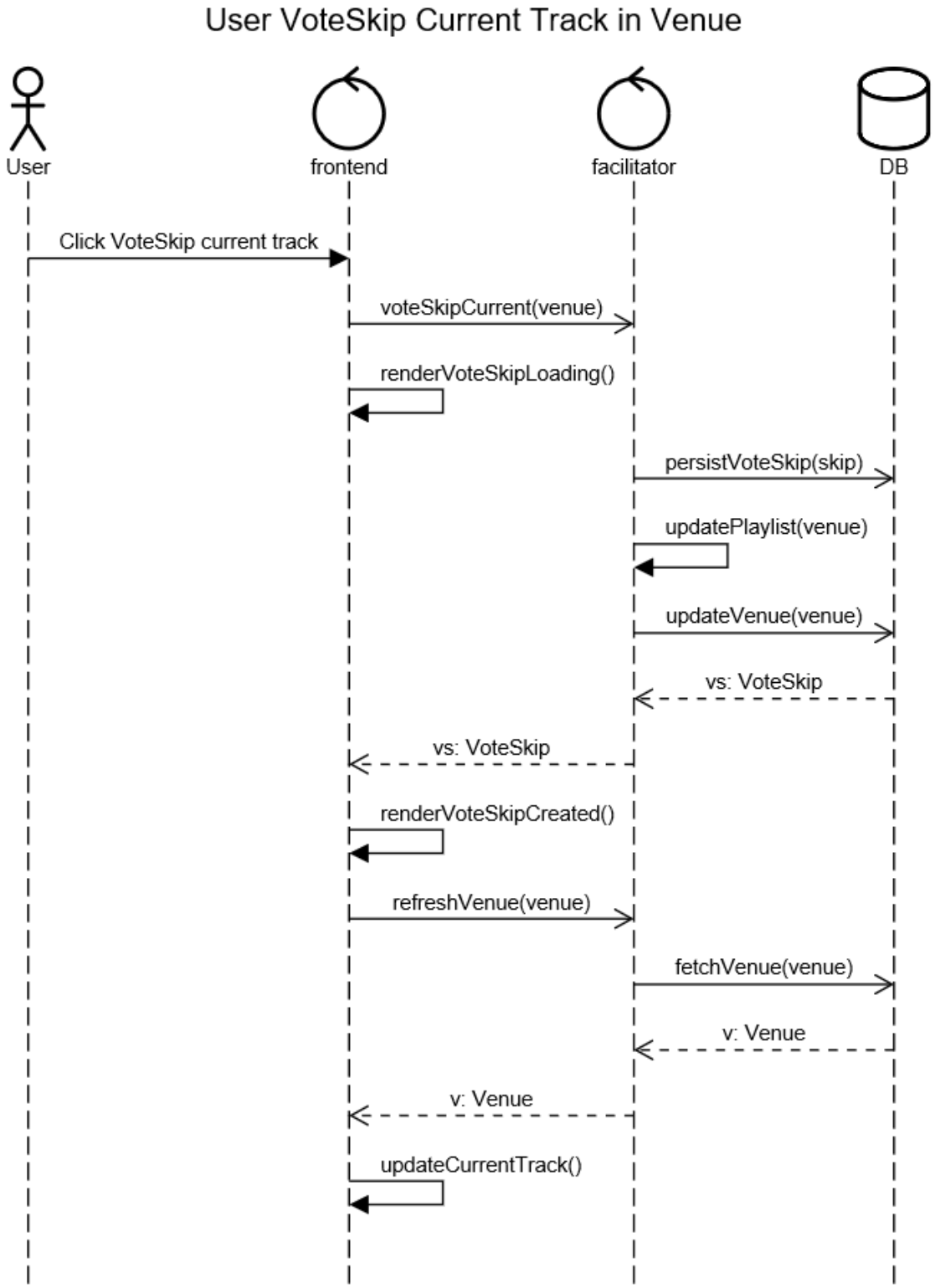

Figure 4: VoteSkip current track sequence diagram (asynchronous)

3.4 Interface Interaction

Figure 5: UI interaction mockups

4 User Stories

1. Client UI Flow

   As a client, I want to know how to flow between various UI views so that the user has an intuitive experience.

   Given there are various views/pages in the client, the client will flow between them accordingly.

2. Server <-> DB Interaction

   As a server I want to interact with my DB to persist venues, find venues by location, and persist voting records so that I can create/list venues and vote(skip)/propose songs.

   Given that there is a server and a DB, server will interact with DB to persist and search records.

3. Client <-> Music API Interaction

   As a client I want to interact with music API to search for songs by name, get song ID, get song by ID, and play a song, so that I can reference songs to the server in a normalized way and play songs.

   Given there is a client and a music API, client will interact with music API to search songs by name, get song ID, get song by ID, and play a song.

4. Client <-> Server Interaction

   As a client I want to be able to communicate with the server and receive data so that I can stay up-to-date with server data.

   Given there is a client, client can communicate with server and receive valid data back.

5. Client List Venues

   As an audience user I want to interact with my client to dispatch a venue list request using my GPS location so that I can list nearby venues.

   Given there is at least one nearby venue, when client refreshes, an request will be dispatched to the server and the client will await results and render.

6. Server List Venues

   As the server (facilitator) I want to respond to venue list requests to list venues so that audience clients can see nearby venues sorted by geographical distance.

   Given that a venue list request was dispatched to the server, server will enumerate nearby venues from DB and respond with venue list.

7. Client Fetch Venue

   As an audience user I want to interact with my client to dispatch a venue fetch request for a certain venue so that I can get up-to-date about the venue.

   Given there exists a venue, when client fetches that venue, an request will be dispatched to the server and the client will await results and render.

8. Server Fetch Venue

   As the server (facilitator) I want to respond to venue fetch requests to fetch venues so that audience clients can see information about a venue in progress.

   Given that a venue fetch request was dispatched to the server, server will find venue from DB and respond with venue info.

9. Client Create Venue

   As a host user I want to interact with my client to dispatch a venue creation request to the backend so that I can begin hosting my venue.

   Given that a user wants to create a venue, when the create venue button is pressed, then a request will be dispatched to the server and the client will await results and render.

10. Server Create Venue

    As the server (facilitator) I want to respond to venue creation requests to create venues so that audience clients can see the venue and participate in song selection.

    Given that a venue creation request was dispatched to the server, server will persist a venue object into the DB and respond with the newly created venue.

5 Technologies Utilized

Spring Boot: framework upon which facilitator service is built

Maven: build tool used to build facilitator service

Docker: used to containerize both facilitator service & frontend service

Google Kubernetes Engine (GKE): used to deploy and orchestrate facilitator & frontend containers, create services, and create load balancers (in conjunction with GCP load balancers)

Google Cloud SQL: used to host an instance of PostgreSQL

PostgreSQL: data storage layer; stores venues, tracks, votes, and stats

Hibernate/JPA: ORM tool

Create React App: create backend-agnostic React apps

Node.JS/npm: Used to develop, build, and test frontend service

nginx: Used to host the frontend service built by Create React App & npm

Travis-CI: CI used to build, test, and deploy frontend & facilitator services

Mockplus: Program used to design and implement the mock-up for the UI