Final Project

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

June 2, 2019
## Contents

1 **Project Background**  
2 **Project Outcome**  
2.1 Assumptions ........................................... 2
3 **Planning**  
3.1 Milestones ............................................. 3
3.2 Components ........................................... 3
3.3 Architecture ........................................... 4
3.4 Interface Interaction ................................. 8
4 **Sprint Retrospective Overview**  
4.1 Sprint 1 ................................................ 9
4.2 Sprint 2 ................................................ 9
4.3 Sprint 3 ................................................ 9
5 **Challenges Faced** ................................... 13
6 **Remaining Features** ................................. 13
7 **Developer Commits**  
7.1 Frontend ................................................. 14
7.2 Facilitator .............................................. 15
8 **Testing**  
8.1 Facilitator ................................................. 16
8.2 Frontend ................................................. 16
9 **User Stories** .......................................... 17
10 **Technologies Utilized** ............................ 24
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 Entrusting the venue host with choosing a DJ that will know and then play what the crowd desires is simply outdated in the age of smartphones.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 ($\text{num\_upvotes} - \text{num\_downvotes}$) 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 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

4. users will be interacting using a supported platform
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)
User VoteSkip Current Track in Venue

Click VoteSkip current track

voteSkipCurrent(venue)

renderVoteSkipLoading()

persistVoteSkip(skip)

updatePlaylist(venue)

updateVenue(venue)

vs. VoteSkip

renderVoteSkipCreated()

refreshVenue(venue)

fetchVenue(venue)

v. Venue

updateCurrentTrack()

vs. VoteSkip

v. Venue

Figure 4: VoteSkip current track sequence diagram (asynchronous)
3.4 Interface Interaction

Figure 5: UI interaction mockups
4 Sprint Retrospective Overview

The following details focus items identified in each sprint; each retrospective document is attached in the following 3 pages.

4.1 Sprint 1

During the first sprint, we made it a priority to allocate time every day to work as a group together. We planned how we would use codebunk to collaboratively work on the project concurrently and we made time to meet every day for at least 30 minutes.

4.2 Sprint 2

After Sprint 1, we identified that our stories and tasks were sometimes lacking in depth with acceptance tests and deliverables. We then ensured new stories had acceptance tests that were clear but very specific and had others review our tests. We made sure deliverables were also clear and we made sure that when a task was done, it actually met the deliverable before marking the task as Accepted.

4.3 Sprint 3

After Sprint 2, we noticed that we had quite a bit of switching around between tasks when we were blocked by other team members. We then ensured that task blockers were clearly documented and we took extra care in picking tasks to guarantee that the chances of blocking another team member were minimal, and a developer’s tasks led sequentially into their next one.
next Sprint 1 Retrospective

April 16, 2019 - April 26, 2019

Demos:
Steven: frontend/backend Kubernetes deployment
Edward: Napster PoC
Kerem: API functionality
Rami: mockups/wireframes
Cris: frontend

What worked?
● Steven: Learned Javascript (had trouble with callbacks), got a lot of insight into server containerization and TLS.
● Edward: Learned a lot about NodeJS and got docker containers working.
● Rami: Learned a lot of different client frameworks, design/mockup went well.
● Cris: Got hands dirty with React, figure out how to make requests in React, learned Napster API.
● Kerem: Got Napster PoC working, documented server backend clearly.

What didn’t?
● Steven: Had some trouble getting GCP billing/free credits working.
● Edward: Had trouble understanding the various limitations for certain music APIs.
● Rami: React was more complex than expected.
● Cris: Handling user states in React is complicated.
● Kerem: SSL certificates & load balancing for frontend/backend was a pain.

What should change?
● Steven: Use JS that was learned during sprint one to help more with frontend development.
● Edward: Help integrate the napster api into the frontend
● Rami: Make sure each task on Pivotal has a clear deliverable.
● Cris: Document more communications on Slack so they aren’t forgotten.
● Kerem: Have more communications with frontend devs so API is clear.

Focus item:
● Tasks and stories must have clear deliverables and acceptance tests.
next Sprint 2 Retrospective

April 29, 2019 - May 10, 2019

Demos:
Steven: frontend & facilitator CI build, test, & deployment
Edward: song skip feature
Kerem: song suggest menu
Rami: song up/down voting & selection algorithm
Cris: geolocation integration

What worked?
● Steven: Getting CI working end-to-end to ease deployment speeds up development & testing
● Edward: Learned about Spring configuration hierarchies
● Rami: Learned about scheduling refresh tasks in React
● Cris: Learned how to use and test with react-geolocate libraries
● Kerem: React documentation is awesome, was able to implement easily

What didn’t?
● Steven: Reached quota limit for Pivotal private tracker
● Edward: Had a little problem deciding how configuration files should be set up in production.
● Rami: Bombarded facilitator API too much when testing
● Cris: React geolocate library was hard to mock during testing
● Kerem: Sometimes was hard to plan tasks to avoid blockers

What should change?
● Steven: Need to have a better idea of deliverables for tasks
● Edward: Had to change tasks because of blockages
● Rami: Should share things we learned about libraries with each other
● Cris: Should set up Slack notifications so we don’t miss any messages
● Kerem: Take better care to avoid blockers and try to take tasks that are related.

Focus item:
● Plan ahead better to avoid blockers by taking tasks that are related to each other (cohesion).
next Sprint 3 Retrospective

May 12, 2019 - May 24, 2019

Demos:
Steven: song streaming through Napster SDK
Edward: keeping audience content up to date
Kerem: mobile rendering optimizations
Rami: statistics & venue closing
Cris: enhancing DB relationships

What worked?
- Steven: Learned a lot about HLS streaming and got DRM player working
- Edward: Learned about horizontal scaling pods with GKE
- Rami: Learned about transactional & atomic DB queries
- Cris: Digged deeper into Hibernate/Spring JPA; learned a lot about ORMs
- Kerem: Rami was very helpful with CSS advice

What didn’t?
- Steven: Discovered Apple’s limitations on streaming within Safari
- Edward: Had trouble authenticating with GKE
- Rami: Had a race condition in some DB queries that took a while to debug
- Cris: Lots of information to process when learning ORMs
- Kerem: Prioritization of tasks was harder when little time left in sprint

What should change?
- Steven: Task deliverables need to be clearer
- Edward: Documentation should always be kept up-to-date
- Rami: Need to set aside at least a few hours every day to dedicate to project
- Cris: More frequent demos to share knowledge
- Kerem: Need to better understand which tasks are higher priority

Focus item:
- Take better care prioritizing tasks so we need to know what needs to be done now and what can wait.
5 Challenges Faced

A problem we faced early on in the development cycle was finding a music streaming service that would be free to use, or would at least offer free trials to developers. While obviously no music streaming service can offer subscription-free listening, we were able to leverage the Napster API using a premium developer trial account. Although using a developer trial account is sufficient for the first three sprints and hence the demo, we are limited by the number of concurrent sessions available as well as a rate limit on global API calls. In the future, we would likely want the host of a venue to authenticate with their own music streaming service credentials (such as Apple Music, Spotify, or Napster) and we would also be able to access the host's music library. Additionally, the Javascript SDK provided by Napster for streaming music was quite lacking in quality & documentation, and bug-ridden, so we were forced to actually manage the internal state of the stream ourselves, while only using the DRM streaming framework the SDK provided.

6 Remaining Features

We have many great proposed features that would bring great value to the product and attract users, including but not limited to:

1. Leverage ML to identify the genre of songs proposed and voted on in a venue to recommend other songs or curate default playlists for venues.

2. Allow users to authenticate themselves using Napster to access their personal music library.

3. Allow certain users to be "verified", where venues they host would automatically show up as prioritized in the venue listing. (could be useful for restaurants, etc).

4. Add an explicit song filter so that certain venues are not allowed to play explicit songs.

5. Add permission gating so that the host can whitelist only certain users to propose songs.

6. Add an account feature so that users can see each others’ scores regarding how many songs proposed, how many songs were accepted, how many were skipped, etc.
7 Developer Commits

While development was done using codebunk, and committed under a single GitHub ID, each commit can be more or less attributed to a single person.

7.1 Frontend

<table>
<thead>
<tr>
<th>Commit</th>
<th>Date</th>
<th>Time</th>
<th>Message</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>7fa2a45</td>
<td>Mon, 20 May</td>
<td>13:39:56</td>
<td>add current song image to venue listing</td>
<td>Rami</td>
</tr>
<tr>
<td>5e03ad1</td>
<td>Mon, 20 May</td>
<td>18:15:58</td>
<td>pass functions to song play look</td>
<td>Cris</td>
</tr>
<tr>
<td>3a21c55</td>
<td>Wed, 22 May</td>
<td>13:27:15</td>
<td>Create README.md</td>
<td>Edward</td>
</tr>
<tr>
<td>4bb4a79</td>
<td>Tue, 21 May</td>
<td>17:49:07</td>
<td>redo venue info design</td>
<td>Kerem</td>
</tr>
<tr>
<td>cde56ff</td>
<td>Tue, 21 May</td>
<td>17:16:43</td>
<td>add headphones icon for audience</td>
<td>Steven</td>
</tr>
<tr>
<td>43aaa88</td>
<td>Wed, 22 May</td>
<td>11:55:44</td>
<td>allow audience to see progress</td>
<td>Rami</td>
</tr>
<tr>
<td>92f4f28</td>
<td>Mon, 20 May</td>
<td>18:29:19</td>
<td>Revert &quot;pass functions to song play look&quot;</td>
<td>Kerem</td>
</tr>
<tr>
<td>1c25475</td>
<td>Fri, 17 May</td>
<td>10:50:15</td>
<td>use slider to seek song</td>
<td>Kerem</td>
</tr>
<tr>
<td>71e5216</td>
<td>Thu, 16 May</td>
<td>00:58:36</td>
<td>adjust vertical for time</td>
<td>Edward</td>
</tr>
<tr>
<td>ece310a</td>
<td>Thu, 16 May</td>
<td>00:55:53</td>
<td>update css for seek bar</td>
<td>Steven</td>
</tr>
<tr>
<td>ebdcb9f</td>
<td>Wed, 15 May</td>
<td>11:29:01</td>
<td>add progress bar seeking</td>
<td>Edward</td>
</tr>
<tr>
<td>d96ab14</td>
<td>Mon, 20 May</td>
<td>14:54:00</td>
<td>add basic slider</td>
<td>Kerem</td>
</tr>
<tr>
<td>34c93b5</td>
<td>Thu, 23 May</td>
<td>01:20:26</td>
<td>explicitly set project name</td>
<td>Steven</td>
</tr>
<tr>
<td>ed86e3f</td>
<td>Sun, 12 May</td>
<td>21:55:21</td>
<td>gcloud auth before cluster auth</td>
<td>Rami</td>
</tr>
<tr>
<td>dbe9065</td>
<td>Sun, 12 May</td>
<td>21:49:23</td>
<td>update statefulset with new image</td>
<td>Edward</td>
</tr>
<tr>
<td>4db6f66</td>
<td>Thu, 16 May</td>
<td>01:02:05</td>
<td>tweak css again</td>
<td>Edward</td>
</tr>
<tr>
<td>71e5216</td>
<td>Thu, 16 May</td>
<td>00:58:36</td>
<td>adjust vertical for time</td>
<td>Edward</td>
</tr>
<tr>
<td>ece310a</td>
<td>Thu, 16 May</td>
<td>00:55:53</td>
<td>update css for seek bar</td>
<td>Steven</td>
</tr>
<tr>
<td>ebdcb9f</td>
<td>Wed, 15 May</td>
<td>11:29:01</td>
<td>add progress bar seeking</td>
<td>Edward</td>
</tr>
<tr>
<td>d96ab14</td>
<td>Mon, 20 May</td>
<td>14:54:00</td>
<td>add basic slider</td>
<td>Kerem</td>
</tr>
<tr>
<td>34c93b5</td>
<td>Thu, 23 May</td>
<td>01:20:26</td>
<td>explicitly set project name</td>
<td>Steven</td>
</tr>
<tr>
<td>ed86e3f</td>
<td>Sun, 12 May</td>
<td>21:55:21</td>
<td>gcloud auth before cluster auth</td>
<td>Rami</td>
</tr>
<tr>
<td>dbe9065</td>
<td>Sun, 12 May</td>
<td>21:49:23</td>
<td>update statefulset with new image</td>
<td>Edward</td>
</tr>
<tr>
<td>d422e7f</td>
<td>Sun, 12 May</td>
<td>21:48:09</td>
<td>tag image with commit id</td>
<td>Kerem</td>
</tr>
<tr>
<td>bfb5ca8</td>
<td>Sun, 12 May</td>
<td>16:06:48</td>
<td>only update kubectl component</td>
<td>Rami</td>
</tr>
<tr>
<td>280c61a</td>
<td>Sun, 12 May</td>
<td>16:04:33</td>
<td>push image to gcloud</td>
<td>Rami</td>
</tr>
<tr>
<td>73d9947</td>
<td>Sun, 12 May</td>
<td>15:28:29</td>
<td>add gcloud to travis ci build</td>
<td>Rami</td>
</tr>
<tr>
<td>f197c02</td>
<td>Sun, 12 May</td>
<td>14:57:20</td>
<td>remove debug statements from dockerfile</td>
<td>Steven</td>
</tr>
<tr>
<td>86cc7e9</td>
<td>Thu, 9 May</td>
<td>13:42:55</td>
<td>add test initialization file and geoloc mocks</td>
<td>Kerem</td>
</tr>
<tr>
<td>f1df7e6</td>
<td>Thu, 9 May</td>
<td>13:22:45</td>
<td>npm install before test</td>
<td>Kerem</td>
</tr>
<tr>
<td>7008732</td>
<td>Thu, 9 May</td>
<td>13:20:42</td>
<td>fix typo</td>
<td>Rami</td>
</tr>
<tr>
<td>a10475d</td>
<td>Thu, 9 May</td>
<td>13:19:01</td>
<td>Add basic CI config</td>
<td>Edward</td>
</tr>
<tr>
<td>85cd2fe</td>
<td>Thu, 9 May</td>
<td>13:11:55</td>
<td>allow for async trackProgress events</td>
<td>Rami</td>
</tr>
<tr>
<td>12a21a1</td>
<td>Mon, 6 May</td>
<td>11:11:48</td>
<td>Add containerization</td>
<td>Steven</td>
</tr>
<tr>
<td>f9e32a1</td>
<td>Sun, 5 May</td>
<td>18:27:46</td>
<td>Add voteskip features and song playing</td>
<td>Edward</td>
</tr>
<tr>
<td>5094da9</td>
<td>Fri, 3 May</td>
<td>01:24:48</td>
<td>Add basic Napster SDK integration</td>
<td>Kerem</td>
</tr>
<tr>
<td>c903d7a</td>
<td>Thu, 2 May</td>
<td>14:20:03</td>
<td>upvote/ downvote icons</td>
<td>Steven</td>
</tr>
<tr>
<td>184f128</td>
<td>Thu, 2 May</td>
<td>12:02:42</td>
<td>Allow track proposal and track user ID</td>
<td>Kerem</td>
</tr>
<tr>
<td>32d9a45</td>
<td>Wed, 1 May</td>
<td>21:51:27</td>
<td>Initial commit</td>
<td>Kerem</td>
</tr>
</tbody>
</table>
7.2 Facilitator

ff65fcb Thu, 23 May 2019 00:12:16 -0700 utilize cascading to save venue stats - Steven
5974391 Wed, 22 May 2019 23:24:53 -0700 put closed in venue listing - Rami
e2a152e Wed, 22 May 2019 23:07:28 -0700 track stats & add venue close handler - Edward
3fb20c2 Mon, 20 May 2019 12:56:01 -0700 fix json property name - Steven
3b0de0e Mon, 20 May 2019 12:46:28 -0700 add time update handlers - Rami
4e80c09 Fri, 17 May 2019 11:10:54 -0700 Revert addition of time fields - Cris
65338bf Sun, 12 May 2019 22:33:33 -0700 tag maven image in gcr format - Steven
34070ef Sun, 12 May 2019 22:28:33 -0700 build docker image during install goal & push image - Edward
19ab56e Sun, 12 May 2019 15:00:08 -0700 rename default config file - Rami
3a61689 Sun, 5 May 2019 16:59:06 -0700 configure tests better - Steven
9e10c55 Sun, 5 May 2019 16:56:14 -0700 configurable voteskip threshold - Cris
4e80c09 Fri, 17 May 2019 11:10:54 -0700 Revert addition of time fields - Cris
f2f36f0 Fri, 17 May 2019 12:10:23 -0700 add time update handlers, DB interaction, & tests - Steven
561fe5c Thu, 17 May 2019 11:58:24 -0700 Add next-song functionality - Steven
c27ac52 Thu, 2 May 2019 18:10:24 -0700 fix delete vote endpoint typo - Edward
204c6b8 Thu, 2 May 2019 11:18:43 -0700 accept host and user IDs for venues & votes - Edward
a328c15 Thu, 2 May 2019 00:30:53 -0700 set track<->vote FKs to not be updated on vote insert - Edward
2140926 Tue, 30 Apr 2019 22:51:40 -0700 add basic input validation - Rami
2dc12b5 Tue, 30 Apr 2019 22:24:45 -0700 compare by venue age second - Cris
5486160 Tue, 30 Apr 2019 15:24:15 -0700 add null checks for creation time, fix column config - Rami
304c6e3 Tue, 30 Apr 2019 15:06:48 -0700 update docs, add age field - Cris
437c02d Tue, 30 Apr 2019 14:58:28 -0700 java SortedSet uses compareTo for equality, fix this - Cris
bca9294 Tue, 30 Apr 2019 14:45:39 -0700 allow CORS & add hostname field for venues - Edward
b388888 Mon, 29 Apr 2019 13:47:17 -0700 add IT updates - Cris
109ff9e Sun, 28 Apr 2019 11:1:17 -0700 add failsafe plugin to run ITs - Rami
5c2bc57 Sun, 28 Apr 2019 10:41:19 -0700 formatting and fix type in model class - Kerem
e5c76e8 Sun, 28 Apr 2019 10:41:03 -0700 add new endpoints and update API docs - Cris
d8310c4 Sun, 28 Apr 2019 10:40:44 -0700 add probe controller for k8s readiness checks - Edward
5067c7f Sun, 28 Apr 2019 10:40:16 -0700 add JPA repositories for Track, Venue, & Vote - Edward
af2c192 Sun, 28 Apr 2019 10:39:56 -0700 let IT parse the JSON response so we can chain future ITs - Kerem
b011df3 Sun, 28 Apr 2019 10:39:23 -0700 add maven plugin to build docker image - Kerem
e7c4e60 Thu, 25 Apr 2019 11:53:39 -0700 add new venue, track, vote endpoints and DB integration - Steven
6c831ef Thu, 18 Apr 2019 20:00:06 -0700 maven wrapper removed, use executable - Rami
59e8b27 Thu, 18 Apr 2019 21:57:54 -0700 add travis configuration - Steven
5b4f38c Wed, 17 Apr 2019 23:59:28 -0700 UNIX-ify target executable jarfile name - Edward
eb7e0e5 Wed, 17 Apr 2019 23:58:48 -0700 Update API documentation for venue creation request - Steven
1564c9c Wed, 17 Apr 2019 23:54:59 -0700 add basic controller & service. add venue creation route. basic IT - Kerem
280553e Tue, 16 Apr 2019 18:45:36 -0700 Spring boot skeleton - Kerem
5cc4e7a Fri, 5 Apr 2019 14:07:13 -0700 Update LICENSE - Edward
de2c499 Fri, 5 Apr 2019 14:06:16 -0700 Update README.md - Kerem
3e0e18e Fri, 5 Apr 2019 13:59:08 -0700 Initial commit - Edward
8 Testing

8.1 Facilitator

Facilitator is testing using SpringRunner for integration tests and JUnit for unit tests. Tests are dispatched using Maven Surefire Plugin for unit tests and Maven Failsafe Plugin for integration tests. Test files are defined in src/test/java/buzz/song/facilitator/, where WebIT is an integration test that tests every route exposed and ServiceTest tests the lifecycle for venues. Code coverage is generated using JaCoCo, where every branch except error branches, and every method except for toString, hashCode, etc. is tested:

Stress testing/simulation was not done nor was chaos testing the system as a whole.

8.2 Frontend

Frontend is testing using react-testing-library, jest-dom, & async-wait-until. Tests are dispatched using Create React App's react-scripts test script. Test file is defined in src/App.test.js and mocks are in src/__mocks__. The src/App.test.js does some basic testing of the venue listing and venue information components but does not test the actual streaming of the song. Code coverage was generated using npm t -- --coverage --watchAll=false and coverage is relatively low since we are not testing the song playback.
9 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.
   Demo: mockups: [http://run.mockplus.com/TFRg7wqFYmVXdeeT/index.html](http://run.mockplus.com/TFRg7wqFYmVXdeeT/index.html)

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.
   Tests: Facilitator ITs demonstrate DB connectivity. API is live at [https://api.song.buzz](https://api.song.buzz) and demonstrates integration with GCP postgres instance

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.
   Demo: frontend is live and can interact with API

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.
   Demo: frontend-poc is live at song.buzz and in repo poc-frontend

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.
   Demo: frontend is live with list venue functionality

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.

Demo: frontend is live with list venue functionality

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.

Demo: WebIT::controllerTests

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.

Demo: frontend is live with create venue functionality

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.

Demo: WebIT::controllerTests

11. **Expose frontend over HTTPS**

As an audience user I want to interact with my client over HTTPS so that I can be assured that my connection is secure and other requests I make to external requests are also secured.

Given there exists an audience user and a client, the audience user will be able to access the client over HTTPS and not HTTP.

Demo: load balancer exists and frontend can be accessed at https://song.buzz

12. **Server Receive Song Upvote**

As the server (facilitator) I want to respond to song vote requests so that a venue playlist will have the corresponding song's vote incremented if the requester has not already voted on that song.

Given that a song vote request was dispatched to the server and the song exists in the playlist, server will increment that song's votes by 1 and respond with acknowledgement of addition.

Demo: facilitator WebIT & live on song.buzz
13. **Expose facilitator over HTTPS**  As a client I want to be able to communicate with the server over HTTPS so that the communication channel is encrypted and follows best practices.
   Given that there is a client and a server, the client will be able to make requests to the server over HTTPS.
   Demo: load balancer exists and API can be accessed at https://api.song.buzz

14. **Server Receive Song Proposal**
   As the server (facilitator) I want to respond to song proposal requests so that a venue playlist will have the new song added to it and its votes set to 0.
   Given that a song proposal request was dispatched to the server, server will append the song to its playlist, allow users to vote on it, and respond with acknowledgement of addition.
   Demo: facilitator WebIT & live on song.buzz

15. **Client Propose Song**
   As an audience user I want to interact with my client to dispatch a song proposal request so that I can submit my song to the queue where other audience members can vote.
   Given that there exists a venue, when user selects song and clicks propose, a request will be dispatched to the server and the client will await results and render acknowledgement of receipt.
   Demo: facilitator WebIT & live on song.buzz

16. **Client Upvote For Songs**
   As an audience user I want to interact with my client to dispatch a song vote request so that I can upvote songs and increase their weight in the queue.
   Given that there exists a venue and the song exists in the playlist, a request will be dispatched to the server and the client will await results and render acknowledgement of receipt.
   Demo: live on song.buzz

17. **Client Unique Identity**
   As a client I want to be able to identify myself uniquely to the server during interactions so that I can associated myself with venues, tracks, and votes.
   Given that there is a user on a device, user will have a unique identify when sending requests to the server. Demo: live on song.buzz

18. **Server Handle Client Uniqueness**
   As a server, I want to be able to associate a client with venue, track, and voting actions so that I can uniquely identify them and limit actions to being done only once and identify them to other users.
   Given there is a request incoming to the server with client identity information associated with it, server will attach that identity to the action and prevent duplicate actions.
   Demo: facilitator WebIT & live on song.buzz
19. **Server Receive Song Downvote**

As the server (facilitator) I want to respond to song vote requests so that a venue playlist will have the corresponding song's vote decremented if the requester has not already voted on that song.

Given that a song vote request was dispatched to the server and the song exists in the playlist, server will decrement that song's votes by 1 and respond with acknowledgement of subtraction.

Demo: facilitator WebIT & live on song.buzz

20. **Client Song Voteskip**

As an audience user I want to interact with my client to dispatch a song voteskip request so that I can vote to skip the current song being played.

Given that there exists a venue and a song is being played, an request will be dispatched to the server and the client will await results and render acknowledgement of receipt.

Demo: live on song.buzz

21. **Venue Host Name**

As an audience user I want to see the name of the host of a venue so that I can be sure I am joining the right venue.

Given a venue exists and an audience user lists venues, the audience user will see the name of the venue host.

Demo: facilitator WebIT & live on song.buzz

22. **Client Downvote For Songs**

As an audience user I want to interact with my client to dispatch a song vote request so that I can downvote songs and decrease their weight in the queue.

Given that there exists a venue and the song exists in the playlist, an request will be dispatched to the server and the client will await results and render acknowledgement of receipt.

Demo: live on song.buzz

23. **Containerize facilitator**

As the facilitator application, I would like to be containerized and set up to run on GKE so that updates can be easily pushed, CI can build and deploy new versions, and replica count can be updated easily.

Given a GKE cluster and a facilitator application, the facilitator application will be containerized and run on GKE.

Demo: Travis-CI for facilitator builds docker image

24. **Containerize frontend**

As the frontend application, I would like to be containerized and set up to run on GKE so that updates can be easily pushed, CI can build and deploy new versions, and replica count can be updated easily.
Given a GKE cluster and a frontend application, the frontend application will be containerized and run on GKE.
Demo: Travis-CI for frontend builds docker image

25. **Server Receive Song Voteskip**
   As the server (facilitator) I want to respond to song voteskip requests so that a venue playlist will have the voteskip count for the current song incremented by 1 and skip song if necessary.
   Given that a song voteskip request was dispatched to the server and a song is played, server will increment song voteskip count by 1 and skip song if voteskip threshold was reached.
   Demo: facilitator WebIT & live on song.buzz

26. **Host Resume Song**
   As a host user I want to interact with my client to resume the current song so I can resume playback of music.
   Given a song is paused, the audience user will be able to resume the current song.
   Demo: live on song.buzz

27. **Host Seek Time**
   As a host user I want to interact with my client to change the current time of the song so I can skip back or forward.
   Given a song is playing, the audience user will be able to seek the current time of the playing song.
   Demo: live on song.buzz

28. **Audience Current Time**
   As an audience member I want to see the progress time of the currently playing song so that I can know how long until the next song is played.
   Given a venue exists, the audience user will be able to see the current time of the currently playing song in the venue info menu.
   Demo: live on song.buzz

29. **Close Venue**
   As an host user I want to interact with my client to end my venue so that other audience users know that the venue is over and can no longer vote on tracks.
   Given that there is a venue that a host is the creator of, if the host requests to end the venue, the venue will be deleted and users notified.
   Demo: live on song.buzz

30. **Facilitator Testing**
   As a developer, I want to be sure the facilitator application is free of bugs and works as expected so that in production, users can access it the way that is meant to be accessed.
Given the application can be built, integration tests will be run against it to make sure all works properly.
Demo: test suite defined in

31. Currently Playing Song
As an audience member I want to see the cover image of the currently playing song so that I can know if the music being played at a venue is that of my taste.
Given a venue exists, the audience user will be able to see the image of the currently playing song in the venue list menu.
Demo: live on song.buzz

32. Frontend Testing
As a developer, I want to be sure the frontend application is free of bugs and works as expected so that in production, users can access it the way that is meant to be accessed.
Given the application can be built, integration tests will be run against it to make sure all renders and works properly.
Demo: React tests defined in
https://github.com/cs48-next/frontend/blob/master/src/App.test.js

33. Client Play Song
As an host user I want to interact with my client to play songs so that I can actualize the requests of audience users to listen to their selected song.
Given that there is a venue that a host is the creator of, if there is a song next in the playlist, the host client will fetch and begin to play the song.
Demo: live on song.buzz

34. Stats
As an audience member I want to see stats about venues such as: total songs proposed, total up/down votes, total voteskips, total songs skipped, total unique voters so that I can have a better understanding of how my peers are interacting with a venue.
Given a venue exists, the audience user will be able to see the aforementioned stats on venue display.
Demo: live on song.buzz

35. Server Receive Playlist Pop
As the server (facilitator) I want to respond to playlist pop requests so the client will be able to get the next song in the venue playlist.
Given that a song playlist pop request was dispatched to the server and a venue exists, server will pop and respond with head of playlist.
Demo: live on song.buzz & integration test in WebIT
36. **Host Pause Song**
   As a host user I want to interact with my client to pause the current song so I can stop playback of music.
   Given a song is playing, the audience user will be able to pause the current song.
   Demo: live on song.buzz

37. **Demo**
   As the next team, I want to be able to know how to present to the class so that my demo will be AWESOME!
   Demo uploaded to youtube: https://www.youtube.com/watch?v=ha0fkMNc-_s
10 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