Updated 5/22 - all changes are highlighted

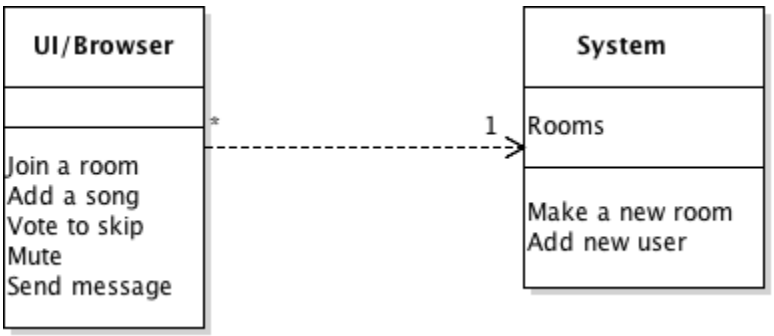
SharedMusic

Eric Nhan - ericnhan
Adam Stephenson - jasteph
Keith Yang - keith619
Kevin Fan - kpyfan
Gunnar Onarheim - onarhg
Reggie Jones - reggiej7
Svetlana Grabar - sgrabar
Tanner Coval - tcoval

System Architecture

Customer View

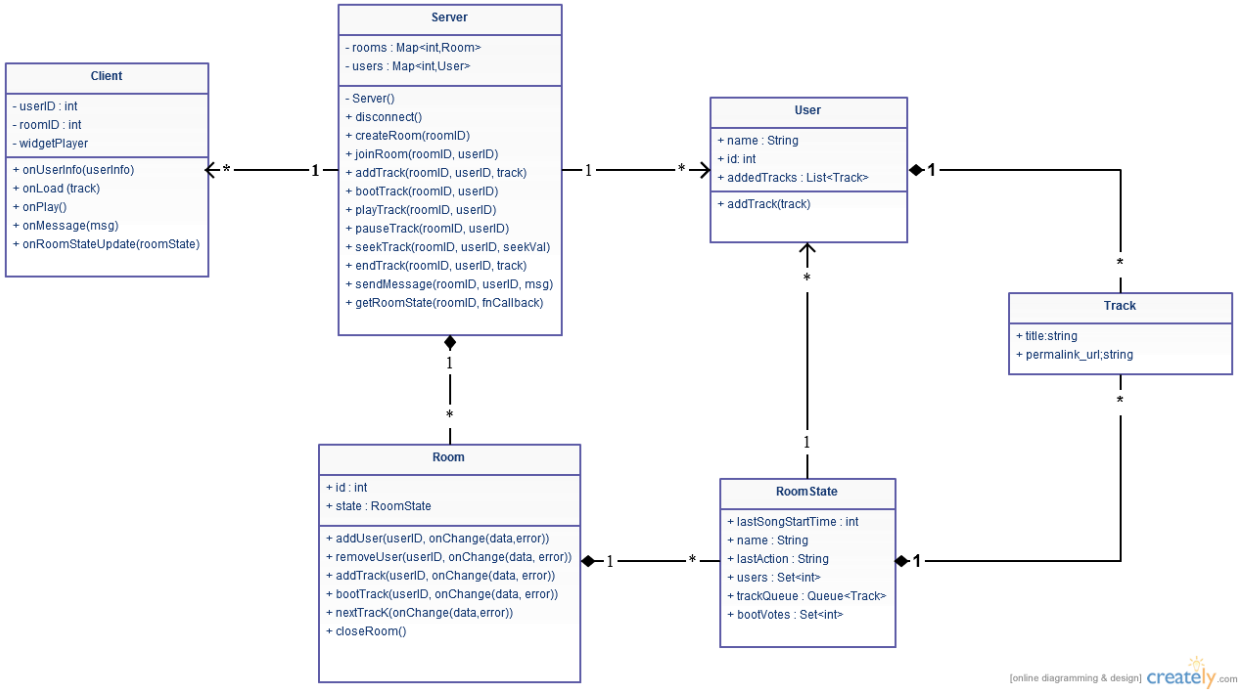
A customer view of the application contains two main components, the user interface (browser) and the system. The user interface displays a room which is kept track by the system and is updated whenever a new user joins. The browser updates the room whenever a song changes or a button is pressed.



Developer View

Major modules we will have in our architecture are client, server, user, room, room state, and track. Currently all the data is stored by the client and the server. The client holds userid and roomid for the server to identify. The client communicates with a server which contains information about rooms and users. These rooms have data about room state. The room state module contains information about users and tracks. The users contain a name and id.

The main function of the client is to provide an interface to connect the server to the user. The main purpose of the server is to manage all of the rooms and updating clients when room states change. Both server and room state contain user information to be used for verification.
The main changes that have been made for the beta since the original document have been the adding more modularization to the implementation of the client. We now have modules for playing the music, searching for songs, and socket management. This was made because it allowed for easier separation of tasks. The main change since the beta release is actually streamlining all elements of the system architecture to work together. Because we now have components for the back end and front end as well as UI we are now ensuring that all of these pieces work together with the desired functionality. Though we have fixed some bugs and implemented features there have not been many major changes to our system's architecture.



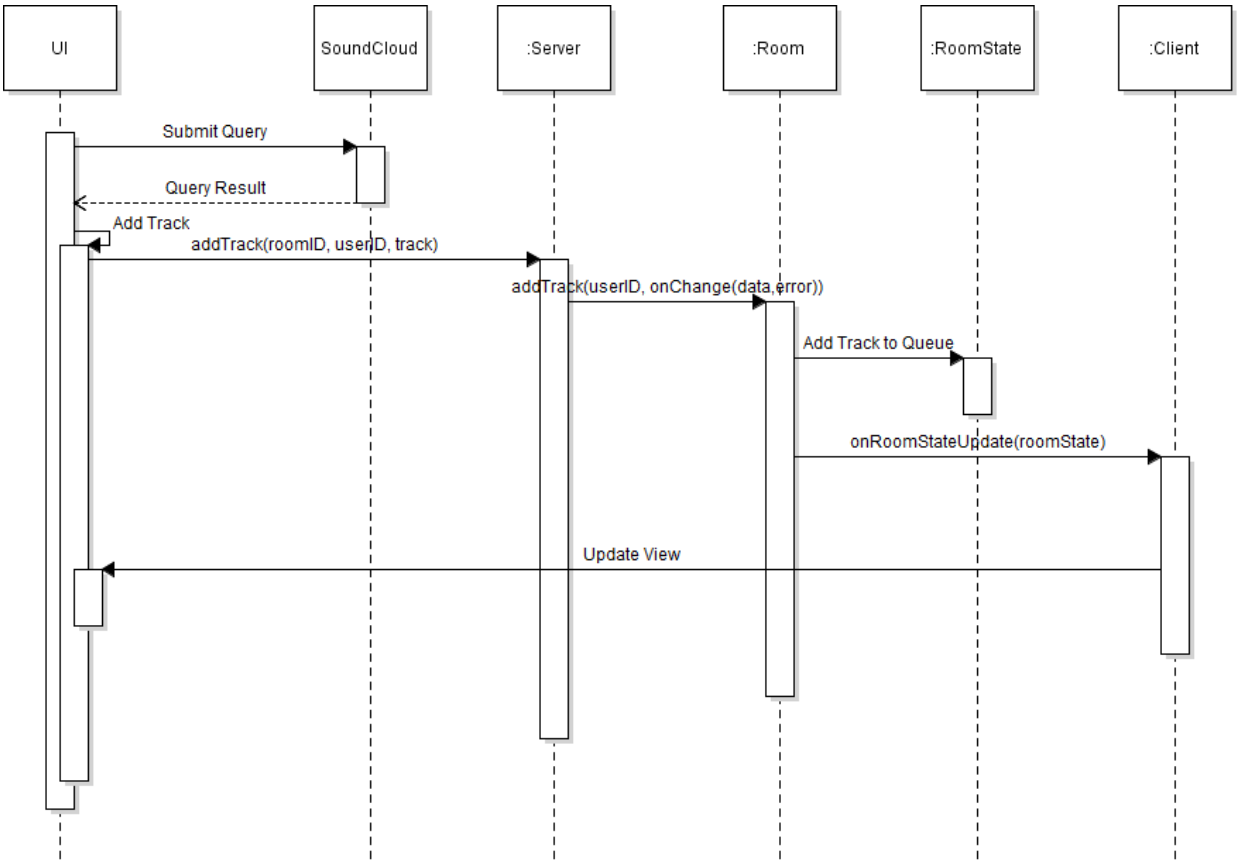
Alternative Design Decisions
One design decision that we made was between allowing our users to pause/play or having a continuous stream of music instead. What we chose was a continuous stream because even though most music players have some option to stop a song, we felt that this would interrupt the group atmosphere that we wanted and a user will still be able to mute a song or vote to skip.

Another design decision that was discussed was how we would have the initial user enter a room. The first way was to require the user to enter a name before starting a new room, the

other was creating a room and then having the user provide a name. The one we decided to use was to create a room and then ask for a name that way we can treat the initial user the same as any other new user joining the room.

Use Case Diagrams

Case 1: Searching and adding a song

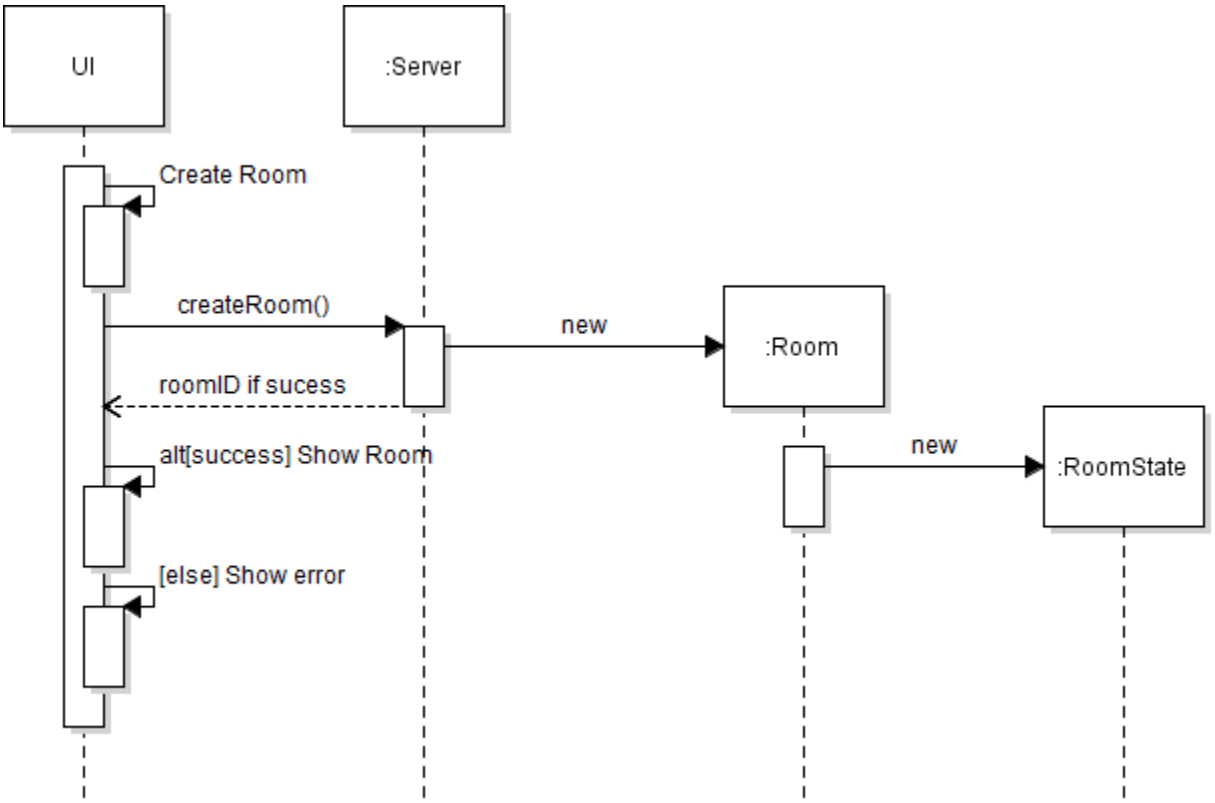


Pseudo-code:

query sent to soundcloud
results displayed in view
user selects track to add
server receives message addTrack(roomID, userID, track)
server finds room with roomID and sends message addTrack(userID, track)
track is added to queue in the room's roomstate
client is notified of a change in roomstate

update is reflected in view

Case 2: Creating a room



Process

Risk assessment

Our most serious risk involves the time it takes to learn new languages. Based on our architecture, we will be working a lot in Javascript, which not everyone is familiar with. This is guaranteed to be an issue. This is predicted to have a low impact because we have team members with Javascript experience already, which means less people need to learn it. The team members with Javascript experience also have agreed that it is not a particularly difficult language to pick up, so the time required to gain a basic understanding should not be too high.

A second risk for us is feature creep. There are many small features that we would like to eventually have as a part of our product. However, pursuing too many of these initially can easily cause us to fall behind. This can potentially have a high impact on project completion. In general, we have adopted the philosophy of creating a minimum viable product first, then adding features based on the amount of time we have left and the priority we give to each feature. We believe this has reduced the risk of feature creep occurring to a low level.

A third risk for us is usability issues. This has a medium chance of occurring. In terms of actually turning something in for our project, this does not have a particularly high impact, though it has a medium impact on having a strong, usable product. We are mainly concerned about having usability issues due to the fact that we will not have very much time for usability testing. We believe that our group reaching consensus on usability decisions as well as feedback from our TAs should be enough to mitigate this risk. If we still have usability issues, we will seek additional help in redesigning.

Another major risk is the proper synchronization of music across multiple clients. This is somewhat of a technical challenge as our product needs all of the clients in a given room to be roughly in sync. Not being able to make this work has a very high impact on our project completion. We feel that this has a low likelihood of occurring as the APIs we are using should allow us to synchronize clients. In addition to this, we are building in some features that should allow us to keep all clients within a small window of acceptable desynchronization.

Finally, we believe that we are at risk of not properly building for expansion. As we have decided to go with a schedule based around creating a minimum viable product and then adding to it, we feel that there is a strong likelihood that there will be portions of the project that were not properly built to be added to and iterated upon. We believe the total impact of this problem is fairly small as there are more things we would like to do this quarter than we can feasibly accomplish so there will be other things we can attempt if we cannot add a specific feature. That being said, we are trying to mitigate the likelihood of this occurring by knowing ahead of time what features we would like to implement in which order, which allows us to plan ahead for what we need to leave open to expansion.

These risks are mostly the same as from our SRS. However, we have added some new risks that were not present there. We feel that at the time of the writing of our SRS, we had adequately planned for the existing risks.

Project Schedule

External Milestones
Minimum Viable Product Beta Testing Release - May 15th
Minimum Viable Product Release - May 22nd
Final feature Beta release - May 29th
Final feature release - June 2nd

Internal Milestones
Room creation and sharing - May 1st
Room joining and leaving - May 5th
Song search, queue, and playing - May 15th
Testing client side code
Integrating server and client side code
Testing end to end
100% Green on Tests
Refinement from beta to MVP release - May 22nd
Additional feature additions - May 29th
Refinement from feature beta to final release - June 2nd

These internal milestones include both testing and development as we plan to have testing developed alongside feature development. All internal deadlines include testing being finished for the features in question. The deadlines are in order of dependency. The first must be completed before the second, and so on and so forth. At the point where we are adding additional features, each feature will be dependent on the MVP being there, but there is not necessarily any dependency between additional features. Effort estimates are the basis for the time between each internal milestone and as such indicate the relative effort we feel development and testing for each portion should take.

Since the beta release, there has been significant work done on all aspects of the project. One of our short-term goals at this point was to get all tests that we have written to pass. This is something that we were able to accomplish prior to the feature-complete release.

Team structure

Our overall project manager will be Keith. He was the one who came up with the original idea, and as such, we feel that he has a strong vision of what our product will be. He also has been core in getting meetings and other coordination technologies set up. We feel that he will fit well in a role that helps to drive the group as a whole.

Each team member has selected a technology or series of technologies that they are interested in. These are technologies that we expect each group member become proficient in. When the team member is working on a feature, they will be responsible for the portion of the feature that corresponds to their technology. That being said, each member has the freedom to also help out and work on other portions of the project as needed. The roles are defined so we have a concrete number of people working on each thing and so that we have leaders for each portion of the project.

The technologies are as follows:
MongoDB (Gunnar)

ExpressJS (Tanner, Keith)

AngularJS (Kevin, Svetlana)

NodeJS (Adam, Tanner, Reggie, Keith, Eric)

The frontend technology is primarily AngularJS with a possibility of other UI libraries included. This means that Kevin and Svetlana are spearheading the development of our frontend. Adam, Tanner, Reggie, Keith, and Eric will be working on the ExpressJS and NodeJS backend portions of the project. Gunnar is responsible for anything database related, though we do not currently have too much planned that requires the database. As such, he is also treated as a flexible member that can jump in to any task where needed.

After most of the server implementation being completed by Adam and Tanner the group got rearranged with Keith and Eric have moved to the client-side code to help Kevin. We have decided to not include a database so Gunnar is now implementing tests for the server code. This initial division of the group did not work out as intended and the work was not evenly distributed.

For the feature-complete release we rebalanced some of the project workload among everyone to ensure that tasks would be completed on time. Tanner's task included both client-side and server-side work during this section. Adam also did more client-side work in this section. Reggie worked on testing for the front end and code coverage. Other than these changes, the roles stayed fairly similar to what they were previously.

As each task and milestone on our schedule involves both frontend and backend technologies, we expect that everyone will have a role in each portion of the project. The frontend and backend portions of each milestone will be developed simultaneously. With proper communication and modularization, there should be no issues with the way the frontend and backend mesh together.

Our team will communicate primarily through our Google groups email list. We also have a Facebook group that our team members have access to for smaller, less official notifications. Our weekly meetings are schedule for Wednesday at 12:30 pm and a backup meeting time of Friday at 12:30 pm has been scheduled as well. Should we need to, we can use the Friday time

to hold a second meeting each week. Additionally, we can easily organize further meetings between subgroups of the team for each feature.

Test plan

- Unit test strategy
    - Unit tests for all the back end code. The purpose of unit tests is that they usually result in loosely coupled modules.
    - The tests will be developed individually. Each person is responsible for writing unit tests for the code they commit.
    - We plan to set up CircleCI for continuous integration.
- System test strategy
    - We will implement end to end integration tests. The purpose of this is to emulate user scenarios of how they will use the product.
    - These tests will be developed as full features get implemented. We don't want to initially denote someone a "tester" role, instead we will all share the load--but that may change that down the road if need be.
    - Again, these will be frequently run via Circle. We suppose once per night will suffice, but will make changes as we fill necessary.
- Usability test strategy
    - The purpose of usability tests is to test the user experience.
    - These tests will be developed by exploiting our friends to get feedback on the usability. Since many of them are tech savvy, this should give us good insight on our layout and design.
    - These will be run near the later portion of our project as we are getting our UI and functionality in place.
- We think our testing strategy is adequate as we will be able to ensure 100% passing tests with help from outside frameworks and tools. We have decided we will use Jasmine for testing as it appears to be the most common in the JavaScript community.
- For a bug tracking mechanism, we will be using Trello. This will allow us to have lanes of "new bugs", "fixed", and "confirmed" so we are able to see the bug through the lifecycle.

Bug tracking has moved more to just using the one available on GitHub.

We have implemented adequate testing for the server and still need to have front end tests before we implement end to end testing. Some of the front end unit testing that we have decided to use is mocking user input to make sure correct calls are made and correct information is displayed.

For the feature-complete release we still need more testing for the client-side. However, we have a functioning CircleCI build that tests our code whenever we push to the repository. Additionally, we have gotten to a point where all of the tests we have written pass when the system is built by CircleCI.

Documentation Plan

We will be providing a help page on the website with instructions on the general use cases such as creating a room, joining a room, and adding songs to the playlist in the room. This will help new users quickly learn how to access the basic functionalities of the site. An admin guide will be provided that documents how to install and run the website on the server. If we expose our API in the future, man pages will also be provided.

Coding style guidelines

Languages:

- JavaScript: http://javascript.crockford.com/code.html
- HTML: https://make.wordpress.org/core/handbook/coding-standards/html/

To enforce these standards and legibility of code, we will be making sure at least one other teammate gives a code review before merging the pull request into the master branch. If we find this becomes a problem than we plan to implement more forceful and rigorous code reviews before committing. However, since JavaScript has pretty well defined and known coding standards, we do not foresee this being a big issue.