OurMusic Documentation

Members
Haralabos Karayiannis (karayia2)
Fidel Trejo (fatrejo2)
Ralph Berrabah (rberrab2)
Daphne Agustin (dagust3)
Jacob Strafford (straffo2)
Rupali Raju (rdraju2)
John Terwilleger (jterwil2)

Table of Contents

Project Description

Application Overview

OurMusic is social music web app that allows users to create and join virtual online spaces called rooms. Inside rooms, users can search for music videos and add them to the playlist queue, which plays synchronously for all the people in the room. Songs in the playlist can be upvoted or downvoted to give a track higher or lower preference in queue or remove it entirely. As a collective group, the users in the room can select certain songs to be the room favorites; when no track is queued by a user, the room plays songs based on the collectively favored songs or recent songs played in the room. A chat window in the room also enables people in the room to converse and share their thoughts.

Motivation

OurMusic sets out to socialize the music listening experience. We feel that socializing with friends and listening to music are a given among college students, teenagers, and young working adults. At any given point in the day we have the opportunity to listen to music - whether it's walking to class, driving back from work, doing homework, or cooking dinner with your favorite tunes - but often are unable to do so with friends due to busy schedules. This is where the idea arises, what if we can combine them? Sharing music can be tedious, especially when you share a single song from one of your random playlists to your group of friends. Sure you can send the YouTube link through a Facebook group message, but why not just pick the song and let everyone listen without anyone having to click on a single link. To address these needs, OurMusic was conceptualized so the music listening experience can be socialized.

Process Followed by Team

Why Scrum?

Our group preferred Scrum over XP which we used last semester for several reasons. Perhaps the single most important reason our group prefers Scrum is that it stresses daily communication and involvement in the project. In a project of this scale where the code base is being built from the ground up, it is highly important to be up to date on what everyone else is working on. Daily check ins force developers to do more regular scheduled daily work or maybe help developers realize their inactivity. It is also a great way for team members to know what every member is working on and help hold them accountable. In addition, another reason that our group prefers Scrum is that teams work in sprints. Having face-to-face sprints will force developers to allot time to work on the project and also make it easier for developers to get help on technical issues from other team members. Lastly, we chose Scrum because it does not necessitate the strict use of any particular engineering practices like XP. Teams can independently choose if they want to use pair programming or use test-driven development. It puts trust in developers and teams to choose and implement their own best engineering practices. Our team prefers to have the ability to decide whether or not we should apply features of XP in our development like code review and extensive unit testing. Because of this freedom and because of Scrum's stress on daily progress, we believe Scrum will maximize our group's ability to create efficient, working code.

Modified Scrum, Refactoring, Testing and Collaborative Development

As stated, our group adopted a modified version of Scrum. Initially we chose to have one mandatory in person meeting for the whole group on Tuesdays at 6 PM and three weekly check-ins on our wiki by midnight on Monday, Wednesday, and Friday. However, we found that three weekly check-ins were too much and that members did not have enough updates to commit so we further reduced this to Wednesday and Saturday. We found this to be fairly effective in helping us maintain iterative and collaborative development. Furthermore, our team split into backend and frontend teams. The backend team had four people (Fidel, Haralabos, John, and Jake) and the UI team had three people (Ralph, Daphne, and Rupali). These teams met outside weekly group meetings to do sprints, fix bugs, and collaborate on details. There were no regular meeting times for these sub groups and sprints were arranged out of necessity.

No formal refactoring guidelines or formal code review practices were followed, but refactoring was done in each iteration to get unused, duplicate, and broken code. Each member looked after their and edited their code accordingly. Unit tests and jQuery tests were done for each of the classes on the backend and manual testing was done for the frontend.

Requirements and Specifications

Introduction to Use Cases

OurMusic deals with a heavy amount of user interaction. Those that are using the application are doing so through the music rooms, where they are communicating with each other via the chat room. In addition, users are sharing music with each other by adding YouTube videos of their music through the queue. All the users in the room can see the queue and are thus aware of what videos are playing currently and will be playing next, which is how the application "communicates" with the user, per-say.

Several forms of this communication and interaction is evident in our use cases that are implemented in the project.

To begin, users must first have an account in the application. The use case "Log In" covers this requirement. A user can log in on the page and is taken to a menu from which they are able to resume a previous room session, join a room, or create a new room. There are fields on the log in page which allow the user to enter their username and password. The user can then click the submit button to log in. If the user does not have an account, the user clicks the "Register User" link and is taken to a short form to create an account:

Use Case: Register User

- **Primary Actor:** User
- **Goal in Context:** Logging in to the website
- **Scope:** System
- **Level:** Sub-Function/User goal
- **Preconditions:** User (1) has an account with the system and (2) is not logged in.
- **Stakeholders and interest:**
  - *User: wants to log into our system*
- **Trigger:** User clicks the "Submit" button
- **Main Successful Scenario:**
  - User enters username into the appropriate field
  - User enters the matching password in the appropriate field
  - User clicks the submit button
  - User is logged in to the system
- **Alternative Flows:**
  - Alt1: Incorrect Username or Password
    - User enters a username and password into the fields that do not match
    - User clicks submit

        - Log in attempt fails and user must retry
    - Alt2: User doesn't have an account
        - User clicks the "Register User" link and is taken to a short form to create an account

As mentioned before, all forms of user interaction take place in rooms, but in order for said rooms to be used for such communication, they must first be created by the user. The use case “Creating Room”, covers the requirements and expectations out of fulfilling this use case.

In this use case, a user selects the option to create a new room from the main menu. The user will enter information into the form such as the name of the room, specify the settings as private or public room, enter optional genre tags, and additionally have the option to invite other users to the room. They will click submit which will successfully create a room with an empty playlist, where the user creating the room will be the room administrator.

Use Case: Creating Room

- **Primary Actor:** User
- **Goal in Context:** Create a room with an empty playlist and set basic permissions, set tags, send user invites.
- **Scope:** System scope
- **Level:** User level use case
- **Stakeholders and interests:**
    - Users: want to create or join a virtual place for multiple people to create a playlist.
    - Room Administrator: want to have control of the newly created room to moderate quality.
- **Precondition:** User is logged into system.
- **Minimal Guarantees:** Room will be created according to default settings unless user specifies differently in room creation form. New room will have empty playlist to add music to.
- **Main success scenario:**
    - **Triggers:** User clicks "Create Room" button, fills out form, and clicks "Submit" button
        - User clicks "Create Room" button.
        - User fills out name, public/private settings option, sets optional genre tag, and sends out room invites.
        - User clicks submit.
        - User input is validated.
        - System creates new room with empty playlist.

- **Alternative success scenario:**
    - User cancels creation of room
        - User hits "cancel" button
        - User returns to main home screen

The main use of the OurMusic application is for users to be able to share and listen to music with each other. In order to accomplish this, users are allowed to add songs to the queue in the music rooms. After the user logs in and either joins, resumes, or creates a room, the user may add a song to the queue. The song is added to the queue by clicking on the "Search for Song" button which opens a search box to search for a specific song. Once the song has been found, the user clicks "Add to Queue" option.

Use Case: Add Song to Queue

- **Primary Actor:** User
- **Goal in Context:** Adding a song to queue
- **Scope:** System, Spotify
- **Level:** User goal
- **Stakeholders and interest:**
- *User:* wants to add song to the queue in order to listen to it when it is at the top of the queue
- *Other users:* want to listen to music selected by other users
- *Youtube:* retrieves the song
- **Preconditions:** 1) Logged in. 2) created, joined, or resumed a room 3) Is currently in a room
- **Minimal guarantees:** The song the user selected is added to the queue
- **Trigger:** User clicks "Search for Song" button
- **Main successful scenario:**
    - The user clicks on the "Search for Song" button
    - The user then searches for a song
    - YouTube API retrieves matching result.
    - The user selects the song by clicking on the option, "Add to Queue"
    - The system adds the song to the queue
- **Alternative Flows:**
    - Alt1: User cancels search
        - User clicks the exit button the searching window
    - Alt2: Application cannot find the song
        - Window displays "no songs found"
        - User can search for another song

Users also interact with the application by upvoting or downvoting songs that are on the queue. This interaction directly corresponds to how all the users in the room order the priority of songs that are played on the queue. To do this, users click the upvote or downvote button next to a song in the queue. User will only be allowed one vote. If user clicks upvote button, that will give it preference and move it up the queue. If the user clicks the downvote button, that will move it down the queue. The queue will be adjusted in real time according to cummulative user votes. Additionally, if song receives a very high downvote count--which will be calculated by a formula in the system--it will be removed from the queue.

Use Case: Upvoting and Downvoting Songs In A Room

- Primary Actor: Users
- Goal in Context: Upvote and downvote songs in queue to move songs with preference up the queue in real time.
- Scope: System
- Level: User level use case
- Stakeholders and interests:
    - Users and Room Administrator - Both want the ability to only play the most popular music in the room.
- Precondition: User is logged into system and in a room with a song in queue.
- Minimal Guarantees: When upvoting or downvoting songs, queue will be readjusted according to user vote giving songs with more upvotes preference. Songs with high downvote scores will be removed from list.
- Main success scenario:
    - Triggers: User clicks "Upvote" or "Downvote" button
        - User clicks "Upvote" or "Downvote" button.
        - User input is validated and voting choice remains highlighted.
        - System moves song up queue or down queue in real time.
    - Alternative success scenario:
        - User clicks already clicked and highlighted "Upvote" or "Downvote" button
            - Users vote is rendered obsolete

In addition to the user-application interaction in OurMusic, the primary means of communication that the users have with each other is the chatbox. In the chatbox, users enter the text they wish to send to the other users in the room into the text field and click “Send”. The text is then visible to all users in the music room, and users reply to each other in the same manner.

Use Case: Chatting within a Room

- Primary Actor: Users
- Goal in Context: Allow the users of a song to be able to chat together within the room.
- Scope: System
- Level: User level use case
- Stakeholders and interests:
    - Users and Room Administrator - Both want the ability to chat with the fellow room members
- Precondition: User is logged into the room where the chathub is located.
- Minimal Guarantees: User's can send and receive messages from other users.
- Main success scenario:
    - Triggers: User writes a message and clicks "Send"
        - User types the desired message into the text box then clicks the "Send" button
        - Messages will be appended to the top of the current conversation.

Architecture and Design

The architecture of OurMusic revolves around three main components: a user (or an admin), a room, and songs.

Design:

OurMusic is designed to be intuitive. A user can quickly create an account when the user first opens the page. Once the user goes through a simple registration process (which asks for a name, user name, and password) they are redirected to the homepage of OurMusic. The home page serves three purposes: allow the user to view and join open rooms, allow the user to create a room, and allow the user to manage their account. If the user decides to join a room, they can do so by simply clicking "join" next to the name of the room. If the user decides to create a room, the user clicks on a link which redirects the user to another page. This page allows the user to create a name for a room and click on a confirm button which creates a new room. If the user decides to manage their account, they click on the manage account link. This link redirects a user to a page where the user can change their password by entering their previous password and creating a new one.

The room is where the functionality of the site lies. The room allows a user to view a YouTube video player, search music and add it to the queue, view a music queue (which allows for upvoting and downvoting of a song), view other users in the room, and chat with other user using a chat box.

The YouTube player is the bread and butter of the app. The video is determined based on the items in the queue. The first video that is played is a default video but the videos after that are retrieved from the queue.

The queue can be populated by anyone who is currently viewing a specific room. In order to add a video, the user types the name of a video, clicks search, and the Youtube Data API fetches the video title and id to display to the user.

The user can then decide to add that video to the queue, or to search for another video. If the user clicks on add to queue, then it will be added to the queue.

Any video in the queue can be upvoted or downvoted by any user in the room. The more upvotes a video has, the faster it climbs to the top in the queue. If a video is downvoted, then keeps being pushed lower in the queue.

A list of people is shown in the page. This allows other user to know who is currently in the same room as you.

A chat feature is also implemented so that users can communicate.

If the user is the admin of the room (better known as the creator of the room), they have extra functions in the room. The admin can: delete a song from the queue, remove a user, and delete the room.

A song can be deleted from the queue by clicking the x button next to the song in the queue.

An user can be removed from the room by the admin clicking the x next to the user.

The admin can delete the room by clicking on a link that says delete room. Clicking on the link takes the admin to a confirmation page. If the admin decided to confirm the delete, the room no longer exists. Users still in the deleted room will no longer have additional videos playing from the queue as it does not exist.

Architecture:

The live, real-time broadcasting of information and updates necessary to make this app work led us to SignalR, a Microsoft-supported library for .NET web developers that allows bi-directional communication between the server and clients. SignalR was also designed to fit nicely with .NET MVC design patterns, allowing us to delegate different work within the group without breaking each others code.

SignalR utilizes C# `Hub` classes on the server, which manage connections using Web-sockets as well as other compatible technologies for connection management. The key feature of SignalR is that server-side C# code can call functions such as:

```
Clients.Group(roomName).sendMessage(message);
```

That line of code would make all clients associated with the group “roomName” to call a JavaScript function that was defined on page load:

```
rHub.client.sendMessage = function (message) {...};
```

Similarly, client-side JavaScript can call a C# function on the server:

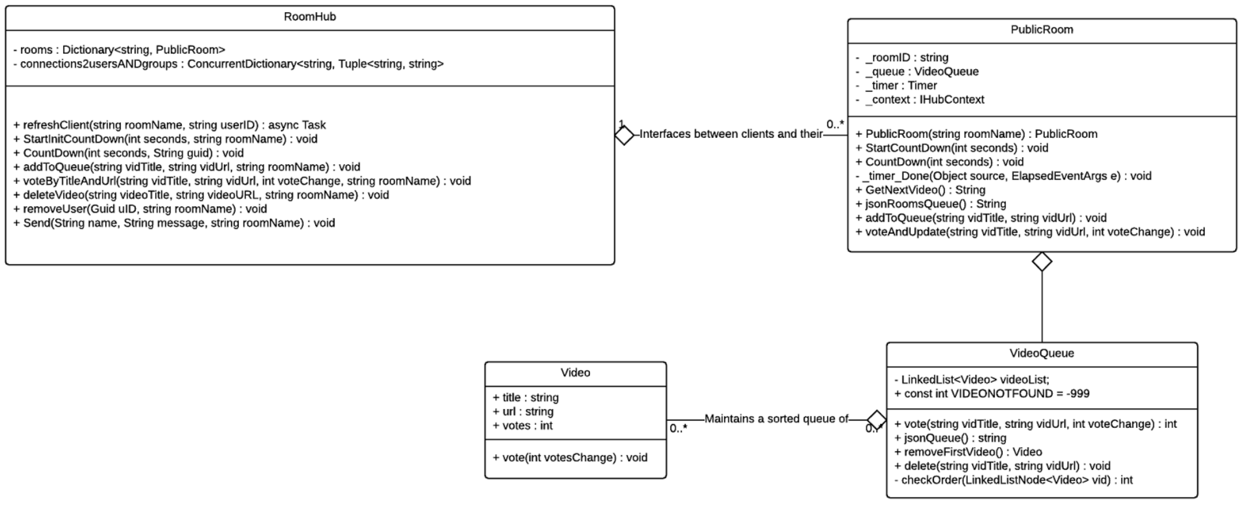
```
rHub.server.deleteVideo(videoTitle, videoURL, roomName);
```

While SignalR does a great job supporting “Group” functionality, SignalR doesn’t store any accessible data mapping users to groups, and leaves that for the developer. Notice that `deleteVideo()` sent the roomName as a parameter. This allows the corresponding C# function to quickly know which group to broadcast that information to. Each C# function only knows a ConnectionID for the calling client, and no more. We need to notify users when somebody in their room disconnects or leaves the room, so we store a thread-safe `ConcurrentDictionary<string, Tuple<string, string>> connections2usersANDgroups` which maps a SignalR ConnectionID to a userID and a roomName. We only use this dictionary for managing `OnDisconnected()` events, because any function called by an active user can pass the roomName as a parameter, which is faster than the lookup.

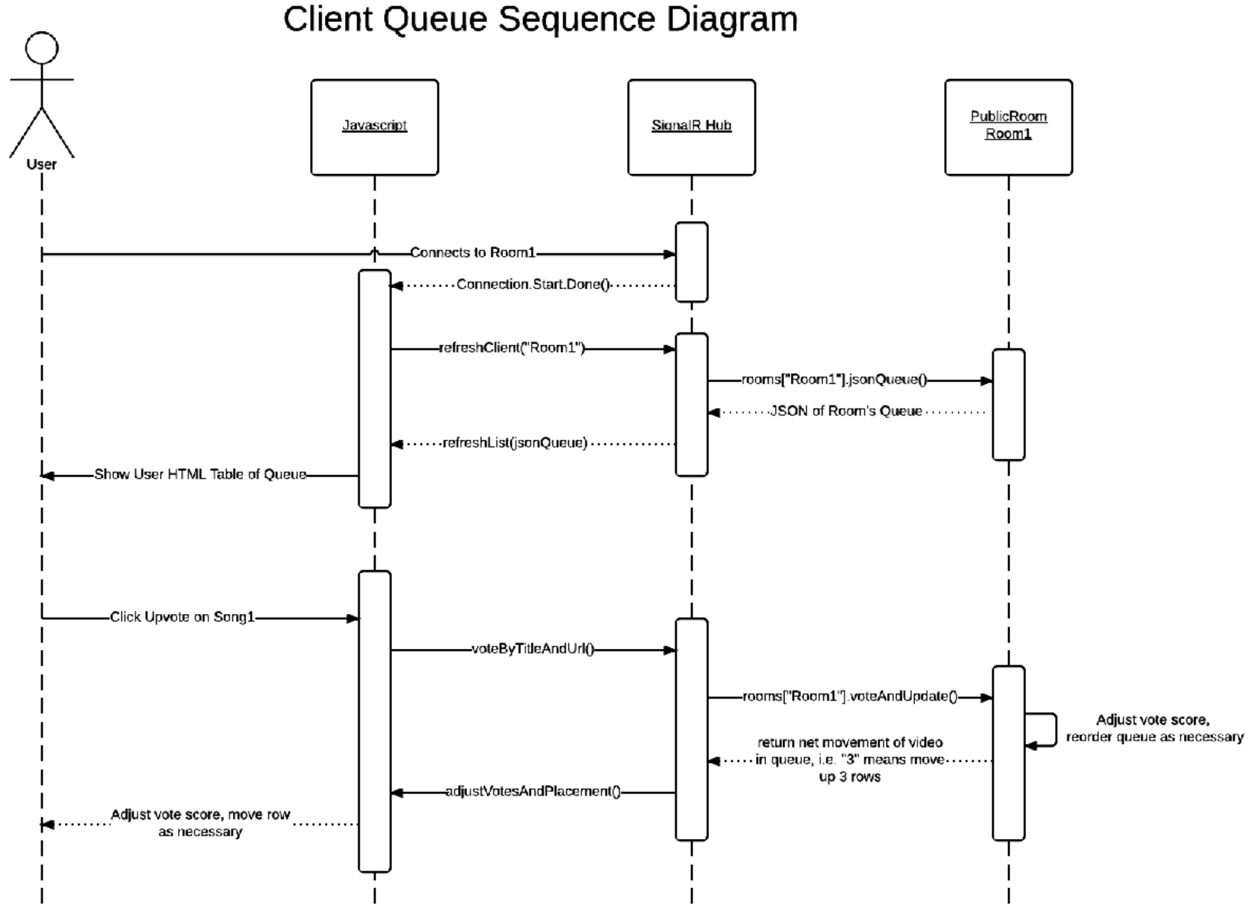
Our initial implementation of the room functionality only accounted for one room, and all functions were broadcast to all users. Since the purpose of OurMusic was multiple separate rooms, we were able to easily manage the `Clients.Group()` functionality, but we then had to introduce a `PublicRoom` container class, which store the queue and separate timer events for each room. Then the `RoomHub` can use a `Dictionary<string, PublicRoom> rooms` on the Hub, to map the roomNames passed as parameters to a `PublicRoom` object.

We have a very quick and responsive web app, that works just as intended. However, all data is stored in server memory, and we do have serious concerns about capacity and scaling ability. We developed the initial implementation of the app under time constraints and we worked with logic that we were relatively familiar with. For OurMusic to facilitate usage on a larger scale, we would definitely need to move much of the data storage to databases, likely non-relational NoSQL databases. We looked into this method initially, but none of us were familiar with the usage, and the learning curve seemed too steep to stay on pace for future iterations. We are happy with the quick, small-scale app that we have though.

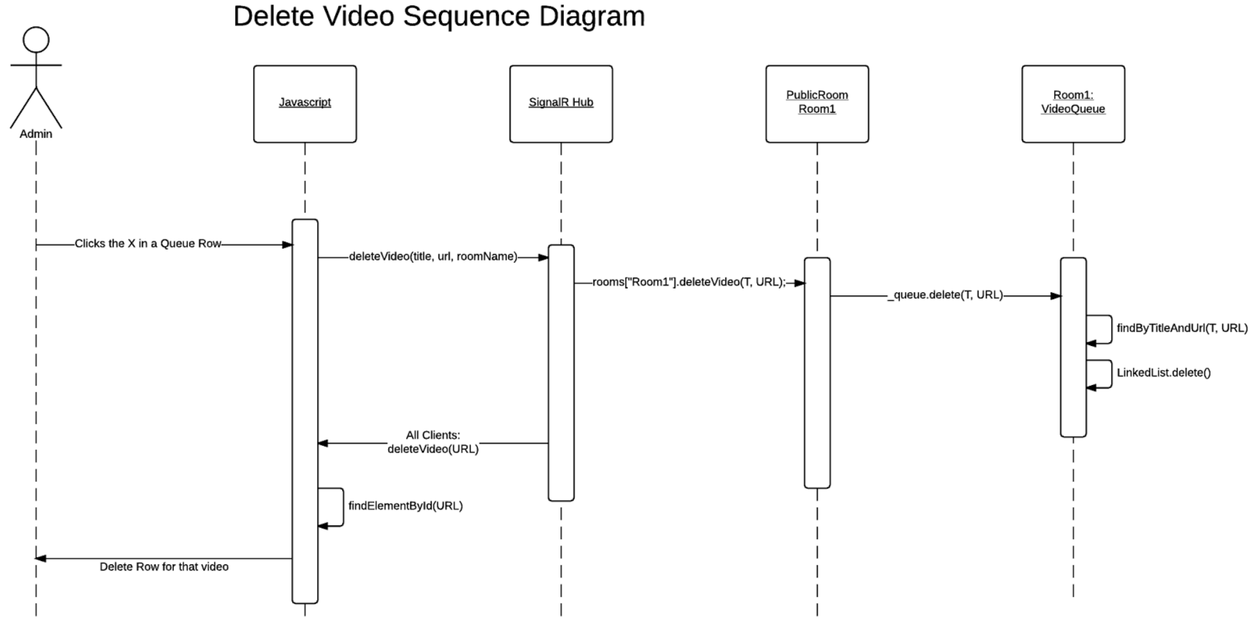
OurMusic Rooms Class Diagram


RoomHub
- rooms : Dictionary<string, PublicRoom>
- connections2usersANDgroups : ConcurrentDictionary<string, Tuple<string, string>
+ refreshClient(string roomName, string userID) : async Task
+ StartInitCountDown(int seconds, string roomName) : void
+ CountDown(int seconds, String guid) : void
+ addToQueue(string vidTitle, string vidUrl, string roomName) : void
+ voteByTitleAndUrl(string vidTitle, string vidUrl, int voteChange, string roomName) : void
+ deleteVideo(string videoTitle, string videoURL, string roomName) : void
+ removeUser(Guid uID, string roomName) : void
+ Send(String name, String message, string roomName) : void
1
Interfaces between clients and their
0..*
PublicRoom
- _roomID : string
- _queue : VideoQueue
- _timer : Timer
- _context : IHubContext
+ PublicRoom(string roomName) : PublicRoom
+ StartCountDown(int seconds) : void
+ CountDown(int seconds) : void
- _timer_Done(Object source, ElapsedEventArgs e) : void
+ GetNextVideo() : String
+ jsonRoomsQueue() : String
+ addToQueue(string vidTitle, string vidUrl) : void
+ voteAndUpdate(string vidTitle, string vidUrl, int voteChange) : void
Video
+ title : string
+ url : string
+ votes : int
+ vote(int votesChange) : void
0..*
Maintains a sorted queue of
0..*
VideoQueue
- LinkedList<Video> videoList;
+ const int VIDEONOTFOUND = -999
+ vote(string vidTitle, string vidUrl, int voteChange) : int
+ jsonQueue() : string
+ removeFirstVideo() : Video
+ delete(string vidTitle, string vidUrl) : void
- checkOrder(LinkedListNode<Video> vid) : int

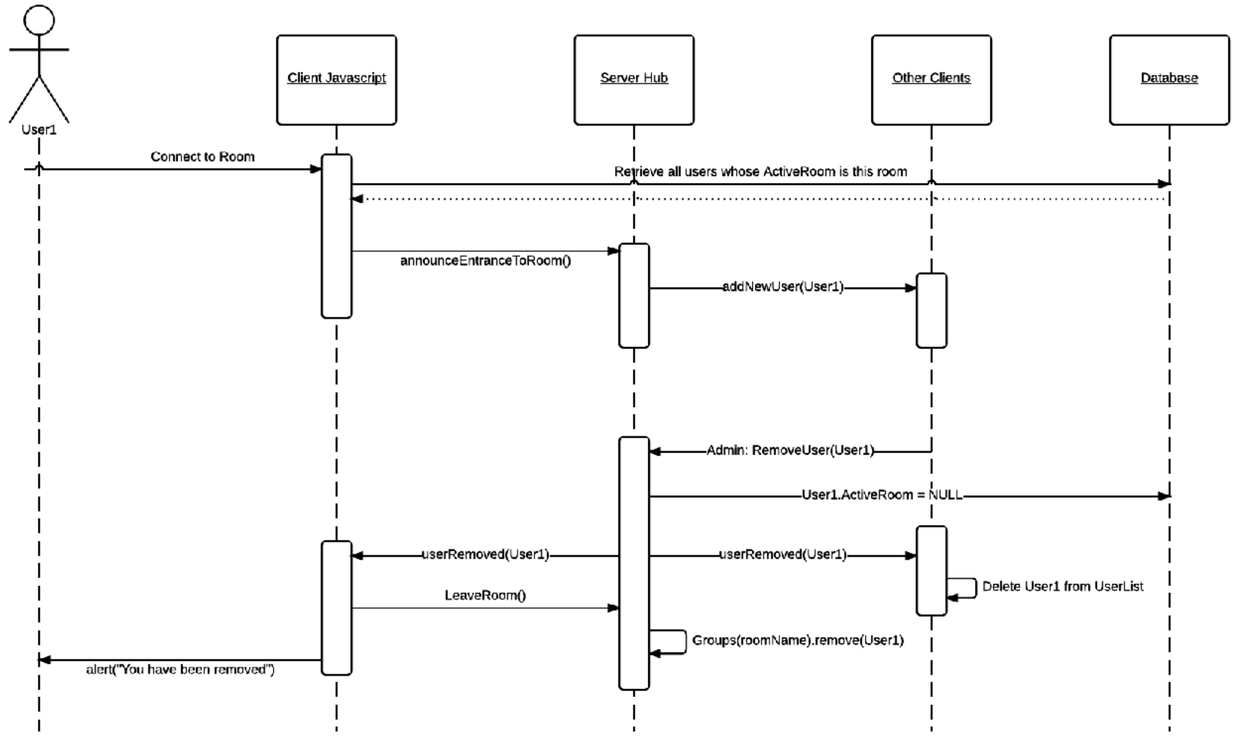
Client Queue Sequence Diagram


User
Javascript
SignalR Hub
PublicRoom Room1
Connects to Room1
Connection.Start.Done()
refreshClient("Room1")
rooms["Room1"].jsonQueue()
JSON of Room's Queue
refreshList(jsonQueue)
Show User HTML Table of Queue
Click Upvote on Song1
voteByTitleAndUrl()
rooms["Room1"].voteAndUpdate()
Adjust vote score, reorder queue as necessary
return net movement of video in queue, i.e. "3" means move up 3 rows
adjustVotesAndPlacement()
Adjust vote score, move row as necessary

Delete Video Sequence Diagram


Admin
Javascript
SignalR Hub
PublicRoom Room1
Room1: VideoQueue
Clicks the X in a Queue Row
deleteVideo(title, url, roomName)
rooms["Room1"].deleteVideo(T, URL);
_queue.delete(T, URL)
findByTitleAndUrl(T, URL)
LinkedList.delete()
All Clients: deleteVideo(URL)
findElementById(URL)
Delete Row for that video

User List and Remove a User Sequence Diagram



Future Plans and Member Reflections

Future Plans

At this time we have no immediate plans to build on our app. However, the technology used in this project gave us a great introduction to a host web technologies which we found really useful and most of the group plans to use in the future.

Haralabos Karayiannis (karayia2)

I'm really surprised that we were able to create a package of this size. None of us all of the technologies thoroughly when we first started. We each knew bits and pieces and needed each other to finish it. I was intimidated by the amount of documentation we had to read to get going on the unfamiliar parts, but once we did everything started moving at a great pace. I'm glad I finally figured out how to use Visual Studio and Github because at the beginning of the project I kept overwriting others' commits and had to revert several times. Having opportunities like this to interact with different types of people is always a plus for me too. Other than that I got some more exposure to SignalR and JQuery which are things I'd like to get to know even more in the future.

Fidel Trejo (fatrejo2)

I absolutely loved working on OurMusic. This is the first time I was able to create an idea from scratch, build it, and actually see it work. To me, this still blows my mind. I know that this would have not been able to be accomplished this if not for the amazing team I worked with. I learned so much about working within a group that had to work remotely for the most part. Since we were all busy students with our other classes and students organizations, we had to communicate almost entirely through text. The only other time we communicated in person was a single in person meeting each week. But this worked out well for us. We managed to keep up with our work and allow other team members to continue their work.

The technology we worked with was technology that I had previously worked with. MVC, .NET, GitHub, Bootstrap: these are all technologies that I had used while I was at my internship at Microsoft. The technology that I had not used but knew about was Signalr. I knew about it because of a multiplayer game that I played while I was at Microsoft. I wanted to discover the capabilities of this technology and that's exactly what we did. I feel confident that I could create another application using Signalr. It's a very handy tool that does a lot of the work for you. Although I will admit that it took way too long to figure it out, it was definitely worth it.

Ralph Berrabah (rberrab2)

I had a lot of fun working on OurMusic this semester. The idea for the project was great and everyone on the team was awesome to work with! Compared with the project in CS427, this project felt "real" in the sense that it gave me an idea of what it would be like to work on a professional project with a lot of developers. In my job I only have to work with one other software developer, so this experience was especially useful for me. I had also never used Scrum before, so implementing it for a project was a definite learning experience for me. I felt that Scrum worked out very nicely for this project, especially considering all of our busy schedules. It allowed us to get work done without having to be in the same room constantly.

I had used (and still use) MVC5, Git, and Bootstrap in depth before this project, but I had never even heard of SignalR until I saw it in the project proposal. At a first glance it sort of reminded me of Node.js. When I started to try to learn it, it gave me a huge headache because of the multi-threaded, asynchronous aspect of it. I had a lot of problems understanding the nuts and bolts of how it worked, but after a while things started to fall into place. A lot of real time web applications will probably start making use of libraries like SignalR, so it was great to get the experience of a new technology like that. I actually have ongoing projects that I am trying to use SignalR with now. Overall, great technology, great team, and I had a lot of fun.

Daphne Agustin (dagust3)

This project really gave me a taste of an application that was more applicable to the things that we would make in an out-of-school setting, like once we graduate and move on to the real world. Compared to a lot of things that I've done in school, this project seemed a lot more applicable to my career, and as someone who wants to go into their tech career specializing in user interface, this project was quite fun to use for practice. The entire concept of the project was just so enjoyable and applicable to my interests as someone who avidly listens and shares music with my peers, and it was very interesting and fun to work on.

I will be the first to admit that I had never used any of the technology that was used in this project before, so as far as utilizing such technology, it was a huge learning curve. As far as working with the UI team for this project, it gave me a good crash course on Boostrap for web development, which was very heavily used in the interface for this project. The things I learned from helping with the user interface was very valuable in being able to learn the nooks and crannies of making the user interface aesthetically pleasing as well as practical and user-friendly.

Jacob Strafford (straffo2)

I'm very proud of what we have accomplished working on OurMusic. The project turned out to be far more difficult that we initially expected, and involved a lot of new technology for all of us. I specifically remember sitting around a table in the Siebel basement with a few team members, each of us scratching our heads, struggling to make any progress on figuring out how the heck we were going to make this work. It was a tense few hours as we sat there frustrated. Hundreds of google search tabs later, the app works.

I had done very little web development prior to taking this class, and I feel completely confident in my ability now. I learned an incredible amount of Javascript this semester. The first few weeks I couldn't get anything to work. I watched a few hours of videos on Javascript, and learned some pretty advanced functionality, such as the benefits and power of the loose types, lambda functions, and prototypal inheritance. I also learned that making DOM logic work is a completely different beast than theoretical Javascript.

I was also new to Visual Studio and .NET development, and after enjoying the progress I made this semester, I will be spending my summer as an intern at Research Park doing .NET development. This class threw me in the deep end of MVC, dynamic server-client architecture, and Visual Studio, and I've learned to swim pretty well in the process.

Rupali Raju (rdraju2)

This project gave me a great introduction into the SCRUM development process. Working with a group regularly over the past semester, gave me insight into a team development effort. Overall, I would say I really enjoyed all the technologies I was exposed to which include Javascript, MVC architecture, and Bootstrap in particular as well as the company of a very dedicated and motivated group of people.

My only regret is that I felt I could have done more for the project. Early on, I was assigned to the UI team and learned Bootstrap which definitely developed my web dev skills. Using that in conjunction with the MVC architecture was a new experience for me. However, there was a disproportionately low amount of work to do for the UI team in the later weeks as the backend had to be implemented. Then when I tried to join the backend team, the learning curve was too high for the amount of time left to complete the project. Looking back I do wish I had been on the backend team so that I could have better spent my time implementing an additional use case. All that being said, I do think this project was a great learning opportunity for me and will help me become a better developer in the future.

John Terwilleger (jterwil2)

Working on this project was great practice for me, because it is exactly what I will be doing for my job after school. This project allowed me to get a good view on how to effectively program in a group and ensure everything has very explicit guidelines so there are no ambiguity between individual's work. The process we used was SCRUM, which was useful so there was documented progress on each part of the project accessible to the entire team.

The programming aspect of the project also taught me a lot. Being a Computer Engineer, I do not have a lot of exposure to the higher level/user interface aspects of programming (html, javascript, etc...). During this project I was able to get good experience with html, javascript, and the different ways to pass variables in between the two and the server c# code. On top of getting more experience with javascript, we used the SignalR library. SignalR is used to allow users to be updated simultaneously. It is a very interesting setup and use of hubs to disperse data, and is knowledge I will definitely use in my career. This project gave me knowledge that isn't taught in my major, but I will definitely use in the future as a Software Developer.