WASHINGTON STATE UNIVERSITY
VANCOUVER

CS 320 Course Project Final Report

for

Sound Recorder

Prepared by

Group Name: Tune Deaf Interactive

Shawn Hillstrom	**11544366**	**shawn.hillstrom@wsu.edu**
Brad Hendrickson	**11515344**	**bradley.hendrickson@wsu.edu**
James Keirnan	**11505377**	**james.keirnan@wsu.edu**

Date: December 13th, 2018

Contents

1 Introduction

1.1 Project Overview

Sound Recorder is a simple, web-based audio mixing tool. The goal of the project is to provide a simple, free alternative to its more expensive, more complicated, non-web-based counterparts. Sound Recorder provides a user with the ability to record and download audio files to each of a total of five channels. The user can play and pause each channel individually or play the mixed result all together. Additionally, the Sound Recorder application provides full documentation of use for new users on its landing page.

1.2 Definitions, Acronyms and Abbreviations

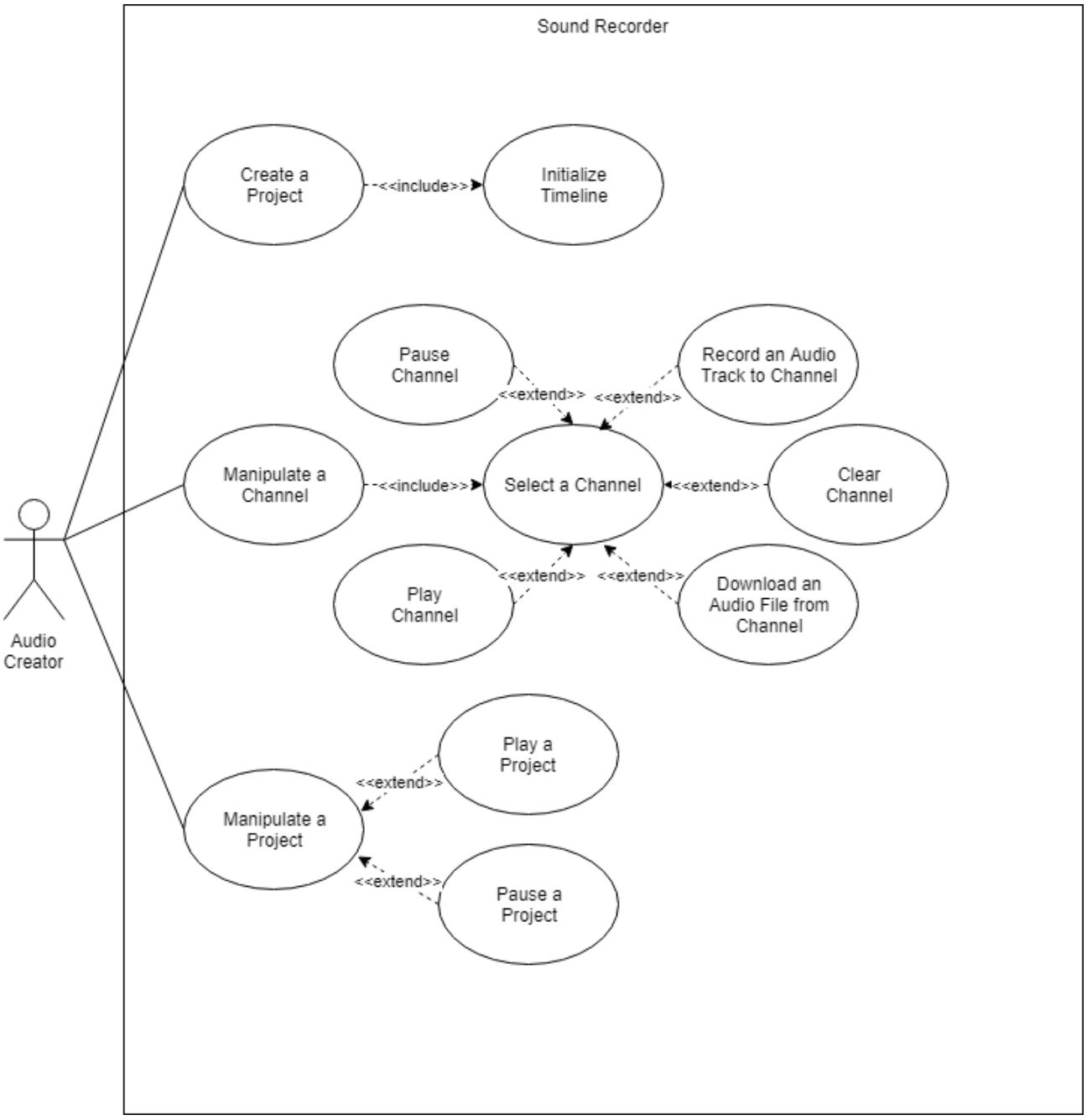
No special definitions, acronyms, nor abbreviations are used in this report.

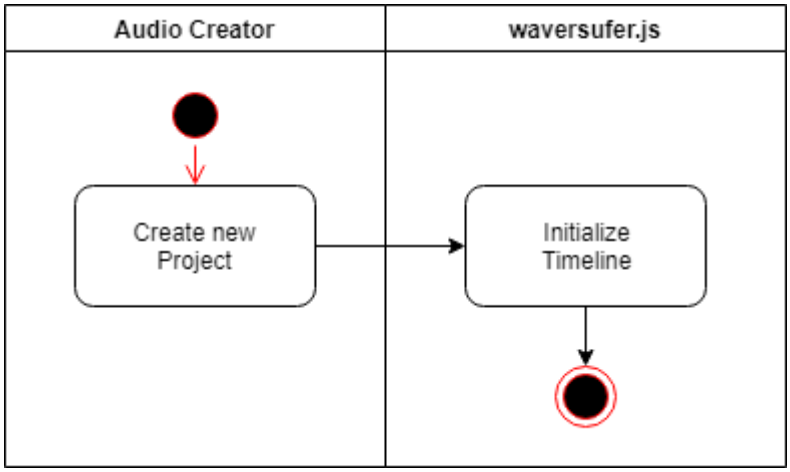
1.3 References and Acknowledgments

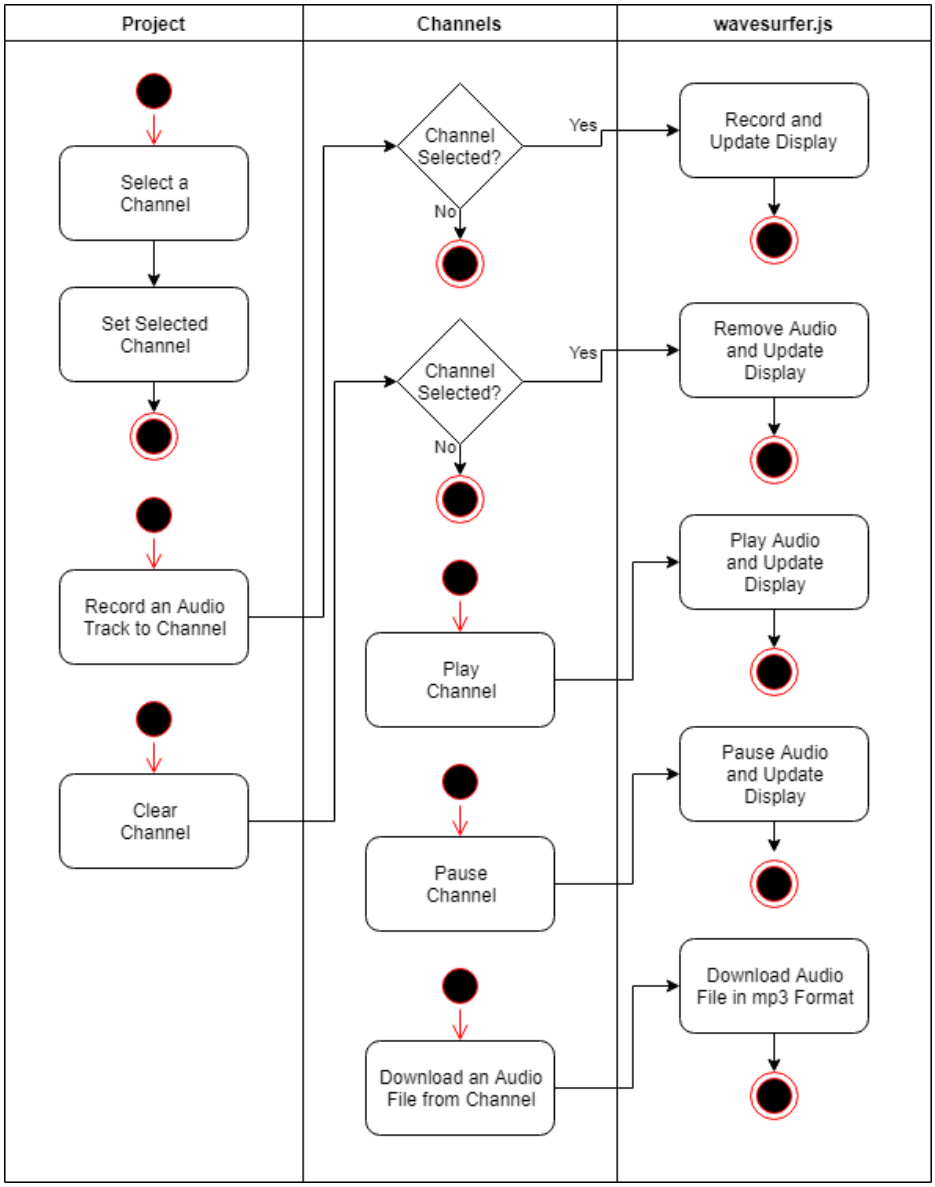
[1] katspaugh, “Overview,” wavesurfer.js. [Online]. Available: https://wavesurfer-js.org/. [Accessed: 03-Dec-2018].

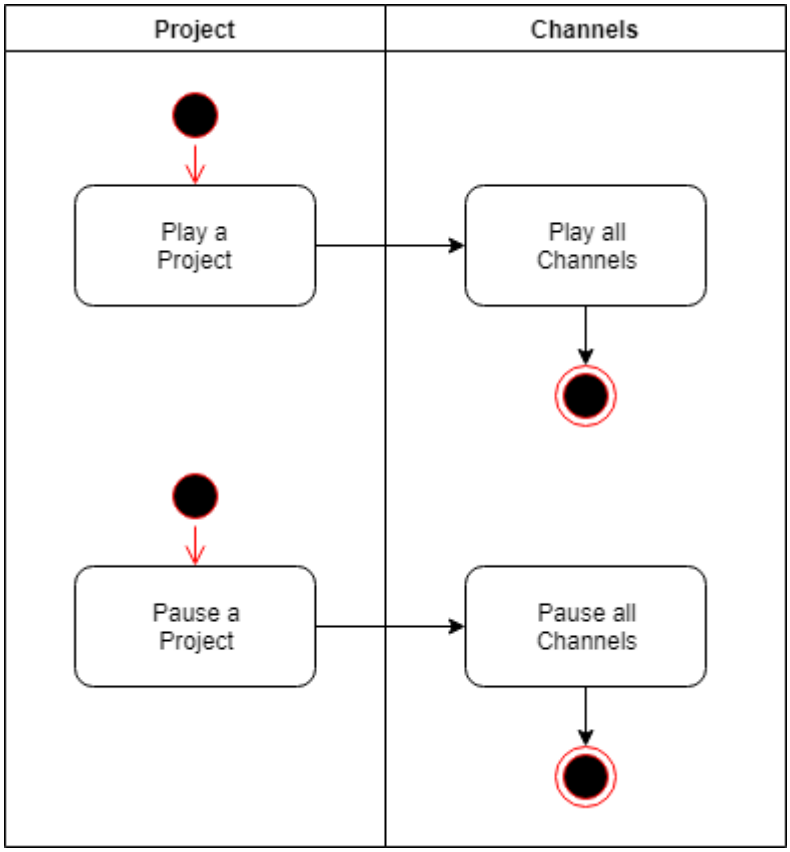
2 Design

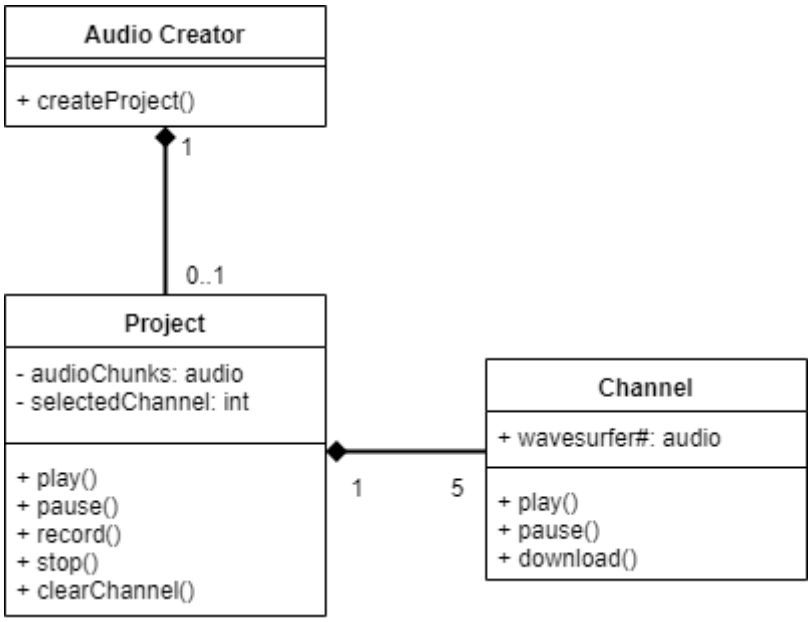
2.1 System Modeling

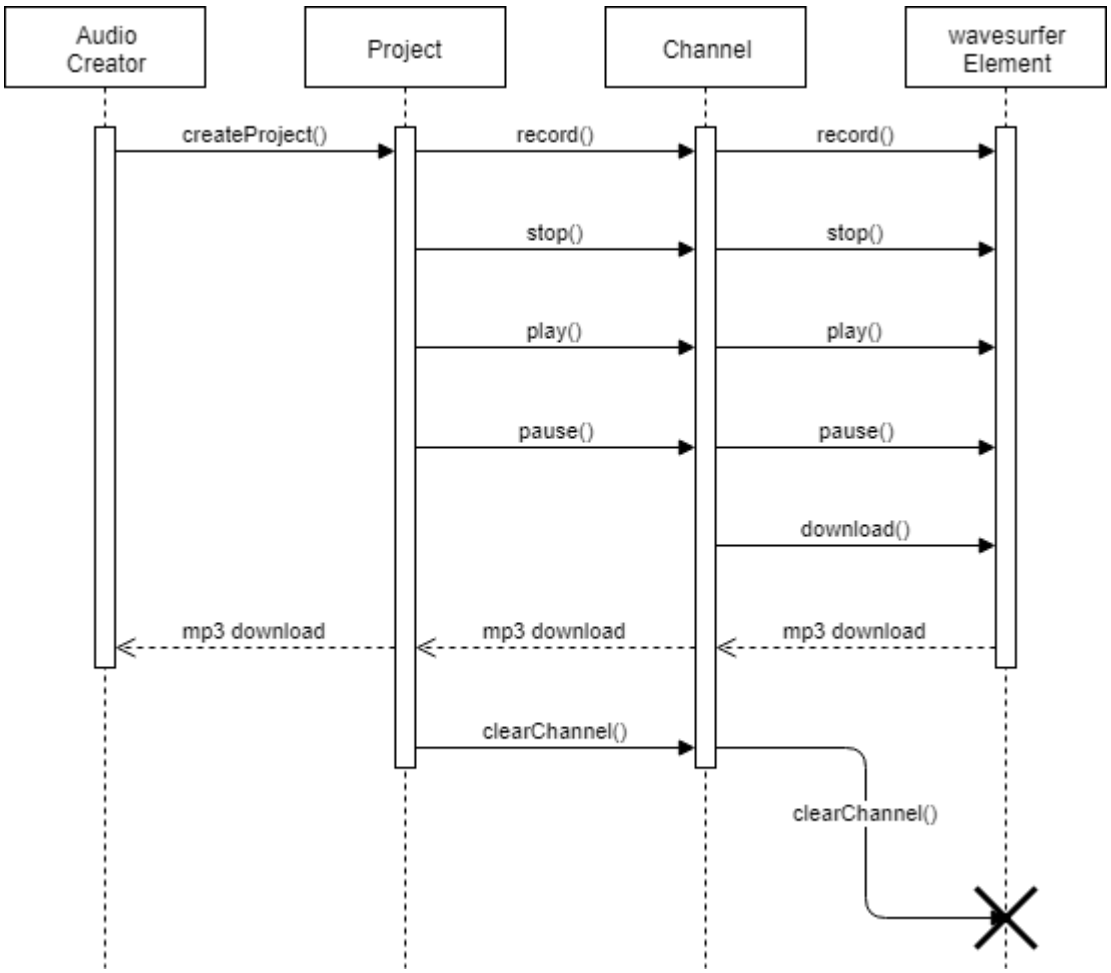



Audio Creator
wavesufer.js
Create new Project
Initialize Timeline

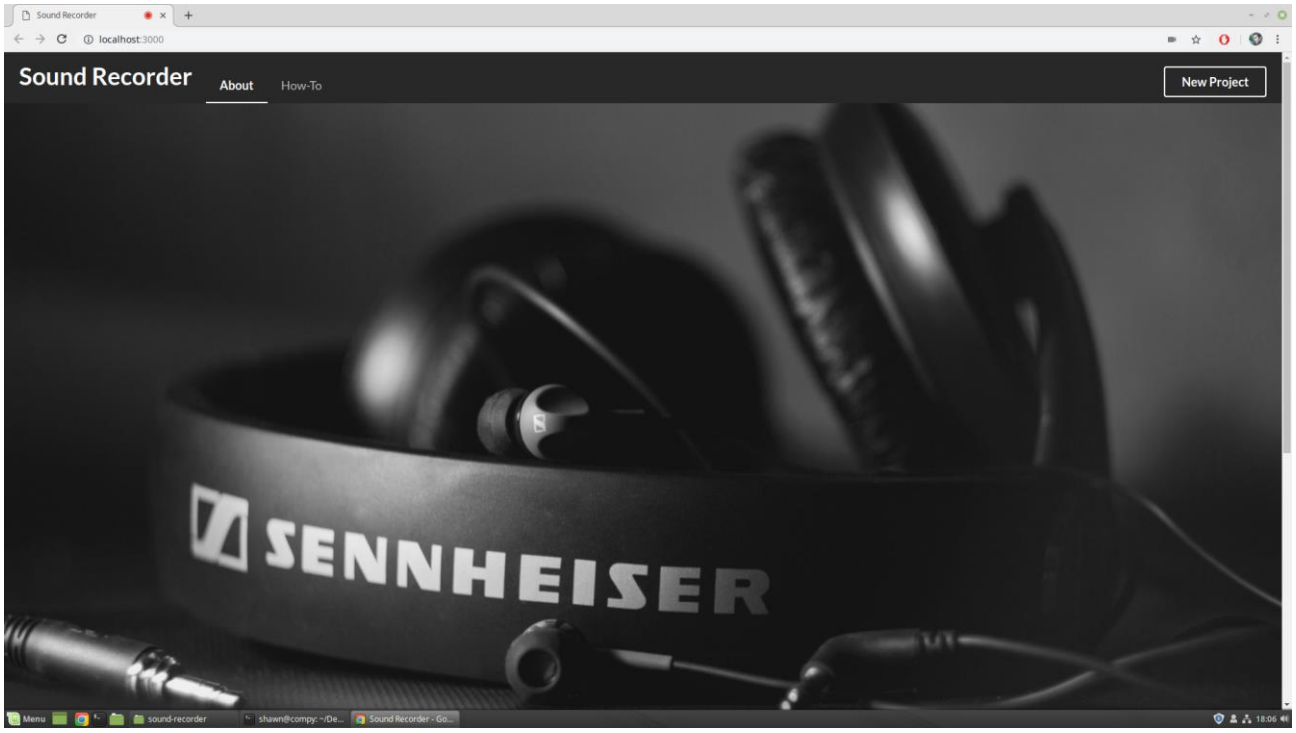

Project
Channels
wavesurfer.js
Select a Channel
Set Selected Channel
Record an Audio Track to Channel
Clear Channel
Channel Selected?
Yes
No
Channel Selected?
Yes
No
Play Channel
Pause Channel
Download an Audio File from Channel
Record and Update Display
Remove Audio and Update Display
Play Audio and Update Display
Pause Audio and Update Display
Download Audio File in mp3 Format

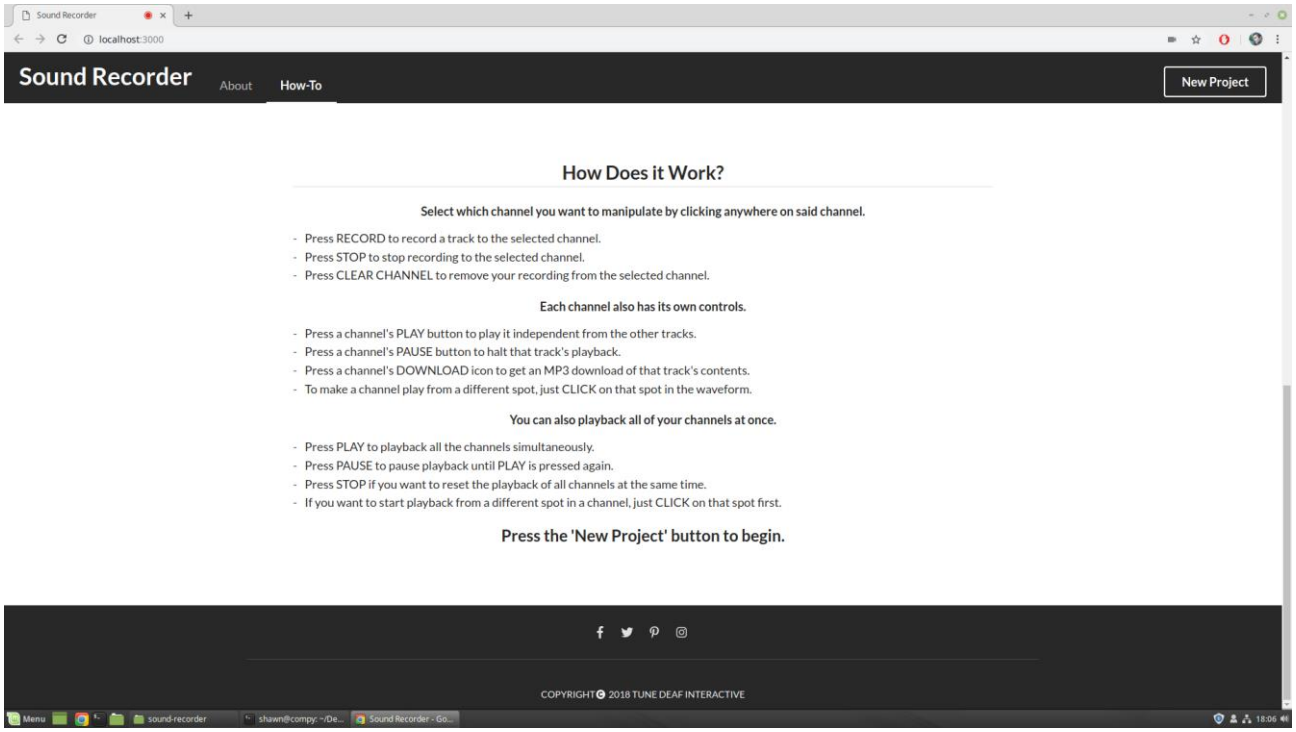

Project
Channels
Play a Project
Play all Channels
Pause a Project
Pause all Channels

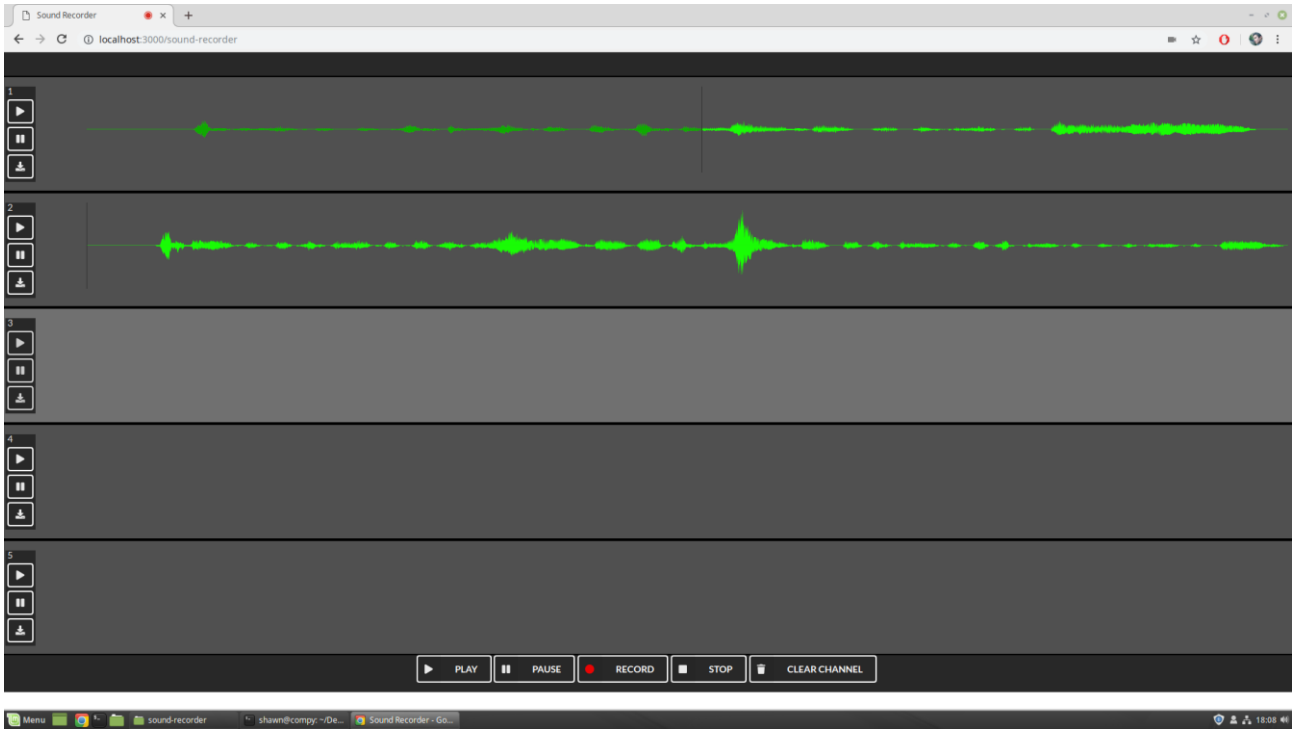

Audio Creator
+ createProject()
1
0..1
Project
- audioChunks: audio
- selectedChannel: int
+ play()
+ pause()
+ record()
+ stop()
+ clearChannel()
1
5
Channel
+ wavesurfer#: audio
+ play()
+ pause()
+ download()


Audio Creator
Project
Channel
wavesurfer Element
createProject()
record()
record()
stop()
stop()
play()
play()
pause()
pause()
download()
mp3 download
mp3 download
mp3 download
clearChannel()
clearChannel()

2.2 Interface Design






Sound Recorder
localhost:3000/sound-recorder
1
2
3
4
5
PLAY
PAUSE
RECORD
STOP
CLEAR CHANNEL
Menu
sound-recorder
shawn@compy: ~/De...
Sound Recorder - Go...
18:08

3 Implementation

3.1 Development Environment

This project used HTML, CSS, and JavaScript in association with the Meteor web application framework. The team used IntelliJ with ESLint for Code Quality assurance. Additionally, the Semantic UI library was used for the UI implementation and wavesurfer.js [1] was used to implement audio elements.

3.2 Task Distribution

Shawn Hillstrom was responsible for some UI implementation and non-audio related back-end implementation. Additionally, Shawn was responsible for code refactorization.

Brad Hendrickson was responsible for audio implementation including the discovery and implementation of wavesurfer.js.

James Kerinan was responsible for UI implementation and helped out with any back-end features that he was called on to add.

3.3 Challenges

The biggest challenge encountered during this project was learning how audio works in HTML and trying to integrate wavesurfer.js [1] with Meteor. Another challenge, although not as time consuming, was trying to make the UI look nice.

4 Testing

4.1 Testing Plan

Other than testing the overall layout of the UI features through trial and error, the features included in the main application page for Sound Recorder are tested in order of completion, indicated as follows:

1. Channel selection.
2. Start recording.
3. Record to selected channel.
4. Stop recording.
5. Play all channels.
6. Pause all channels.
7. Play each channel individually.
8. Pause each channel individually.
9. Download files from each channel.

4.2 Tests for Functional Requirements

All of the features implemented in Sound Recorder required Acceptance Testing so they were tested using console logs and/or trial and error tracking.

Channel selection was implemented early on in the project and required very little testing. Each channel is both a row element and a button and when each channel is clicked a variable containing an integer is set on the backend to the corresponding channel number. The resulting value contained in said variable is logged to the console to confirm functionality.

The **start recording** button was the next to be tested. Functionality was implemented fairly easily but was outputting errors whenever recording was started without a channel being selected. In order to remedy this, state checking was implemented with console logs to indicate path decisions.

Being able to **record to selected channel** was a slightly more complicated challenge and required several conditionals to decide where to store said recorded track. Regardless, little testing was required other than implementing a console log indicating that the HTML audio element could not be assigned to a channel.

The **stop recording** button was next and it produced similar errors to those encountered with the **start recording** button. The same tests were conducted: surrounding the stop actions with conditionals checking for recording state and printing to output if the state prevented stopping.

Implementing the **play all channels** button required very minimal testing because of the convenient api provided with wavesurfer.js [1] and no test cases ended up being written for this feature. The same was also true for the **pause all channels** button.

The **play each channel individually** and **pause each channel individually** buttons have functionality which is implemented within each HTML audio element. This was additionally true for the **download files from each channel** button, although it did require linking to the elements created to store audio files client side. Testing, again, was minimal for these features.

4.3 Tests for Non-functional Requirements

The only non-functional tests incorporated in this project were the built in Meteor tests checking client and server functionality when running the application. Other than this, there were no non-functional requirements which needed to be tested for the features implemented. The only non-functional requirements are listed below in Section 4.4.

4.4 Hardware and Software Requirements

- Access to a browser capable of dealing with HTML audio elements.
- Access to the built in JavaScript output console in said browser.
- Access to a working and connected audio input device (e.g. a microphone).

5 Analysis

Milestone 1:

Shawn Hillstrom – 4 hours

Brad Hendrickson – 4 hours

James Keirnan – 4 hours

Milestone 1 was mildly time consuming because it was the group's first time working on the project together and the members did not really know exactly what the project was going to turn into yet. The members were able to complete it with approximately 12 labor hours between everyone involved.

Milestone 2:

Shawn Hillstrom – 2 hours

Brad Hendrickson – 2 hours

James Keirnan – 2 hours

Milestone 2 was much easier than Milestone 1 because the members of the group had put much more thought into the project and now had an idea of where to start and how to organize the system on a high level.

Milestone 3:

Shawn Hillstrom – 25 hours

Brad Hendrickson – 25 hours

James Keirnan – 25 hours

Milestone 3 was by far the hardest project milestone out of the three. This milestone involved the members of the group figuring out how to actually implement the system in Meteor and dealing with changing requirements and time constraints along the way. The group worked tirelessly over the course of three weeks putting in 75 labor hours between everyone involved.

6 Conclusion

There have been several highs and lows throughout the project. One thing we wish we had done was select our external resources for the project earlier. If we had found wavesurfer.js [1] and figured out how to make it work with Meteor earlier in the course, the final project would have taken a lot less time to complete. Additionally, our requirements changed quite a bit during the process which is generally normal for any team but due to time constraints we were not able to re-visit our design documents; instead we were required to simply work with what we could in the time alloted. This taught us that design documents are great, if you have completely thought out the project before you begin. In practice, though, often a team will have to adjust to new constraints in the moment rather than re-visiting the design documents for the project.

Appendix A - Group Log

Throughout the completion of this project, the group did not meet in person. Instead, the members chose to use Discord as an online forum for project discussion. Communication was frequent and constructive and progress on the project was equally frequent as can be seen on the project GitHub respository which can be viewed here.