A Tribute to John Cage

Group 12

Alper Aydin, Jacob Pfaffenbichler, Jason Tan, Luke Blanchard, Yoannier Hermida
Sponsor: Dr. Richard Leinecker

February 11, 2020

'John Cage picking mushrooms in the woods', William Gedney Photographs and Papers, David M. Rubenstein Rare Book & Manuscript Library, Duke University. [1]

Table of Contents

1 Executive Summary 1

2 Project Overview 2

2.1 Project Description 2

2.2 Who was John Cage? 3

2.2.1 1912-1953 3

2.2.2 1954-1992 7

2.2.3 Mycology with John Cage 13

2.3 A Few Works by John Cage 16

2.4 Statements of Motivation 18

2.5 Goals and Function 25

2.6 Initial Ideas 28

2.7 Objectives 35

2.8 Constraints 38

2.9 Broader Impacts 39

2.10 Legal, Ethical, and Privacy Issues 39

3 Specifications and Requirements 41

3.1 Application Frontend 41

3.1.1 Requirements for Web Page 41

3.1.2 Requirements for Mobile Application 41

3.1.3 Stretch Goals 42

3.2 Backend API 44

3.2.1 Requirements 44

3.2.2 Stretch Goals 45

4 Division of Labor 47

4.1 Jacob Pfaffenbichler (Project Manager) 47

4.2 Alper Aydin 47

4.3 Jason Tan 48

4.4 Luke Blanchard 49

4.5 Yoannier Hermida 49

5 Research 51

5.1 Frontend Technology 51

5.1.1 Web Page 51

5.1.2 Mobile Application 52

5.2 Backend Technology 68
5.2.1 MERN Stack 68
5.2.2 Node.js Dependencies 69
5.2.3 GridFS 72
5.3 Audio Processing Algorithm 72
5.3.1 John Cage Influences 72
5.3.2 Digital Audio and Audio Formats 74
5.3.3 Audio Encoding and Decoding 76
5.3.4 Audio Analysis 77
5.3.5 Digital Audio Effects 80
5.3.6 Consulting the *I Ching* Oracle 83
5.3.7 *I Ching* Audio Modification 94
5.4 Hosting 95
6 System Design 97
6.1 Application Frontend 98
6.1.1 Mobile Application 98
6.1.2 Web Page 115
6.2 Backend API 120
6.2.1 Server 120
6.2.2 Audio Processor 123
6.2.3 Database 127
6.2.4 API Routes 129
6.2.5 Class Diagram 134
7 Testing Details 135
7.1 User Feedback 135
7.2 Continuous Integration Delivery (CI/CD) 135
7.3 Unit Testing 136
7.4 Integration Testing 137
7.5 System Testing 138
7.6 Mobile Development Testing 139
7.7 Backend Development Testing 140
8 Finance 142
9 Installation Instructions 142
10 Milestones 143
11 Future Plans 146
11.1 Frontend Features 146

11.2 Backend Features 146
12 Project Summary and Conclusion 150
13 References

1 Executive Summary

We are gathered this year of 2020 to celebrate the 108th birthday of American composer and music theorist John Milton Cage Jr. His works and philosophy explore the realms of the indeterminacies in both sound and music. He greatly influenced the genre of electroacoustic music and used instruments in unorthodox ways to generate unusual sounds or timbres. His compositions were truly original and were very far reaching into the deep philosophical gray areas involving music and sound. One of his works titled *4'33"*, was performed in such a way that the intention was to capture a sound that was, "unintended." It involved a pianist sitting down in front of a piano and not playing the piano in a usual way. This work was intended to be used in an effort to produce indeterminate sound and label it as a musical composition. If we think of what may happen when some people don't experience what they are expecting to experience at a performance or even in life, that phenomenon in itself can beautifully illuminate his philosophy of sound and quite possibly life. His opinions of music were very avant-garde, one his quotes read, "Which is more musical, a truck passing by a factory or a truck passing by a music school? Are the people inside the school musical and the ones outside unmusical? What if the ones inside can't hear very well, would that change my question?" [2-Pg.40] We begin to observe unusual patterns in the silence of sound.

Our sponsor Richard Leinecker got the opportunity to attend a John Cage Meta Music Festival at the University of Miami where he heard him speak. There was also a special advanced session for graduate composition students that he attended. Dr. Leinecker's experience at this session must have been quite enlightening as he mentioned, "He held everyone spellbound, and every sentence seemed profound." At the music festival John Cage also received a ten minute standing ovation at the general session before he began.

Some of his works and compositions involved the use of computer technology to modify and mix different sounds to accomplish an indeterminacy. The methods he used for his randomizations were unusual, one of them involved the implementation of the hexagrams within an ancient chinese classical text called the *I Ching* or *Yi Jing.* An example of one of these works of randomization, *HPSCHD* (1967-1969 John Cage and Lejaren Hiller), involved the use of a harpsichord and computer generated sounds. [3] His works spread into areas of mathematics, computing, and modern technology. Randomization algorithms were used to provide indeterminacy to a vast array of sounds, and now in the world where modern technology has advanced even more since his time, we are given an opportunity to use it to bring his works to life and make them everlasting.

2 Project Overview

2.1 Project Description

“A Tribute to John Cage” allows composers to create their very own John Cage like composition. This composition will be created by an audio mixing computer algorithm that attempts to replicate the style of John Cage, each audio input will be nondeterministically blended into each other. Composers can create a “room” once they wish to create a composition; once a room has been created, four to eight performers can enter the room to record whatever sound they are currently overhearing. The individual tracks will be blended into one composition using the algorithm and sent to the database. The website will then be able to play back the composition and show the recording information. The website will have a search feature so that anyone can go in and listen to any composition.

There are four main components to our project. The first component is the mobile application. The mobile application allows composers to make compositions by creating a room in which audio is recorded, as well as an optional pin that is associated with the room. If a performer enters in the correct pin, if there is one, they are allowed to join in the composition. Once the composer starts the composition, the performers will then begin recording the audio around them. The mobile app also allows listeners to overhear ongoing compositions, as well as view completed compositions on an archive. The second component is the mixing algorithm which will be stored on a cloud based server. The mobile applications will each individually send about twenty seconds of audio at a time to the server and the algorithm will make the necessary modifications to the sound and combine them before sending them to the database. The third component is the database which is also stored on a cloud based server. Once the final recording has been composed, it will be pushed to the database along with the details of the composition. The database will also store the account information of the composers. The fourth and final component is the web application. The web application allows users to create accounts and view their own compositions, and contains an archive of all previous compositions that users can search for and listen to, as well as listening in to compositions that are in progress.

2.2 Who was John Cage?

2.2.1 1912-1953

John Cage was born in Los Angeles, California on September 5, 1912. His father, John Milton Cage Sr. was an engineer and inventor, his mother, Lucretia Harvey worked as a journalist for the *Los Angeles Times* [3]. His family tree goes way back to the time when America was founded, with some accounts of an ancestor also by the name of John Cage helping George Washington survey the colony of Virginia. [4-Ch.1] Cage's first experiences with music were during his childhood where he took private piano lessons. He graduated as valedictorian from his highschool in Los Angeles and enrolled at Pomona College in Claremont, California interested in becoming a writer. [3] In 1930 he dropped out thinking that a trip to Europe would be more beneficial. He describes in his 1991 autobiographical statement:

> I was shocked at college to see one hundred of my classmates in the library all reading copies of the same book. Instead of doing as they did, I went into the stacks and read the first book written by an author whose name began with Z. I received the highest grade in the class. That convinced me that the institution was not being run correctly. I left. [5]

Touring Europe, he experienced many forms of art including Gothic and Greek architecture, painting, poetry, and music. He ventured to many places in France, Germany, and Spain, also visiting Capri, and Majorca where he started composing. His first compositions were created using mathematical formulas but he was displeased with his work and left them behind. After his time in Europe he was motivated to travel back to America after reading Walt Whitman's *Leaves of Grass.* [5]

He moved to Santa Monica, California in 1931 where he gave private lectures on contemporary art. By 1933 he decided to concentrate more on music than painting. [6-Pg.4] He sent some examples to Henry Cowell who suggested that Cage study with Arnold Schoenberg.

That same year, before approaching Schoenberg, Cage traveled to New York and took preliminary lessons from Adolph Weiss who was a former Schoenberg pupil. [3] A few months later Cage knew enough about composition to approach Schoenberg. During

their meeting, Cage told Schoenberg that he could not afford his price, and so the older composer asked whether Cage would devote his life to music. After Cage replied that he would, Schoenberg offered to tutor him free of charge. [5]

After I had been studying with him for two years, Schoenberg said, "In order to write music, you must have a feeling for harmony." I explained to him that I had no feeling for harmony. He then said that I would always encounter an obstacle, that it would be as though I came to a wall through which I could not pass. I said, 'In that case I will devote my life to beating my head against that wall.' [2-Pg.260]

Cage ended his studies with Schoenberg after he told his students that he was trying to make it impossible for them to write music. "When he said that, I revolted, not against him, but against what he had said. I determined then and there, more than ever before, to write music."[6-Pg.6] Between 1936–1938 Cage changed numerous jobs, and landed the title Dance Accompanist, which was the beginning of his lifelong association with modern dance. He produced music for choreographies and at one point taught a course on "Musical Accompaniments for Rhythmic Expression" at UCLA. [3] He also assisted filmmaker, Oskar Fischinger, with preparing to write music for one of his films. Mr. Fischinger told him one day, "Everything in the world has its own spirit which can be released by setting it into vibration." After hearing this, Cage recounts starting to hit, rub, listen, and bang on everything which then got him to write percussion music and play with his friends. [5]

In 1938, Cage drove to San Francisco to meet composer Lou Harrison. The two composers shared an interest in dance and percussion and hit it off very well. They formed a working relationship that continued for several years. Harrison also introduced Cage to the *I Ching* and it was years later that he started using it in his compositions. [3] His good friend helped Cage secure a position at Mills College, teaching Musical Accompaniments and collaborating with choreographer Marian van Tuyl. Many famous dance groups were present during his time at Mills College and Cage's interest in modern dance grew further. [3]

His association with Harrison in San Francisco helped him secure a position as a composer and accompanist for choreographer Bonnie Bird at the Cornish College of the Arts in Seattle, Washington. This was an important time in his life as he met many people who became lifelong friends, one of those being Merce Cunningham who was to become Cage's lifelong romantic partner and artistic collaborator. Cage also organized a

percussion ensemble that toured the West Coast and brought him his first fame. In 1940 his reputation was enhanced further with the invention of the prepared piano which was a piano that had its sound altered by objects placed on, beneath, or between the strings. *Bacchanale* was one of his earlier compositions created for the prepared piano. Cage left Seattle in the summer of 1941 after the painter László Moholy-Nagy invited him to teach at the Chicago School of Design. [3]

While teaching there he worked as an accompanist and composer at the University of Chicago. Eventually, because of his reputation as a percussion composer, he landed a commission from the Columbia Broadcasting System to compose a soundtrack for a radio play titled *The City Wears a Slouch Hat* by Kenneth Patchen. Hoping to find more commissions, he left Chicago for New York City in the spring of 1942. [3]

Cage went to New York with his wife Xenia Kashevaroff (married June 7, 1935), he met many important artists such as Piet Mondrian, André Breton, Gypsy Rose Lee, and Marcel Duchamp, with the help of painter Max Ernst and Peggy Guggenheim who the Cages were staying with. [3] After a falling out with Guggenheim and a great performance at the Museum of Modern Art he and Xenia spent the rest of the summer of 1942 with dancer Jean Erdman and her husband Joseph Campbell. [3] Without the percussion instruments, Cage again turned to prepared piano, producing a substantial body of works for performances by various choreographers, including Merce Cunningham, who had moved to New York City several years earlier. Cage and Cunningham eventually became romantically involved, and Cage's marriage to Xenia, already breaking up during the early 1940s, ended in divorce in 1945. Cunningham remained Cage's partner for the rest of his life. [3]

Gita Sarabhai, an Indian musician who came to New York to study western music had become a part of Cage’s circle of artistic acquaintances in New York in 1948. There was a point in time Cage asked her what her Indian teacher thought was the function of music. She replied, “the purpose of music is to sober and quiet the mind, thus making it susceptible to divine influences.” [4-Pg.90] In the early 1950s, D. T. Suzuki lectured on Zen Buddhism in Columbia, which Cage attended:

> In the course of a lecture last winter, Suzuki said that there was a difference between oriental and european thinking, that in european thinking things are seen as causing one another and having effects, whereas in oriental thinking this seeking of cause and effect is not emphasized but instead one makes an

> identification with what is here and now. He then spoke of two qualities: unimpeded-ness and interpenetration. Now this unimpeded-ness is seeing that in all of space each thing and each human being is at the centre and furthermore that each one being at the centre is the most honoured one of all. Interpenetration means that each one of these most honoured ones of all is moving out in all directions penetrating and being penetrated by every other one no matter what the time or what the space. [3, 7]

In early 1951, Cage crossed paths with an ancient classical text called the I Ching which a friend gave to him, it was commonly used for divination, but for Cage it became a tool to compose using chance. [3, 5, 7] These chance operations applied to the sounds in a composition yielded works in which sounds were almost free from the composer's will:

> When I hear what we call music, it seems to me that someone is talking. And talking about his feelings, or about his ideas of relationships. But when I hear traffic, the sound of traffic—here on Sixth Avenue, for instance—I don't have the feeling that anyone is talking. I have the feeling that sound is acting. And I love the activity of sound ... I don't need sound to talk to me. [8]

Some of his first works involving the *I Ching* were *Imaginary Landscape No. 4* for 12 radio receivers, and *Music of Changes* for piano. The *I Ching* became Cage's standard tool for composition: he used it in practically every work composed after 1951, and eventually settled on a computer algorithm that calculated numbers in a manner similar to throwing coins for the *I Ching*. [3, 9]

In 1952, Cage composed his widely-known and controversial piece *4'33''*. The score instructed the performer not to play the instrument during the entire duration of the piece—four minutes, thirty-three seconds—and is meant to be perceived as consisting of the sounds of the environment that the listeners hear while it is performed. David Tudor even performed *4'33''* on August 29, 1952 at Woodstock, New York, which caused an uproar in the audience. This may have been the inspiration for the book titled *The Roaring Silence: John Cage: A Life.* [3, 4, 7]

Around this time Cage also had taught classes at the avant-garde Black Mountain College just outside Asheville, North Carolina. In 1952, he organized what has been called Theatre Piece No. 1, the first “Happening”, or “the event”. This was known by some as the beginning of aleatory music and dance. [3] A Black Mountain lecturer named M.C.

Richards and the poet Charles Olsen read poetry from ladders, Rauschenberg's "White Paintings" hung overhead while he played Edith Piaf records on an old phonograph, David Tudor played the piano, Merce Cunningham danced in and around the audience being chased by a barking dog, and Cage sat on a step ladder for a duration of two hours reading a lecture on the relation of music to Zen Buddhism, on and off, sometimes listening silently. The experience was very random and full of sensory input, this was perhaps what Cage was trying to accomplish. From 1953 onward, Cage composed pieces used in modern dance, some of these especially, were for the dances of Cunningham who had also adopted chance. [3,5]

Headshot of Merce Cunningham from the April 1961 issue of *Dance Magazine*, commemorating the magazine's annual award winners [10]

2.2.2 1954-1992

Here is a detailed timeline of John Cage’s life from 1954 and on that the New York Public Library provided: [11]

1946:

- Begins composing *Sonatas and Interludes*.

1948-1949:

- Meets Robert Rauschenberg and Buckminster Fuller.

- Teaches at Black Mountain College, North Carolina.

- Meets Pierre Boulez, in Paris.

- Receives awards from The Solomon R. Guggenheim Foundation and The American Academy.

1949-1950:

- Composes *String Quartet in Four Parts*, which premiers on August 12.

- Becomes part of the *New York School* of composers with Morton Feldman, Christian Wolff, and David Tudor.

- Begins to use chance operations and the *I Ching.*

1951-1952:

- Composes *Music of Changes*, among his first using chance operations and compositional decisions determined by the *I Ching.*

- Composes *Theatre Piece No. 1*, considered to be the first happening.

- Teaches summer courses at Black Mountain College.

- The composition of *4'33"*, which premiers on August 29, marks the admission of silence into Cage's compositional repertoire.

- Applies chance operations to electronic composition.

1953:

- Becomes director of the Merce Cunningham Dance Company.

1954:

- Meets Jasper Johns and Karlheinz Stockhausen.
- Moves to Stony Point, New York.

1956:

- Starts teaching courses at the New School for Social Research, New York City.

1958:

- Meets Luciano Berio.
- Twenty-five-year retrospective concert at Town Hall, New York City.

1960-1964:

- *Bacchanale*, *String Quartet* and *4'33"* published by Henmar Press.
- Becomes a fellow at Wesleyan University, Connecticut.
- Writes *Silence.*
- Tours Japan.
- Co-founds the New York Mycological Society.
- Composes *Variations IV*.

- Explores the independence yet co-existence of music and choreography.

- Goes on world tour with the Merce Cunningham Dance Company.

1964-1968:

- Introduced to the writings of Henry David Thoreau.

- Becomes composer-in-residence at the University of Cincinnati.

- Starts composing *HPSCHD*.

- **Begins incorporating computer-generated chance operations into his compositions.**

- Becomes an associate at the Center for Advanced Study, at the University of Illinois.

- Elected to the American Academy and Institute of Arts and Letters.

1968-1975:

- Creates first works of visual art.

- Becomes artist-in-residence at the University of California at Davis.

- Writes *Notations*.

- Composes "*Songbooks*" which premieres on October 26 and is published that same year.

- Becomes a fellow at the Center for Advanced Study, at Wesleyan University.

- Moves back to New York City.

- Composes *Child of Tree*, which premieres on March 8 and is published that same year.

1978:

- Teaches first printmaking sessions at Crown Point Press, in San Francisco.

1980:

- Becomes Regents Lecturer at the University of California at San Diego.

1982:

- Cage is feted in celebrations worldwide marking the occasion of his 70th birthday.

1983-1984:

- Composes and publishes *30 Pieces for String Quartet*.

- Composes *8 Whiskas*.

- Composes and publishes *Nowth Upon Nacht* Premieres *30 Pieces for String Quartet* in July.

- **Begins working with a computer to make large-scale computer-assisted compositions.**

1985:

- Composes and publishes *But What About the Noise…*

1986:

- Awarded an honorary doctorate by the California Institute of the Arts.

1987-1988:

- Extends concepts from his collaboration with Merce Cunningham to include

independent but co-existent elements of theatrical production such as lighting, sound, and costumes.

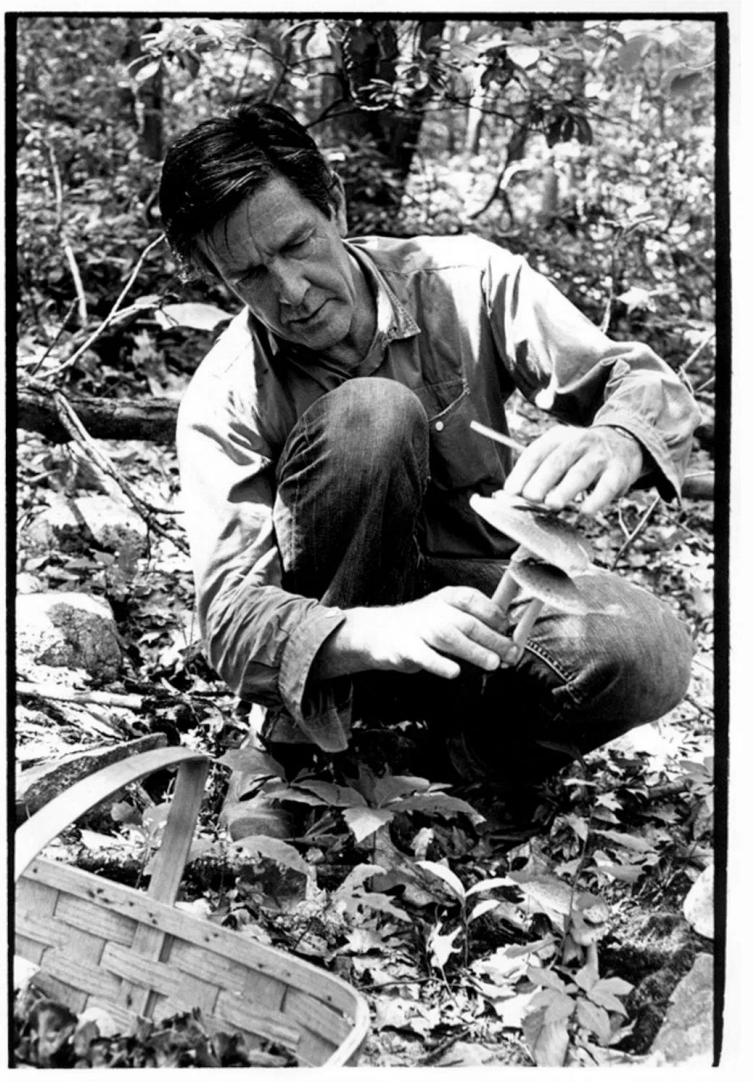
- Turning to a new genre, Cage begins composing his first opera, which becomes part of the *Europeras 1-5* series.

- Becomes Norton Professor of Poetry at Harvard University.

In the course of the 1980s, Cage's health worsened progressively from arthritis, sciatica, and arteriosclerosis. He suffered a stroke that disabled his left leg and in 1985 broke his arm. On August 11, 1992, while preparing evening tea for Cunningham and himself, he suffered another stroke. He died the morning after at St. Vincent's Hospital in Manhattan, He was 79.

To honor his wish, his body was cremated and his ashes were scattered in the Ramapo Mountains just like his parents' ashes were, near Stony Point, New York where he had gone on his mushroom picking adventures in the forest. Merce Cunningham lived for 17 more years before dying of natural causes in July 2009.

'John Cage picking mushrooms in the woods', William Gedney Photographs and Papers, David M. Rubenstein Rare Book & Manuscript Library, Duke University. [1]



2.2.3 Mycology with John Cage

Here is an excerpt from a web page that contains a few of John Cage's comments out of a 1968 book titled, *The God Who is There* by Francis Schaeffer:

> In 1954, the sculptor David Weinrib and his wife moved into an old farmhouse on a tract of land in Stony Point, Rockland County, forty miles from New York, which the Williamses had brought. Cage lived and worked in an attic room that

> he shared with a colony of wasps, and often took long, solitary walks in the woods. His eye was caught right away by the mushrooms that grew so abundantly in Rockland County, in all shapes, and sizes and brilliant colors. He started to collect books on mushrooms and to learn everything he could about them, and he has been doing so ever since. After all, mushroom hunting is a decidedly chancy, or indeterminate pastime.
>
> No matter how much mycology one knows—and Cage is now one of the best amateur mycologists in the country, with one of the most extensive private libraries ever compiled on the subject—there is always the possibility of a mistake in identification. "I became aware that if I approached mushrooms in the spirit of my chance operations, I would die shortly," Cage said not long ago. "So I decided that I would not approach them in this way!"
>
> In other words, here is a man who is trying to teach the world what the universe intrinsically is and what the real philosophy of life is, and yet he cannot even apply it to picking mushrooms. If he were to go out into the woods and begin picking mushrooms by chance, within a couple of days there would be no Cage! [12]

Following are a bit more passages about a few of John Cage's experiences with mushrooms, written by Sally Wilson:

> By the late 1950s Cage's career as a composer was well-established. His early works followed the conventional rules of music theory, but progressively he scrutinised those rules, challenged and unravelled them. He altered pianos with nails and rubber bands to produce new sounds, played flower pots, cow bells and frequency oscillators, pioneered the electroacoustic movement and, in 1952, composed his famous piece *4'33"*, which was musically silent. With this work Cage's purpose was to focus his audience towards the ambient sounds of their environment. "Whether I make them or not, there are always sounds to be heard and all of them are excellent," he remarked.
>
> In 1959 Cage taught a Mushroom Identification course at the New School in New York City, alongside his course in Experimental Music. When classes ended Cage, together with author and horticulturist, Guy Nearing and a loose gang of student enthusiasts, decided to revive the New York Mycological Society. This

happened officially in 1962. Their charter was to do exceptional things, like: go on wild mushroom foraging expeditions in the woods, within public transport distance from New York, and then cook collective dinners with the plunder. The hot ticket item on the Society's social calendar is the Annual Banquet, held in early winter at an NYC restaurant, which often makes the culinary pages of the *New York Times.*

But the masterpiece was *'Morels à la John Cage'* cooked with sweet butter, a ½ cup of flat champagne, heavy cream, morels, and salt and pepper.

In 1972 Cage's Mushroom Book was released, a copy of which is held in the collection at MOMA. More of Cage's recipes, observations and diary entries jostle for room in the book, alongside quotes from Buckminster Fuller and Henry David Thoreau. "Sandwiches of leftover mushrooms," Cage reports at one point.

"I'm what you would call an amateur mushroom hunter, and so far I haven't killed myself or killed any other person," Cage once confessed in an interview with German-American composer and filmmaker, Henning Lohner. Sure enough he never killed a dinner guest, but poisoned them he did. Cage was a fossicker of all wild things and once ruined half the staff from the Museum of Modern Art with a dish of skunk cabbage (Symplocarpus foetidus), or wrongly identified poisonous hellebore. "I ate more than the others did in an attempt to convey my enthusiasm over edible wild plants. After coffee, poker was proposed. I began winning heavily. M.C. Richards left the table. After a while she came back and whispered in my ear, "Do you feel alright?" I said, "No, I don't. My throat is burning and I can hardly breathe."

Cage was taken to the hospital, his stomach pumped and was administered adrenalin to keep his heart beating. His doctor said, "Fifteen minutes more and he would have been dead."

Cage had the type of mind that did not shut off. He thrived in the woods and felt at home on the hunt for wild mushrooms. "Clothes I wear for mushroom hunting are rarely sent to the cleaner. They constitute a collection of odors I produce and gather while rambling in the woods. I notice not only dogs (cats, too) are delighted (they love to smell me)."

"We remain greedy: we never find enough. We keep on looking for mushrooms until we're obliged (an engagement or the fact the light's falling) to stop. Only for some such reason do we leave the woods (unless, by then, we're lost)."

Cage hunted wild mushrooms throughout America, in Italy, Finland and the Soviet states. The practice offered solitude and also community. Friends often joined Cage on the hunt and in his Mushroom Book he recalls conversations from those times as having the same intimacy as discussions in the living room. There was a simplicity to hunting that he respected: "We only need boots, baskets, paper bags, and knives," he once wrote. With full baskets he and his revolving troupe would return to the two-room cabin in Stony Point and cook. When they visited, Cage even convinced *Vogue* to gather its own lunch, and eat barefoot.

"Often I go into the woods thinking after all these years I ought finally to be bored with fungi. But coming upon just any mushroom in good condition, I lose my mind all over again." This is what Cage wrote in his diary, published later as *How to Improve the World (You Will Only Make Matters Worse)* (2015, Siglio Press). It's hard not to feel like minded in autumn and winter, as the saffron milk caps and slippery jacks make themselves known and our appetites turn to sources of warmth and satisfaction. [13]

2.3 A Few Works by John Cage

Freeman Etudes (1977-1980, 1989-90):

The *Freeman Etudes* is a complex set of violin etudes composed as part of John Cage's etudes series in the mid to late '70s. John Cage designed these etudes with the intention of representing "the impossible", mirroring it with his perspective on the ever-present social and political conflicts of the world. Initially composed for violinist Paul Zukofsky at the behest of Betty Freeman, Cage channeled the set's 'indeterminacy' approach using star charts to embed incredible musical detail. As a result, Zukofsky was unamused by its seemingly unplayable status and Cage thereafter left the pieces in hiatus. This was disproved by János Négyesy and Irvine Arditti in the '80s and surged Cage's passion once more, allowing him to finish the etudes by 1990. [9]

Etudes Boreales (1978):

A work concentrated on the cello and piano, the *Etudes Boreales* is yet another etude set in John Cage's etude series. It was composed in dedication to cellist Jack Kirstein of the LaSalle Quartet and his wife, Cincinnati pianist Jeanne Kirstein. In its cello etudes, it bears similarity to the *Freeman Etudes* in the sense that their composition involves using chance. Their pitch ranges are limited but quick to change, posing a challenge for conventional cellists. *Etude Boreales* also contains piano etudes that were written using star charts (the chosen chart was *Atlas Boreales*, the set's namesake). The chart bears no indication of pitch, instead alerting where on the piano one must play, transforming the piano into more of a percussion instrument. [9]

Imaginary Landscape (1939-1952):

Imaginary Landscape is a series of five pieces, all of which are reliant on electricity, with the pieces starting off as using electrical elements as just an instrument, but the later pieces featuring only electrical elements. *Imaginary Landscape* no. 1, 2, and 3 were mainly percussive pieces that also feature electrical elements, such as a coil of wire and an electric buzzer. *Imaginary Landscape* no. 4 and 5, to contrast, do not feature any "traditional" instruments, opting instead to use radios and phonographic records respectively. [3]

Sonatas and Interludes, for prepared piano (1946–48):

This piece was a cycle of 20 short pieces for prepared piano. It consists of four sets of four sonatas, each set separated from the next by one or two interludes. The cycle requires nearly four dozen "preparations" of the piano strings, which yield a range of gentle modifications of the timbres while retaining the melodic quality of the instrument. Most of the sonatas are in a binary form, while the interludes either have no repeated passages or follow a four-part aabbccdd structure. [3, 14]

Music of Changes, for piano (May – December 1951):

Music of Changes is a reference to the Chinese oracle book called the *I Ching*, or Book of Changes, of which Cage made extensive use in composing the piece. Another, more personal, reference is perhaps seen in the changes taking place in Cage's overall compositional language at the time. For this work, Cage employed *I Ching*-derived chance operations to create charts for the various parameters, i.e. tempi, dynamics, sounds and silences, durations, and superimpositions. With these charts, he was able to

create a composition with a very conventional manner of notation, with staves and bars, where everything is notated in full detail. The piano is played not only by using the keys, but also by plucking the strings with fingernails, slamming the keyboard lid, playing cymbal beaters on the strings, striking the keyboard lid, etc. Use of the pedals is also notated in full detail. The notation is proportional, where 1 inch equals a quarter note. The rhythmic structure is 3, 5, 6 3/4, 6 3/4, 5, 3 1/8, and is expressed in changing tempi, including the use of accelerandi and ritards. This work may be seen as the first step of Cage's voyage into the world of chance composition. For Cage, this was a necessary first step in the giving up of individual taste and memory, as well as other previously meaningful traditions in the making of art. This development, in part, came as a result of his encounter and informal studies with Gita Sarabhai (Indian philosophy) and attendance at the lectures of Daisetz T. Suzuki (Zen Buddhism) in the late 1940s and into early 1950s. However, chance here only applies to the process of composition. The actual result, or composition, that derived by these means, along with the performance, are fixed and determined, things which Cage would also later abandon in subsequent compositions. [3, 15]

2.4 Statements of Motivation

Alper Aydin:

When I first heard Dr. Leinecker's pitch on the John Cage Tribute, my interest was immediately struck when I saw the word *I Ching* in his presentation. I have always been very interested in Asian philosophy. Over winter break I went to the Netherlands to visit some good friends of mine and one of them showed me a copy of the Yi Jing while walking me through his library. Ever since that encounter I have seen it coming up in multiple places and readings, including in Dr. Leineckers presentation. I reviewed the other project pitches but this project was the one that was the most interesting. This project merges with my personal interests to study this ancient Chinese divination text. I am also very much into the psychedelic arts and influences, one of them being music. I have a little bit of a percussion and bass guitar background, but even more so I partake in listening to beautiful music. Music of all kinds and all realms, as long as they sound good to my ears. I enjoy music from Elder Scrolls, Dark Souls, Lord of the Rings, Star Wars, Hans Zimmer, Nordic music, Celtic music, John Williams, music from various movies and tv shows, underground hip hop, asian ambient music, Sanskrit mantras, Buddhist chants, and a lot of other ambient mixes. The latest group I would suggest to check out is

a group by the name of *Too Many Zooz*. Initially researching John Cage and seeing a performance of his *4'33"* I was skeptical, but then I watched some more of his performances and interviews and just from his mannerisms I could tell he was onto something and knew what he was doing.

After watching some interviews I read some of his quotes about sound and his interpretation of music and I immediately reprimanded my skepticism. He was describing an idea that I had come across many years ago that had to do with silence and sensitivity. When we decrease or limit stimuli in our environment, we become very perceptive and sensitive to any kind of stimulus that occurs. Supple like bamboo, not breaking in the wind like the oak. The bamboo picks up the slightest change in wind. I think this is exactly what his philosophy and perception of music described, and this resonated deeply within me because it is one of my own observations. A few years ago I was reading about these things called sensory deprivation tanks that cut off all the senses in an effort to experience the inner world. Some people claim that spending just twenty minutes inside can make you hear very loud noises even though it is not that loud to an observer sitting outside the tank. Everything is vibrating, or so we observe, everything has energy, and so if we can tune out blatant sounds as anomalies that occur from the background sound, we can enter the domain of vibration and feel the instance. The quote in the page above about the truck passing by a factory and a music school is one that effectively expresses his perception of music. My personal statement to any critic would be to have the ability to entertain an idea without fully accepting it. Then they may very well be able to see and understand the larger meaning within his work.

I was listening to a composition of his titled *Bird Cage* (1972) one day when I was working in the kitchen. In the beginning of the piece there was a woman asking, "What's your name?" continuously, then the verse stopped playing and other sounds and blends began to occur. After a long time into the piece, I heard the woman again say, "What's your name?" and I immediately thought it to be very trippy and kind of scary. What I took from listening to this piece was that the randomization was intense, things could come back into the piece even after an hour of absence. I thought that this definitely was a coding area. Also by extending my idea about silence that I mentioned above, he enlightened me to turn that way of thinking into the domain of music and sound. This really made me appreciate music way more, by influencing me to be mindful of silence which in turn increases the senses, especially to different sounds. The musical compositions I hear now, will be more fulfilling due to this mindfulness. Also quite possibly the beautiful pieces of Bach, Mozart, Beethoven, and Shostakovich were

composed by chance events too! What are the chances that the Earth was close enough to the Sun and had enough water and polarity to support life? And on this Earth men like Mozart, Bach, Beethoven, Shostakovich and many more were supported so that they could create their beautiful compositions, I think this may very well have all been chance.

In my opinion, his philosophy that all sound is music describes music in its most fundamental form of sound, nature, chance, and that all sounds are beautiful. Does a performer play any two pieces the same way each time? Are we here by some randomization or is our destiny predetermined, is that predetermination based on another chance?

Jacob Pfaffenbichler:

Music has always been an important factor in my life. Some of my earliest memories include me enjoying music. I have been avidly listening to music since I was about seven years old, and now I listen to a variety of types of music. Music is something that I use to keep myself focused while I do my work or when I wish to relax. As I grew older, I also began to understand that music is a form of communication that transcends language itself. I personally believe that two people can speak two completely different languages and still get the same feeling when listening to a piece of music. It has the ability to tell a story without saying any words.

My interest in music really started to expand at the beginning of the seventh grade when I decided to join my middle school's band. When I first made this decision, I had just moved to Florida, so joining the band gave me a group of people to hang out with that I would not otherwise have. My parents, neither of them being all that musical, just assumed I would drop it after a year or so. I decided I really enjoyed it, and ever since then, I have been playing music for the past ten years, and I have been a part of both my high school and college marching bands for the past nine years, and I was one of the alto saxophone section leaders for the Fall 2020 season at UCF. I was enjoying it so much that I inspired both of my younger brothers to pick up music as well. In this past decade, my relation to music has evolved from just something that I enjoy in my spare time to something I am fascinated with. Along with a degree in computer science, I also plan on graduating this fall with a minor in music as well.

When I first heard about the John Cage Tribute senior design project, I instantly knew that this was the project that I wanted to work on. It was a project that combined my top

two interests, technology and music. This is something that not many other projects could accomplish, and something I didn't even believe would be included among the Senior Design topics. Another reason I gravitated towards this particular project was for its uniqueness. This project, unlike quite a few projects that I was interested in, was building something new from the ground up, as opposed to building upon something that was already created by others. When I was first looking through the list of projects provided, one of the key points I was looking for was something that I could look back on and feel proud. I have a feeling that this project can accomplish this goal of mine.

When it comes to actually listening to music, I believe that I have an eclectic, or varied taste when it comes to what kind of music I listen to. In general, I can listen and even enjoy most forms of both classical and contemporary music. However, I would have to say that I am very partial to music with jazz influences. This can very much be attributed to the instrument that I play, the Alto Saxophone. I also enjoy listening to video game soundtracks, as the way they are meant to be played on a loop allows them to serve as background music as I go about my everyday life. But overall, I can listen to pretty much anything and enjoy it.

Jason Tan:

Music has always been a part of my life for as long as I can remember. I started learning to play the piano when I was about 8 years old. Throughout elementary school and middle school, I took lessons from 2 different teachers (I had to switch because, if I remember correctly, the first teacher began having health complications of some sort and decided to take a break from teaching thus I moved on to the second). Around the beginning of high school, I took a hiatus mostly because I was getting involved with other things like high school band and tennis but also due to the fact that I felt as though I wasn't enjoying what I was learning and playing in my lessons. However, sometime in my junior year, I happened upon a certain pianist named Kyle Landry through YouTube who became a major inspiration for me. I could identify with him because, in addition to playing the piano, he was also a gamer that found his own inspiration in the themes from games that he played such as Final Fantasy, Kingdom Hearts, and Pokemon. After this discovery, I decided to continue my piano playing and came to appreciate the lessons I had in the past which built a foundation that I could progress further from on my own. Since then, I've followed in Kyle Landry's footsteps playing songs that he's covered like the Battle Theme from the Piano Collections of Final Fantasy VII and even some of his own arrangements of songs like Passion from Kingdom Hearts 2.

Along with piano, I also played the trumpet in middle and high school band. That was a different experience in music for me because I had to learn how to stay in sync with everyone else and follow the conductor. When I played the piano solo, I could decide how fast or how loud I wanted to play but while playing in band, that was all decided by the conductor. Then, on top of that, I also had to learn how to march and stay in formation when I got into high school band. As you can see, this is all vastly different from playing the piano but it helped me become a better musician in general because I could play music and nurture my skills in an environment with other people who loved doing the same. However, by the end of my high school career, I decided I would forgo the trumpet. I wasn't progressing as much as I would've liked, and I realized I preferred playing the piano more anyways given how much longer I'd been playing. Still, I will always hold onto the memories and experiences I had playing it in band.

With all of these experiences that I've laid out above, I think it's only logical that I'd want to participate in a project involving something I know and enjoy working with. I mean, sure, there's all sorts of projects on VR/AR and computer vision and all and that's cool, I guess. But then there's music? Yes. I'll take it. Granted, the style of John Cage is quite far from the conventional music that I'm used to playing and hearing but at its core, music is music. And that's good enough for me.

Luke Blanchard:

At the age of 11 I first started taking piano lessons which began my interest in music. After taking lessons for many years I began playing piano in my church band which deepened my love for music and composition. I taught myself drums and began teaching piano and drums as a side job which I did for several years. I've always heard that teaching a subject is the best way to learn it, and I found that to be very true as I taught music to people who had never played before. Around this time I began playing around with composition, mostly with free Digital Audio Workstations I found online. I was intrigued by the ability of music to make you feel a certain way. I found this in movies, video games, and regular music tracks, although I was never sure of the method to which this was achieved.

I have been greatly inspired by the works of an artist called Sleeping at Last, who grabbed my attention through an album inspired by the planets of the solar system. This artist has many songs inspired by seemingly random things, such as cardinal directions or

the 5 senses. I was amazed by how these songs took a single concept and expanded it into a complete song that could invoke emotion from the listener. It is this process that inspired me to look deeply at the meaning of music and how it affects us at different levels. Music's ability to resonate and connect to people is unique and powerful yet very little is understood about it.

When it came to the selection of my Senior Design project, it was hardly a question what would be my first pick. While I had never heard of John Cage before, it didn't take long for me to realize that he was a revolutionary composer and his work needed to be preserved. This project will allow us to not only build a technically complex system, but one that captures John Cage's ideals and philosophies. We will be exploring not just how to generate random music from random sounds, but how to capture the feeling and emotion in John Cage's work. The most exciting thing about this project to me is not building a tool for people who want to make cool sounding music, but a tool for people who want to make music that evokes feeling just like the music of John Cage.

Yoannier Hermida:

Rewinding to the year 2002, my interest in music began to sprout during my exposure to video games. Despite not strictly following the guidelines of conventional music composed for the general public, it bore very alluring properties. Some would, if one's eyes were closed, evoke the exact atmosphere that the levels they inhabited would share. Imagine a never-ending expanse of ocean, decorated to its brim with lush vegetation and dangerous aquatic creatures. Next, picture an absolutely massive yet quiet and clackety clock, whose second hands march punctually to a resolute and firm beat. Such imagery was in the immediate grasp of the video games that lay soundly in my memories. On another note, they exhibited this: not a mere evocation of emotion, but an entire feeling that would have otherwise remained entirely dormant in one's mind. If a frantic and almost off-key theme ensued, it would send any curious person in 2002 in a craze. An endless flurry of button inputs would be sent straight to the game! We can take a slow, reflective tone from Star Fox 64 (such as its Level Select) echoing throughout the depths of space as an example. In my shoes, one would be left in awe. How curious it is for such a seemingly microscopic story to be overshadowed by the bewildering and endless universe!

Among the myriad of games that I remember fondly from long ago, Super Mario 64, Star Fox 64 and Donkey Kong 64 were a few prime examples that confidently demonstrated

such kinds of properties in their music. First and foremost, any atmosphere or scene that unfolds over the course of these games is accentuated by its relevant tune. A good number of these songs have nestled themselves in my memory up until this day, demonstrating how close to heart they can be. This, however, only explains my desire to lend an ear to music. My opportunity to interact with the development of music itself follows shortly.

I was introduced to my first set of instruments, including the flute-like recorder and keyboard, around the end of my elementary school years. I was less satisfied with the recorder but enjoyed the keyboard's wide array of unique settings. Entire days of mine would pass by as I played around with the most obscure sounds (for e.g., a chorus, UFO noises, utensil noises) found in the keyboard. Although I never dabbled into the actual theory of music until more recently, the prospect of music composition itself was readily planted into my brain from that very moment. After about a year or so, I was left dejected when the keyboard was thrown away. My interaction with music composition laid dormant for years to come, until I was reintroduced to it in the form of Digital Audio Workstations (DAWs).

I had always been keen on purchasing it but lacked the financial stability to afford one. As a gift, however, my family purchased me an Apple MacBook Pro laptop for education purposes. Much to my surprise, Apple pre-installed their flagship DAW, GarageBand, into them. I hastily installed SoundFonts (instrument sets that belong to specific video games) in order to breathe life back into the fond memories of the games I played long ago. Once I had grown accustomed to GarageBand's interface, I opted to serve as the music composer in my Artificial Intelligence for Game Programming group in Spring 2019. This video game project, in tandem with my practice with a DAW, emboldened a desire to develop music as a hobby after graduation.

With regards to this Senior Design project, the John Cage Tribute stood out the most when considering innovative capacity. There are few projects that not only bear the property of being a "first generation" project (not being inherited from previous SD groups), but also possess the need for developing a solution that has never been attempted before in software history. The idea of allowing countless John Cage fanatics to develop their own music in his likeness heavily emphasizes the implication of software innovation. John Cage is specific about his musical developments, leaning toward a more avant-garde and ambient structure. This resonates with video game music that I have been exposed to, alongside some of my favored compositions that I listen to as I study.

Some of the works of GAS, a musician, includes Untitled (Pop) and Königsforst. These albums are laden with the ambience that I have surrounded myself with for well over a year now. They were the first to come to mind when Dr. Leinecker pitched this idea. I hope to pay tribute to John Cage's ideals while embracing the style of GAS to perfect the art of ambience with our project.

2.5 Goals and Function

Alper Aydin:

The goal of the application is to have a certain feel and a particular function in order to mimic and pay tribute to the compositions and philosophy of John Milton Cage Jr. Both the feel and the function is of paramount importance and this may have an effect on the technical implementation of the application. Dr. Leinecker mentioned that he would like software that could play the recording in real time on the website, I thought that this may be one of these technical implementations that keep Cage's technique in mind. All of his performances like to capture the current environment as it is during the performance, I think Cage may have preferred it to be in real time also. If this is at all not possible or a difficulty to accomplish real time arises in the development stage, then it is acceptable to make it as close to real time as possible, as Dr. Leinecker mentioned. John Cage would have liked it to be in real time, but adding in a computer generated delay just might do the trick into slicing not only musical pitches but time itself. For now we will aim for real time, or very close to it. The design of the frontend is another vital factor in giving it a John Cagey feel, both the web application and the mobile application's user interface could incorporate images changing at certain rates just like from some of his works, *HPSCHD.*

Another main goal is to have a solid server set up that can pass signals between the database and the web and mobile application. When sounds are recorded from the mobile side they will need to be passed to the server and modified there before being sent to the database and the web application. The server side will need a sophisticated, programmable audio mixing algorithm that closely mimics the compositions created by John Cage and/or other compositions that are based on the philosophical view of his works.

With these implementations using modern technology, we hope to offer a medium in which an audience can digitally attend and join in on an original John Cage composition. We hope that in the future this application can be upgraded and expanded on to better fit any kind of need in the realm of Cage.

Jacob Pfaffenbichler:

This software, for the most part is not meant for casual use, as in we are not expecting the primary use of this software to be just recording things on a whim. We intend this to be used for performances. While theoretically anyone could use our application for the sole purpose of listening to others creations, most of our software is designed around making the compositions. Because of this, our primary audience will be those with more of a musical background, as opposed to a more technical background. The layout of both the web and mobile application will have to keep this in mind, and should be as simple as possible.

Jason Tan:

The paragraphs surrounding this one explain the overall goals and functions of this system better than I ever could so, to avoid being redundant with what would quite frankly be an inferior explanation, I'll just continue with my own personal goals for the project as the front-end web developer. First and foremost, the web application must have all necessary functions in order for users to properly interface with our system. It must be able to take in all information that is required, operate on it efficiently, and display the output correctly. This is all a given. The web application is where user accounts are made and recording sessions are listened to, it is the starting point for all usage of the entire system. If it cannot perform correctly, then there is not even a chance for the user to move forward and experience the rest of work that the group put in, all because of my lackluster job. But the second most important goal is that the website must look good. It needs to look presentable. First impressions are a defining factor in any sort of interaction with anything so it is imperative that I get it right. Then, on top of looking good, I would like to, at the very least, attempt to style the website in such a way that it conveys the personality of John Cage, if it's even possible at all to express the personality of a human being through web design. Granted, Cage's was one that is extremely distinct so I at least have something to work with.

Luke Blanchard:

Since this project is intended to pay tribute to the composer John Cage, the resulting system's functionality must align with his ideologies. His ideologies developed over the course of his life, and he is well known for being very open about his principles. Many of these principles and ideas are explored in the book "The Roaring Silence: John Cage: A Life" which is a biography about Cage.

A quote from Cage is "My favorite music is the music that I have not yet heard. I write music so that I can hear the music that I have not yet heard". This strongly emphasizes his desire for uniqueness in his music. Cage often inserted randomness into his compositions with the desire to never hear the same piece of music. With this project it is vital that we inject randomness into the music being made. The level of randomness could be adjustable by users, but some level should always be present. It is also important that the system produces unique sounding music, with lots of variation between pieces.

Cage was well known for using non-conventional sounds and 'instruments' in his music. He is quoted saying "All sounds are useful in music" and "Composers should direct their search to sounds which hitherto have not been considered musical". This shows his desire not just to use non-conventional sounds in his own music, but for the musical community to use them as well. Cage even went as far as to say "People were so protective of the term music that it might be better to find another word", preferring the terms "art of noise" or "organization of sound". I think these terms speak volumes into the desired functionality of this project. We are not trying to create music as it is generally defined but music as it was defined by Cage, as an organization of sound. We don't want to haphazardly combine sounds together either though. Organization implies intention which means we have to be very intentional with how our system combines sounds together. This can be achieved through analyzing the individual audio streams and comparing their qualities to make intelligent mixing decisions.

Yoannier Hermida:

In my particular role for the John Cage Tribute project, I aim to deliver a mobile application that grants users the opportunity to be in John Cage's shoes. The user will have the ability to join an existing recording session (being dubbed a "performer") that is due to begin. Upon the session's initiation, they can record their surroundings to add to a collection of other performers' audio recordings in the same session. This collected

ambience is then transmuted with an algorithm that gives the resulting composition its "John Cage-ness." If the user had created the session instead, they would be dubbed a "listener" and would listen to the ambience as it is being constructed, lacking the ability to record audio. My priority for the mobile aspect of the project is to ensure this process is as painless for the user as possible.

Subsequently, there must be some form of enumeration of users' past recorded compositions and each must be accessible for playback if they so desire. Specifically, they will be capable of viewing the compositions that they themselves have made, alongside the rest of the user base. There will be a search function added that can ease the search process based on each composition's own info, such as its name and added tags. Upon selection of a particular composition track, there will be a play, pause, and slider that can decide what time exactly in the audio file the user would like to start playback. In the event that a user is disinterested in the composition info that they provided to one of their pieces, they will be capable of editing them. In the event the entire composition is not to their liking whatsoever, a delete functionality will also be built in for them to take advantage of.

The application is expected to be capable of being presented with a relaxing interface akin to that of modern mobile applications seen nowadays. Although the app will sport minor differences between its Android and iOS iterations, this is with respect to Apple's interface guidelines. The majority of the interface is expected to have a "Cupertino" style to it, which is explained easiest as an "Apple-y" looking style. The Android platform will use Flutter's regular interface enhanced by the relaxing style mentioned earlier.

Lastly, the application's listening/recording mechanics are largely dependent on its ability to communicate quickly enough with its server. The current consensus is that (almost) real-time audio listening will occur during each recording session. Upon further operation on the Tribute, if this appears to be unviable, then an initial and somewhat significant delay will be required at the beginning of a session. In the worst case, the "listening" mechanic may have to be scrapped altogether and the resulting audio will simply be constructed and dropped off in the composer's completed compositions.

2.6 Initial Ideas

Alper Aydin:

- The incorporation of moving fractal images in the web and mobile applications, like those seen in the video for *HPSCHD* on Youtube. In the web application while playing a composition, a fractal can be randomly chosen to be played along as a visual effect to the composition. On the mobile side, while the sound is in the active process of being recorded, a moving fractal image can be shown on the mobile screen. If we want to really upgrade the application, we can make the fractal/visual effects change based on the wavelengths of sound being played or recorded.

- The duration of the entire composition can follow a range of possibilities, even recording and mixing for an indefinite amount of time until a user stops recording. The duration can be very short, medium ranged, hours or days long (although this would not be feasible for database storage). The cool thing is that we can not only run the sounds in a loop, but instead randomly come up with points in time during the composition where a change in the randomization variables would occur, in essence we would roll the dice all over again in the middle of a composition. We could even influence this re-rolling of the dice if a particular timbre of sound is recorded in a particular way. The possibilities are endless. The frequency with which these points would occur that change the randomization variables would also be random. It would be a never-ending loop that changes the sizes of the intervals at which these randomization variable changes (re-rolling of the dice) occur. So if we were to have a very long or infinite piece, the program would continue to randomly generate points in the future of the recording where the randomization variables would change, this would occur for the whole duration (definite or indefinite) of the composition.

- We thought about recording sounds, sending them to the server for mixing and then sending them to the database for present or future playback. Another thing I thought about was after a certain amount of time (possibly randomly chosen), sending previously made compositions back into the server or instance of John Cage to modify the composition all over again to make the compositions everlastingly random. We would not have to delete the old compositions but just use them to generate new compositions. This may fill up the database as time goes on in which it would be feasible to delete the old composition after the new one was automatically generated.

- We could implement a code that makes the final composition a different length than the original. This way users could record very short versions of sound to make longer compositions, and we can implement this by using loops and randomly changing

intervals that change the values of the initial randomization variables. This way we would be able to make compositions of random length. By incorporating randomization at every layer, and by following the Yi Jing divination strategy for each one, we can really make this thing close to random.

- Creating dependencies between the randomization variables in random ways. Some would have dependencies, others would not. At those points in time where the randomization variables are changed randomly, the dependencies would also change.

- The elements of the input stream that could be modified in random ways and turned into randomization variables could be many things such as the volume of the sound, the duration that each sound will be played (could include frequent or infrequent rests). Some things to take into account in our randomization algorithm could involve the volume that the sound was recorded in, if the recorded sound was very low in volume and the others were very loud we may slightly modify the lower sound to be louder or cut out other sounds or make them less frequent. Additionally, we can also measure the frequency of the sound before rolling the dice. If one frequency is very high then we might want to pair it with a lower frequency wave without modifying frequency at all. We might also randomly choose if we want to do the above mentioned things or not, at every layer we could incorporate a randomization for whatever we could think of. Some other things that could influence the randomization variables and their dependencies could include barometric pressure, number of steps taken that day, geographic location, time of day, time of year, the age of the participant, etc.

- One idea I got for the mixing algorithm while Luke was reading a John Cage quote to me; the quote read something along the lines of constantly wanting to hear sounds or compositions that have never been heard before or that one has not heard yet. John Cage mentioned that he wanted to do something that had not been done before. “Two musicians making the same kind of music is one music too many.” So it got me thinking, are two of the “same” pieces performed ever the exact same? Impossible, they cannot be because of the number of diverse variables in the environment of the performance. The composition is not the only thing taking place, life is moving on, the sounds, the energy of the environment. Two samples of similar sound that are run through our audio processing algorithm should never sound the same on two different runs. The samples are fundamentally different but come from the exact same sound. Keeping this in mind, I drew up a hash table in my mind and thought of different probing strategies while Luke read me these quotes. Two samples are fundamentally different, but

of the same sounds. That John Cage had said he wanted to hear something that he has not heard before, each time! And so this got me thinking of mapping out each composition into a hash table based on certain types of variables and the dependencies between them. These variables could be pitch, volume, and duration (is it an ambient sound or is there some form of beeping occurring). We can hash based on this and use a probing strategy on a very large table to never mix a composition in the exact same way. We would have random array ranges that are chosen using a random number generator, but if the generations were to fall into a similar place on the table (from a previous similar composition) we would roll the dice again (probing) and fall on a new randomization to ensure that we create very different pieces each time. It would be important to use a large table with high ranges in our random number generators, this way we can have many compositions that sound different. The table would map out all of the possibilities that could occur given a certain number of dependencies matched with certain "**intensities**" of the variables. Just like the King Wen sequence in the Yi Jing! Imagine there was an array with 100 indices to determine what rhythm the sound would be played in, with index 10 indicating a whole note and index 90 indicating 64th notes. This is why I used the word intensity above, because for each randomization variable we could consider an array indicating the intensity of the variable and incorporating that into a hash table that maps out all of the combinations possible. The indices for each variable should be randomly chosen.

- As a final addition to the idea above, Luke had created a diagram of sound effect modifications we could use in this project. These effects could be implemented in an array of its own type so that it can easily be incorporated into the hash table and array implementation above. If there are a total of 5 different modifiers then we can randomly choose a number from 0-5 that represents the number of modifiers to be used at the same time. According to John Cage, he wanted to hear sounds that he had not yet heard. He would have liked to hear blends that were not done yet. So in order to achieve a unique blend each time we could implement a hash table that represents each possible blend and keep track of the blends that have occurred previously. The way to derive this would be to multiply the size of each array that represents each randomization variable's intensity on the spectrum. If we had 3 randomization variables and three arrays of randomly chosen size: 45, 2, 84. Then we would multiply 45*2*84=7560, there would be 7560 possible ways to blend the sounds. If we incorporated Luke's sound effect modifications in an array and there were 5 modifications and that we could do all of them at the same time, this would change our counting as we could not just multiply 7560 with 5 which is the size of the array. Instead we would have to multiply it by 2^5=32 because each index

in the array has two options; turn the effect on, or don't (0 or 1). If we have five effects or an array of size 5, that gives an option to choose 0 or 1 in each index.

- Implement King Wen Sequence from the Yi Jing into the program and allow it to modify the sounds.

Jacob Pfaffenbichler:

- Singular tracks will be pushed to the database and resent to the user hosting the performance.

- Composers can check a box to make the algorithm be able to detect human voice and distort it to make the human speech unrecognizable.

- When launching the mobile application, a prompt will appear stating that the user is responsible for the music that they create in order to deter the use of copywritten material.

- Possibly incorporate machine learning into our mixing algorithm so that it can gradually develop into sounding more John Cage like as it receives continued use over time.

- After the database has stopped receiving new information from the mobile application, it will store the completed audio file to be accessible by using the web application.

- Incorporate previous works of John Cage such as Imaginary Landscape Number 5 or the numbers series into our algorithm.

- Once the recording has finished, the software will generate a score inspired by the score of John Cage's Imaginary Landscape number 5 based upon what the algorithm generated for the recording.

- Add tags and collegiate affiliations to the finalized recordings stored in the database so that it is easier to search for a specific recording that you would want to listen to on the archive.

Jason Tan:

- The website could possibly support making appointments for recordings, groups for the appointments, and finally users for the groups. Additionally, the website might also be the place where final mixed output can be searched for and listened to.

- Thus, the website could have pages for:
    - login/register
    - user overview
    - user settings
    - group overview: Appointment-making (date + time)
    - group settings
    - music library with search/sort functionality: Downloading

Luke Blanchard:

- To reflect the quote from John Cage, "*My favorite music is the music that I have not yet heard*", the output of the system should be completely unique every time.

- Use sensor data from the mobile devices recording sound such as GPS and barometric pressure to influence the effects applied to that sound.

- Partition the incoming audio stream into chunks of a fixed number of audio samples. The chunks are analysed and audio qualities such as wavelength, amplitude, and frequency are determined. Based on these qualities, effects are applied to the audio in such a way that when multiple streams are mixed together, they sound 'good' to the listener.

- The Audio Processing Server will employ a microkernel design architecture. The base functionality would be to receive multiple streams of audio and combine them together into a single output stream. The additional functionality would be to add in audio effect plugins that would modify each audio stream before they are combined together.

- The audio effect plugins could include but are not limited to: frequency content modification (equalization), dynamic compression, panoramic position, volume level (gain), reverberation, delay, and pitch shifting.

- Allow the importing of Virtual Studio Technology (VST) plugins to be used by the Audio Processing Server to modify the audio streams. This would allow for an extreme number of possibilities of how the audio could be processed as well as allow advanced commercial audio effects to be used and not built from scratch.

- Allow the importing of Web Audio Modules (WAMs), a similar design to importing VST plugins. WAM is a newer format and is less commonly available, but would be easier to adapt to running on a server as they natively run in webapps.

- Build our own audio plugins using SOUL (SOUnd Language), an embedded programming language designed completely for audio and music manipulation. This is an extremely new language with very little documentation, but has huge potential.

- Build our own audio plugins using a Node.js package such as audio.js or node-core-audio.

Yoannier Hermida:

- Design additional functionality to connect and disconnect users over the duration of their recording session, allowing for omission of unwanted noise.

- Implement a variation feature that allows for additional modulation of the recorded audio, including speed change, pitch shift, and reverb.

- Add a countdown before allowing users to press the record button for improved preparedness to record.

- Include a verification function from the mobile app, sending a confirmation message to the user's confirmed email that was used to log into the app.

- Set a strict time limit for the length of users' recordings to disallow extensive storage usage.

- Present a series of entertaining images as the user awaits the end of their recording or ends it prematurely (these images would reference John Cage's abstract and avant-garde thinking).

- Add reminders for returning users to improve awareness of upcoming or potentially canceled recording events.

- Set up features that allow users to listen to their previous compositions on the mobile app as well, reducing required time spent accessing the web application for them.

- Include alerts for members of a team during recording that a member or more has disconnected or reconnected to the recording session.

2.7 Objectives

Documentation:

- Regularly keep up with lecture material and dynamically apply what we learn into our project.

- Meet once or twice a month with Dr. Leinecker for updates and performance meetings.

- Meet once or twice a month with TAs for help and advice.

- Prepare and maintain a presentation of the entire project for design reviews and presentations.

- Regularly work on the final design document for submission at the end of Senior Design 1. Also keep the final design document updated after Senior Design 1 to account for any future modifications and presentations.

- Produce all supporting charts and diagrams that are required.

- Make sure the document is polished and professional.

- Update document to note any changes that might have taken place during the development process.

Learning Technologies:

- Learn how to connect to MongoDB and transmit data to and from the frontend.
- Learn GridFS for storing audio to database.
- Learn the MERN stack adequately enough for project development.
- Learn how to use Flutter/Dart for iOS and Android implementation.
- Learn how to use the needed frontend technologies and brush up on JavaScript.
- Learn any additional technologies that will be needed along the way.

Prototype Wood (Phase 1):

- Set up database schematics (users and compositions).
- Set up the website that can create user accounts, play compositions without having to be logged in, search for compositions by title and tags.
- Make sure rooms are able to successfully be created and hold listeners and performers connected via sockets.
- Audio successfully passes from the recording room to listeners and to the frontend live recordings page.
- Pass sound files from 2 mobile phones to the server and database, play both sounds at the same time on the website.
- Run as many tests as you can to ensure the objectives were met for Phase 1 (unit tests, integration tests, end-to-end).
- Fully deploy the web application and push the mobile application to the iOS and Android store.
- Evaluate Phase 1 product, update future objectives if needed, and move on to the

next Phase.

Prototype Fire (Phase 2):

- Pass sound files from 1-6 mobile phones to server and database. Make sure the notifications and codes are being sent to all of the phones from the web application for confirmation.

- Implement a more advanced audio mixing algorithm into the server's code and send the modified sound as one to the database and website. Be able to search for and play the final composition on the website.

- Polish website and mobile application user interface making it very user friendly.

- Run as many tests as you can to ensure the objectives were met for Phase 2 (unit tests, integration tests, end-to-end).

- Fully deploy the updated web application and push the changes to the iOS and Android store.

- Evaluate Phase 2 product, update future objectives if needed, and move on to the next Phase.

Prototype Divine Intervention (Phase 3):

- Pass sound files from 2-8 mobile phones to server and database.

- Further expand the audio mixing algorithm and make sure it is exceptional. Send the final composition to the database and the website.

- Further polish the frontend of the website and the mobile application, possibly adding visual effects. Make sure the frontend is exceptional.

- Implement stretch goals if time allows.

- Run as many tests as you can to ensure the objectives were met for Phase 3 (unit tests, integration tests, end-to-end).

- Fully deploy the updated web application and push changes to the iOS and Android store.

- Evaluate Phase 3 Divine Intervention and make necessary changes. If accepted by the whole group, move on to preparing the presentation.

Prepare Presentation of Divine Intervention:

- Powerpoint.

- Find participants to take recordings on presentation day.

- Test a few more times before the actual presentation date, ensure the application is working.

- Populate the database and the website with compositions.

2.8 Constraints

- Time: This project has a strict deadline and must be completed and ready for presentation on December 3rd, 2020.

- Knowledge: As computer science majors, music is not necessarily in our wheelhouse, especially when the music is as unconventional as the music of John Cage. We will have to spend a lot of time in the beginning phases of our project researching John Cage and his works.

- Mixing Algorithm: Even once we have a good base knowledge about music, the mixing algorithm is still going to be the most complicated part of our system.

- Server: The server is going to have to take input from up to potentially eight points from the mobile application, so the server is going to need to have enough bandwidth to support that.

- Userbase: Our intended audience is musicians, so they might not be as

knowledgeable in the field of technology. We are going to have to design our user interfaces to compensate for this.

- Covid-19: During our first semester as a group, the United States started to enforce lockdowns due to the Covid-19 pandemic. On top of all the damages it has cost families across the United States, it has limited the productivity of the project, as we were effectively no longer able to meet in person.

2.9 Broader Impacts

Ultimately, the goal of this project and of this system is to contribute to the legacy of John Cage and his musical philosophies, as is the goal of anything that might be called a "tribute". Thus, the intended impact would be to ensure that everyone who uses the application is left with a significant impression and understanding of who John Cage was and what his philosophies were. Our sponsor Richard Leinecker proposed the idea of promoting the app at different music schools which would grant needed exposure. Of course, at the end of the day, this will most likely remain as a small project and, by that I mean, it is very unlikely that this will become anything close to "mainstream". However, that's okay because I doubt anyone involved really expects it to be the case. John Cage's music and philosophies are simply too abstract and esoteric to become popular in this day and age. In fact, when I thought to use the word esoteric, I only had a rough understanding of what it really meant so upon searching it up on the internet, I discovered it was actually the perfect word, meaning "intended for or likely to be understood by only a small number of people with a specialized knowledge or interest". With that said, for the relatively small group of people that have an interest in John Cage or those who stumble upon the app by chance, the rest of the team and I hope to make this app an interesting and inspiring experience to remember John Cage by.

2.10 Legal, Ethical, Privacy Issues

When dealing with recordings, especially when the recording is going to be viewed as a performance, there are several legal, ethical, and privacy issues that arise. The most important of which would be the recording of copyrighted content. There is nothing stopping our customers from turning on a piece of copyrighted content, pressing record, and starting to stream music that is legally owned by someone else. Because of this, we

will have to disclose to our customers that they are legally responsible for what they create. Another way we have tackled this issue is by increasing the minimum amount of users from 2 to 4, and by distorting the audio more when there are less performers recording. Additionally, there is a likely possibility of someone being recorded without their knowledge or their consent. Something we can do if we choose to go down the machine learning route is to train the algorithm to detect if it can clearly hear a human voice and distort it before it gets mixed. Some recordings might warrant the use of someone talking or singing, so it might need to be an option given to the composer when initially creating the recording. With regards to privacy, we must ensure that the passwords are not only stored properly but also encrypted so that in the case that a leak occurs, it is much more difficult to retrieve user information.

3 Specifications and Requirements

3.1 Application Frontend

3.1.1 Requirements for Web Page

- The homepage of the website will provide an explanation of the project and allow users to create accounts and listen to live recordings. Any user, with or without an account, will be able to search for a composition on the homepage by entering a tag, name of performer, or title name of a composition.

- On the account creation page, the user will be prompted to enter their email address, a username, and a password. Once they successfully create an account, they are led to the user dashboard page.

- On the user dashboard, users will be able to listen to personal recordings from their previous sessions and edit the associated composition information

- The search functionality will also show lists of compositions that match the search criteria, and a user will be able to choose a composition from that list to listen to.

- A statement appearing somewhere within the phone application and on the web page stating that any person using the application is personally responsible for any copyright infringement they might commit while using the John Cage Tribute software application.

3.1.2 Requirements for Mobile Application

- The mobile app will immediately direct users to its dashboard, which describes the general intention of the John Cage Tribute on its foreground. It will allow seamless navigation from the dashboard to other screens.

- A section that dedicates itself to the recording performance. It will initially greet

the user with a screen that requests a PIN code in order to direct them to the appropriate session. The recording session's screen will be slightly different, depending on whether a user that intends to listen to the audio (a listener) or to record audio for the session (a performer) navigated to the recording screen. When the session is complete, the creator of the composition will be greeted by a form that allows them to enter data based on the session, such as its name, relevant tags to search it by, etc. If the user fails to enter data, the composition will sport default information (name: Untitled, tags: none).

- Another section focused on recordings that were already completed (known as "compositions"). The user can access their previous compositions if they verify their status as the session maker. These can be modified, deleted, or replayed as they see fit. In addition, they may search for other users' compositions with a search function based on their names or tags. Thereafter, they may play the audio back to hear what others have created.

- An audio player component that allows for playback of the selected composition in question. It supports pause, play, and a seek slider that can be dragged to play the composition at a certain point in its duration (think YouTube's video slider). The screen will show information pertaining to the composition itself in an organized manner.

3.1.3 Stretch Goals

The mobile application is not without its room for improvement. Its lingering weaknesses are largely dependent on the presence of the appointment system and its limitation on users' desires to immediately begin recording. An in-depth explanation is below:

Host-based System:

- *This stretch goal was achieved.*

As opposed to tracking a day and time for recording, a user can instead create a room and host it for other users to join. This requires a(n):

- Navigation screen that grants a user the option to host a room or join a room.

- PIN code auto-generated by a room so that users who enter the PIN may enter.

- Duration field that specifies how long the recording session will last.

- "Ready" button that each user in the room must press before a session may begin, signifying the preparedness of the users.

- "Record" button that only the composer is authorized to press to initiate the session, which shall initiate a countdown to the session's execution.

- Mute button to allow users the option of providing no audio to the audio-mixing server (through the closing of their individual web socket).

- List of icon indicators (where each indicator represents a user) that glow a specific color to indicate whether that user is currently active, muted, or disconnected from the recording session.

This overhaul reduces the amount of info to be stored in the database due to the lack of necessity for an appointment schema and replaces that with a room-ID and room-PIN schema instead. In turn, the web application would lack the need of an appointment page and would become more streamlined. Inversely, the audio-mixing server would now require micromanagement of sockets and an indicator of whether or not a recording session has begun. Despite this, the investment would guarantee an increase of user traffic with consideration to the decreased difficulty in initiating a recording session.

Aside from the host-based system, the mobile app would be significantly enhanced with:

- Suggestions list in the "Search Compositions" screen based on how frequently users searched for a particular tag or tags.

- Friend list and simple chatrooms to allow ease of PIN distribution between friends.

- Arrangement in a users' own compositions in the form of albums at their behest, allowing for distribution of whole albums instead of individual pieces.

- (Based on previous bullet) Expand "Search Compositions" screen to include the search of albums.

- Advanced search features that allow the user to search for a composition based on the range of duration, date recorded, etc.

- Easy deletion/disposal of a recording that is deemed unwanted, provided as an option after the recording is complete.

- Inclusion of “favorites” or “playlist” options for users to save their favorite compositions and play them in their leisure.

- Selection on the audio player to sequentially play every track saved for convenience of the user, akin to a playlist.

- “Contact Us!” widget or screen that quickly allows users to relay unintended bugs or behavior present in the John Cage Tribute project, along with potential UI/UX flaws or critiques that may benefit the project in the future if dealt with.

- Incorporating visual effects into the web and mobile application while a session is recording. If the user navigates to the recording page while a session is recording there would be some moving images of fractals or lines. The playback of previously recorded compositions may also have visual effects.

- At the end of a recording session the application could display a score of each of the sounds showing what the algorithm’s output was specifically.

3.2 Backend API

3.2.1 Requirements

- The server shall support up to 8 connected mobile devices at a time.

- Each composition will be a maximum of ten minutes in length. Once ten minutes has been reached, the compos

- The database will store the final product of the composition and the website will make it accessible to users wishing to play the recorded compositions. The database will also store composition and user account information. Passwords will be hashed.

- The spirit of the software will be its sophisticated audio mixing algorithm. We will need to implement an algorithm that mixes the sounds in such a way that it closely coincides with John Cage's way. The server will take in multiple audio inputs at one time, modify them by using randomization techniques to create indeterminacies in their sound, and blend them all into each other as one composition. This final piece will be sent to the database and be accessible from the website and mobile application.

- The backend will need many functions that perform create, view, list, update, and delete operations on user and composition information.

- The server shall support TCP socket protocol to connect to mobile devices.

- The server shall support 16-bit 22050 Hz Mono WAV encoding and decoding.

- The server shall support MP3 encoding.

3.2.2 Stretch Goals

- Allow the composer to adjust the level of randomization.

- Create a way in which no recording can literally sound the same, may need a large hash table to map out a range of the possibilities.

- Incorporate an email verification system into the software so that it can ensure that proper emails are being registered on the website.

- Implement stronger password requirements while creating an account.

- Come up with a strategy to make the application open source so that users can easily create their own randomization modules and plug them into the application to record their own pieces.

- Display the "score" of the recording as listener listens to the recording

- Allow listeners to download the completed recordings to their devices.

- Allow users to upload the modules they create onto their accounts and share them with other user accounts on both the web and mobile applications.

- Play the recordings in real time on the web application, so that users can listen to the compositions as they are being created. If the task is not feasible to implement, try to make the recordings as close to real time as possible.

4 Division of Labor

4.1 Jacob Pfaffenbichler (Project Manager)

- Schedule group meetings and setting up interactions with our sponsor as well as those in the field of music.

- Develop a backend database that stores the user information and the information of the finalized compositions and has connections with the front end web application, the mobile application, and the mixing algorithm stored within the server.

- Assist in the development of the mixing algorithm once the backend has been completed.

- Assist in developing the layout of our final documentation so that the document reads cohesively.

- Have the database be able to store MP3 files so that it can be listened to later in an archive.

- Assist in setting up the archive on both the web and mobile application so listeners can search and listen to previously recorded compositions by themselves or by other users.

- Layout out structure of final presentation, as well as the video for the Senior Design Project Showcase.

4.2 Alper Aydin

- My responsibility for this project will be to come up with an acceptable audio mixing algorithm that makes the user feel like John Cage is in our server messing with the sounds. To do this I will dive into some history and uncover the different compositional styles and techniques he used throughout his life. After studying his works

and understanding how he applied the I Ching and other techniques to his compositions, I will come up with an algorithm that aims to do the same.

- In order to mend this algorithm into the application, I will need a good knowledge of how to maneuver around the backend of the application. After all, the audio mixing will be done on the server. My goal is to rapidly progress my knowledge in the backend of the system so I can properly support my group if they need any help. I will begin creating a simple backend that can connect to the database and pass signals, and then begin figuring out how to transmit data being recorded by a mobile phone. After creating the bridge of signal transmission, I will start integrating the audio mixing algorithm in three stages (simple to complex).

- If any of my group members need help or assistance with any tasks that are on the other ends of the application, I will help them out in any way I can.

- As we go along in the development of the project, I will actively test the software for bugs and keep a log of all of the issues that arise.

- This semester I took on the task of designing the layout of the final design document with a table of contents that hopefully presents the material in an orderly fashion. I also helped merge everyone's pages together so that the document reads cohesively.

4.3 Jason Tan

- Design the front-end web application using React.js to implement the following functions:
    - creating and managing user accounts
    - Live recordings page
    - accessing previously recorded compositions from the user's own library as well as other users

- In addition to implementing the above functionalities, I must also design the website in such a way that it is aesthetically pleasing and easy to use.

4.4 Luke Blanchard

- Design and create a Node.js application to send and receive streams of audio data.
- Implement an npm module to encode and decode WAV and MP3 audio files.
- Design a Node.js module to take multiple streams of audio and mix them together
- Assist with mobile app and database development to help total system integration.
- Create a socket manager to receive incoming audio streams and communicate recording times with the mobile app.
- Research digital audio and audio processing techniques.

4.5 Yoannier Hermida

- Construct an immersive mobile application using Dart and Flutter for users to experiment with their interest in the John Cage Tribute, whose code is well-documented for potential future modification.
- Augment pre-existing Flutter audio recording packages to allow for the ability to transfer raw audio data from the client to a server.
- Research Java and Swift development practices to modify aforementioned Flutter packages' unique methods of handling platform-specific audio behavior (where Java and Swift correspond with the Android and iOS platforms, respectively).
- Design a comfortable and modernized user interface that combines the qualities of "John Cage-y" or avant-garde visuals with a familiar style akin to that of the web application for consistency purposes.
- Instill real-time capabilities during recording sessions through the use of web-sockets.

- Mediate the user's ability to access and play completed compositions, composed by others or by their own groups, through individual screens in the app.

- Assist server development's web-socket behavior and audio-mixing algorithm, due to their intricate relationship with the mobile application's real-time expectations.

5 Research

5.1 Frontend Technology

5.1.1 Web Page

React.js:

React is a JavaScript library specifically for interactive UI development. Essentially, it takes HTML and puts it inside JavaScript, especially with the use of JavaScript XML(JSX) which literally allows you to write HTML tags within JavaScript code that is compiled using the Babel compiler. Behind the scenes, it utilizes a virtual Document Object Model (DOM) of the webpage which is built on the back-end server, reducing the load that is handled by the browser. When a change is made, the system is able to compute the differences using a diffing algorithm and update the virtual DOM components accordingly, further improving the performance.

React works primarily through the use of components which are rendered as HTML elements in the virtual DOM. They can be implemented as either classes or functions which is important because, like a class or function in any programming language, they can be written once and then called or constructed anywhere else in the program, even inside other components to allow for the composition of different components. This reusability of components is great for data abstraction because components can be imported and used without actually needing to see the original code nor does the code that's using the components. It also helps with shortening and organizing code if you have a component that needs to be used multiple times. Instead of copying and pasting the entire HTML element, you can simply copy and paste the React component tag.

Another important detail of React is the existence of state within each component. All components can make use of an object called state which can be changed to render updates in the component. This can be used for a myriad of different purposes such as changing the text in a text field by a component's inner HTML dependent on a JavaScript string held in state or even changing whether a component is rendered at all by making it dependent on a Boolean variable held in state. In previous versions of React, the ability to

maintain state was limited to class components. However, as of React version 16.8, function components were also given state through the use of "hooks". These hooks give you access to variables that serve as state and functions that allow you to change that state just like that of a class component. Additionally, both types of components have what are called properties which are basically JavaScript values passed to the component using a syntax similar to HTML attributes. These values are held in an object that is received in the function parameters or class constructor parameters from which you can extract the needed data.

A similar alternative to React.js that I looked into is Angular.js, another web UI framework. This alternative was rejected for a few reasons. First of all, Angular is regarded by many as having a higher learning curve than React although, this is largely because it is a more complete framework. Also, React is dependent on certain companion libraries to do certain tasks such as Redux for global state management so in some cases, this difficulty probably cancels out. Another important reason is that React is generally faster than Angular due to its use of a virtual DOM as opposed to Angular which uses a real DOM. However, this is nullified if you consider the second version of Angular in which they implemented a method of change detection, bringing the performance of the real DOM up to speed with the React.

Now, you might be wondering why I'm countering every reason I bring up. It's because, as said in the article, [19]"Angular vs React: Which One to Choose for Your App" by Oleg Romanyuk, "*There is no better framework. Both are updated continuously to keep up with the competitor… In the end, React vs. Angular is all a matter of personal preference, a matter of skills and habits."* So, looking at all of the details of each one on my own, I think it would've been very hard to make a definitive decision on which framework to choose for this project. Luckily, I do have one reason that blows all others out of the water. The project sponsor himself requested a MERN stack and the customer is always right. Granted, it is sometimes possible to persuade the customer otherwise if you can make a significant argument for an opposing opinion. However, I have neither the experience nor the confidence to formulate an argument against Richard Leinecker in favor of using Angular or any other web UI development frameworks therefore, React it is.

5.1.2 Mobile Application

Overview of Technologies:

The mobile technologies to be incorporated for this project currently include the Dart language and Flutter framework for Dart, both developed by Google. They decide the infrastructure of the application's code and how it will be written. The technologies to complement the two are Visual Studio Code, Android Studio, and XCode. These, in turn, allow the code to be written and packaged into executables. All our mobile technologies listed, except for the App Store/Play Store app-upload process, are completely free. Although each of them can be set on any operating system, XCode is the prime exception, which is available on MacOS systems only.

The syntax of the Dart language, which borrows heavily from C, Java, and JavaScript, make it a prime selection for our group. Our curriculum, as C.S. majors, commonly expose us to the three languages mentioned earlier, thereby allowing us to learn Dart in a shorter time frame. Dart is also an object-oriented language, aligning it relatively closer to the Java language than the other two. The object-oriented programming course is a necessity as a C.S. undergraduate, which lends to our aptitude in using Dart's object-oriented paradigm in Flutter.

Flutter is a framework for Dart that is prepackaged with numerous components (known as "widgets") supplemented with complementary videos to bring curious developers up to speed. The core functionality in Flutter is carried over from other client-side frameworks, making it rather quick for web development veterans to acclimate with. State management in Flutter is trivial with respect to its BLoC pattern (business logic component). Unlike the ever-popular React Native, which requires an external container for its state to be manageable (using Redux), Flutter can use BLoC out-of-the-box. The BLoC pattern is represented as a sort of "global scope" class that handles all state management, making it quite easy to set up as well. In the event that an application requires phone-specific behavior to be used correctly, one must construct what is known as a native API (see subsequent section titled "The Native API"). Android phones' functionalities are largely built into libraries for Java and Kotlin, whereas iOS phones find their behavior locked exclusively into Objective-C and Swift.

Visual Studio Code is the essential text editor with incredible support from avid developers over the years. With plentiful plugins and extensions handy, VSCode can simplify the process of setting up needed technologies on any computer and operating system. It conveniently supports dual interaction between its code body and instances of the Terminal (Powershell or Command Line for Windows and Linux, respectively).

Installing packages to run the app and delivering content via Git for storing our app's code has become an almost immediate action through this setup.

Android Studio is the primary platform for exporting .apk files for Android users to download. Similarly, XCode is Apple's desired platform to produce iPhone applications. Both platforms contain phone emulators that allow developers to run their Flutter applications to bypass the requirement of buying the phones themselves. As such, a developer can comfortably test their mockups without needing to repeatedly export app files for running it on their own phones. Despite XCode only being accessible on MacOS systems, the act of exporting our mobile application on both iOS and Android allows for a greater target audience to enjoy the genius of John Cage.

Analysis of Dart:

Dart is the selected programming language for the mobile app aspect of the John Cage Tribute. Dart was developed by Google on October 10th, 2011 and designed by Lars Bak and Kasper Lund. It is an object-oriented, class-based, garbage-collected language that supports static- or dynamic-typing. One of its largest perks is its familiar syntax, bearing C-style syntax in conjunction with functionality that is also present in Java and JavaScript. As such, programmers with a background in these languages will find themselves at home with Dart. Despite its inclination to object-oriented code, it is actually a multi-paradigm language; it can support functional, imperative and reflective behavior when the need arises. In fact, when used in tandem with Flutter as its framework, Dart is seldom used without a combination of object-oriented and functional patterns.

The language is laden with numerous benefits that make it a promising candidate for the Tribute. Due to its cross-platform nature, Dart finds itself inhabiting applications with the intention of being released on multiple platforms. This is a crucial component for the John Cage Tribute's goal as a multi-platform application. Android and iOS, two linchpins of the mobile industry, frequently associate with Dart-based applications and as a result, blossom a plethora of discussions that improve Dart documentation. Evidently, the Dart language has a hand in audio-related software as well, which can prove handy during the construction of the mobile app's audio-recording component.

Like JavaScript, Dart supports the use of Promises (referred to as "Futures") and Promise resolutions. Continuing that trend, the implementation of the "async" and "await"

keywords, labels that, like Promises, indicate an asynchronous series of events, have found themselves in Dart as well. The use of lambda functions and their JavaScript shorthands, arrow functions, are frequently encountered in robustly-made Dart applications for the added effect of conciseness and simplicity. Dart may be compiled in numerous ways, with its compilation into JavaScript serving as the convention for Dart-based web tools. In other scenarios, it may rely on its own VM, the Dart VM, for compilation or even with ahead-of-time compiling into byte code. This AOT-compilation approach is commonplace for Dart applications that rely on Flutter's framework.

One of Dart's most unique perks is its implementation of a ready-to-read data structure known as a StreamController class. The StreamController is capable of receiving data from numerous sources (in other words, numerous functions that pass data to it) by being "listened to." In addition, it sports three other invaluable properties: a Stream class (an object that can provide access to any data passed to it and is automatically updated in that vein), a Sink class (another object, used for passing data into it), and a StreamTransformer class (used to directly modify any data passed to it under certain conditions that the developer chooses). This is an effective tool for representing any data that may be received from the audio-mixing server. The Stream updates in real-time just as the server intends to pass mixed audio for the listener.

Lastly, it supports multi-threaded capabilities through the use of Isolates. Isolates are workers that rely on their own segments of memory to perform their assigned tasks. These threads are unable to communicate directly, instead using a message passing system to allow tasks with shared parameters to intertwine between them. As of August 2018, Dart 2.0 introduced an additional feature to augment its concurrency property: web workers, a feature also present in JavaScript. Web workers allow for script processing and execution in background threads without the need to interfere with foreground processes, such as the user interface. As such, the audio recording process can be assigned as a background process as the UI maintains its normal behavior.

Analysis of Flutter:

In tandem with Dart, Flutter is the framework to embolden Dart's toolkit for the John Cage Tribute's mobile app. Flutter is an open-source UI application framework for the Dart language. Under the codename "Sky", Flutter was unveiled to the public in 2015 and eventually released by its developer, Google, in May 2017. Flutter, like the language it builds upon, is still a relatively new tool. This youthful property, alongside its

developer's reputation, marks it as a rather alluring gem in the eye of ambitious developers today.

Flutter's architecture consists of its aforementioned language platform, its engine, its expansive library, and a fundamental aspect of the framework, known as "widgets". With thanks to the Dart VM, Flutter gains a just-in-time execution (computer code executing on runtime) that allows for a "hot reload", a quick reload of applications upon saving any changes to their code. Flutter's engine retains the cross-platform interest like Dart, communicating with prominent platforms such as iOS and Android. It supports a myriad of essential features that are expected of professional applications, including network I/O (which includes streaming), plugin extensions, graphics, animations, and accessibility services. In the wake of the Tribute's completion, it can be augmented even further with use of these features to meet substantial standards of the average high-quality app. Flutter's widgets are the crux of its library, serving an assortment of purposes in a typical Flutter application. They can be likened to fleshed-out Java classes or components in ReactJS, imitating an object or behavior that shape the application's interface.

Widgets come in a variety of shapes and sizes, borrowing well-known design implementations found in HTML and stylization in CSS to illustrate familiarity and efficacy. The Flutter library and API is, in essence, the widget compendium. Its documentation on each one is considerably detailed and action-oriented, a reasonable expectation from Google. The library divides its widgets into two categories: material widgets (a typical interface widget) and Cupertino widgets, which adhere to Apple's Human Interface Guidelines iOS design. This is a particularly promising event, considering the process of Apple's App Store requiring apps to be confirmed to be valid and meaningful in order to be hosted. The Flutter widget is capable of managing the state behavior of its application by listing itself as "stateful" or through a newly supported formal paradigm known as the "BLoC Pattern." First and foremost, these widgets, whether stateless or stateful, use a design function known as a "builder" to decide when and how information should be presented to the user. These builders rely on state to execute, with stateful widgets interacting with builders with ease. Stateful widgets manage their state directly in the component that depends on them. The BLoC pattern is difficult to master, but abstracts micromanagement of state to a single file or a series of state management-exclusive files, communicating skillfully with builders and other widgets when designed with success.

Rejected Alternatives for Mobile App Development:

The mobile app industry revels in its rich diversity of languages and frameworks to accommodate their development needs. Similarly, the Tribute's requirements are broad but decisive, allowing for considerable technology flexibility. Among the numerous available mobile application development environments, a few options were labeled as worthwhile candidates: JavaScript (programming language), React Native (JavaScript framework) and Mobile Angular UI (JavaScript framework). As it appears, JavaScript has gained tremendous traction as an environment for app development, whether on the web or on mobile. The John Cage Tribute has found itself nearly catering to this route, as well.

React Native is Facebook's flagship for creating mobile apps and with good reason. Its seamless flow between screens and intuitive structure are major perks amongst mobile frameworks. Its immediate familiarity with the React library would also make it quick to be picked up by our members that wish to study the MERN stack. In fact, the "hot reload" behavior that Flutter exhibits, a means to quickly reload an app after having modified its code, is also supported by React Native as well. Despite these numerous advantages, React Native still suffers in regards to its native support amongst the platforms it works under. It is compatible with iOS and Android devices, but this is a bit of an overstatement. Its capability to interact in a native manner with its device's platform, in other words, with the features found in the device under its platform's lens, is limited by its developer. Some of this core functionality is not abstracted away and is therefore up to the developer to determine how to interact with these features through the construction of Android- or iOS-exclusive modules (expected to be written in Java or Swift). Seeing as these tools are not out-of-the-box, it is an easy medium by which a mobile developer in-training can feel overwhelmed or construct an application just barely held together by its mediocre code.

Mobile Angular UI is another framework by Google under the interest of complementing their AngularJS web framework (the "predecessor" to their Angular web framework). It takes advantage of Bootstrap and AngularJS to allow quick and effective creation of HTML5 mobile applications. From a glance, this appears to be quite helpful for diving headfirst into the mobile app's creation under a shorter time frame, allowing for further prioritization of the server's music-mixing algorithm. As a result, however, applications borrowing from this framework tend to appear more simplistic and less sleek, eliminating one crucial factor of constructing a mobile UI that bears visual similarity and quality to the web application aspect of the John Cage Tribute.

Lastly, the rejection of the interest to utilize JavaScript as the language to wrap the whole mobile application together was understandably difficult. With usage of JavaScript on the mobile side, there would be a result in a unanimous use of one language across the entirety of the John Cage Tribute, allowing for incredibly fast reaction times to work completed among the frontend of the web app, mobile app, and backend. In addition, the learning of the framework associated with JavaScript would also be quick to garner, reducing learning time and increasing productivity. The conclusion against these advantages was precisely the status of Dart and Flutter as entirely different entities from the familiar JavaScript tools. An undertaking of this magnitude demonstrates a masterful skill-set obtained from having trudged through an entirely new set of languages and frameworks. This is further magnified with consideration to Dart and Flutter's emergence as up-and-coming tools. Although their documentation is limited, they are rapidly increasing in popularity, just as JavaScript itself has done in the past. The industry will bear a newfound sense of pride toward individuals that exhibit technical depth as well as technical breadth. Dart and Flutter are key tools to blossom such perspectives to life.

The Native API:

At times, there is certain behavior that cannot be construed whatsoever in the user interface. As such, Dart and Flutter may not quite serve as enough of a service to provide for this behavior. A primary example of behavior that is not available in these services is audio recording, a functionality that is absolutely essential to the John Cage Tribute. It requires platform-specific code to be executed in order for the user interface to understand that the user wishes to record audio. To achieve this, the mobile application must support what is known as the "native API", or the "platform-specific API."

Evidently, Flutter is aware of this. Upon the generation of a project, Flutter will construct two subdirectories, labeled "android" and "ios", to make room for a developer's platform-specific code. To develop Android-specific code, one must open Android Studio and open the MainActivity.kt or MainActivity.java file located in the kotlin/java subfolder of the aforementioned android folder. For iOS, the similarity is nearly perfect. Xcode must be opened, navigating to the ios folder of the project. Thereafter, one must select Expand Runner > Runner in the Project navigator and open AppDelagate.swift (or AppDelegate.m for Objective-C) and begin the work. These files, by convention, are renamed to the overall purpose of the native API, such as FlutterAudioRecorderPlugin.

For the sake of modernity for the Tribute, Swift will be used for iOS and Kotlin will be reserved for Android. A detailed explanation of their selection over their counterparts is written in detail below.

Analysis and Decision for Swift over Objective-C:

Swift is Apple's flagship programming language and the answer to its prayers toward its use of Objective-C since 1988. Serving multi-paradigm (supporting imperative, object-oriented, functional, and more) and compiled functionalities, it is meant to target a wide audience and behave as the future engine that will power all subsequent Apple products. Swift's development began in July 2010 by developer Chris Lattner, citing its inspiration as countless numbers of popular languages at the time. It was unveiled on June 2nd, 2014 at Apple's Worldwide Developers Conference as a gradual replacement for Apple's Objective-C foundation. Since 2015, it has remained open-source and can be accessed at apple/swift on GitHub.

The language specializes in melding together some of Objective-C's appreciated abilities, including dynamic dispatch, late binding, and extensible programming, all the while providing easier opportunities to catch accidental bugs in its software. It is heavily invested in providing syntactic-sugar as a means of simplifying large amounts of code for the sake of aptness and avoiding what is known as the "pyramid of doom." In short, when code begins to reveal gradual and excessive tab usage over its creation, the tab spaces form a sort of pyramidal shape, known as the pyramid of doom.

Its syntax is highly familiar, demonstrating most of C's operators and by implication, Objective-C's. However, it lacks the necessity of semicolon usage, header files, and more. Some of its most essential changes from Objective-C pertain to its disallowance of pointer handling (without specific declarations) and integer overflows. Countless developers across time have groaned in dissatisfaction at the overwhelmingly large amount of memory handling issues that go unnoticed while using C-like languages, which Swift aims to swiftly dispatch. It also labels assignment operations as operations that return no value, preventing the ever-common error of writing one equals instead of two in conditional statements.

In conclusion, time is of the essence for the John Cage Tribute. Finding effective and easily readable shorthands while maintaining an incredible degree of safeness against memory failure means that there will be a significantly decreased amount of debugging

necessary to ensure the minimal work necessary for the app to work. Although it may take additional research to grasp Swift's initially tricky techniques, this saves ample time for the focus of our mobile developer, Yoannier, to remain longer in the user interface aspect of the app. The native API does not necessitate substantial attention if its sole purpose is to provide an outlet for requesting and delivering microphone-related functionality from the phone to the app itself.

Analysis and Decision for Kotlin over Java:

Kotlin is a cross-platform programming language with the expectation to be used for a wide variety of applications (labeled general-purpose). It is statically typed, meaning that any usage of variables requires a definitive type in order to be used. It was unveiled in July 2011 by JetBrains (previously known as IntelliJ Software) to respond to some of Java's lack of features that Scala apparently had, but did not have the compilation quickness that Java bore. Its name stems from Kotlin Island as a nod to Java's name possibly stemming from the island of Java in Indonesia.

Kotlin is what some may refer to as the "cooler brother" of Java. Kotlin's aforementioned properties are found in Java as well, making their primary purposes practically identical. However, it sports the ability to declare raw types for preparing against abstract forms of data, typically found when receiving data between client and server. A universal pitfall that all developers have certainly come across, known as The Billion Dollar Mistake, is the error of accessing a member of an object instance that is null. In Java, this is known as a NullPointerException. When this occurs, there is no means for Java to protect against the app throwing an error and exiting its current event loop. Kotlin's remedy for this is support with explicit calls for handling such kinds of errors, along with an operator (written as !!, also found in Dart as ??) to have a default result to fall back on in the event that this occurs. It can also integrate Java code into itself and process it, allowing Java development teams to transition to Kotlin far more easily than other languages. In addition, Google has even made an official statement in 2019, declaring Kotlin as its preferred language for Android development over Java.

Similarly to what is specified for Swift over Objective-C, the intent of the native API is to construct the required essentials necessary to initiate audio recording and data passage to the client side. Speed and lack of fallacious code behavior is exactly what the mobile app can benefit most from for its native API. Learning Kotlin will likely be far simpler due to its similarity to Java being closer than Swift's similarity to Objective-C.

Relevant Flutter Plugins:

Shortly after completion of a valuable Udemy course on the study of Dart, the necessity of audio capability for recording purposes became apparent. As expected of the youthfulness of Flutter, only a limited quantity of audio-oriented plugins were present either through GitHub or Flutter's website. A few notable examples include: audio_player (a well-maintained audio player, not a recorder), flutter_audio_recorder (supports WAV files), and flutter_sound (lacks lossless/uncompressed audio support, allows byte access and synced with audio_player). Inspection on the remaining audio plugins concluded with discoveries of mediocre API and plugin documentation, duplicate plugins of the aforementioned three above, or incomplete/inoperable plugin projects.

- Flutter_sound:

The plugin flutter_sound lacks the desirable lossless audio property that is tantamount to the server's drive toward a quick conversion process, but supplies a rather robust example of an audio recorder/player combination for building off of. Its example app borrows its capabilities from the audio_player plugin itself and has constructed its own recorder, to boot. It can serve as a basis for constructing a more professional-grade audio recorder screen that users consider intuitive to its purpose. The flutter_sound plugin supports a number of methods that innately allow a "subscription" to the audio being recorded. In other words, it can "listen" to the audio and peer into snippets of the recording audio for a given period of time, which is the subscription itself. Seeing as this is not available in flutter_audio_recorder, it can prove useful to simplify the process of extracting these bytes for usage in a stream.

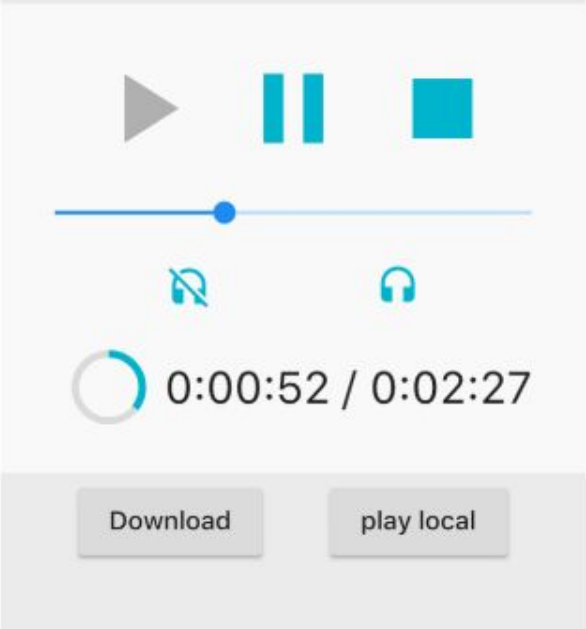


Figure 1: The audio_player borrowing Flutter's versatile widgets. This lends credence to Flutter's high quality UI design.

- Flutter_audio_recorder:

The plugin flutter_audio_recorder is the plugin that will be used for allowing the recording process to actually occur. The data recorded from the mobile user's environment into the plugin comes in the form of bytes of data, listed as 16000 bit depth and with 44100 samples per second. These bytes of data, however, are not constructed natively through Dart alone. The flutter_audio_recorder development team used alternate programming languages for each required mobile platform (Java for Android and Swift for iOS) in order to garner these bytes of data and allocate them to a buffer for general use. The files that mediate these events can be found in the android folder and ios folder, respectively. The Android file's name is (as of the completion of this document) FlutterAudioRecorderPlugin.java, whereas the iOS file's name is SwiftFlutterAudioRecorderPlugin.swift. In order to accommodate the lack of access to the individual bytes, these files will have to be modified to support such behavior. Thereafter, the flutter_audio_recorder.dart file can be adjusted to contain methods that access the modifications. This process can be accomplished with assistance from the flutter_sound's subscription methods, as mentioned in the above section.

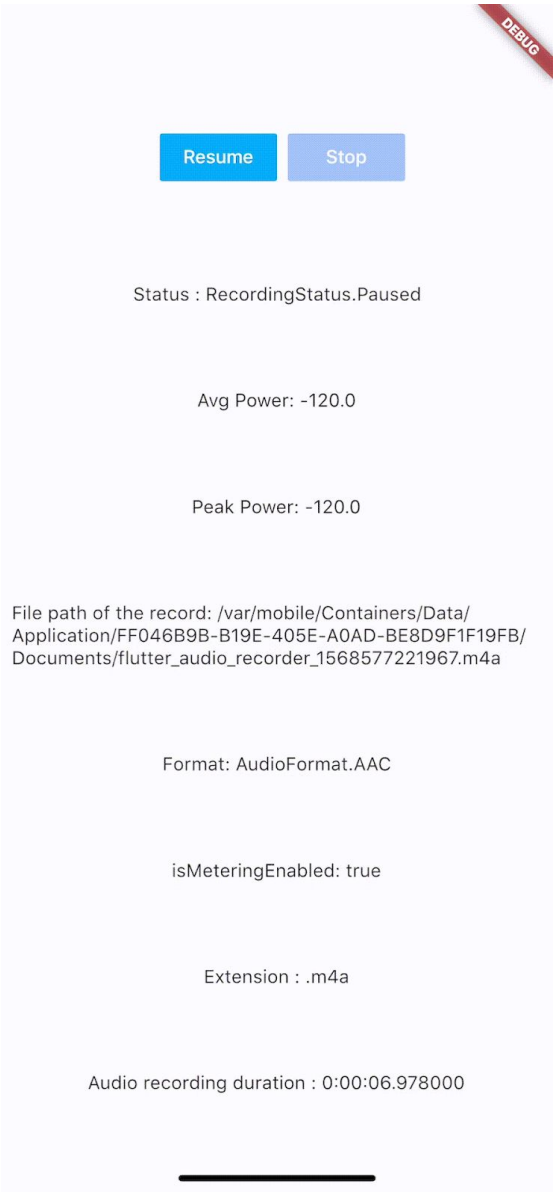


Figure 2: A still taken from flutter_audio_recorder's demo. Note the audio format and duration. These are valuable aspects of the plugin.

- Permission_handler:

The permission handler is a simple yet effective plugin for ensuring the security measure of asking a user for permission to use their microphone for recording purposes. Its methods are easy-to-use, returning a failure value in the event they decline and a success value if they accept. As the response is being awaited, a "Future" value is returned, which indicates a pending answer.

- pin_code_fields:

Any intention to design a PIN-entering system is thoroughly quashed with special thanks to a Flutter developer's special plugin. Pinput is a widget composed of smaller widgets that act as single-character fields for inputting values into. For an example, see the "Mobile App: Authentication Screen" page.

- Web_socket_channel:

This plugin is Flutter's version of a websocket-handler that can connect to any active websocket available. Its actual websocket is conveniently represented as one of Dart's own data structures, a StreamController class. Thus, emitting data through the websocket is as simple as interacting with the StreamController's sink and receiving data is identical to checking the controller's stream. The concepts of sinks and streams are extensively discussed in the "Analysis of Dart" section.

- Email_validator:

An out-of-the-box email validation plugin that dynamically checks a user's email for validity without the use of regular expressions. Using the EmailValidator class and its validate method, email checking is nearly a few lines away!

ReactiveX:

A ubiquitous and popular collection of open source projects, it finds its applications in countless modern-day company products and programming languages. ReactiveX is short for "Reactive Extensions", signifying the intent its projects all share under its collection. Its projects are prefixed with a shorthand, "Rx", before addressing the language that its project is suited for. In general, it aims to remedy the common pitfalls that most stream-supported programming languages that software developers run into. As such, ReactiveX shifts its interest to the asynchronous programming world. A frequently encountered example would include the lack of fluidity that some programming languages' streams contain, including their observability, lack of summarization of multitudes of streams, and simplicity of use. A few notable programming languages that ReactiveX has developed for include:

- JavaScript
- Java
- Clojure
- Python
- Go
- Swift
- Kotlin
- Dart
- And so on...

ReactiveX attempts to almost rewrite these stream classes and its associated components altogether, renaming them as “Observables.” The observers bear an underlying observer that subscribes to the sequence of observables, carefully obeying the order that they were initiated. This ensures that the expectations of the developer are met. This also reflects the aim of ReactiveX to re-represent these streams as simpler-to-understand “iterable items” (N.B.: similar but not identical to Dart’s Iterable class).

This iteration pattern is how the labeling of subscribed observables is now far easier to attribute error handling, data success handling, and intermediate data handling. Of notable interest is the “intermediate data handling.” A software engineer may apply functions to modify a received observable to return a subsequent observable, further operating on more observables that may be subscribed to at a later period. This chain of the manipulation of observables creates seamless data modification in the matter of a few lines of code, whereas the same process without ReactiveX would warrant numerous separate documents to achieve.

For the context of the John Cage Tribute, few of the numerous projects that ReactiveX has worked on will be expected to be applied. Of the ones that are particularly important are RxDart (UI language), RxSwift (native API code for iOS), and RxJava/RxKotlin (native API code for Android, only one of the two will be relevant). RxDart, excluding the audio recording plugins, will most certainly function as the most important plugin that this project will require. This, alongside its complexity, is what warrants its expansive research in comparison to other plugins,

RxDart:

ReactiveX set its sights on augmenting Dart’s occasionally onerous Stream classes

around the final months of 2016. Incidentally, this was shortly before the advent of Flutter in the year 2017, which helped skyrocket RxDart's popularity in the programming community as a result. The project is entirely open-source, being available as a repository on GiHub via ReactiveX/rxdart. As per its readme file, its primary function is to reinforce Dart's Stream and StreamController classes. Interestingly, RxDart had initially sought to replace Streams altogether with Observables, but eventually reversed this decision to simply make the Stream class more robust. In summary, the plugin directly impacts three essential functions of Dart:

- Stream Classes
- Extension Methods, and
- Subjects (modified StreamController Classes)

With RxDart, Streams now come in a wide assortment of flavors, each for their own unique purpose. Streams may now be merged together, send requests intermittently upon failure, await subscription from an observer before generating itself, emit a value after awaiting some duration, and more. These added properties breathe life into one's Dart-based applications without reinventing the wheel and potentially causing an error. Among the most needed Streams from this list is the CombineLatestStream, which performs the aforementioned merging. As an example, a developer may merge Streams that listen to a user's inputs of an email and password, designing one large Stream that automatically allows entry once these two criteria are successfully met. This provides endless UI-related ideas that make navigation more elegant than imagined.

Extension Methods, as their name implies, are methods meant to operate on an existing Stream instance. They cause the source Stream they operate on to become a new Stream, retrofitted with specialized event behavior. Certain operations include buffering sequences, test-case event handling for Streams, delaying, setting a default value for Streams, etc. Some other primarily used extension methods are best summarized as "list comprehensions", or a shorthand manipulation of lists of data in the form of mapping, reducing data to a single value, accumulation, and more.

Last but not least, RxDart supplies an augmentation of Dart's StreamController classes, masking them as "Subjects." These Subjects first and foremost polish versus Dart's implementation is the abstraction of Dart's design redundancies and preservation of the StreamController's innate qualities. They essentially act as "wrappers" to this class, bringing along with it numerous variations that parallel the diversity of RxDart streams.

Of note is the forefront of the Subjects, the PublishSubject. PublishSubjects act as Dart's broadcast StreamController (where a broadcast StreamController is one that can be subscribed to more than once) while fusing the functions of a stream and a sink together. In other words, one can now access their constantly-evolving data and access the ability to add data or modify pre-existing data in the same object. The BehaviorSubject class is tunnel-visioned for emitting the most recent data it receives as the first thing any subsequent listener encounters when it begins listening to it, while also exhibiting a broadcast property. Lastly, the more complex ReplaySubject class attempts to bundle up every added item (up to a specified maxSize, if given) into a field to give any subsequent new listener, acting as a verbose variant of the BehaviorSubject.

The App Export Process:

The John Cage Tribute's mobile application is to be exported on both Android and iOS platforms. The source code will be designed entirely within the confines of Visual Studio Code. Although it can be written in XCode or Android Studio, these two applications will instead relay themselves the task of exporting the appropriate files to their respective platforms. As the project is expanded upon, the prototypes (Wood, Fire, and Divine Intervention) are to take advantage of the exporting process for user feedback purposes.

Android Studio is home to all things Android. It provides access to an Android phone emulator of numerous kinds, each with their own capacity to support different versions of Android firmware. With such a beneficial tool in hand, we can mimic many kinds of users' phones and even replicate niche errors they may encounter, should the need arise. Currently, our workspace utilizes the Pixel 3 while operated with the latest Android firmware available. Android Studio's export process for testing purposes is particularly simple. Through its Export command, an APK file can be produced (with a file extension of .apk). This APK can be installed and run on most modern Android phones, while may also be distributed via email. Exporting mobile apps to the Google Play Store is relatively straightforward. After creating a Google Developer account, the developer must use or create a new keystore. If selecting Create Keystore, they must pay a one-time dev fee of $25 and generate a sign on their APK. Then, one may create their official application through Android Studio. After selecting the Create Application command, its name and graphics assets can be set for the app to export. Finally, some info pertaining to the app can be filled before selecting the APK file to publish.

XCode is Apple's IDE equivalent to Google's Android Studio. It is, in my perspective,

rather cumbersome to create applications on compared to Android. At the very least, XCode also provides a simulator developer tool of its own. The Simulator is a separately run program that can be accessed on XCode's startup screen. Nearly every iteration of the iPhone can be simulated here. Flutter's hot reload can come in handy to reload one's test app, but these prototypes cannot be distributed or published without an Apple Developer account. TestFlight, an application that produces up to 100 copies of a prototype for distribution and testing purposes, will also require an Apple Developer account. Unfortunately, this is where Apple's inconvenience rears its head. To enroll in the Apple Developer program, an iTunes Connect account must be made. An account in the Apple Developer program requires an annual fee of $99, in which different kinds of developer programs may be selected, but only the standard account is required. The Enterprise program is not required for the scope of this project, as it is only meant for private distribution amongst employees in a company for internal or proprietary use. Once the account is set up, iTunes Connect will behave nearly identically to Android Studio's Create Application command. Before a valid .ipa file may be selected for upload, it must receive a distribution certificate signature, which requires a Distribution Provisioning profile, both of which can be completed in XCode. Once the .ipa file is uploaded, Apple will review the mobile app and publish it if it obeys Apple's guidelines. Otherwise, they will return a list of reasons why they may find the app unsafe to publish, which can be appealed. Only then may the app be entirely available on the App Store.

5.2 Backend Technology

5.2.1 MERN Stack

For this project, we will be using a MERN stack for our web application. MERN is an acronym for its four main components: M(ongoDB), E(xpress), R(eact), and N(ode.js). We considered using other stacks, such as a LAMP stack or a MEAN stack, but we ended up going with a MERN stack because some of the team members were already familiar with it.

- MongoDB is a NoSQL document-oriented database. MongoDB uses JavaScript Object Notation (JSON) like objects in the form of documents composed of key-value pairs.

- Express is a Node.js framework. It is used to make writing code in a Node.js

environment easier than writing it in just Node.js.

- React is a JavaScript library. React is used in the development of the front-end web application. It also allows users to code in JavaScript and create UI components.

- Node.js is a JavaScript runtime environment. It allows users to execute JavaScript code outside of a browser.

5.2.2 Node.js Dependencies

Along with Express, there are several other dependencies that will be used to develop the backend:

- Express-validator:

Express-validator is a set of express.js middleware that wraps validator.js validator and sanitizer functions. The backend will use express-validator to make sure that the parameters added by those trying to post information to the database.

- Mongoose:

Mongoose is a MongoDB object modeling tool. As such, it is the primary communicator between the server-side and the data stored in the project's database, created with MongoDB. It handles synchronous and asynchronous code, using JavaScript's Promise class to represent any asynchronous data. It creates connections with the database and constructs schemas to imitate the data that would be received from it, passing them to the client-side when it requests it. Due to MongoDB's status as a NoSQL program, it lacks stringent relationships. This is now Mongoose's job to establish them in the event that the project requires it. Examples include registered users possessing their own compositions, which means a relationship between the two must be made.

- Bcrypt.js:

Bcrypt.js is a library used to hash passwords. In our backend, it will be used to hash the passwords of our users who create an account.

- Jsonwebtoken (JWT):

The JSON Web Tokens, or JWTs, are individual components of data that are composed of singular, albeit lengthy, hash value strings encoded in JSON format. Their intention is to secure an interaction between parties, usually for authentication purposes. The actual module associated with these tokens is called Jsonwebtoken, simplifying the security process tremendously with out-of-the-box methods such as jwt.sign (for signing and generating a JWT).

- Config:

Config allues to set default parameters throughout the backend. One of its main purposes is to set global variables throughout multiple files. This is used to store the JWT secret token, as well as the mongoURI.

- Multer

Multer is a node.js middleware used to handle multipart/form-data, which is primarily used when uploading files. There are two parts to a multer object, a body, which contains JSON like information, and file or files, which contains one or many files, respectively. Since a major component of our project is to store MP3 files on a database, having the middleware is essential to our project's success.

- Node-Fetch

We have several api calls made directly from the server itself. To accomplish, we utilize Node-Fetch, which

- Wavefile:

An npm module for decoding and encoding WAV files. The main functionality we utilize from this module is the encoding of raw PCM audio data into a WAV file buffer. This is needed to transform the raw audio data from a recording session into something we can save to our database and playback in the future.

- Node-Lame

An npm module that allows Node.js processes to utilize LAME, an open source MP3 file encoder. We use this module to transform WAV files generated by recording sessions into compressed MP3 files to save in the database. It works with WAV file buffers and produces MP3 files with any bitrate you choose.

- Child_process

A module included in Node.js that allows the use of multithreading. Node.js by default runs in a single thread at all times, but since the server will be doing multiple intensive processes at the same time it is necessary to create multiple threads in order to ensure stability. This module allows for the creation of separate Node.js processes that can communicate with each other but still operate as separate threads.

- Socket.io:

In order to have communication between the different components of our system, we need to utilize socket connections. Socket.io is a library that enables real-time bidirectional event-based communication using socket connections. The connections between different components of a system are based around a single Node.js server, and any number of web based or Node.js clients. Information can be passed from the server to clients and back, making this perfect for our purposes. Any serializable data structures can be sent, meaning we can pass audio sample data between components. For our system, the Audio Processing Server will be the socket server, while the mobile app, website, and database will establish client connections. This will allow full connectivity between all components of the system, with every piece able to communicate with the others. Socket.io most importantly is used to host rooms that performers and listeners can enter using a pin from both the mobile application, the website will host a live recording page that shows a list of joinable rooms. Just enter the pin and you're ready to go.

- WS:

This module (stylized as ws) is a counterpart to socket.io. Its socket communication is built for speed, but supports less complex behavior as noted in the socket.io library. As a result, this module is of considerable importance in the event that socket.io is not of sufficient speed for the project's audio transfer functionality. In exchange, the sockets will become more rigid and only support basic behavior, such as "onConnection" (execute something when connection ensues), "onMessage" (when a message from the client's socket is received), server creation, and client authentication.

5.2.3 GridFS

Normally when storing files, you would store them on the server itself as opposed to the database. With MongoDB, however, you can use gridfs in order to store files on a MongoDB database. Files on MongoDB are stored in two fields. The first is gridfs.files, which when created, a unique ObjectID is generated, and the information of the file, such as the type of file being stored and the length of the file in bits. The second is gridfs.chunks, which is the raw binary information of the file, as well as the ObjectID generated by gridfs.files. Normally there is a limit to the size of objects that can be stored on MongoDb, which is 16 MB. Gridfs.chunks gets around this by breaking up the binary information of the file into smaller chunks and then storing each one separately in gridfs.chunks.

In order to add files to a MongoDB database from a multer object, **multer-gridfs-storage** is used. Using

5.3 Audio Processing Algorithm

Our research was broken up into two main categories: researching the technologies for our software, and researching the works of John Cage. When going into this project, we knew that the biggest hurdle we would have to face is understanding enough about the works of John Cage in order to accurately replicate his style of compositions. For that reason, we dedicated a lot of our time looking into John Cage and his different works. One very helpful resource that we had was Dr. Thad Anderson from the UCF school of music. He did his doctoral dissertation on John Cage, and he pointed us in the right direction for some of John Cage's works that we should check out.

5.3.1 John Cage Influences

With this project being a tribute to the late John Cage and his many compositions, it only makes sense to use some of his works as a model for what our own mixing algorithm will look like. Going through his library of works, there is quite a lot that we can potentially choose from. I would like to talk about the two compositions that we have eventually narrowed down to.

One of the potential influences on how our mixing philosophy that we narrowed it down to is one be modeled is John Cage's Imaginary Landscape number 5. The Imaginary Landscape series, which started off as a mix of both percussion and technology, shifted to just being focused on technology by the fifth composition. Imaginary Landscape number 5 was originally created with the use of records, and these records would be played when the score dictated them to be played. Each grid represents a specified unit of time. The score may also feature numbers below the grid that range from one to eight, which dictate the dynamics, or volume, that the tracks would be played at. These dynamics could be static throughout when the track is meant to be played, or they can change throughout the duration of the sample. The units of time that each track would play at as well their dynamics was determined by the *I Ching*.

Like Imaginary Landscape Number 5, our algorithm will be dealing with recordings instead as opposed to more traditional instruments. In our case, the individual tracks would be the recordings that the performers create. To simulate the I Ching, we could use anything from a simple random number generator to using the coordinates of the performers and having that generate a seed. And like Imaginary Landscape number 5, this randomness will determine when, how long, and how loud the samples would be played at.

Another potential influence for this project could be modeled after John Cage's Number Pieces. Cage composed music up until the day he died. He received so many commissions at this time that he didn't have time to write a traditional score for them. So instead, he gave instructions of what to play and gave them a range of times for them to start and stop playing. The actual notes and rhythms were up to the performers, so the performers would improvise and play off of what the other performers were playing. The reason that these pieces were known as the Numbers series was because of the naming scheme John Cage used for the individual pieces. It would be titled with an alphabetical representation for the number of how many performers it was composed for, with a numeric written in superscript indicating which version it is. For example, John Cage's

second work for three performers would be titled Three[2]. These compositions ranged from solo pieces to full orchestral pieces compositions that necesitate over one hundred performers.

An example of a Numbers piece that we are trying to accomplish would be Four[4]. Four[4] is a piece for 4 percussionists in which each performer chooses around 5 different instruments and then plays them when instructed by the score. For example, the second performer might be instructed to start playing an instrument at a time between twenty and forty seconds, and to stop playing between one minute and a minute and twenty seconds in. What we could do to simulate this in our software is to have the recordings played back to either the composer or the performers and have them modify the individual tracks as they see fit.

Both John Cage's Imaginary Landscape Number 5 and his Numbers series are great reference points and there is a chance that we are going to likely include elements from both of them in our algorithm. Both of these models come with their own strengths and weaknesses. For the Imaginary Landscape model, the biggest drawback is that it takes away control from the performers, instead it leaves everything up to the *I Ching* style of randomization. Whereas with the Numbers, it allows for a certain amount of control that can be given to either the composer or the performer. The biggest problem with the Numbers model is that with the delay, it could be really hard to mix the recording from the composer's end, and would be nearly impossible on the performers end due to the fact that they would be hearing their own past recording. The Imaginary Landscape model does not have this issue.

5.3.2 Digital Audio and Audio Formats

When processing audio it's good to have some understanding of how digital audio works and what the different digital audio formats are.

Sound in its natural state is a wave, like a sine or cosine wave. To store sound digitally we need to represent a continuous wave as a finite set of values. These values will individually represent the position of the sound wave at a specific point in time. In digital audio these values are referred to as *samples*, with the number of samples in each second of audio referred to as the *sample rate*. These samples are stored in binary values with the number of bits storing each value referred to as the *bit-depth*.

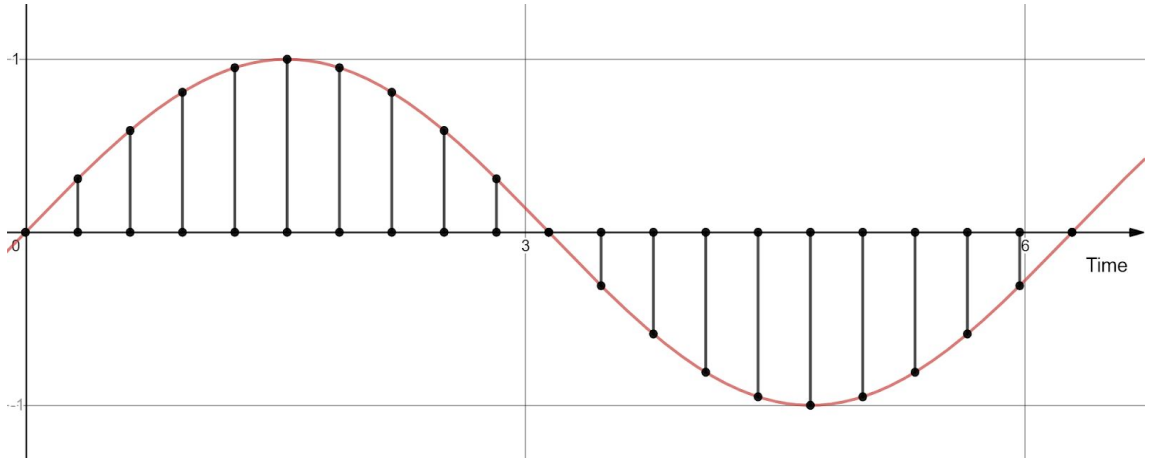


Figure 3: Here we see a sine wave being sampled 20 times in the span of its period.

In digital audio when the bit-depth and sample rate are larger the quality of the recorded sound is higher. However, larger bit-depths and sample rates will also increase the audio file sizes.

The most common bit-depth and sample rate for storing audio is 16-bit 44.1 kHz, which is the same quality as a CD. We've chosen 16-bit 22050 Hz as the quality for our system as it offers very high quality while still allowing for manageable file sizes.

The method of representing an analog sound wave as digital samples is known as Pulse Code Modulation, or PCM. PCM sample data is stored sequentially as it was recorded.

For example, in figure 3 the resulting PCM data would be:
[0, .31, .59, .81, .95, 1, .95, .81, .59, .31, 0,− .31,− .59,− .81,− .95,− 1,− .95,− .81,− .59,− .31, 0]
Stored as binary data.

PCM audio data cannot be stored and played back by itself however, it needs some additional information so that a computer can recognise it. To accomplish this we use the Waveform Audio File Format or WAV. This file format contains a header that tells a computer things it needs to know about the audio data such as the bit-depth and sample rate. A WAV file can store different types of audio data, but for our purposes it will contain 16-bit 22050 Hz PCM audio data.

The 16-bit samples are read as signed 16-bit integers, meaning they have a range of -32768 to 32767. These are the numbers that we will be working with directly to

analyze, manipulate, and mix the sound.

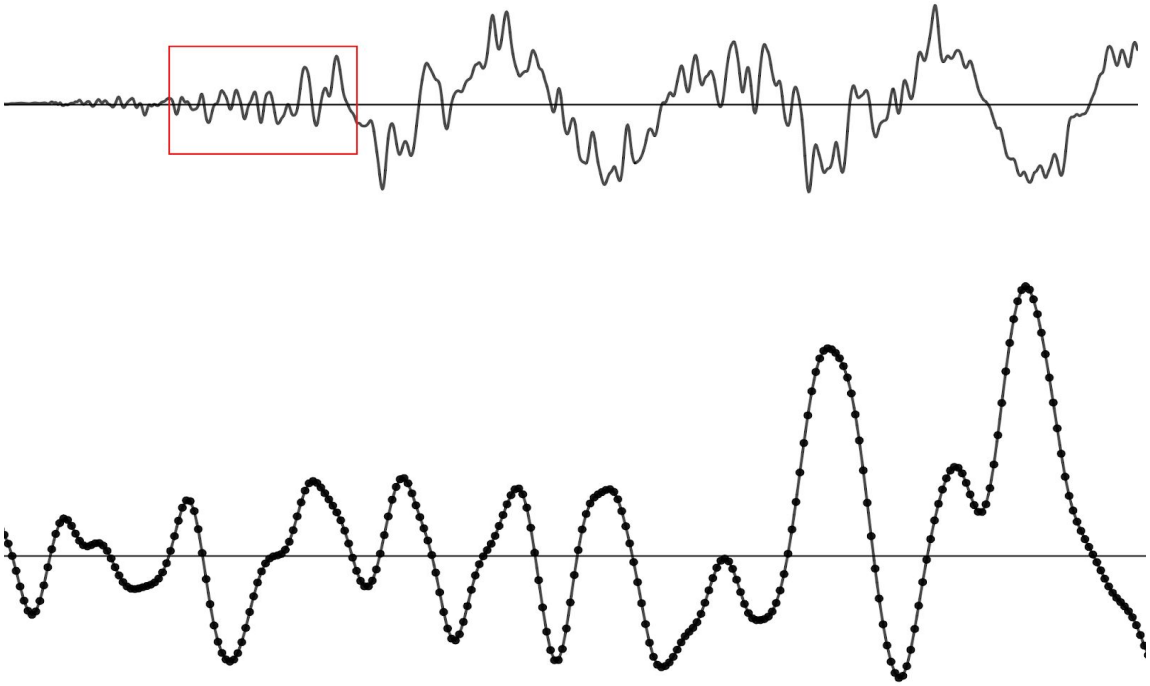


Figure 4: A graph made from the PCM samples from a 16-bit 44.1 kHz piece of audio. The area in the red outline is shown zoomed in with each point showing an individual sample.

In these examples we are working with a single audio wave, known as *mono* audio. Certain microphones can record 2 audio waves simultaneously similar to how our ears hear sound. This is known as *stereo* audio and it takes up twice the space of mono audio. For this reason and the fact that most mobile phone microphones cannot record in stereo, we have decided to use mono audio exclusively.

5.3.3 Audio Encoding and Decoding

The process of extracting the raw audio data from a file such as WAV or MP3 is called decoding. Similarly, the process of writing raw audio data into a file such as WAV or MP3 is called encoding. Encoding and decoding audio takes processing time, and such we would like to avoid it unless absolutely necessary.

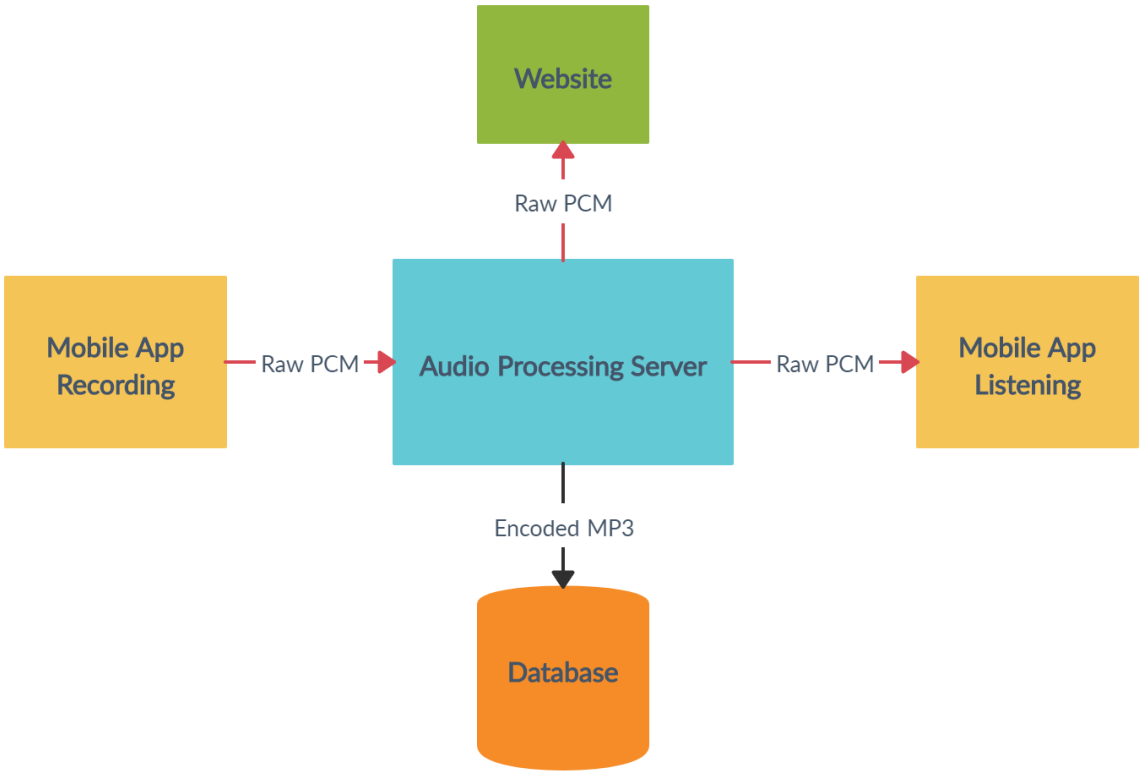


Figure 5: A depiction of the change in the form of the audio data as it moves around the different components of the system.

As we can see with the above diagram, the only time we need to encode the audio data is when we are writing it to the database. Since we will always use 16-bit 22050 Hz PCM audio, we can safely pass it around the other components of the system without encoding. From when the audio is captured by the mobile app, to when it is processed in the server, and finally when it is played back by listeners, it does not need to be encoded at any of these stages. Since we are not encoding at any of the stages when real time playback is occurring, we don't need to decode at any of these stages either.

When encoding to MP3 for storage in the database, the mixed track is buffered until recording is finished. Then the audio can be compressed and encoded to the MP3 format for space efficient storage using the Node-Lame module.

5.3.4 Audio Analysis

An initial approach to this system would be to take all the recorded sounds and mix them

together no matter what the recorded sounds were. In many of John Cage's compositions he combined sounds in a randomized way, and he valued randomness in almost all of his work, so using this approach would be a valid representation of John Cage. However, Cage also liked to combine sounds intentionally with a hint of randomness. Cage also loved using technology in his compositions, so it seems fitting to allow the use of technology to provide some intentionality to the mixing process.

Audio mixing typically involves an audio engineer who has a highly trained ear making slight adjustments to different audio tracks until it has the desired sound. In our case we will have no audio engineer doing the mixing, so we will attempt to perform basic mixing without any human involvement.

In order to automatically mix our recordings together, we will need to first analyze the sounds and understand their contents. Some of the most useful qualities of sound for our system will be *amplitude* and *frequency*.

Audio Amplitude

The amplitude of a sound is experienced by listeners as the loudness of a sound. With respect to PCM audio data, the absolute value of each sample represents the amplitude of the sound at that exact point in time. In order to better understand the contents of a piece of audio, we want to know the maximum amplitude that occurs in that sound.

To find the maximum amplitude of a sound x(t) with n samples:

$a = max(\{|x(t)| : t = 0, ..., n\})$

This amplitude value will be useful for determining the gain levels when mixing.

Audio Frequency

Frequency represents the number of times a sound wave repeats itself in a span of time. More precisely, the frequency of a wave is the inverse of its period. Meaning for a sine wave with a period of $\frac{1}{10}$, the frequency would be 10 Hz. By default sound waves are in the *time domain*, meaning we view them as they change with respect to time. A sound wave can also be viewed in the *frequency domain*, meaning we can now view them with respect to their frequency content.

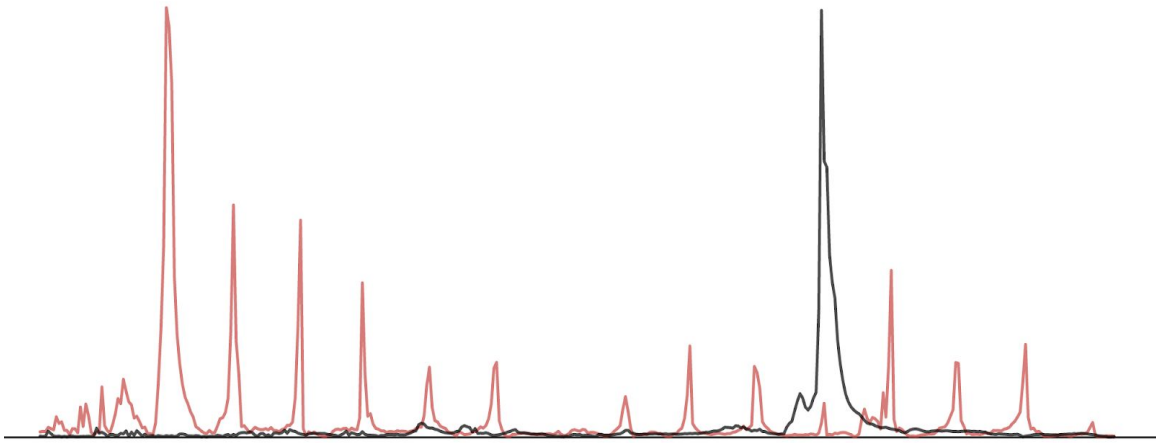


Figure 6: 2 sounds graphed in the frequency domain. The red line is a sound with fundamentally lower frequencies, while the black line is one with higher frequencies.

This is useful to us because the frequency content of a sound directly correlates to how it is perceived by our ears. Sounds with more low frequencies sound lower in pitch and sounds with more high frequencies sound higher in pitch.

Transforming audio from the time domain to the frequency domain and back is an extremely useful tool in audio analysis. To achieve this we can perform a short-time Fourier transform (STFT) on a chunk of PCM audio data to go from time domain to frequency domain, and an Inverse STFT to go back to the time domain. This means that modifications can be made in the frequency domain and translated back into the time domain, making this a useful tool in the mixing process in addition to analysis.

Audio Envelope

Another quality of audio that will be useful when analysing the incoming sounds is the *envelope* of the sound. The sound envelope is loosely defined as the overall shape of a sound wave.

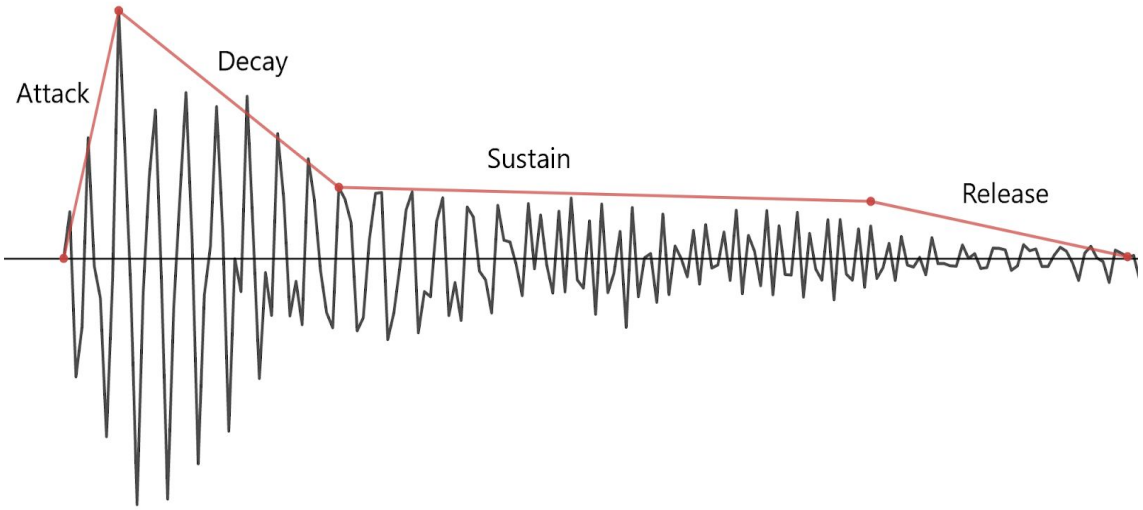


Figure 7: An example sound envelope with its 4 main components labeled: attack, decay, sustain, and release.

The attack of a sound is perceived as how percussive it is. A violin or other stringed instruments will have a long attack while drums will have a very short attack. The decay is how fast the sound drops off from its attack, while the sustain is how long the sound holds a consistent volume. Again, stringed instruments will have a slow decay and long sustain, while drums will have a fast decay and almost no sustain. Finally, the release is how fast a sound fades away.

These qualities of sound could be useful pieces of information for the server to make decisions about how to process the sounds it is given. It could also be useful for comparing given sounds to examples from John Cage to try to mimic his music, as the envelope of a sound plays a large role in its perception. We could attempt to modify sounds to have a similar envelope to that of John Cage's music.

5.3.5 Digital Audio Effects

Equalization

Audio Equalization is the process of increasing or decreasing the strength of individual frequencies in an audio signal. This adjustment allows certain qualities of a sound to be made louder or quieter. Frequencies are generally grouped by how they sound to human ears with the frequency groups referred to as *frequency bands*. Some of the common

frequency bands are bass (60 to 250 Hz), midrange (500 Hz to 2 kHz), treble (2 to 4 kHz), and presence (4 to 6 kHz). Equalization is accomplished by viewing a sound on the frequency spectrum, making adjustments, then transforming the sound back to the time spectrum (see section on audio analysis).

Modifying these frequency bands can have a dramatic effect on the perception of sounds. This is especially useful when mixing sounds together, as too much sound in a single frequency band can be unpleasant to listen to. In our system the applications of equalization are many, but the primary would be to adjust each audio track so there isn't too much sound in any single frequency band. This will make the mixed audio sound a lot more intentional and less like complete randomness. We could also analyze music from John Cage and try to equalize our tracks to mimic the frequency content of his work. For our project, we will be using the equalizer module included with XSound.

Dynamic Range Compression

In audio, *dynamic range* refers to the range between the softest and loudest points of a sound. *Dynamic range compression*, not to be confused with audio data compression, is the process of reducing the dynamic range of a sound. This is done by making the quieter points of a sound louder and the louder points quieter. The overall effect is to make a sound more consistent in volume and avoid abrupt changes.

The general process of applying compression to a sound is to define a volume that you want to be the average, then all samples are moved toward that average based on how far away from it they are. Compression can be applied in small amounts or large amounts, depending on how consistent in volume the desired output is. For our project, we will be using the compressor module included with XSound.

Delay

Audio delay is an effect that records a sound signal then plays it back after a period of time. This process essentially plays the sound on top of itself with an added delay, creating an echo sounding effect. The main parameters to this effect are the delay time and the feedback decay. The delay time is the amount of time before the sound is played back onto itself. The feedback decay is how much the sound will be quieter every time it is looped back. Just like an echo, the delayed sound can be looped back multiple times. If the feedback decay is low, it will play many times with each time being only a little bit

quieter until it is completely gone. A high feedback decay will cause the sound to only be played a few times before it is gone, with each one much quieter than the last.

Audio delay is an effect that can turn very ordinary sounds into much more interesting ones. Delay must be used cautiously however, as too much of it can cause the track to become crowded with looped back sounds. Delay will be a very useful effect for us to use in the Audio Processing algorithm to transform normal sounds into something more, but it must be used in moderation. For our project, we will be using the delay module included with XSound.

Reverb

Audio reverb, or reverberation, is a more complex effect that is a mixture of several of the effects above. The purpose of this effect is to mimic how sounds reverberate around physical spaces such as a large open room. The same way that physical objects absorb and reflect wavelengths of light to create color, physical objects also absorb and reflect wavelengths of sound. This property means that sound can be modified by the physical space that it is in. In physical spaces this can be a factor in the design of a building, for example a oprah hall may be designed so that sounds from the performance reverberate around for everyone to hear. In contrast, recording studios are generally designed to have as little reverberations as possible, with special coatings on the walls to absorb sound.

For our purposes however, we can introduce reverb into our sounds using algorithms that model how different objects absorb and reflect sound. Similar to delay, a sound is looped back onto itself based on a delay time and feedback decay. Instead of just looping back however, the sound is modified each time to simulate a physical space. For our project, we will be using the reverb module included with XSound.

Distortion

Audio distortion is the degrading of an audio signal causing it to sound choppy or ‘rough’. Originally audio distortion occurred in sound recording due to faulty hardware or bad signal connections. Today with audio technology capable of recording sounds with little to no distortion, distortion can be added as an intentional effect to create new sounds. Audio distortion as an intentional effect is most easily recognisable in the genres of rock and roll or metal music. Electric guitars use distortion effects to create their iconic sound.

For our purposes we will probably not use distortion to the same degree as electric guitars, but in a more subtle way to add depth and feel to our sounds. In addition, much of John Cage's music was created using very old recording equipment so for our compositions a small amount of added distortion could bring us that much closer to sounding like John Cage. For our project, we will be using the distortion module included with XSound.

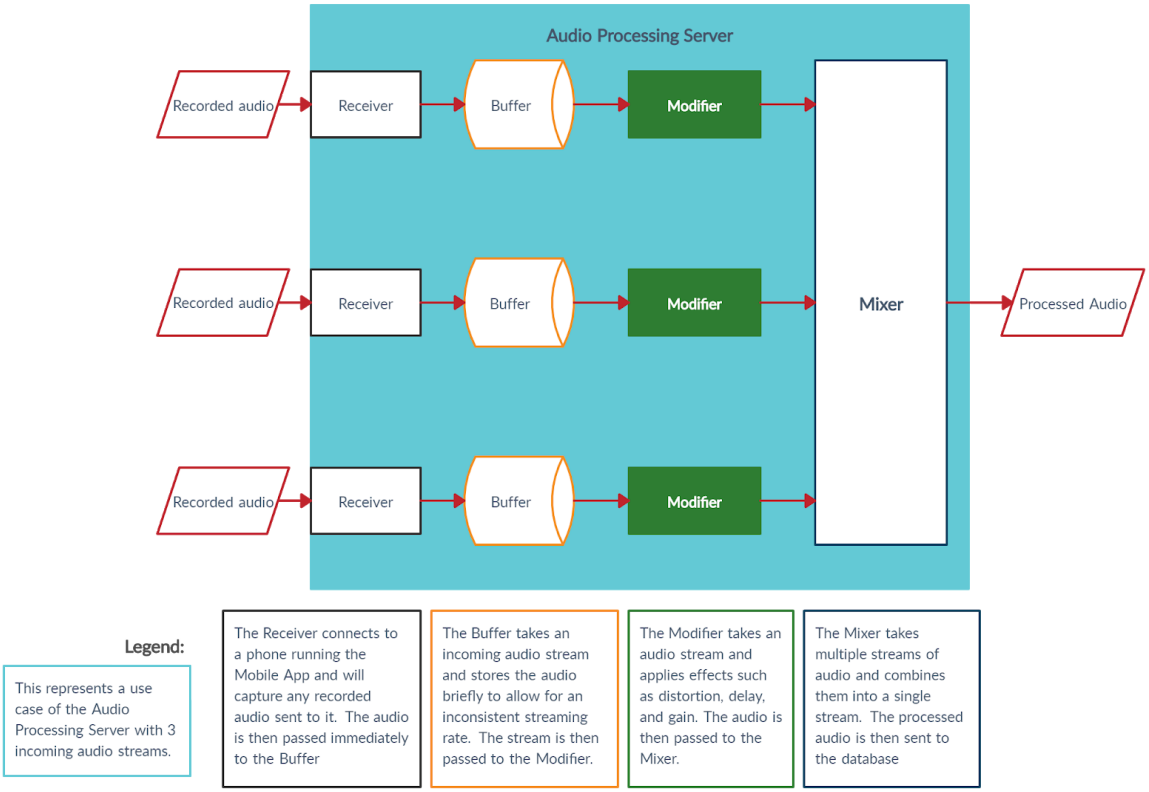


Figure 8: An initial design of the Audio Processing Algorithm running on the server.

5.3.6 Consulting the *I Ching* Oracle

Here is a passage from a web page that contains a few of John Cage's comments out of a 1968 book titled, *The God Who is There* by Francis Schaeffer:

> Back in the Chinese culture long ago the Chinese had worked out a system of

tossing coins or yarrow sticks by means of which the spirits would speak. The complicated method which they developed made sure that the person doing the tossing could not allow his personality to intervene. Self-expression was eliminated so that the spirits could speak.

Cage picks up this same system and uses it. He too seeks to get rid of any individual expression in his music. But there is a very great difference. As far as Cage is concerned, there is nobody there to speak. There is only an impersonal universe speaking through blind chance.

Cage began to compose his music through the tossing of coins. It is said that for some of his pieces, lasting only twenty minutes, he tossed the coin thousands of times. This is pure chance, but apparently not pure enough; he wanted still more chance. So he devised a mechanical conductor. It was a machine working on cams, the motion of which could not be determined ahead of time, and the musicians followed that. Or as an alternative to this, sometimes he employed two conductors who could not see each other, both conducting simultaneously; anything, in fact, to produce pure chance. But in Cage's universe nothing comes through in the music except noise and confusion or total silence. All this is below the line of anthropology. Above the line there is nothing personal, only the philosophic other, or the impersonal everything.

There is a story that once, after the musicians had played Cage's total chance music, as he was bowing to acknowledge the applause, there was a noise behind him. He thought it sounded like steam escaping from somewhere, but then to his dismay realized it was the musicians behind him who were hissing. Often his works have been booed. However, when the audience boo at him they are, if they are modern men, in reality booing the logical conclusion of their own position as it strikes their ears in music.

Cage himself, however, is another example of a man who cannot live with his own conclusions. He says that the truth about the universe is a totally chance situation. You must just live with it and listen to it; cry if you must, swear if you must, but listen and go on listening. [12]

King Wen Sequence and Bagua:

The *I Ching* contains 64 hexagrams each with their own name or chapter in the *I Ching.*

For example the 48th hexagram is named The Well. Turning the page to this chapter (pg.99) we are greeted with a brief but meaningful excerpt, “The well. The town may be changed, But the well cannot be changed. It neither decreases nor increases. They come and go and draw from the well. If one gets down almost to the water And the rope does not go all the way, Or the jug breaks, it brings misfortune.” [15] Each hexagram is made up of 6 lines, which means that each one contains 2 trigrams. One trigram is made up of 3 lines. There are four different kinds of lines:

Figure 9: Yang line.

Figure 10: Yin line.

Figure 11: Transformation Yin line.

Figure 12: Transformation Yang line.

A transformation can be thought of as a negation.

Figure 13: A transformation Yin line = a Yang line.

Figure 14: A transformation Yang line = A Yin line.

In the end, you are left with only Yin and Yang lines.

Figure 15: A trigram contains three lines total, all of the possibilities of the combinations of these 2 kinds of lines can be shown in 8 trigrams [16]:

In Figure 15 above, depicted is what is known as Bagua, which are eight symbols used in Taoist cosmology to represent the fundamental principles of reality, seen as a range of eight interrelated concepts. All of these can be combined to form a total of 64 different possible hexagrams, otherwise known as the King Wen Sequence.

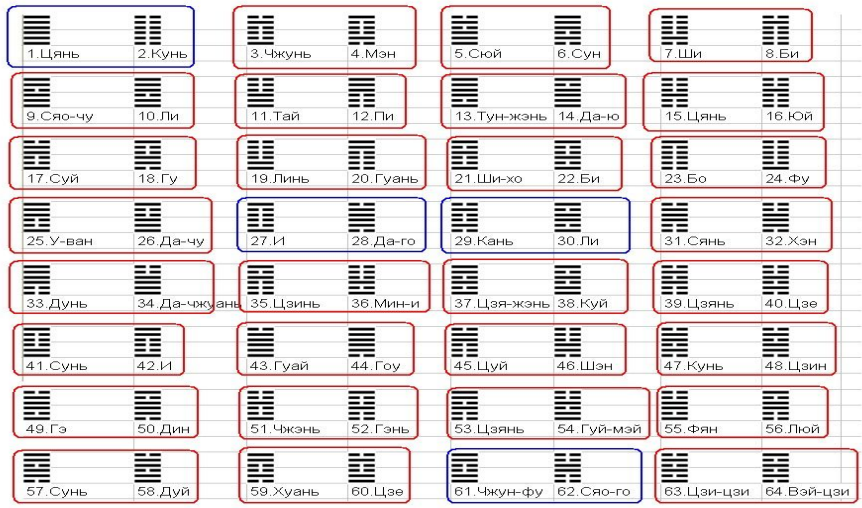


Figure 16: The King Wen Sequence shows all of the different outcomes that the I Ching can provide [17].

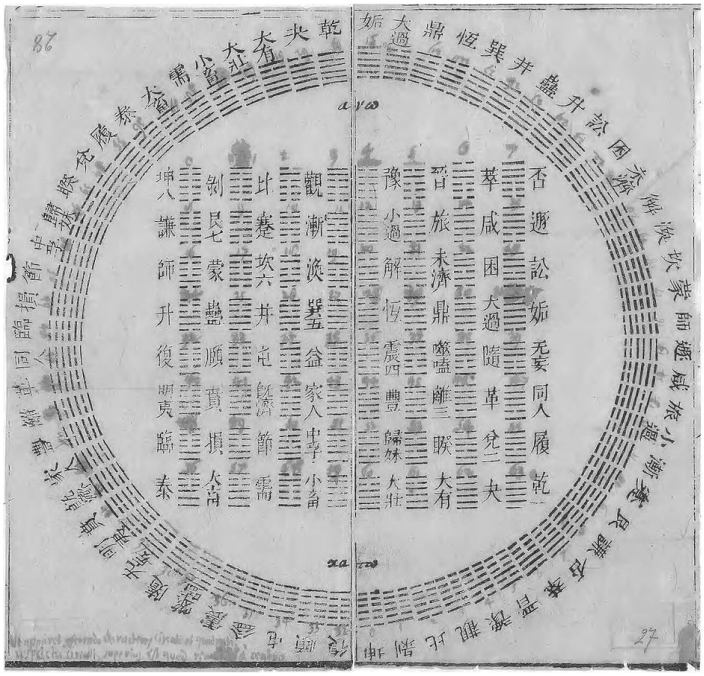


Figure 17: A diagram of *I Ching* hexagrams owned by German mathematician and philosopher Gottfried Wilhelm Leibniz. [18]

The way to consult the Oracle about a question begins by choosing 3 of the same coins,

they must be identical in weight, size, shape, texture, and color. No two coins can be the exact same, but they must be very close to being the same. With your question in mind, throw all three coins on a flat surface. Heads will be equal to 3 and tails will be equal to 2. Add all three of the coin face values up and you will receive a range of four possibilities: 6, 7, 8, or 9.

6 = Yin transformation line → ----- ----- *

7 = Yang line → ----------

8 = Yin line → ----- -----

9 = Yang transformation line → ---------- *

Based on the sum of the values from the coins, write out the line. If you flipped a 6 or a 9, make your necessary transformation or negation and keep in mind which line the transformation occurred on. For example, if line 2 was the only transformation, we keep that in mind for our reading later on.

Toss the coins 6 times to build up 6 lines, the line generated on the first throw is the line at the bottom, the line generated on the sixth throw is the line at the top. The hexagram is built from the bottom up.

Then, split the hexagram up into 2 trigrams (3 lines each), taking the top 3 lines and the bottom 3 lines.

	☰	☱	☲	☳	☴	☵	☶	☷
☰	1	34	5	26	11	9	14	43
☱	25	51	3	27	24	42	21	17
☲	6	40	29	4	7	59	64	47
☳	33	62	39	52	15	53	56	31
☴	12	16	8	23	2	20	35	45
☵	44	32	48	18	46	57	50	28
☶	13	55	63	22	36	37	30	49
☷	10	54	60	41	19	61	38	58

Figure 18: Refer to the hexagram lookup key using your two trigrams, this will tell you the chapter of the *I Ching* that will answer your question

You will see along the left side are the bottom trigram possibilities and on the top are the top trigram possibilities. Map these out to your specific chapter or hexagram number, and then consult the oracle. Keep in mind the lines which had the transformation as those will be the specific passages to consult.

A full example of an *I Ching* consultation follows:

Take 3 identical coins and flip them while asking the question: Should I wash the dishes?

Figure 19: Flip 1.

Three heads = 3 + 3 + 3 = 9 => ---------- * (Yang transformation line)

We have a transformation at Line 1. This will be a Yin Line.

----- -----

Figure 20: Flip a second time while asking if you should wash the dishes.

Two Tails, 1 Heads = 2 + 2 + 3 = 7 => ---------- (Yang line)

----- -----

Figure 21: Flip a third time while asking if you should wash the dishes.

Two Tails, 1 Heads = 2 + 2 + 3 = 7 => ---------- (Yang line)

----- -----

Figure 22: Flip a fourth time while asking if you should wash the dishes.

One Tails, 2 Heads = 2 + 3 + 3 = 8 => ----- ----- (Yin line)

----- -----

----- -----

Figure 23: Flip a fifth time while asking if you should wash the dishes.

Two Tails, 1 Heads = 2 + 2 + 3 = 7 => ---------- (Yang line)

----- -----

----- -----

Figure 24: Flip a sixth time while asking if you should wash the dishes.

One Tails, 2 Heads = 2 + 3 + 3 = 8 => ----- ----- (Yin line)

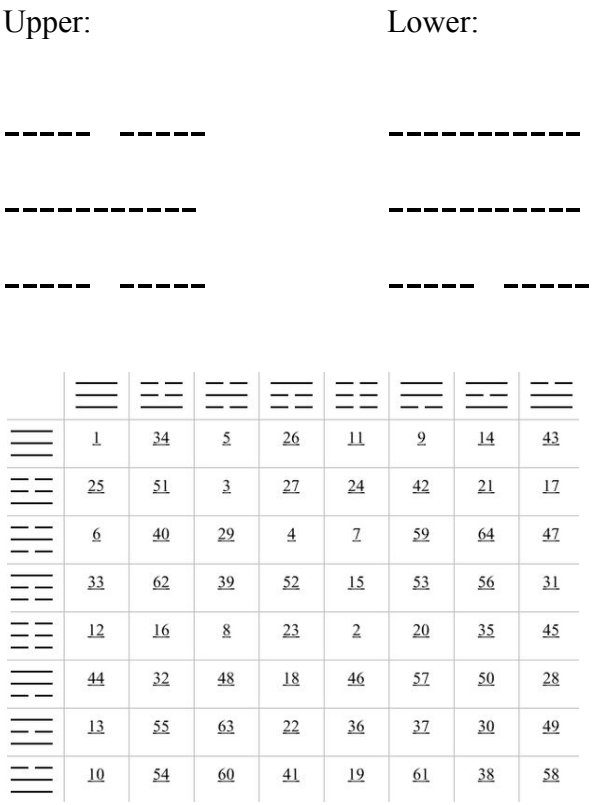
----- -----

----- -----

----- -----

Now let us divide our hexagram into two trigrams:

Upper: Lower:



	䷀	䷓	䷂	䷊	䷇	䷪	䷮	䷡
䷀	1	34	5	26	11	9	14	43
䷓	25	51	3	27	24	42	21	17
䷂	6	40	29	4	7	59	64	47
䷊	33	62	39	52	15	53	56	31
䷇	12	16	8	23	2	20	35	45
䷪	44	32	48	18	46	57	50	28
䷮	13	55	63	22	36	37	30	49
䷡	10	54	60	41	19	61	38	58

Figure 25: consult the key.

……...All I got to say is woah. It is currently 3:51 A.M. in Istanbul and this was my first real run of the *I Ching* and the number 48 came up. Initially if you read the beginning of this section you will see I mentioned the section on The Well which is the 48th chapter. I had randomly chosen this to give an example of a chapter and the excerpt within. Now upon doing this current consultation, in the example I provided, the *I Ching* gave me the Well once again. All I got to say is there is something up here, was this just a coincidence. Washing the dishes came to mind when I was cleaning the coins before flipping. I swear, I am not making this up to add flair to the paragraph. There is something up here. We're not done yet!

Let us go into the *I Ching* and read the chapter:

> The Judgement
>
> The well. The town may be changed, but the well cannot be changed. It neither decreases nor increases. They come and go and draw from the well. If one gets down almost to the water And the rope does not go all the way, Or the jug breaks,

it brings misfortune.

The Image

Water over wood: the image of the Well. Thus the superior man encourages the people at their work, And exhorts them to help one another. [15]

Remembering we had a transformation at Line 1, we check to see the passage that pertains to Line 1, “Six at the beginning means: One does not drink the mud of the well. No animals come to an old well.”

Seeing as for the first line we received a 9 from 3 heads and not a 6, this statement need not apply.

I guess I’ll wash the dishes…….

5.3.7 *I Ching* Audio Modification

Some characteristics of sound:

Some defining characteristics of sound are frequency or pitch, amplitude or loudness, Duration, envelope, and timbre. The envelope of a sound refers to its contour evolving over time. Attack, sustain, and decay are the parts of an envelope. For example an acoustic guitar has a sharp attack, little sustain, and a quick decay. A piano in contrast, has a sharp attack, medium sustain, and medium decay. Other instruments, such as wind and string instruments allow a musician to choose their attack, sustain, and decay constituents of their sound.

The system’s audio input can be analyzed using these five characteristics of frequency, amplitude, duration, envelope, and timbre. In order for a modification to change the sound, it would have to alter one or more of these characteristics.

Modification algorithm:

The system, based on the requirements, will need to alter the sounds randomly. The most desirable outcome would be a system that consults the *I Ching* to do the divination that modifies the sounds. Questions we could ask the book of changes would be:

- How long to play a certain audio input?
- What intervals at which to play a certain input, and for what duration? Should that interval and duration change throughout the recording. Yes or no?
- How can we give the sound a chance to play without any modification?
- What the change in frequency should be, high or low? By how much?
- Should there be a change in the amplitude or the loudness of the sound?
- What special qualities are we looking for in the input audio that may hold special significance over other inputs? Are we looking for those qualities during this run of the program?
- Are we taking the environment variables into consideration? Time, season, barometric pressure, temperature, etc. If so, which ones?

These are a few modifications to consider for a quality randomization. It is important to think of all of the possibilities so randomization can occur at all levels, sometimes dynamically or not. Sound is so complex that the possibilities are endless. In essence, I think it is important to allow a wide range of randomization with many different possibilities.

The implementation of the *I Ching* on a particular run of the program would have to derive meaning from the chapter it lands on for that particular question. There could be a large pool of questions about audio modifications, each one will be asked and each one will consult the *I Ching*. The way I would think of implementing such a thing would be to map out all 64 chapters to each question, manually. If there are 10 questions total, each question could have 64 possible outcomes. The total possibilities would be 640. However, I feel that doing such a discrete implementation would lead to non-random outcomes. “The dao called the dao is not the dao.” Nothing is what it seems and the interpretations of the *I Ching* are based on the senses and feelings. The problem is how to implement this into a computer program. We are starting to go into the area of artificial intelligence, there will need to be some form of omnipresence.

5.4 Hosting

The web application will expect deployment on a hosting service for universal access. The initial consensus on the app’s deployment platform was between Heroku and

DigitalOcean platforms, with an inclination toward DigitalOcean. Heroku is inherently a Platform as a Service (PAAS), ensuring a quick deployment process without additional concerns of overhead. Due to the scope of the Tribute project, it is not anticipated to require substantial power and traffic. As a result, stronger services that fall under the "Infrastructure as a Service" umbrella (IAAS), such as AWS and Microsoft Azure, are not expecting direct use in the project.

Both of the servers have their strengths and their weaknesses. For Heroku, the positives include that it is more flexible digital, while the negatives include it being more expensive than Digital Ocean. For DigitalOcean, the positives include it being cheaper than Heroku, while the negatives include not being as flexible as Heroku.

As it stands, the application's emphasis on non-concurrent recording sessions allows for minimal server traffic. This may, as a result, reduce the necessity of a paid Heroku service and operate on the free service instead. In the event that the application is augmented and its non-concurrency restriction is lifted (via achieving stretch goals), then the paid service by Heroku may become more likely. This, of course, depends on the estimated traffic increase from shifting from non-concurrency.

Database deployment will be accomplished through the use of MongoDB, a NoSQL program. Its JSON-like schemas are significantly more familiar for users working with JavaScript than the conventional row/column programs that SQL programs use. As a result, it fits snugly with the rest of the web application's stack, MERN. In addition, the mobile app's Dart language heavily supports JSON logic, further solidifying MongoDB's efficacy in the Tribute.

MongoDB is laden with platform services that allow for storing large amounts of data. Although a handful of these are paid, there is one, MongoDB Atlas, that requires an account and no fee whatsoever. It is particularly effective in the scope of the John Cage Tribute, so an upgrade to a paid service is unexpected.

For hosting services, the final decision was to go with DigitalOcean. We used NGINX for its reverse proxy capability and PM2 for software management on the server. PM2 provides an easy way to update code on the server.

6 System Design

The components of the system should provide for a smooth interface in which data transfer and requests are processed with ease. Bellow is figure 26, which is our block diagram, as well as figure 27, which is our use case diagram.

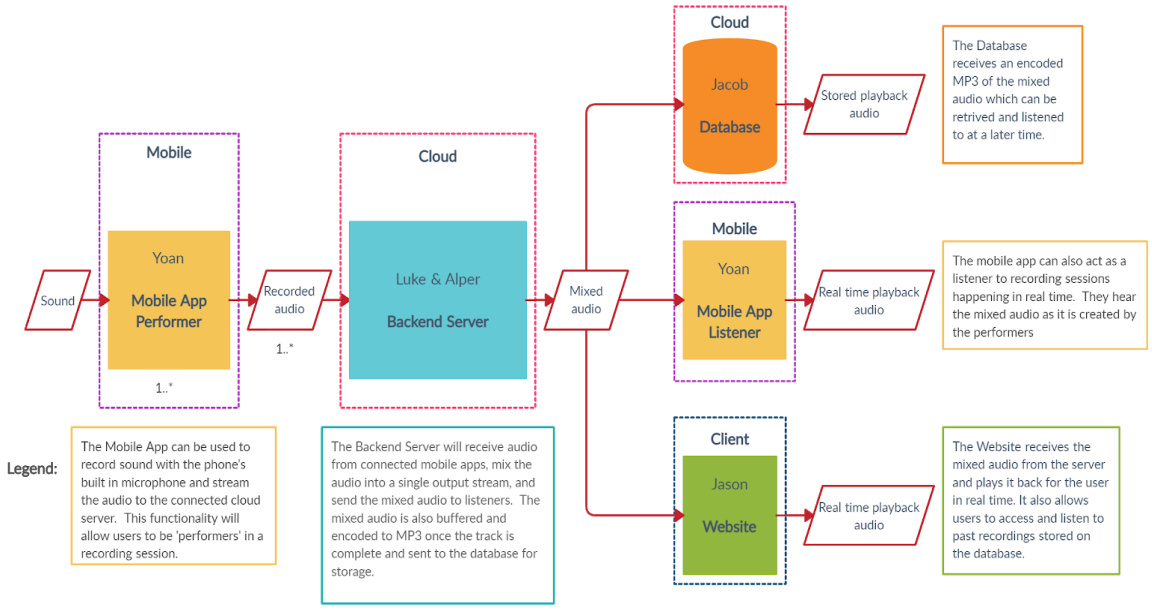


Figure 26: The block diagram of the John Cage Tribute

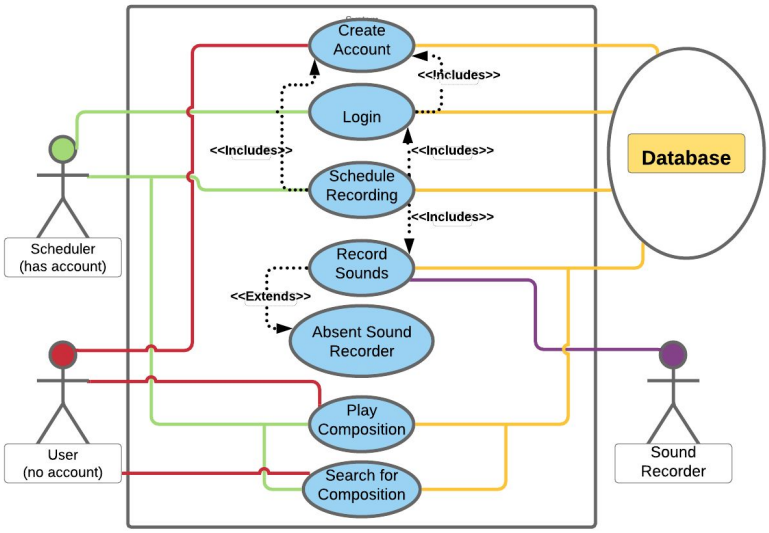


Figure 27: **Outdated Use-Case Diagram**. No scheduler, performer and listeners on mobile application (performers = recorders, listeners = audience). Web application will have a page to listen to live recordings from each room.

6.1 Application Frontend

6.1.1 Mobile Application

Look and Feel:

Despite the Tribute project's status as a "niche" application, this certainly does not excuse it from lacking a comfortable interface that relaxes the eyes. Its look and feel is expected to culminate as a balance between an eccentric and stimulating series of visuals in inspiration of John Cage's peculiar yet innovative ideals and modern-day applications' sleek and simplistic UI design. The two appear to be in juxtaposition, but are in fact laid out independently of one another. In other words, while the modern sleek UI holds the navigation of the app together, the overall function of the app emphasizes John Cage's work flow. This is further illustrated in Figure 28 below.

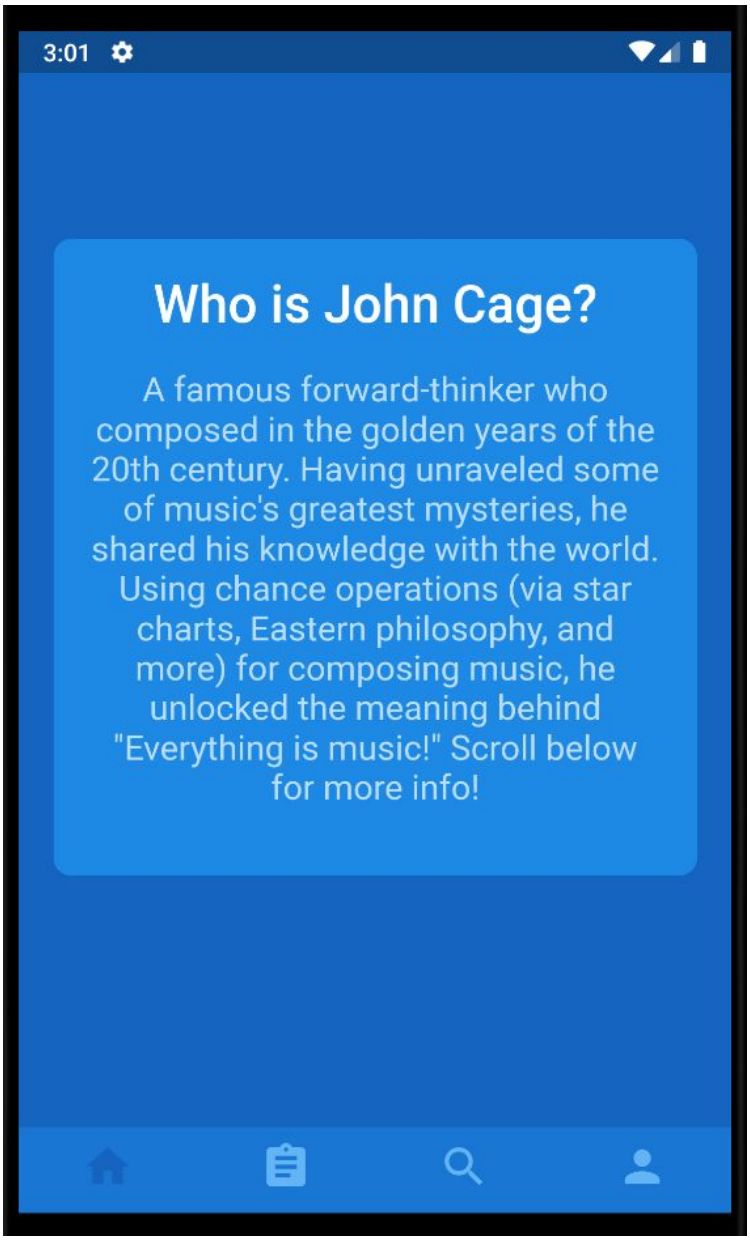


Figure 28: The front page of the mobile app. Predominantly simple and blue.

Simplistic and Sleek Design:

Applications such as DoorDash, Gmail, Discord and iTunes all share the idea of a fundamental color scheme to outline every screen. Additionally, they sport some form of a secondary color to illuminate the parts of a screen that may be interacted with. Any text displayed on the screen tends to directly oppose the color of its surrounding area for

increased readability, but should strictly avoid clashing with the color of the screen (like, say, lime green and bright orange). Any additional icons to complement the screen navigation will sport similar color to any text associated with it. Fluid, quick animations (such as Google's familiar "swipe screen left" animation when entering an email) are in common usage so that the user's attention to navigation is carefully kept at every step. Out of the box, this is already inherently similar to Flutter's Hero animation and the SlideTransition class in Flutter's Widgets library. Lastly, there is a minimal amount of "rough" screen-loading behavior. An example of this is when navigation occurs from one screen to another. The navigation should not be sudden and lack fluidity, otherwise it loses its sleek vibe. As such, the MaterialPageRoute and/or CupertinoPageRoute enables the navigation to "slide" from right to left in a page-turning fashion.

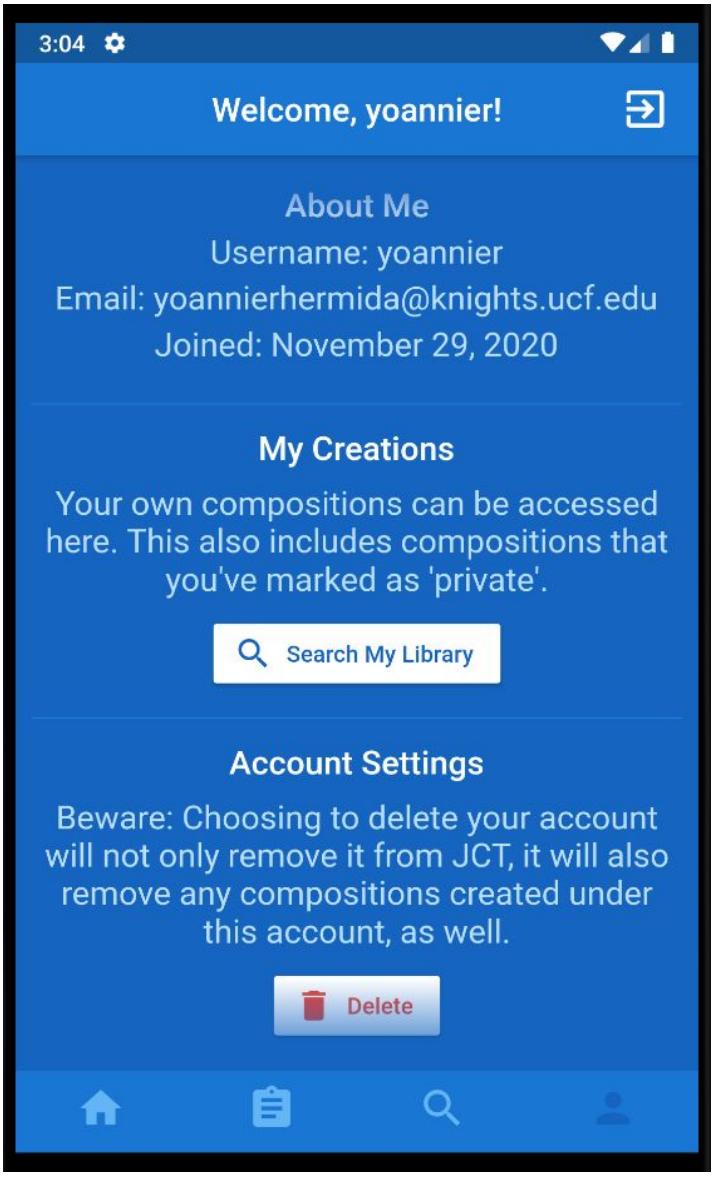


Figure 29: A snapshot of the account screen for the John Cage Tribute. Note its whitespace usage.

As demonstrated above, items in a ListView can be inserted in certain screens that have significant room for decoration and modification. This can also be achieved with animations to occupy a user's visual field as they await an event, such as a recording session or an API call. Conventionally speaking, in Flutter, a widget known as the CircularProgressIndicator (a spinning wheel, if you will) is oftentimes placed in the point of interest to simulate an event's anticipated loading.

Dashboard:

This is the first screen the user is exposed to. It, like many standard and modern applications, allows for access to many other parts of the app. On the center of this screen, a short biography of John Cage and his impact on the Tribute team will be discussed, followed shortly by a basic set of instructions for using the app. Layering this will be a navigation bar at the bottom of the screen. For guests, clicking the rightmost button of the bar, symbolizing the Account screen, will trigger a popup requesting that the user signs in (or sign up if new). Upon signing in or signing up, the screen will become the Account screen, displaying user metadata. The "Sign in" button will then be replaced with "Sign out." This screen will be capable of using JSON Web Tokens (JWTs) to automatically sign the user in the app if they have done so before when reentering. Otherwise, one will be created in the first instance that they do. It will be stored in the user's phone's local storage, being deleted upon signing out or when 10 days of its presence on the device has been detected.

Authentication:

Flutter carries essential components that are presumably borrowed from HTML's simple Text Fields and Text Forms. As such, the screen will make use of the Flutter-made TextField widget to accomplish this task. This widget will require the use of API calls to determine the validity of the PIN code that was entered. An example of some of the room and PIN popup's features is shown below.

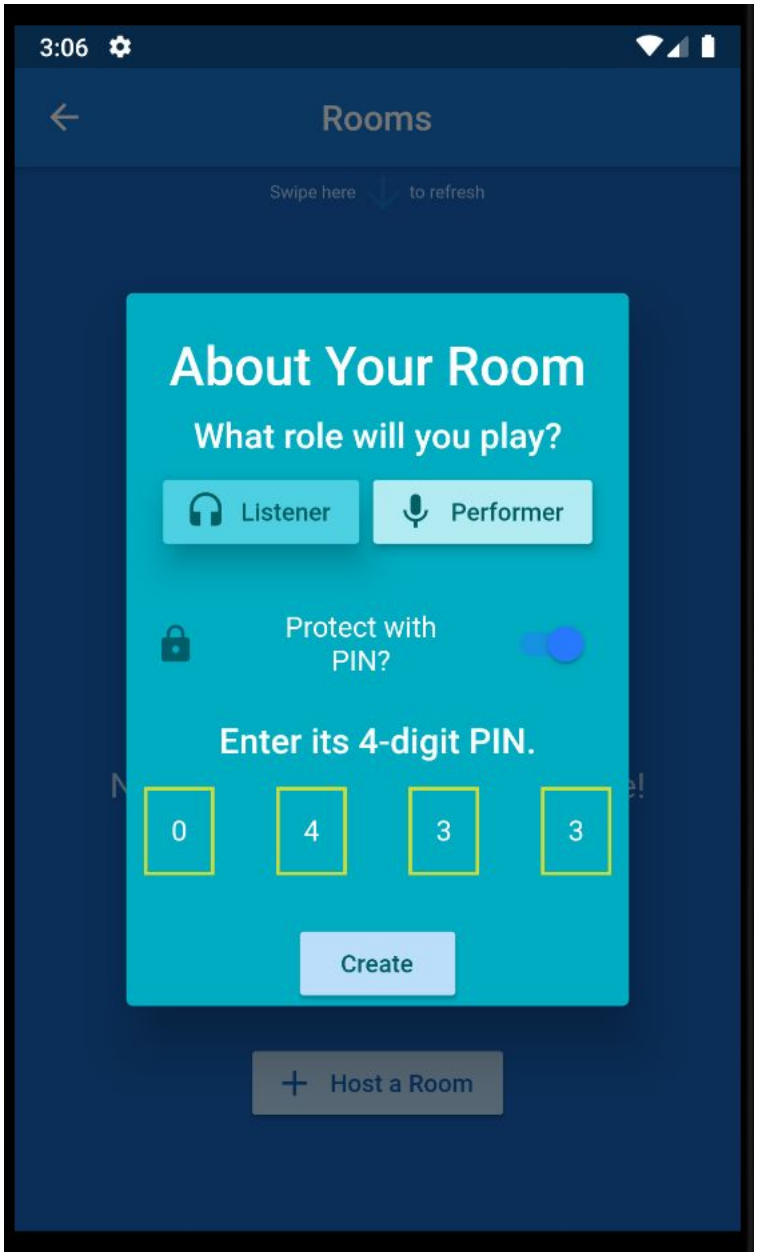


Figure 30: The user's PIN is displayed neatly via a sequence of illustrative blocks. Their decorations change color upon numerous conditions, including being disabled, filled up, or deselected.

A TextField widget will restrict itself to a length of 4 characters. This may be modified in the event that additional security is necessary. The widget will depend on a StreamBuilder (a function that returns a widget based on a given stream of input) to either navigate the user automatically to the recording screen upon success or display a red error message below the series of blocks to indicate failure. The series of blocks is a

more abstract problem. An individual block itself can be constructed using a broad widget called a Container, which is generalized enough to support many kinds of shapes and inputs to modify its appearance. The logic between each block will require clever use of reading the input stream to show or remove information on each block, considering that they are not the same as a run-of-the-mill input field of text.

When the user enters an incorrect PIN, an error text will display to emphasize the mistake they have made. This allows users to quickly recognize the issue at hand and re-enter the PIN they may have mistyped, in the likely event that this has occurred. Due to the nature of the PIN entry process, an API call may be safely made each time the user enters the required quantity of characters, as opposed to each additional typed character in the PIN field. Upon successful PIN entry, users will be greeted by a widget known as the CircularProgressIndicator widget (a small rotating circle to imply a process is currently loading) before the Session screen is loaded.

Gratefully, the example above is an existing Flutter package known as pin_code_fields (a combination of PIN and text fields). It houses Flutter's up-and-coming BLoC-Pattern approach for managing the state, or user input. The package even provides an example application for users to test out or modify. This package was a priority-one for investigating a method of valid authentication in tandem with a smooth user experience.

Pop-ups:

Numerous implementations of pop-ups were used to facilitate authentication. These pop-ups are not quite unique, serving only as a simple UI field for users to enter data. A few notable examples include PIN authentication, prompts to confirm or deny a user's action to leave a room, along with account deletion. The audio player is implemented as a standalone screen, but was initially considered for another potential popup. Flutter contains a widget known as "ShowDialog" that was able to easily accomplish exactly this. In addition, email authentication is performed with a third-party Flutter plugin, the *email_validator* package.

Room Screen:

The Room screen is rather simple, displaying a list of currently active rooms that may be selected for entry. It also displays a "Host Room" button on the bottom of the screen that will only be enabled if the active user is currently registered. Upon selection, the user can

provide their role (Performer or Listener) and a PIN to protect the room if they desire, moving them to the Session screen. If a room is selected instead, a similar popup asking about the user's role, alongside the PIN of the room if a PIN was set for it will prompt. Upon success, the user is navigated to the Session screen, just like the "Host Room" button does.

Session Screen:

On this screen, users will finally be present in a room amongst one another. More particularly, the room that they've selected to be in for a composition's performance. The users, guests or not, are at the behest of the main host of the room itself. They, acting as the composer of the to-be composition, will decide when the composition may begin. The only restriction the composer is confronted with is the minimum performers required in order to initiate a session.

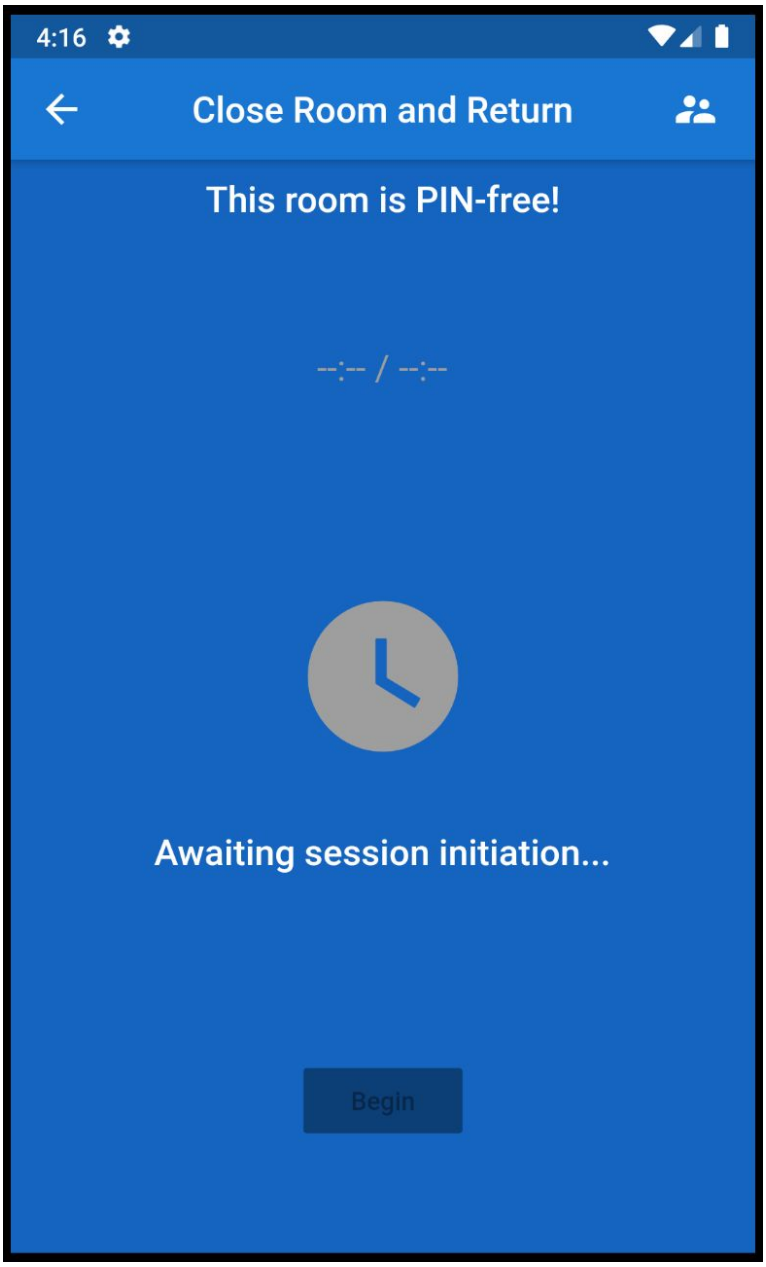


Figure 31: The Session screen, before having initiated the session. Note the greyed icon on the center, which becomes the “action” button for muting/deafening purposes.

Near the top-right is a small “groups” icon that allows for a drawer to slide in from the right, displaying the current members of the room. It updates in realtime as members enter and leave this room. Situated just above the clock is a timer that displays the elapsed time over the duration of a performance, initially displayed as a series of dashes. As the composition progresses, its color changes, beginning originally as black,

becoming green when a composition is capable of being ended manually, then orange and red as the max composition duration of 10 minutes is nearly reached.

The “start session” button depicted near the bottom-center of the screen is the epicenter of the application’s creativity. Once the minimum number of performers have entered the room, the button will glow a vivid white, signaling the advent of a performance. Once selected, a jovial message cheers the members (guests or not) of the room from near the bottom of the screen, assuming they are not the composer. The clock icon finally changes itself into a button for enabling muting/deafening functionality over the course of a session’s duration. Once a session is concluded, the composer is automatically navigated to a short screen to provide metadata about the composition. The remainder of the members are greeted by a message indicating that the performance was successful.

Composition Info Screen:

This screen is essential for making compositions searchable in the first place. It is accessible through a transition for a composer from the Session screen to here, or through the decision to edit an already-existing composition. It prompts the user to enter the composition’s title, description, tags, and whether it should be marked private so other users may not be able to retrieve it. A partial example is shown below.

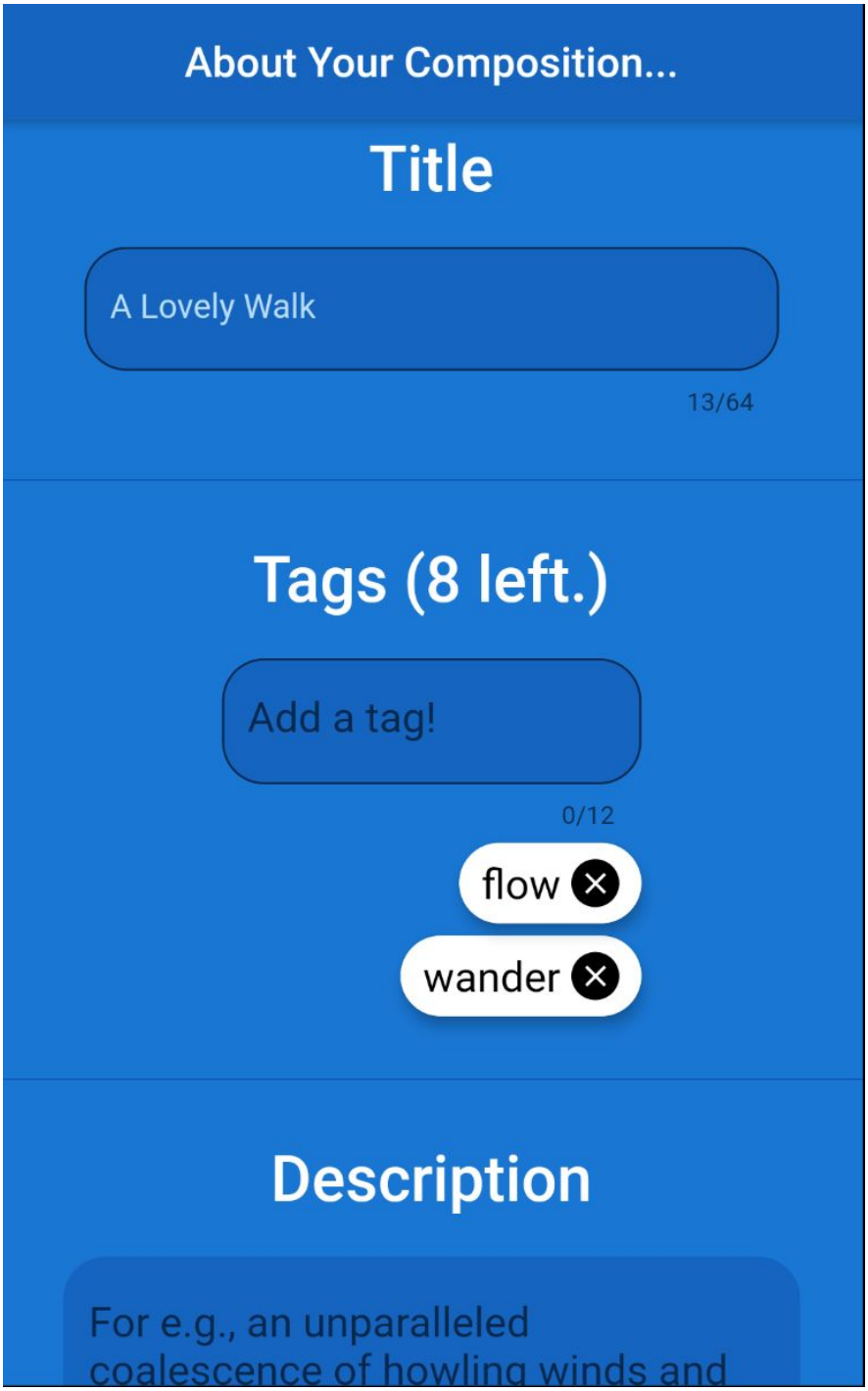


Figure 33: The Composition Info screen is where a composition may finally be given its identity. Due to John Cage's perception of musical indeterminacy, this screen appears ***after*** the composition is finally complete.

As expected of a standard screen, it sports a Submit button at the bottom, which is accessed via scrolling to it. This screen is, in fact, a large ListView, whose items may be scrolled through in order to reach the bottom. Upon submission, the button becomes a CircularProgressIndicator to prevent multiple submission oddities. If the update of the composition fails, an error message displays just above the button. Otherwise, the user on the screen is navigated back to either the Dashboard (if they just finished a composition) or the Library screen (if they simply edited a composition's metadata instead).

Search Screen:

The function of this screen is to provide users with the ability to search for any recorded composition that has been sent to the database. The search process is achieved by searching for these compositions via their name or their tag. Upon clicking a "Search" button, the user will be greeted to a CircularProgressIndicator widget to denote that a search is being performed. As the search completes execution, specialized widgets that symbolize a "temp" composition will immediately load onto the screen. These widgets are gray, box-shaped lines that indicate where the placement of the actual composition will be and their quantity implies the number of compositions that will load. The use of "temp" widgets or components is common in modern applications to give a sense of predictability to the user. In the event that the search fails, a simple error message will display. If the search query garners zero results, a special "no compositions found" message will show itself to the user instead.

Each searched composition consists of: the name of the composition (summarized with an ellipsis if its characters exceed some length), its duration, and its composer's username. Beside each of them is a button engraved with a "Right Arrow" button. This button allows navigation to an audio player that immediately begins to play the chosen composition. The searches themselves are arranged based on the most recent entry and the very bottom of this search list contains the oldest entry that meets the criteria specified by the user's search.

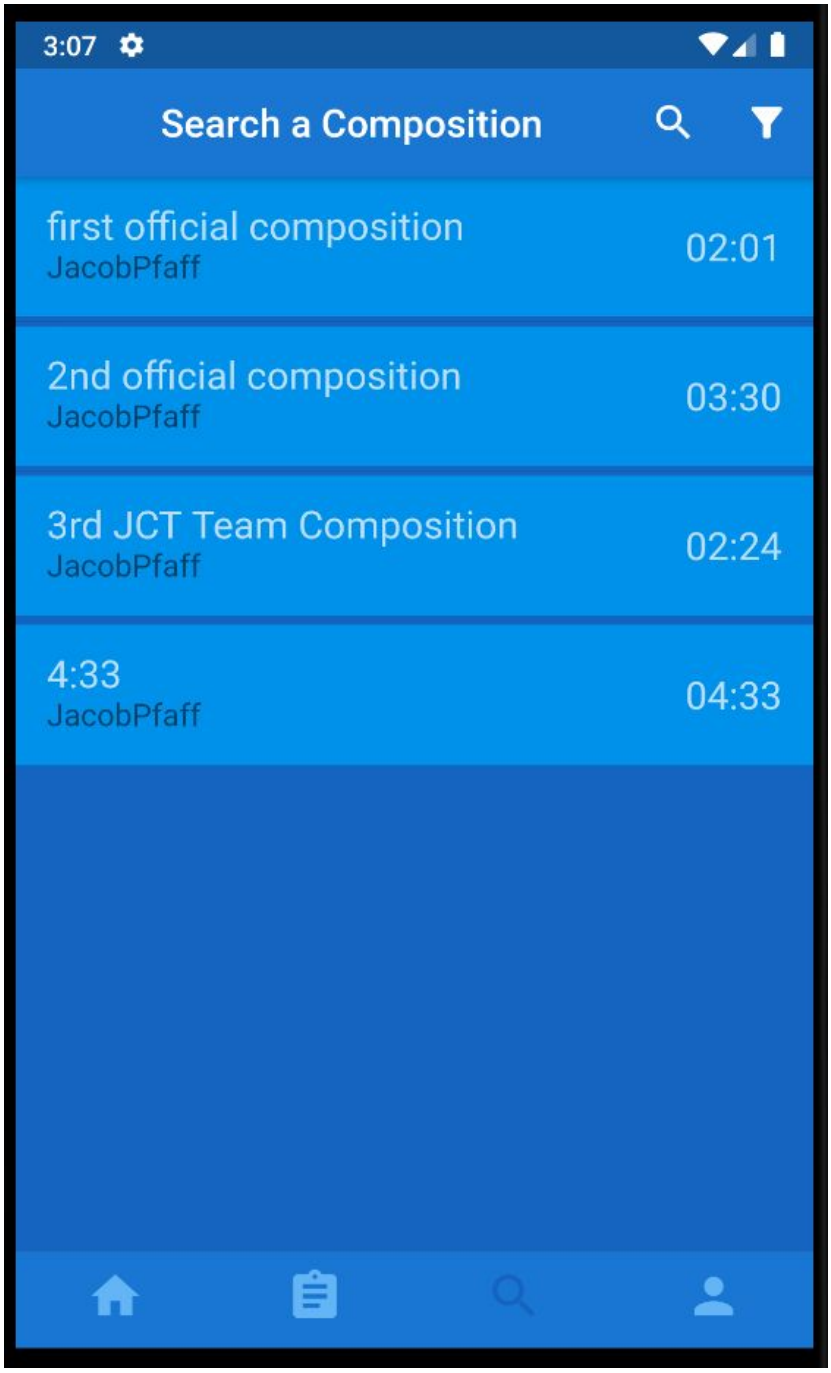


Figure 32: The Search screen, naturally, loads a relevant title to prompt the user to make a decision based on the action buttons all the way to the right. They may select the magnifying glass to enable a search function or choose the filter icon to filter their query based on certain parameters of a composition.

Library Screen:

When a user has successfully composed their own compositions with a performance team, compositions are set up and stored in the database. If curiosity strikes them to revisit their past compositions, the My Library screen will query the database for the current user's compositions and present it in a neat and organized manner. If a user is not logged in, they will be unable to access this screen, due to having no registered compositions under them.

Upon visiting this screen, the application will attempt to retrieve a sufficient quantity of compositions such that it fits in the screen's entirety. This can be accomplished with assistance from a Flutter widget known as a "ListView." The ListView will dynamically load data as the user scrolls or remains on a part of this screen with unloaded composition data. The ListView will then require an "ItemBuilder", meant to describe what actions the ListView should take when data is not yet loaded, about to load, and finished loading. For the initial visit to the screen, the user shall be greeted with a CircularProgressIndicator (similar to when pages are refreshed), followed by the newly retrieved compositions. At the top of the screen, users will find a search bar, huddled next to two small buttons dedicated to the primary functions of the Search/Library screen: searching, and filtering by a parameter. There are four search parameters: Name, Composer (via their username), the Performers involved in the composition (also by their usernames), and Search by Tag. Search by Title will be set by default. This will filter resulting searches by the string typed by the user in conjunction with the search parameter they have selected.

Figure 34: A design of an individual composition's tile. When on the Library screen, a composition tile will allow its composer to play its audio, edit its metadata, or delete it outright as needed.

Each searched composition will appear in the form of a horizontal "tile." The tile will contain special functions specified in Figure 34. These tiles will line the majority of the

My Library screen and be reasonably large enough to make up for those with difficulty tapping small components on their phones.

Personal templates for inspiration include SoundCloud's own Library screen, and Spotify's variant of the screen. These two applications are well-known and have had extensive research made to maximize the user's comfort of accessing their uploaded audio.

Navigation:

Due to the app's intricate layout and sensitive audio data handling, the very navigation of the mobile application requires extensive care. Although it will demonstrate numerous similarities to a typical mobile application that uses authentication, it also bears important screens that cause navigation to change dramatically. The ability to navigate between pages, in fact, will vary yet further between Android and iOS platforms.

For starters, the Android platform grants users a navigation bar at the bottom of its phone's screen, which conceals itself below the app screen when unused. This bar is composed of three buttons: a "go to home" button, a "show opened apps" button, and a "go back" button. This "go back" button, at times, may navigate a user to the phone's own home screen if an app's screen has no previous screen left to navigate to. This establishes Android's personal unique method of navigating. In the context of iOS, a "go back" button, along with the rest of the navigation bar's commands, are foregone altogether, with the exception of a "go to home" button, implemented in different ways per Apple phone kind.

Next comes Flutter's method of initiating navigation. It supports its own form of navigation via an app bar widget, where a "go back" button is wedged at the top left corner of this app bar when enabled to be shown. The behavior of this button is practically identical to its Android counterpart. Flutter also supports the existence of a bottom navigation bar widget. Its appearance beckons similarities to Android's navigation bar, but it itself can be retrofitted for a variety of uses. These implementations are important for the screens they will be associated with.

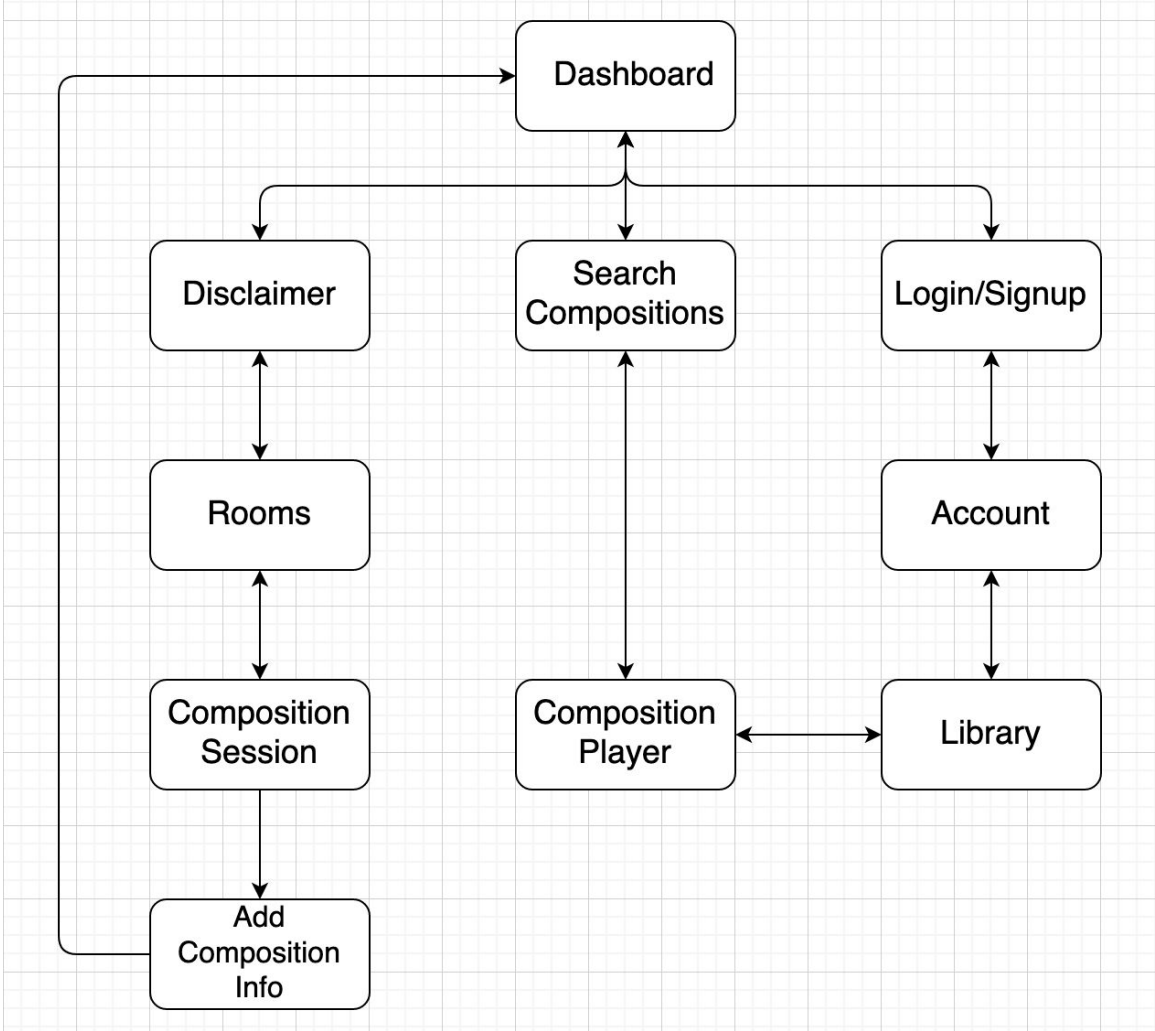


Figure 35: A flowchart entailing the navigation process between the mobile app's individual sections. Note that some arrows are bidirectional, implying back-and-forth navigation. The flowchart does not take Flutter's navigation bar into account.

At the app's front line lies its dashboard. It is functionally designed to greet the user and present them with a variety of options in their selection space. The dashboard, technically, is itself a base screen, providing no form of navigation "back" to another screen. As such, all Android phones' "go back" buttons must route the user to their own phone's home instead. Aside from this, the dashboard is the central hub that takes advantage of the bottom navigation bar to allow users to rapidly switch between the Disclaimer/Room , Authentication, and Search screens.

It will also sport an app bar's "go back" button for the sake of simplicity and recognizability to the user, leading them back to any screen that preceded this current

one. The Android's "go back" button will remain enabled in the event that the user is more comfortable with it than an app's own form of backwards navigation.

The Library screen is another prominent screen that any registered user can navigate to through the Account screen. As such, it is predominantly similar to the Search screen. Users will be allowed backwards-navigation with use of the app bar's "go back" button, the Android bar's "go back" button, and Flutter's very own navigation bar, which will allow it direct access back to the Account screen. When a user selects a composition to lend an ear to, it will direct them to an audio player.

Another prominent screen, the Search screen, is inherently identical to navigate to compared to the Dashboard, Disclaimer/Room and Account screens. It will allow for navigation through the app bar, Android bar, and Flutter's own bar. Just as the Library screen above does, it can lead the user to an audio player for a selected composition.

Two niche sectors of the app, the room authentication popup dialog (via a PIN) and the audio player, are similar in their inability to modify data in a meaningful way and instead query the server (via sockets) or database for loading data. As such, they will obey similar behavior to the prominent screens mentioned above. However, for the authentication's case, it is the guidepoint to one of the most important screens in the project: the Session screen.

The Session screen is essential for the development of a to-be composition. Despite this, as precarious a process as it is, it can be incredibly easy to disrupt one's flow or the app's navigation. When users are in the middle of a performance session, the very last incident they would appreciate is being taken outside of their current screen and to another one. Hence, special precautions must be taken to ensure that the performance goes as smoothly as possible. With that said, Flutter's bottom navigation bar is easy to reach and tap, and thus it is easy to also tap on accident. This bar will not be present on these screens to prevent such an event from occurring. Android's navigation bar's "go back" button, unfortunately, cannot be temporarily disabled. In response, a WillPopScope widget, whose purpose is to catch backwards navigation, will spawn an alert dialog that asks if the user is sure if they'd like to leave their room. This dialog does not open unless the user is in the middle of a performance as a performer, or if they are the composer. Listeners, however, are free to come and go as they choose. The Flutter app bar's "go back" button acts rather similarly to the Android navigation bar's own button, so the WillPopScope widget will be active for both buttons.

6.1.2 Web Page

To begin with, the main objectives of the website in this system is for the purposes of:

- Creating and deleting user accounts.
- Searching for compositions.
- Playing music produced during recording sessions.

The user interface design of the website will be done using React.js as part of the MERN web development stack. React, and MERN in general, was originally chosen by project sponsor Richard Leinecker. The following sections will

Home Page:

This will be the starting point for users when they enter the website. There will be buttons for logging in and signing up if the user doesn't already have an account. The "sign up" button will lead to the account creation page. There will be a button for entering the "System-wide music search" without an account. I have also considered adding something like a short biography of John Cage given that this meant to be a tribute or at least a mission statement to describe the purpose of the system/website as well as a quick overview of how the system works so people, although these are not particularly essential to the function of the website. As a stretch goal, I could implement the page such that users can opt to have their login information remembered and stay logged in, in which case, upon loading the page each time thereafter, the page would simply display a welcome message and a button to go to their user dashboard.

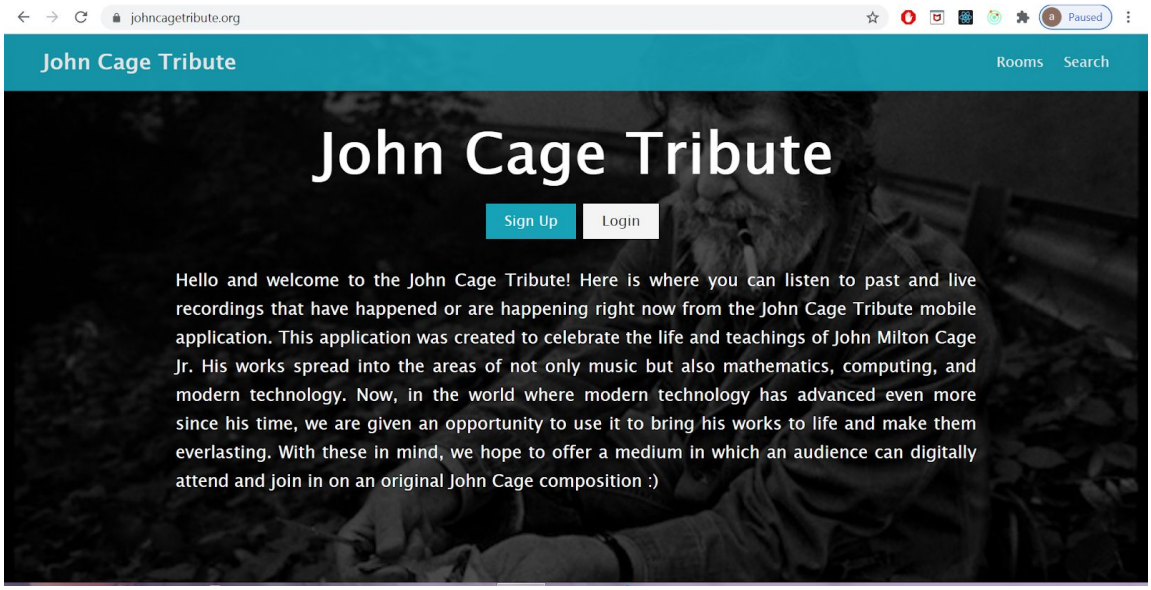


Figure 36: Home page of website

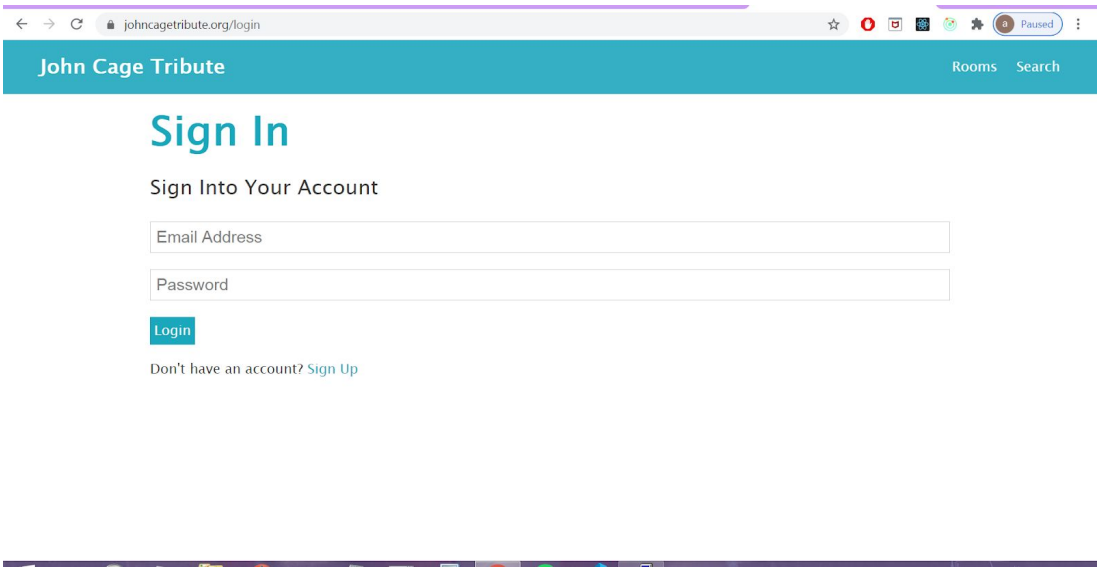


Figure 37: Login Page of website

Account Creation:

As mentioned above, this is the page that users will be taken to after they click the “sign up” button on the home page of the website. The page will display text fields for entering their desired username, password, and email, all of which will be required fields. By required, I mean that the input HTML tags will actually be made using the required

attribute so that, upon submission of the form, the browser itself will check if the user has entered anything in the "required" inputs. In the case that the required inputs are empty, the inputs will output an error message to the user. If the form has been filled out properly, then the page will continue with the submission of the data. After clicking the submit button, the page will make a call to the database to check that the requested username and email are not already in the system. If the information is not accepted, the user will be asked to try with different entries. When all of the information has been accepted, the user will be taken to their dashboard.

User Dashboard:

This page is where the user can view personal information and access all of their recorded compositions. The user will also be able to edit details of the composition like title, tags, description and privacy settings. The privacy settings will be either public or private, private by default, and this will decide whether the composition can be found by other users in the system-wide search. The user will also be able to delete a composition.

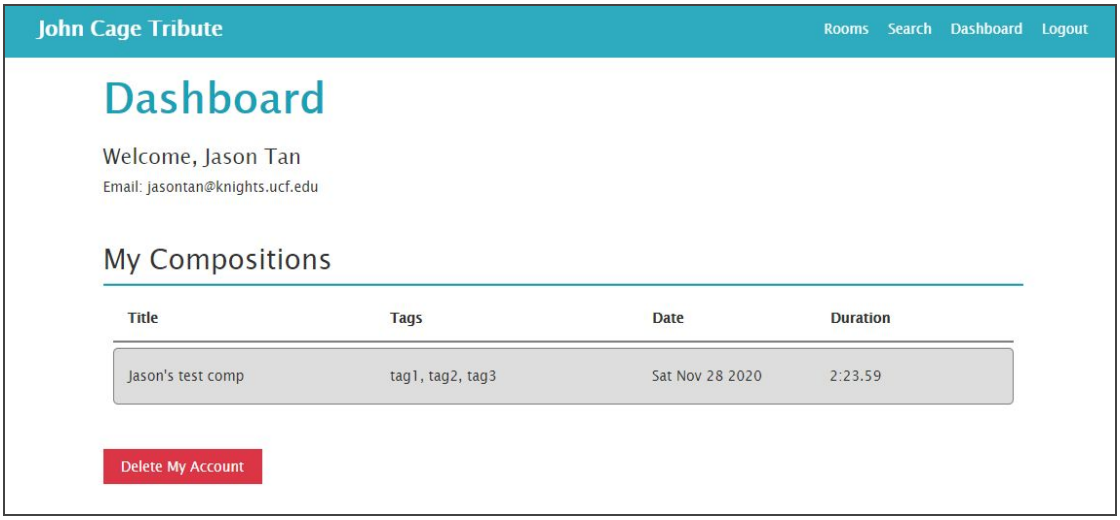


Figure 38: A screenshot of the user dashboard

System-wide Music Search:

This page will allow users to search public compositions made by others. I will not be labeling this page "System-wide music search" when it's deployed, more likely it will

just be “Music Search” or something along those lines. I figured I’d just refer to it as “System-wide” during planning to differentiate it from the search page specific to the user’s own music. The page will have options to search by name, affiliated organization, duration, and date. The result items will display all of the information listed above, with the exception of organization if the composition does not have one. I will be using the HTML audio tag for playing music in the browser. This tag will render controls for pausing/playing the specified audio file, navigating to a specific point in the song (otherwise known as scrubbing or seeking), changing volume, and even downloading the associated audio file.

I had considered several options for how the audio controls rendered by the audio tag will be displayed for the user once they have found the selection they’re looking for. At first, one option I thought was for the controls to be displayed directly on the search results for each result. Another option is for the page to render a lightbox over the search result page with either the controls by itself or a small page with the song details listed in addition to the controls. Yet another option is for the controls to have a single designated place somewhere on the page such as the top or the side of the page, as seen in many audio playing programs, which would change depending on the selection made by the user. I decided this would likely be the best option.

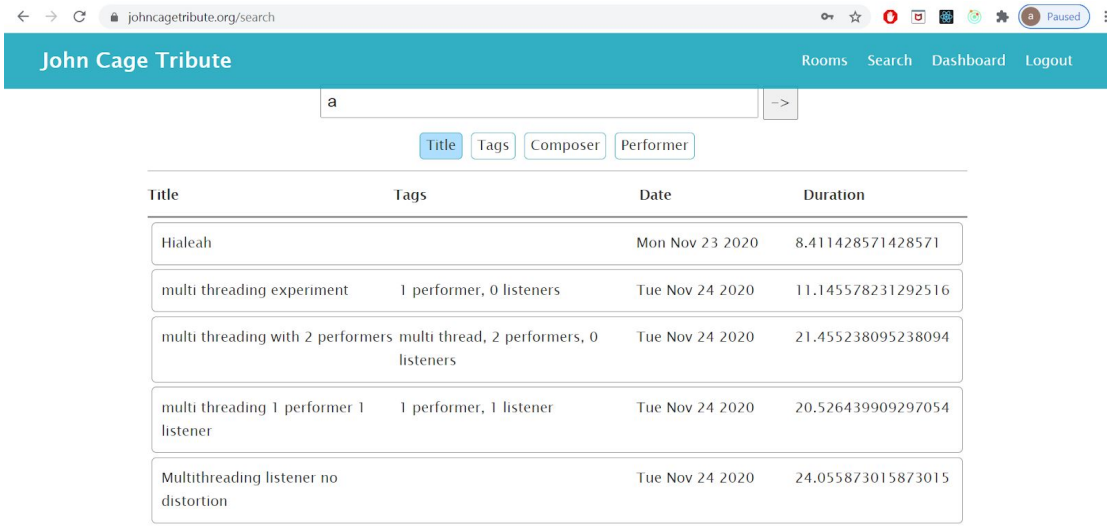


Figure 39: The search page

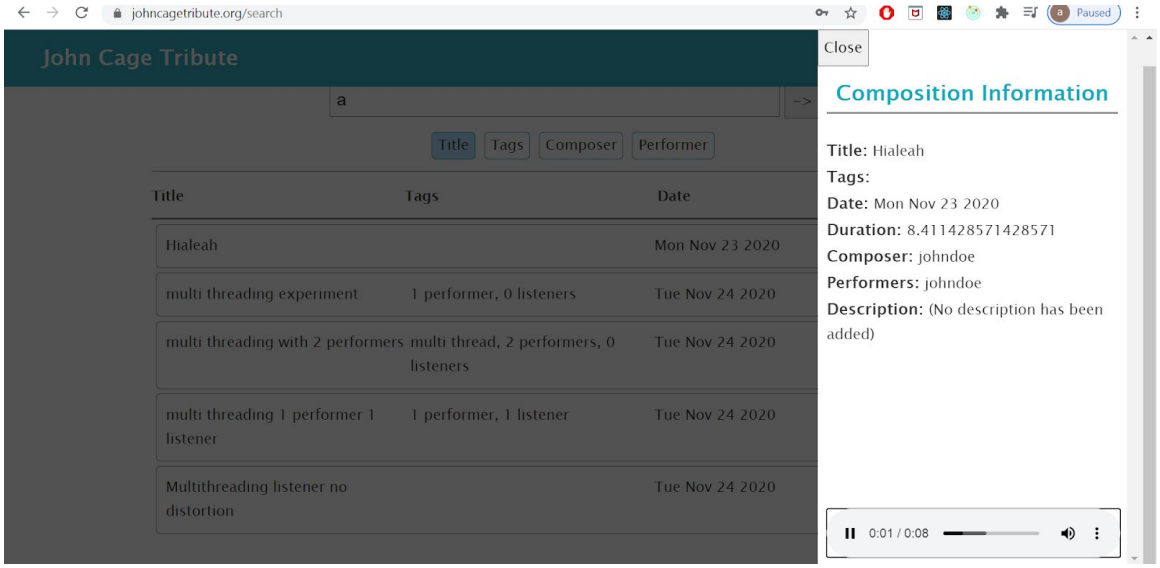


Figure 40: The web page when a composition is selected

Important React Components

CompList: This component is responsible for displaying lists of the compositions on the Search and Dashboard pages after they have been retrieved from the database. It primarily takes a javascript array of composition objects to be mapped into CompListItem components.

CompListItem: This component begins with the element shown in the list with only the title, tags, date, and runtime displayed. Once this element is clicked on, it changes state, triggering the sidebar featuring all of the other composition details along with the audio player with which users can listen to the actual audio file. When the CompList is created on the Dashboard, the user object is passed all the way down to CompListItems to be used for verifying that the user viewing the composition details is the owner of the composition, in which case, the Edit and Delete buttons are enabled.

Search: This component is fairly self-explanatory. It features an input bar for making queries according to the chosen search option. On submission of the search query, the input value is read along with the search parameter to make the appropriate API call in order to retrieve the applicable compositions. The resulting list is then passed to a

CompList to be displayed on the web page.

Rooms: This page was meant to utilize socket.io connections to interact with the server and mobile app in real-time so that recordings could be heard live on the website. Unfortunately, we ran out of time to complete it and it had to be left in an unpolished state. Most of the functionality has been implemented however. Active rooms can be shown on the page where a user can join it directly if the room does not have a pin. In the case that the room does have a pin, an input form is displayed where they can attempt to enter it. Once the user has joined the room, the room modal is displayed where users can see the other members in the room, the host of the room, and the current status. The lists of members are updated in real-time using the sockets and when the host begins the session, the status is also updated in real-time. Unfortunately, the main function of the room, receiving audio, is the part that still requires further work.

6.2 Backend API

6.2.1 Server

For this project, we will be using a cloud based server. The server will be the center of our software, as it will have connections to all of the separate components. The server establishes connections to users on the mobile app using sockets, and can receive, process, and return multiple 16-bit audio streams. Audio from music sessions can be temporarily stored on the server until a session is complete, then it is encoded and uploaded to the connected database. The front-end web application can connect with the server in order to access user accounts, and access the database that contains past recordings.

Server interaction with the system

The server will be the central hub of the entire system. It will send and receive signals from the other components such as mobile app, website, and database. In figure 41 we see how the server interacts with the other components during an active recording session.

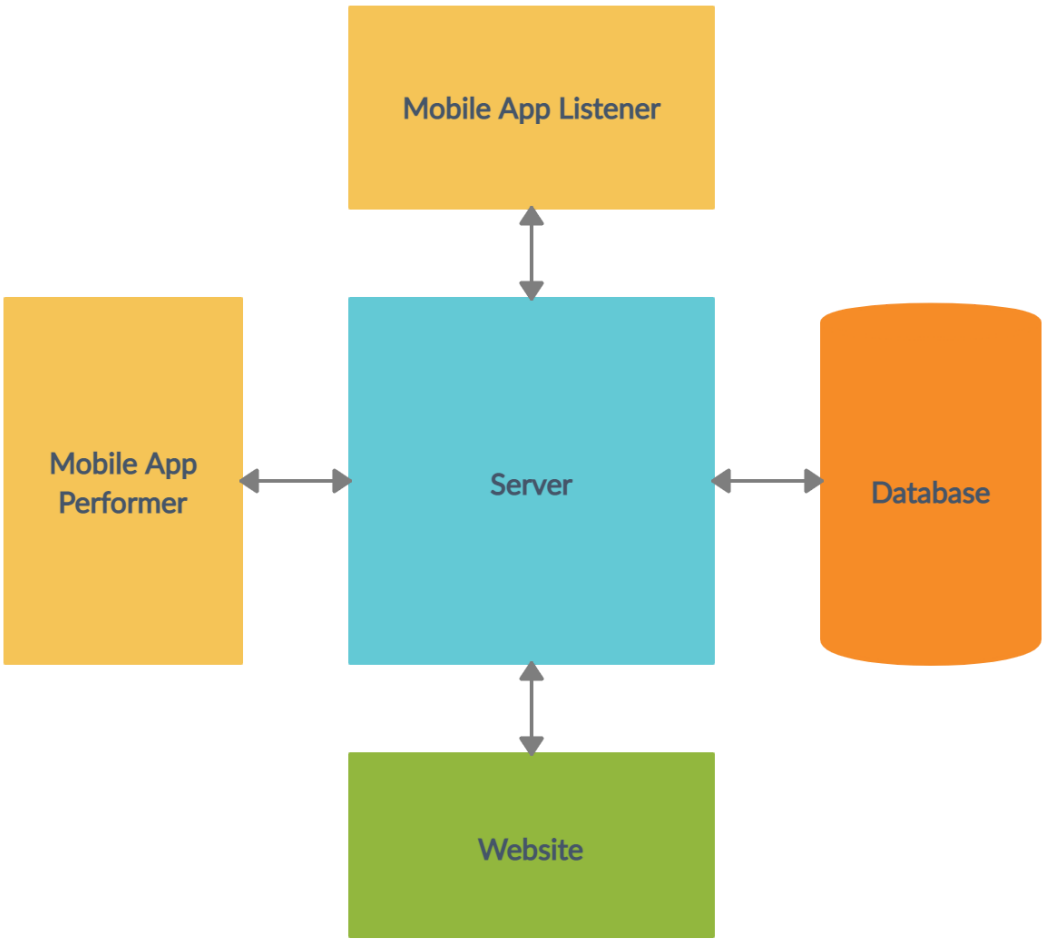


Figure 41: The server's interactions with other components

The server allows for communication between all the components of this system. Users that are performers on the mobile app can stream audio data to the server for processing. Users that are listeners on the mobile app can receive audio data that has been processed. Users on the mobile app and the website can also access the database through the server. These routes are described in greater detail in section 6.2.4.

Socket management

One of the ways the server will communicate with the other components in the system through sockets. The socket connections will be managed by the server which will act as the host of the system. There will be 2 main types of clients that will be making socket connections to the server during operation, performers and listeners. performers are users

running the mobile app on a mobile phone that will be recording sounds using their phone's microphone. Listeners are users with either the mobile app or the website that will live stream the composition as it is created. The server organizes connections into rooms which can be created from the mobile app. Once a room is created, it is visible to other users who can then join it.

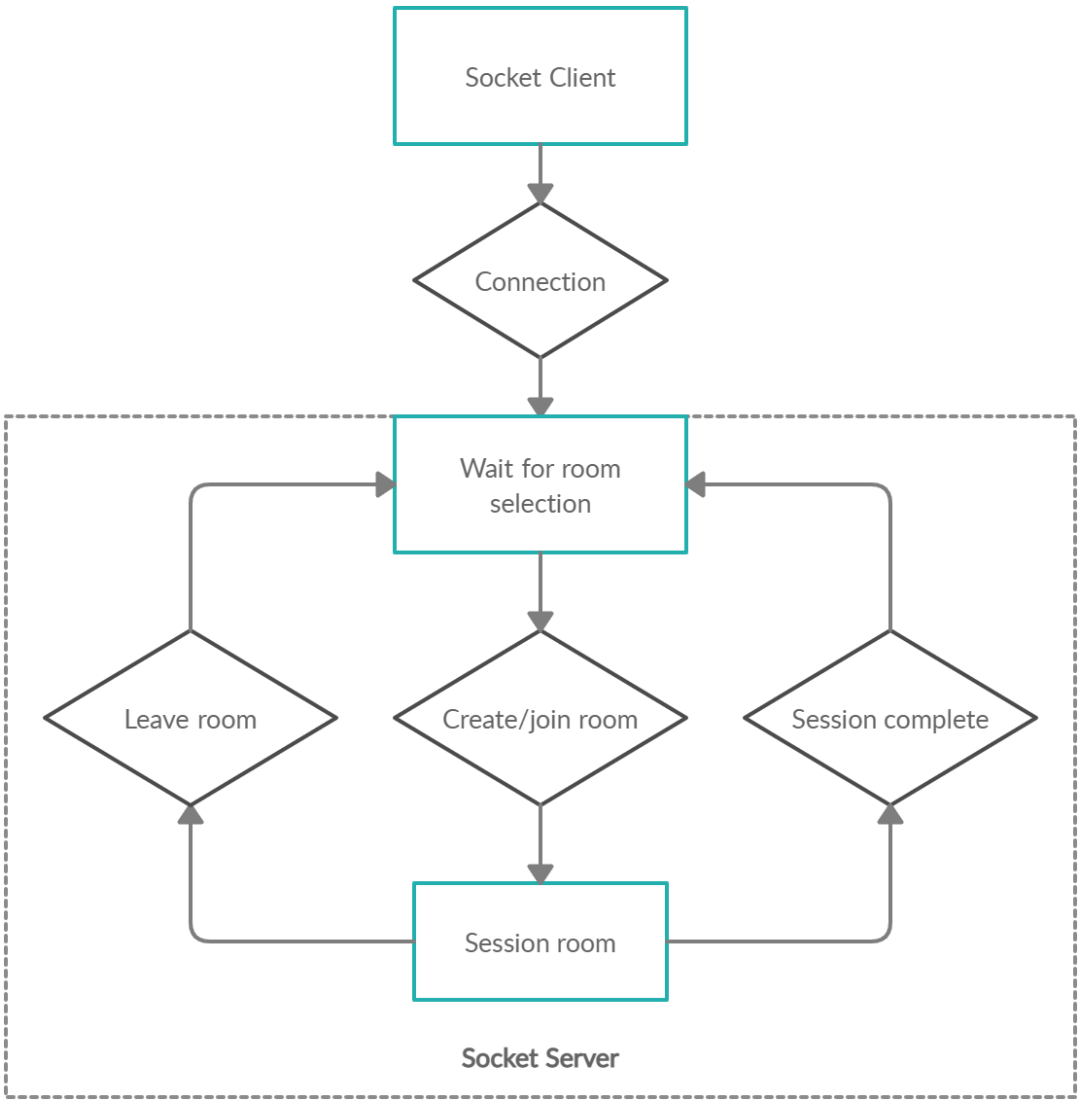


Figure 42: Server socket management

From the room, depending on the clients role, they will either act as a performer and contribute audio to the session, or act as a listener and hear the audio from the session in real time. They can also leave a room without disconnecting from the server, and they

will be returned to the room selection screen.

When a recording session starts, the server will signal all of the members in that room to either start sending audio in the case of performers, or prepare to receive audio in the case of listeners. The audio data is streamed to the server from the performers and mixed together. After the audio is mixed it is streamed to the members of the room that are listeners. When the recording session is complete, the room is deleted and all its members are returned to the room selection screen.

6.2.2 Audio Processor

The Audio Processor is a separate module that receives audio data sent from performers in a room, mixes it together, and returns it. One approach and the original idea for this module would be to manually add the audio data from each performer together. While this would work in mixing multiple audio streams into one, it provides no ability to expand the or scale. We want to process the audio in a way that could be easily upgraded, expanded upon, and is scalable. Doing the process manually would not allow for this, so we decided against it. Instead, we chose to utilize an existing API that would allow for modular expansion of the audio processor to include things like audio analysis and audio modification.

Web-Audio-Api

Web Audio API is a versatile open source system that allows for the manipulation and processing of audio. It is primarily designed to work in a web based environment, so we will be using the npm module web-audio-js that provides a node wrapper around the API to be used in a backend server environment. The basis of Web Audio API is an audio context, which serves as an environment for all audio processing to take place. Inside the audio context, all audio is routed through objects called audio nodes. The routing of audio through audio nodes is modular, allowing multiple nodes to be connected together into an audio graph.

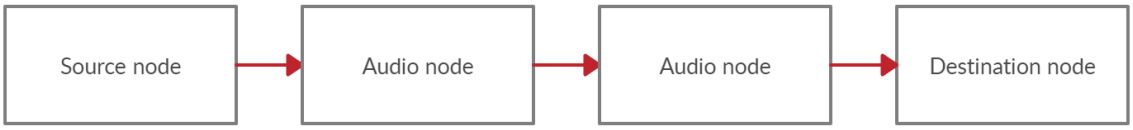


Figure 43: Audio graph example

The audio graph consists of source nodes, processing nodes, and a destination node. In our case, the source nodes are created from the audio data received from performers. We don't currently utilize any processing nodes, however in future development of this project much of the functionality about randomization and audio effects could be implemented using processing nodes. The audio destination node allows for multiple input connections, and automatically mixes the incoming audio together and returns it. We use the output of the audio destination node as the mixed audio that we send to listeners and save to the database.

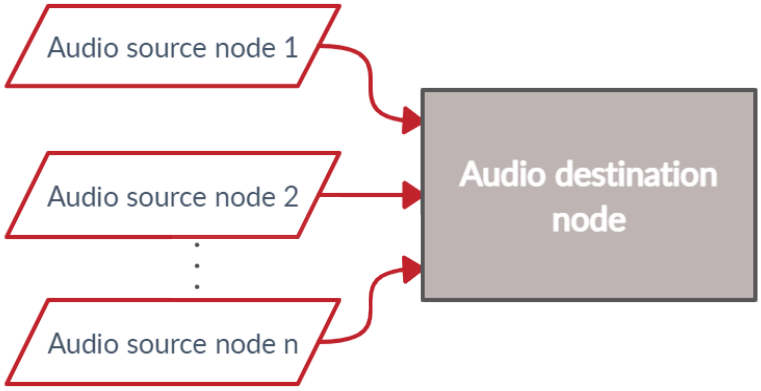


Figure 44: Audio graph for recording session with n performers

Multithreaded Design

Processing audio is a computation-intensive task, especially in our system where we will be working with multiple constant streams of audio data during a recording session. Node.js by default only runs on a single thread with a single main event loop. For our system the main thread and event loop is needed to manage socket connections and rooms, so performing such a resource intensive task as audio processing in the same

thread can cause the whole server to lock up and even crash. In early tests of the server with everything running on the main thread, we frequently saw the server lock up, lose connections to sockets, and crash. To solve this issue and make our system more stable and scalable, we decided to move the computation-intensive task of audio processing to a separate thread.

We achieve multithreading in Node.js by using the built in child_process module. We need to create a separate process running on its own thread, so we utilize the child_process fork() method. This allows for a running Node.js process to spawn a completely separate Node.js process that is connected to the parent process through an IPC communication channel. The parent and child processes can utilize the send() method to send messages and data to each other, and the on() method to listen for incoming messages.

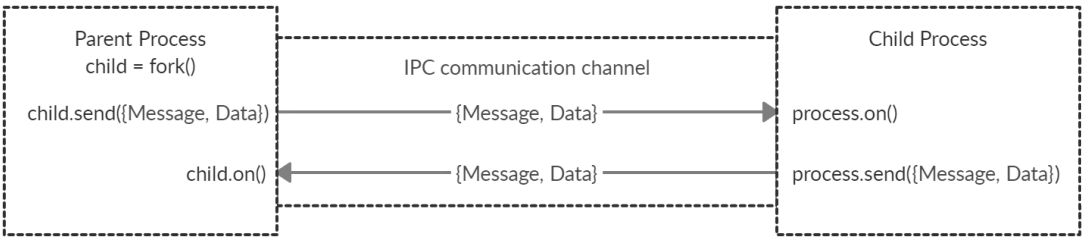


Figure 45: Example of child process creation with fork()

In order to prevent performance issues with the audio processing, we could create a child process with fork(), send all the audio data from performers to it, allow it to mix the audio, and then send it back. This would separate the audio processing task from the main event loop and would solve the problem. This design comes with the major drawback of limiting the server to only one room and one recording session at a time.

To allow for multiple rooms and multiple recording sessions to take place simultaneously, we need to create a thread pool process capable of creating audio processor threads dynamically. This thread pool will need to create an audio processor thread whenever a new session room is created, route the audio data from performers in that room to the associated audio processor thread, and then receive the mixed audio from that thread and send it back to the server. It will also need to kill the audio processor threads for a room whenever that room is closed.

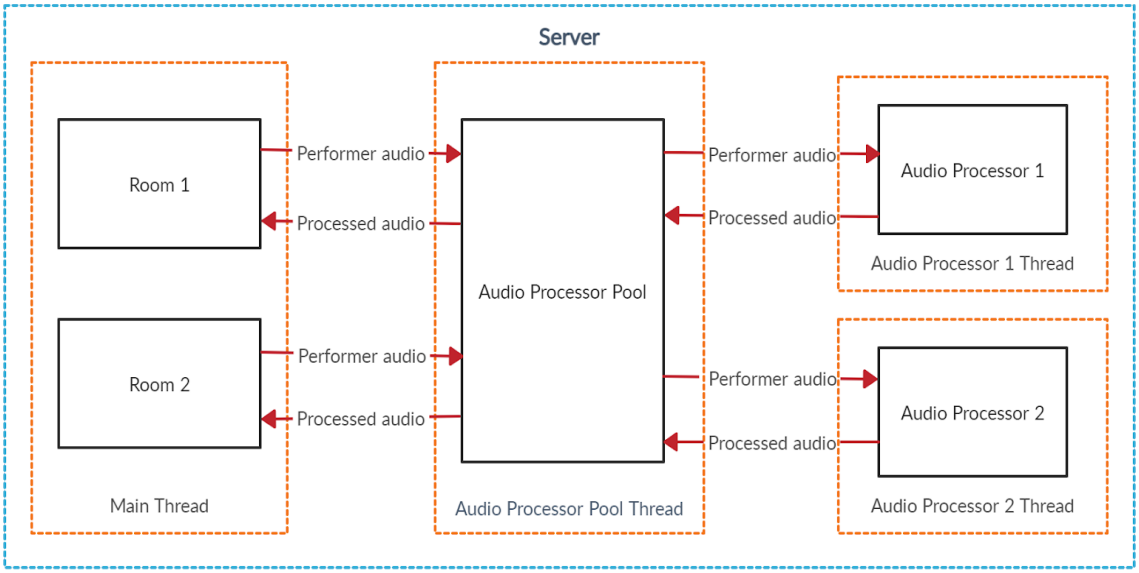


Figure 46: Multithreaded server design with two active rooms

We can see in figure 46 that the audio processor pool handles all incoming data sent from the server and works with multiple rooms. The server doesn't need to interact with or manage the audio processor threads, which is done by the thread pool process. This ensures that we have no bottlenecks in the system, and that none of the threads get overloaded even when there are multiple rooms with many users.

The main server thread and the audio processor pool thread will always be running when the system is deployed waiting for incoming connections and messages. Upon the creation of a new room, the main thread signals the audio processor pool thread to create a new audio processor thread. The audio processor pool maintains a map of connected audio processors organized by the ID of the room they are associated with. Each audio processor thread contains an instance of Web Audio API to process the incoming audio data. Mixed audio from the audio processor thread is returned to the audio processor pool, which in turn sends it back to the main server thread along with the associated room ID so it can be sent to the listeners of that room.

Scalability

With the multithreaded design of the server, the software has no limit to the number of active sessions that can take place simultaneously. However, there are hardware limitations that could impact the scalability of this design. While multiple threads

running on the same CPU core do maintain their own event loops, they still have to share CPU time in order to do work. This means that the number of physical CPU cores available to the server will give us a hard limit on the number of active rooms.

With our current server hosted using DigitalOcean, we have two physical CPU cores available. Our current tests and estimates show a 25% utilization of both cores for a recording session with 4 performers. We estimate that up to 3 concurrent rooms should be possible without overloading the resources available. There is also the potential for any additional audio processing nodes added to the audio graph in the audio processor to add more load to the system for similar use cases.

6.2.3 Database

The database has two primary purposes: store user data, and store the data of previously recorded tracks. Early on in project development, we were also planning on storing the information of appointments, but since we are instead using rooms with socket.io, those have become obsolete, so we no longer have a need to store appointments. The database will be stored on our server, along with the mixing algorithm. As we are using a MERN stack, it will be a MongoDB database, which has the advantage of using Gridfs in order to store MP3 files directly on the database.

First is the composer user accounts. Figure 47 shows the information stored by the user field. In order to create a room to start a composition, an account must be created. When a composer creates an account, they need to provide three things. An email address and a password will need to be provided so that a composer can log in after registering an account. Lastly, a name will be provided, which will be used to show who actually created a composition time. Along with that, the date when the account was created will also be stored.

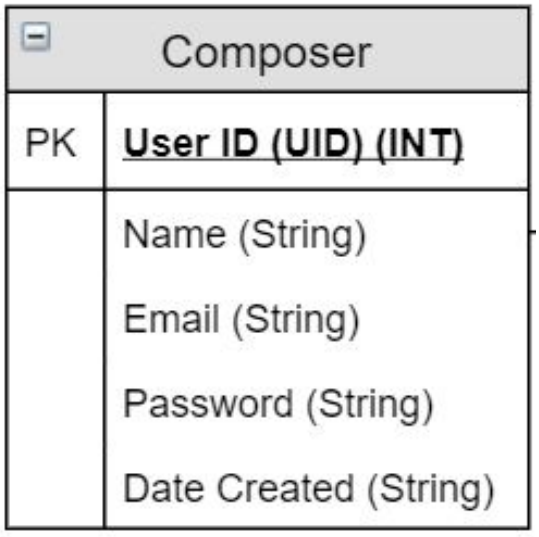


Figure 47: A visual representation of the User model.

Finally, there is the composition itself. Figure 48 shows the information stored in the recording field. The composition will be stored in an archive, so the database will need to store information for both the file itself as well as anything needed for the archive. With regards to the file itself, we will be storing an id that corresponds with the file, the amount of bytes that the file contains, the file type of the recording, which the file is always going to be an MP3 file, and the name of the file, which will be generated by the server. With regards to the archive, there will be information taken from the room in which the composer has created. These fields are the length of time the recording was, the name of the composer, as well as the ID of the composer, and the list of performers that were logged in when the composition was recorded. Along with those fields, there will be several optional fields that a composer can fill out after the composition has taken place in order to help other users search for their recording in the archive. They can enter if the composer wishes to private their composition if they don't want anyone else to have access to it, there is a description if the composer wishes to do so, and tags that the composer can add so users can find compositions with a specific tag. If the user decides not to enter in a title and a description, default values will be stored instead. The date of the composition will be created when the composition ID is generated.

Figure 48: A visual representation of the Composition model.

6.2.4 API Routes

One of the greatest challenges of this project is the passing of information between all of the different systems. To help facilitate this, several API routes are used to contact the server in order to store, update and pass along information as needed. Figure 49 below shows an early version of an entity relationship diagram (ERD) that contains the different elements of the database, as well as how they interact with each other. We initially used this in order to map out the many API routes that we were going to need to create. As the project progressed, we realized that the scope of the project was larger than we anticipated. Figure 50 was created to show a simplified version of the direction that our project was going.

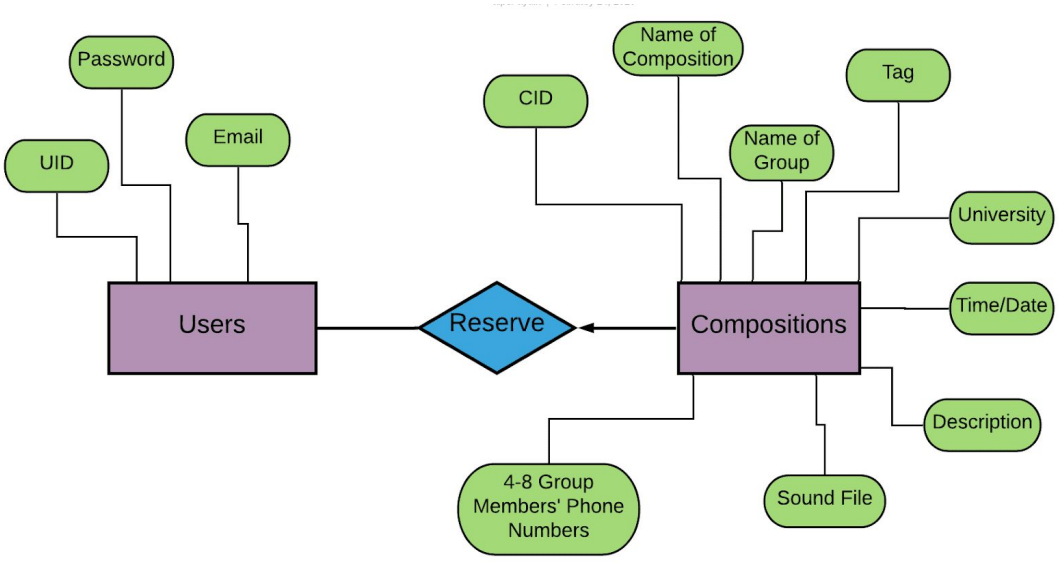


Figure 49: An earlier version of our ERD

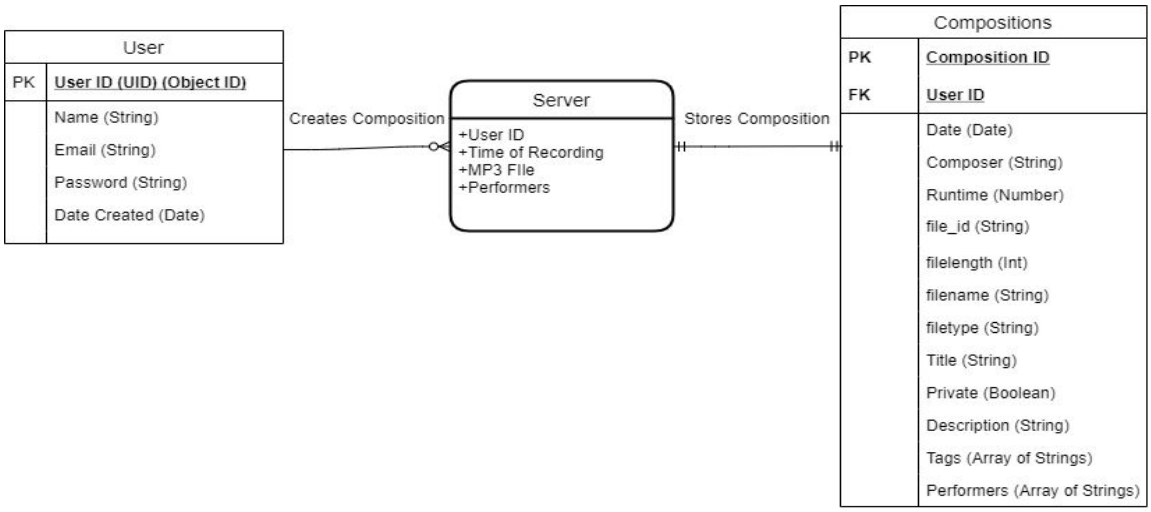


Figure 50: An updated version of the ERD

The first main API route is involved with the composer's information. As stated previously, the composer will need to input their preferred email and password, as well as a username. The backend also makes sure that the email is actually in the form of an email and that the password is at least eight characters long, using express-validator. Once they do that, the backend checks to see if the email they provided is unique. If the email is not unique, then the user. Once it has been confirmed that the email is unique, bcryptJS is used to encrypt the password. The composer profile is then saved as a JSON

web token.

Logging in is similar to registration, in that the user needs to enter in their email and password, though unlike registration, they do not need to enter in a name. Once again, the first step involves searching for that same email, but this time the goal is to actually locate the email, as opposed to making sure it is not there. The program then decrypts the password associated with the email given, and then it compares it to the password given by the user. If the email is not there or if the passwords do not match, the program informs the user that the credentials are invalid.

Once you sign up or log in, a JSONWebToken (JWT) is given. The JWT is formed by taking the user ID and hashing it with a secret key embedded in the config of the backend. This token will be valid for a set amount, after which it will no longer be valid and the user will need to sign in again. This token will be used for authentication, and will be translated back to a User ID when the route requires a user to be signed in to perform certain API calls in the following sections.

The second and final group of routes revolve around the finalized compositions. After the server stops receiving audio data from the compositions, the mobile app will send the ID of the composer, a list of all the performers, and the total runtime of the composition. Once the server has all these things, it does multiple api calls to ensure that the all relevant information to the database is stored. The first thing that should occur is that a composition ID is generated, and this Id, as well as the ID of the composer, are stored. Once generated, it will make two asynchronous api calls. Figure 51 below demonstrates the full process in the form of a flow chart.

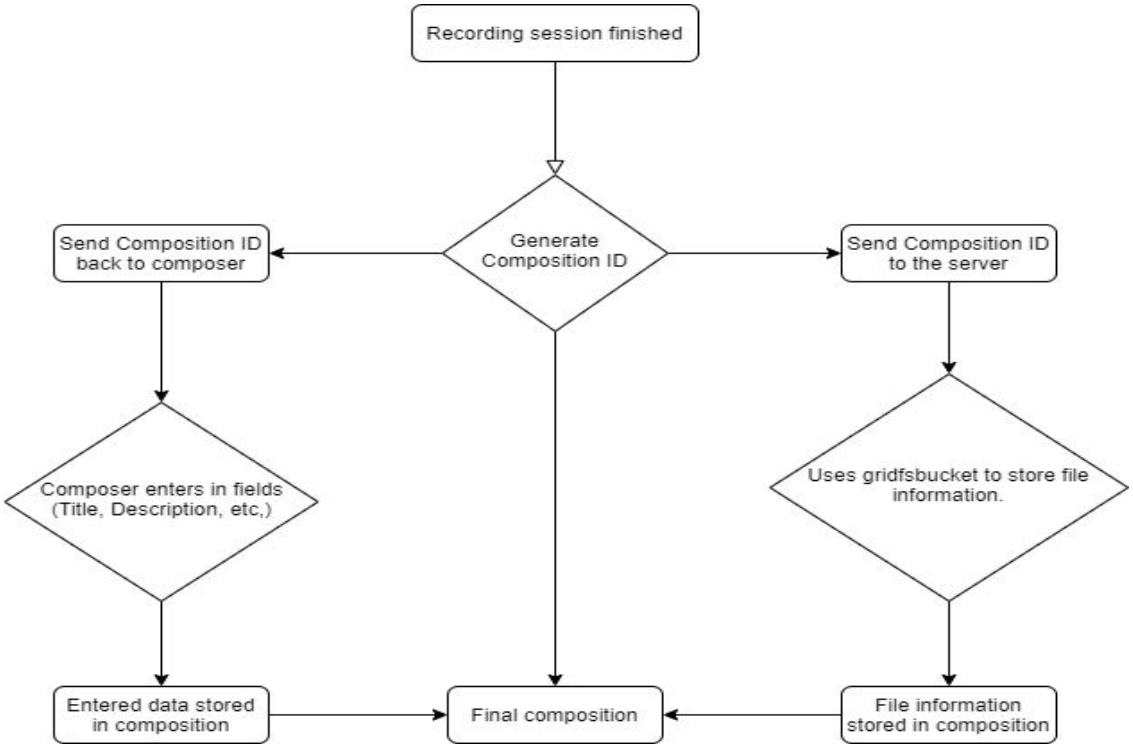


Figure 51: visual representation of composition creation

The first api call will be made back to the composer on the mobile app, giving the composer the composition ID, which the user will add the following fields: the title of the composition, a description of the composition, up to three tags to help search for the compositions, and a true/false tag if the composer wishes to have the composition private or public, respectively.

The second API route called by the server is made once the individual sound files have been mixed together and converted into an MP3. The MP3 file is temporarily stored on the server. This MP3 file, as well as the information sent previously from the mobile app, is sent over to the server. Since we're dealing with files as the input, we no longer could use json as the type of information that was sent over. Instead, we used multipart/form-date, which allows for both file and text inputs. The server will send a Multer object to itself using Node-Fetch. Once received by the server, Multer-gridfs-storage is used to store the file in both gridfs.files and gridfs.chunks, as well as randomly generate a filename using crypto. Once stored in gridfs.files, fileID, file size, filename, and file type are obtained and stored in the composition, as well as the name of the composer and list of performers, are all stored in the database. After this API route is called, the MP3 file that was temporarily written onto the server is then deleted,

as it has no use there now that the MP3 file is properly stored on the database.

After the sound file has been uploaded by the server, the next step is for listeners to be able to listen to the compositions to. First, the archive will display a list of all of the public compositions, sorted by most recently created. There are also routes to filter out compositions based on the title, tags, composer, and performers. Once a listener on either the web or mobile app selects a track, the listener will be able to stream the recording. This is done by using gridfs to locate the file information as well as creating a download stream, which allows the file to be viewable in its given format on both the web and the mobile application. Composers who are logged into either the mobile or web application are able to view a list of both their public and private compositions, as well as delete their own compositions. When deleting compositions, both the composition metadata and the actual file stored in GridFS are deleted. Composers can also delete their own account, which removes the user for the database, as well as all the compositions that contain the same user ID as the user that is being deleted.

6.2.5 Class Diagram

Below is a class diagram for the backend, shown in figure 52.

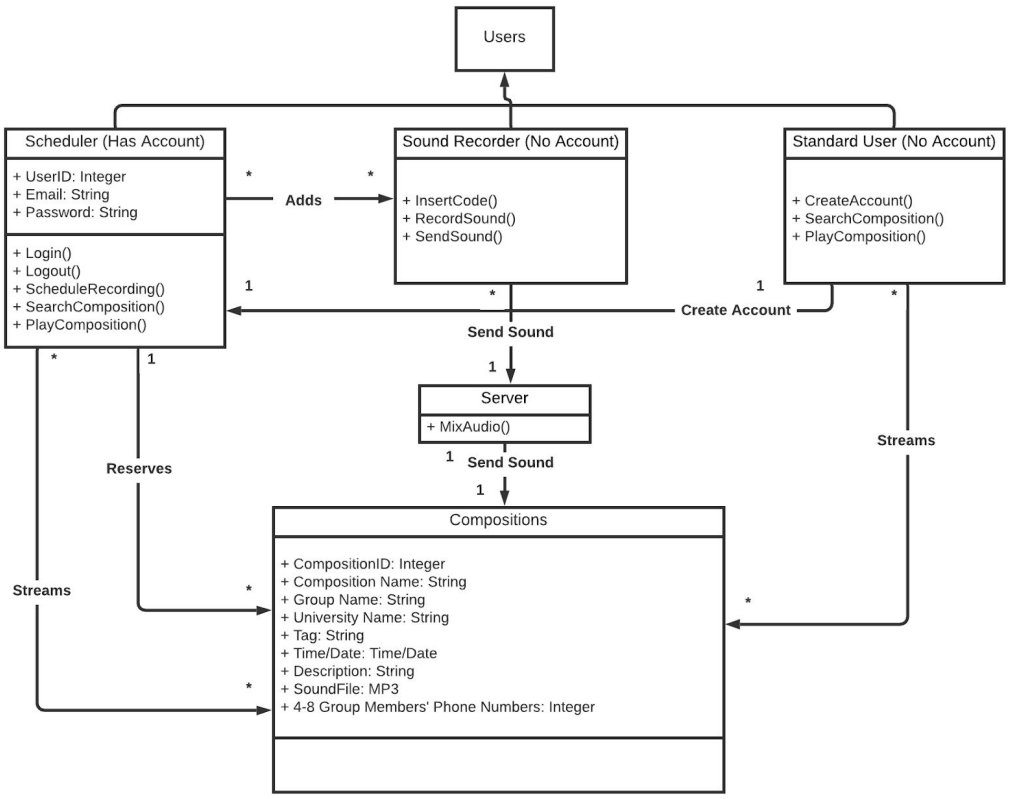


Figure 52: the class diagram of the system representing some of the backend functionality.

7 Testing Details

7.1 User Feedback

During the production of the John Cage Tribute, core audio functionalities cannot be verified without an assortment of kinds of audio input. As such, mock sessions by numerous users (preferably those with a stronger musical background / awareness of John Cage) and collections of the resulting audio files can indicate to the Tribute team far more than individual tests in our local environments.

Similarly, users' interactions with the web and mobile interfaces, along with their critiques and opinions of them, can provide substantial feedback for improving the look and feel of the project before its slated completion in October 2020. Qualities such as color scheme, animation behavior, input latency, and ease of navigation and comprehension are of tantamount importance for maintaining a high standard of John Cage and his legacy.

We may expect to schedule "miniature demos" scattered around any time before our project's completion. These demo phases can be conducted with different groups of individuals and with established expectations. In earlier phases, we can aim to contact close friends and family members to chime in on the Tribute's color scheme, alongside its responsiveness and familiarity. As the deadline slowly nears, we may then turn to contacting members of the University of Central Florida's music department (with special thanks to group member Jacob, who is known in their community). These members would then critique the algorithm's music-mixing scheme, a more specific aspect of the project, than any prior details. This organized system of batch demos can more easily isolate our production from our testing.

7.2 Continuous Integration/Delivery (CI/CD)

In order to establish discrete components of the Tribute as complete, extensive batch testing must be completed in order to label them as complete. Testing these individual

components by hand or with numerous test cases is, at times, a very time-intensive expenditure. The use of a pipeline testing program can become all the more desirable. This allows for new code implementations to be immediately tested and recorded back to the development team to demonstrate whether a change was effective or not. Notable potential programs include the widely used Jenkins and the git-inclined GitLab. Due to overwhelming support for Jenkins, it is currently our expected choice.

Jenkins, authored by Kohsuke Kawaguchi, was one of the answers to the prayer of a development process in the software engineering world, blessed without error-checking by hand.

Jenkins supports an assortment of plugins to ease interaction between certain kinds of applications to be tested (i.e. Flutter apps, Java servers, etc.). However, it requires a decent level of contribution to be learned to its fullest. Once this obstacle is passed, Jenkins's CI/CD promises can be achieved. Continuous Integration is the software engineering concept of testing one's code relentlessly to ensure Continuous Delivery of completed components. As such, Jenkins can be applied to any aspect of our project in slices, ensuring the robustness of each one.

7.3 Unit Testing

For every individual function in our system, we need a dedicated unit test to verify that it is performing in the desired way. Since there may be many hundreds if not thousands of functions in our system, we will need to use an automated unit testing framework. One of the most popular and widely used unit testing frameworks for Javascript is Jest. Jest is an open source unit testing framework developed by

Facebook. It is extremely well documented with a large user base and lots of resources available. It is also very fast with tests being automatically parallelized at run time. It will suit our purposes well with asynchronous function support and built in mocking. We will use Jest for our unit testing as well as some automated integration testing. Since we are using a MERN development stack, most of our system will be based on Javascript allowing us to use this testing framework for most if not all of our unit testing.

7.4 Integration Testing

In addition to individual unit testing, it is necessary to test each major component of our system individually before the final system tests. For our project, the major components will be the mobile app, server, database, and website. Each of these should be tested in isolation from the rest of the system to ensure that all of the functionalities of that component can be integrated together successfully. This is a vital stage of testing because it exercises all of the interactions between the functions in a component. If this stage is skipped and something breaks in the full system test, it is very difficult to find the cause of the problem as it could be from any of the function interactions or the component interactions. By testing the fully integrated components individually, we can know when we perform the full system test that if any issues arise they will be purely from component to component interaction and not from the inner workings of each component.

Integration Testing can be performed with an automated framework or manually. In both cases the techniques are generally the same. In order to remove the dependencies of component to component interaction while still exercising full functionality, it is necessary to use mock objects or components. A mock is a fake version of an object or component that stands in for the real one during testing. Mocks allow a component to run as if fully connected to the system while being completely isolated with respect to dependencies. Mocks generally use fixed or predictably generated data so the functionality of the component under test can be verified. In figure 53 we see the audio processing server performing integration testing. In this case the server is the only ‘real’ component being tested, with everything else being a mock.

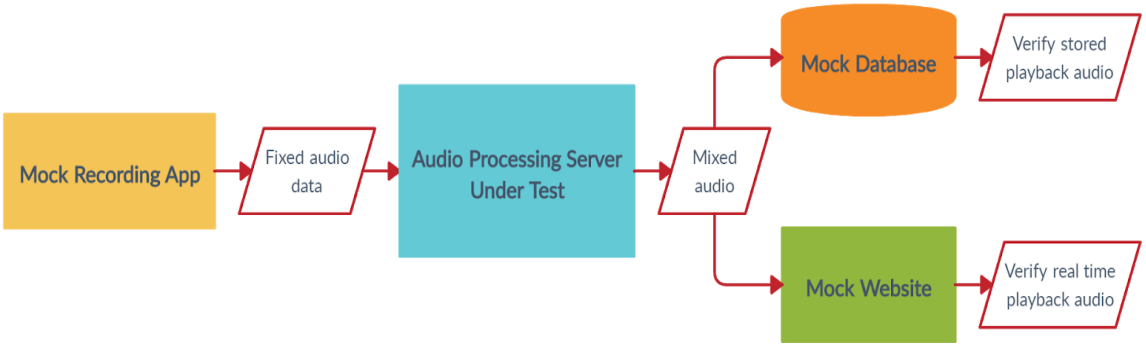


Figure 53: Mock interaction for the audio processing server under integration testing

In the scenario above we can fix the audio data going into the server so we know what should be sent to the database and website after it has been processed. We can then verify that the output is consistent with the designed behavior. Here the server is operating as if it was running with the complete system. This process if done thoroughly on all components will allow the complete system to be connected seamlessly and all major issues will be much easier to track down and fix.

For our project we will need to make mocks of the recording app, audio processing server, database, and website. These mocks should be as simple as possible with little to no logic involved to minimize the number of failure points. They should make all of the same websocket connections as the real components for sending and receiving. When receiving, the mocks should report the data received for validation and if appropriate send a predefined response. For our system both the recording app and the audio processing server send audio data to other components. The mocks of these components should send predefined data of the same format. The goal of the mocks is to allow the component under test to exercise its complete functionality while isolated from the rest of the system.

7.5 System Testing

The final stage of testing is the full system test. This is where each of the individual components (mobile app, server, database, and website in our case) are connected together to test the full functionality of the system. This is generally performed manually in real world situations where the system was designed to operate.

This will be where we deploy our system with the recording app running on multiple phones and the audio processing server, database, and website all running on their respective cloud platforms. Base functionality as well as edge cases and failure cases should be exercised in system testing. This will be the first time all the components will be interacting with each other with real data.

We will want to monitor real world characteristics of our system when in use such as latency, performance, ease of use, and of course the production of real John Cage style music. We will need to establish a baseline with the minimum required devices for the system of 2 phones recording, and ramp up to the maximum of 8 phones recording. If all other stages of testing are performed well, there shouldn't be any major bugs in the system at this stage. The system test is more about the experience of the entire system and the real world usability characteristics. However, bugs caught in this stage are always better than bugs caught after the system is deployed to the users.

7.6 Mobile Development Testing

Aside from user data, tests on individual functions of the mobile application will be necessary to achieve proper compilation and runtime. Impressively, Flutter sports its own testing environment built into its SDK. It can facilitate testing under three primary categories: unit testing, widget testing, and integration testing.

Unit tests consist of testing mere functions or classes. Although any programming language, to an extent, allows for unit testing, Flutter mediates the process by providing packages that initiate mock tests, namely the Mockito package. Mockito supplies mock classes such as the HTTP Mock Client to abstract the requirement of a test server setup to allow for focus on the mobile aspect of the project. In addition, Flutter may automatically run some or all unit tests at the behest of its programmer before loading the app itself.

Widget tests are Flutter's unique way of interacting with any widgets one has created. These are also known as "component tests." These tests generally require a phone medium to operate on, such as an iOS simulator or Android emulator. The tests imitate user actions and initiate widget creation and manipulation, allowing programmers to witness the expected behavior in real-time. This saves tremendous time for the programmer so that they need not test the app themselves and in an arduous, redundant

manner.

Integration tests are the final element of Flutter's testing repertoire. They test a majority of the app, if not the entire app itself. They are built to load services and manipulate widgets while measuring up the app's performance and proper behavior. Like widget tests, they may be tested on phone programs. Additionally, they may even be tested on actual phones. Their primary directory in a Flutter app, *test_driver*, isolates itself from the rest of the tests and app folder to ensure objectivity. The integration tests borrow their functionality from Flutter's *flutter_driver* package.

Manual tests by our development team often consist of running virtual emulators (called simulators on iOS) and our physical devices to load the John Cage Tribute app on them. By doing so, easily-recognized bugs and oddities can be spotted and quickly removed during our development process.

Additionally, there are means to perform tests with the help from internal and external testers through the use of TestFlight for iOS, for emailing application links to registered TestFlight users for the John Cage Tribute. On Android, this is instead performed through manually sending a generated .apk file (the app's file) to users selected for testing and review.

7.7 Backend Testing

Since the backend began development before the DigitalOcean server was launched, we needed some way to test the backend code. Even after the server was launched, a way to test the backend code would still be necessary to test code before integration with the server.

To test the backend routes before we launched our DigitalOcean, and to test backend api routes before adding them to the server code, we needed a way to test them without taking up too much time to write out html tests for each route. To accomplish this, we utilized Postman, which is an application that simulates api calls that are on the backend server code. By running a localhost through the terminal, you are able to use Postman to simulate api calls that would be made by users on the mobile or web application. We mainly tested it to check the passage of JSON and Multiport information from a user to the MongoDB database. The figure below shows the route that is being tested as well as the multipart/form-data body that would be used by the server when storing an MP3 file

in the database.

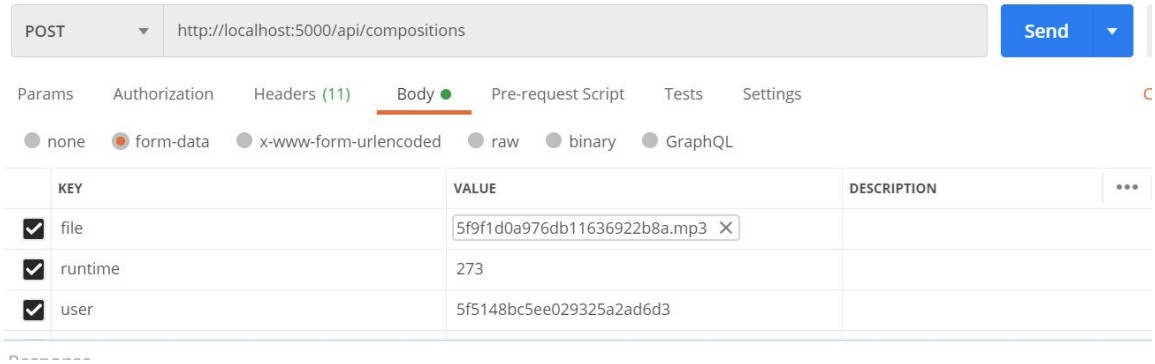


Figure 54: Example test run in Postman

8 Finance

There are a few costs paid for by our sponsor, Dr. Leinecker, in order to maintain the John Cage tribute. First is maintaining the DigitalOcean server, which costs 20$ a month. The second cost is having an apple developer account in order to have our mobile app on IOS devices, which is an annual payment of $99, as well as a one time $25 fee to put an android application on the google play store. Since we are currently using a free tier for mongoDB, there is no cost regarding the database.

9 Installation Instructions

In order for a user to indulge in the John Cage Tribute at its utmost, they must visit the Tribute's website. From here, they are expected to create a personal account, consisting of a username, email, and password.

In tandem, they may proceed to download the John Cage Tribute mobile application on their phone (expected to be launched on Android and iOS). Through this app, they may sign in to access their previous compositions via the Library screen. In addition, they are capable of joining currently active rooms in order to record through their phone or join as the intended listener. Seeing as the project, on its outermost layer, is a web app + mobile app combination, it does not require substantial installation procedures. For developers, the application may be installed via distribution of a .apk generated from the John-Cage-Tribute repository on GitHub or request access to Dr. Leinecker's Apple Development team to run a build of the application via TestFlight on their iOS device, which may be accomplished by signing up as a member or requesting an email link that sends a signed .ipa file to the developer. The link to the repository is at: https://github.com/RickLeinecker/John-Cage-Tribute.git.

10 Milestones

- 01/25/2020 - First official meeting as a team
- 02/03/2020 - Established MongoDB database
- 02/12/2020 - Turned in Initial Design Document
- 02/24/2020 - Project Status with Dr. Leinecker
- 02/27/2020 - Meet with Dr. Thad Anderson for a better understanding of John Cage
- 03/02/2020 - Turned in Design Document for review
- 03/05/2020 - User Information now stored on database
- 03/08/2020 - Basic mockup of John Cage Tribute mobile app was made
- 03/09/2020 - 03/14/2020 - Spring Break
- 03/19/2020 - First audio mixer prototype finished
- 03/23/2020 - Dashboard established in mobile app
- 04/02/2020 - TA check in
- 04/09/2020 - Dashboard, Appointments, and Authentication screens established in mobile app
- 04/11/2020 - Initial John Cage Tribute repository created on GitHub
- 04/21/2020 - Finished first version of design document
- 04/23/2020 - Started integrating initial server audio processor with mobile app

- 04/28/2020 - 08/24/2020 - Summer Break
- 04/30/2020 - Established DigitalOcean Server
- 05/10/2020 - Composition metadata and MP3 files now store in database
- 05/20/2020 - Server can stream audio from 1 performer to 1 listener
- 06/01/2020 - MP3 files on database can now be listened to
- 07/04/2020 - Setup Nginx on backend server
- 07/14/2020 - Transition from scheduled Appointments to realtime Rooms entered full tow
- 07/23/2020 - Socket.IO behavior via rooms and sessions established on mobile app
- 07/26/2020 - Launched initial React application on server
- 08/01/2020 - Modified mobile audio plugins to 22kHz for decreased server pressure
- 08/06/2020 - Implemented composition search routes on backend server
- 09/23/2020 - Started using Web Audio API for audio processing
- 09/29/2020 - Gave practice presentation in front of class.
- 10/10/2020 - Finished integrating Web Audio API into audio processing on server
- 10/20/2020 - Encode session audio to MP3 file when recording session is over
- 10/30/2020 - Fixed audio crackling issue with Web Audio API
- 11/01/2020 - Finalized Transfer of MP3 files between server and database

- 11/03/2020 - Performed live demonstration of application for Dr. Leinecker.
- 11/10/2020 - Touched up on UI as per Dr. Leinecker's advice
- 11/20/2020 - Finished implementing multithreaded audio processing design
- 11/29/2020 - Recorded video for Senior Design Showcase
- 12/03/2020 - Gave Final Presentation on application

11 Future Plans

Even with how much we have done on this project, we still believe that this concept can be pushed so much further. This section will detail features that we would like to see if there would be a part two of this project. A few of these features are stretch goals that we never got around to adding, while others are ideas that we came up with towards the latter part of development.

One thing that we would love to see in a future version of this project is to collaborate with the UCF School of Music. Ultimately, we want this app to be used by musicians, so teaming up with the UCF School of Music would help with getting a deeper understanding of John Cage's works as well as ways to improve the user experience when it comes to making compositions.

11.1 Frontend Features

Mobile

The mobile app, serving as the nexus for the John Cage Tribute, has indefinite potential for new features to be implemented. Of the many, some were of great consideration and frequently talked about with the team. If the app were to load certain musical images or photos of John Cage in its background, it would serve as a considerable upgrade to the sea of blue that pervades the app. Additionally, talks of adding more animations to certain widgets throughout the app would grant it a boost of livelihood that is rather absent in the mobile application to date.

11.2 Backend Features

User Accounts

There are a few things that we would have liked to have added when creating user accounts. The first thing is email verification when creating accounts. As of right now, emails are used in order to provide a unique ID, but having user email verification would theoretically limit a user to how many accounts they could have. Another feature would be the ability to change your name and password after the account has been created.

Plugins

The plugins can perform any function as long as they conform to the protocol that the core system is expecting. In this system, they must inherit from the Web Audio API audio node interface.

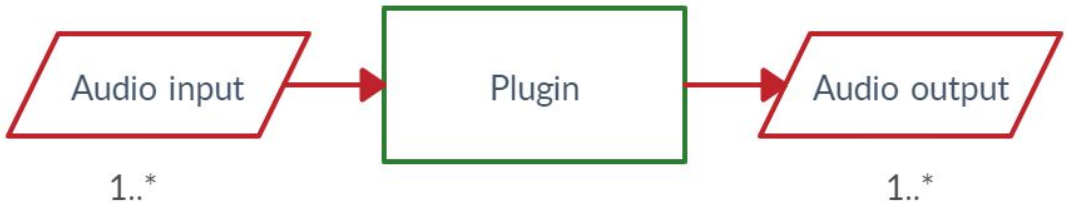


Figure 55: The general format for an audio plugin

Since they inherit from the Web Audio API audio node interface, plugins will be able to take one or more audio streams as input and must output the same number of streams. These plugins could be very simple or very complex performing both audio analysis or audio manipulation. They could be extensions of existing modules made to be used with Web Audio API or custom built. Any number of plugin audio nodes could be added to the overall audio graph of the system to apply audio effects or perform audio analysis.

Audio Modification

A potential improvement to the audio processor would be to modify the sounds recorded on the mobile app with audio effects. Right now the audio processor is built with Web Audio API, so effects that are compatible with this API such as tuna-js should be easy to implement. The goal would be to incorporate these effects as plugins automatically into the audio graph so they can be used in a randomized way. These effects could be chained together in random order and with random settings. These randomized effects when applied to the audio streams prior to mixing will allow us to create something unique and different every time. They will also make the output sound less like random noise and more like John Cage's organization of sound.

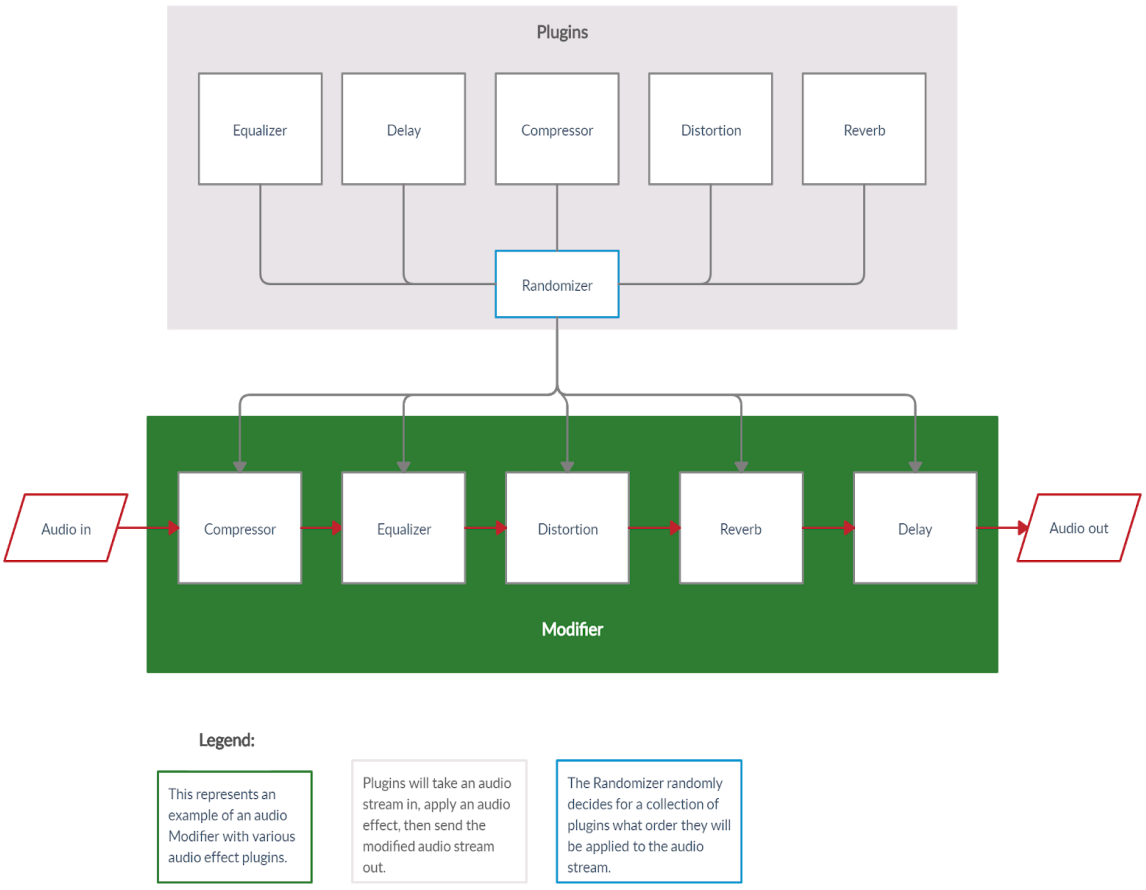


Figure 56: An audio modifier with 5 effects plugins

We can see in figure 56 that we can create a pool of plugins to select from for a given recording session. This will allow the audio processor to grow with the project as we add new effects to the pool. This also adheres to our microkernel architecture design, as new effects plugins can be added and utilized without changing the base function of the system. It is also possible to create custom configurations for the modifier where only certain effects are used. This could be tailored to a specific compositional style of John Cage's, or simply as a new unique way to organize the sounds.

Each room should be able to create its own audio graph with a randomly generated sequence of effects. The possibilities for completely different sounding tracks are nearly limitless and will increase as new effects and configurations are added. This is something that truly matches John Cage's ideals and reflects his musical works.

I Ching Audio Modification

The *I Ching,* I would like to believe, transcends logic. While researching and thinking of the randomization algorithm, a literary issue arose, each chapter can have multiple interpretations. For this reason, the idea to map out each question to each chapter in the *I Ching* manually would defeat the purpose of true randomization. There is the possibility of consulting the *I Ching* sometimes, and not on all of the recordings. However, I think this would make the book of changes insignificant. The problem is in the interpretation, as is with all problems in the world. The program needs to have a kind of way of just knowing whether to do something or not. The program has to be artificially intelligent in consulting the *I Ching.*

The first thing to do would be to come up with good questions to ask the *I Ching*. Find all the possible ways that the audio could be randomized, then use random number generators to answer each of those questions. Each question can have a range of intensity, or a range of indices that determine the intensity of the answer to the question. A random number generator can be used to choose an index in that range to determine the answer. This would be an alternative to consulting the *I Ching*. The implementation of the *I Ching* is still a work in progress…

12 Project Summary and Conclusion

Looking at the entire project as a whole, what we wish to do is to honor the late John Cage by creating a way for people to create their own John Cage inspired compositions by recording. Composers will be given the ability to create rooms so that performers can go out and record, and those recordings will combine and transform into a work that is structurally similar to one that John Cage might have composed. We ultimately hope that this product is to be used by actual musicians and composers all over the place in order to create performances and ensure that John Cage stays remembered for a long time.

Over the past year, the five members of the John Cage Tribute team have come together and have started work on something special. We have brainstormed ideas, begun research into areas that we have lacked knowledge in, and have begun laying the foundation to our tribute of the late John Cage. To our knowledge, what we are trying to do is novel and not really attempted on the scale that we are trying to achieve. We still have a lot of learning, designing, and creating left to do before the end of the year, and if things keep going the way they are right now, there is no doubt that we will be successful in contributing to the legacy of John Cage. This document has laid out not only the content of our design, but how our product has evolved over time, as well as each of our separate contributions to this product. It also lays out how, even though we have accomplished so much, there are still more ways to improve the John Cage Tribute.

In Zen they say: If something is boring after two minutes, try it for four. If still boring, try it for eight, sixteen, thirty-two, and so on. Eventually one discovers that it's not boring at all but very interesting. [2 (pg.93)]

There is no such thing as an empty space or an empty time. There is always something to see, something to hear. In fact, try as we may to make a silence, we cannot. For certain engineering purposes, it is desirable to have as silent a situation as possible. Such a room is called an anechoic chamber, its six walls made of special material, a room without echoes. I entered one at Harvard University several years ago and heard two sounds, one high and one low. When I described them to the engineer in charge, he informed me that the high one was my nervous system in operation, the low one my blood in circulation.
Until I die there will be sounds. And they will continue following my death. One need not fear about the future of music. [2 (pg.8)]

John Cage was here……..

11 References

[1] https://repository.duke.edu/dc/gedney?commit=Limit&q=john+cage&search_field=all_fields&utf8=%E2%9C%93

[2] Cage, John. *Silence: Lectures and Writings.* Wesleyan University press. 1961.

[3] https://cagecomp.home.xs4all.nl/chronology_1912-1971.html

[4] John Cage: The Roaring Silence: A Life

[5] https://web.archive.org/web/20070226123315/http://www.newalbion.com/artists/cagej/autobiog.html

[6] Conversing with Cage.

[7] http://journals.sfu.ca/cjbs/index.php/cjbs/article/viewFile/58/55

[8] https://quotepark.com/quotes/1755311-john-cage-when-i-hear-what-we-call-music-it-seems-to-me-tha/

[9] https://cagecomp.home.xs4all.nl/chronology_1972-1992.html

[10] https://commons.wikimedia.org/wiki/File:Merce_Cunningham_1961.png

[11] http://exhibitions.nypl.org/johncage/timeline

[12] https://thedailyhatch.org/2015/05/22/the-artists-poets-and-professors-of-black-mountain-college-the-college-featured-in-the-film-the-longest-ride-part-5-ceramic-artist-karen-karnes-and-her-husband-sculptor-david-weinrib/

[13] https://theplanthunter.com.au/culture/john-cage/

[14] https://www.britannica.com/topic/Sonatas-and-Interludes-for-Prepared-Piano

[15] The I Ching or book of changes, The Richard Wilhelm Translation. Rendered into English by Cary F. Baynes. Book 1.

[16] https://commons.wikimedia.org/wiki/File:Bagua-name-earlier.svg

[17] https://commons.wikimedia.org/wiki/File:King_Wen_Sequence_Pairs.jpg

[18] https://commons.wikimedia.org/wiki/File:Diagram_of_I_Ching_hexagrams_owned_by_Gottfried_Wilhelm_Leibniz,_1701.jpg

[19]https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/

Permissions:

https://library.duke.edu/about/reuse-attribution

https://commons.wikimedia.org/wiki/File:Merce_Cunningham_1961.png

https://www.nypl.org/help/about-nypl/legal-notices/website-terms-and-conditions
https://commons.wikimedia.org/wiki/File:Bagua-name-earlier.svg
https://commons.wikimedia.org/wiki/File:King_Wen_Sequence_Pairs.jpg
https://commons.wikimedia.org/wiki/File:Diagram_of_I_Ching_hexagrams_owned_by_Gottfried_Wilhelm_Leibniz,_1701.jpg