Audio Canvas: An Audio Visualization Tool

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin–La Crosse

La Crosse, Wisconsin

by

Christian Strauss

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

May, 2022

Audio Canvas: An Audio Visualization Tool

By Christian Strauss

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Kenny Hunt, PhD Examination Committee Chairperson	Date
Dr. William Petullo, PhD Examination Committee Member	Date
Dr. Allison Sauppé, PhD Examination Committee Member	Date

Abstract

Strauss, Christian, “Audio Canvas: An Audio Visualization Tool,” Master of Software Engineering, May 2022, (Kenny Hunt, Ph.D.).

This manuscript describes the development of web based audio visualization tool called “Audio Canvas.” Audio visualizers are created by having some form of audio data animate a visual object. Audio Canvas is designed to manage, edit, and share these audio visualization projects.

Acknowledgements

I would like to express my appreciation to my project advisor Dr. Kenny Hunt for his time, guidance, and support during the length of this project. I would also like thank the Department of Computer Science at the University of Wisconsin–La Crosse for providing the knowledge and experience needed to complete this project. I would lastly like to thank my wife and family for providing me with support and patience during my studies.

Table of Contents

List of Figures

List of Tables

Glossary

Amazon Web Services (AWS)

AWS is a cloud service provider that offers various on-demand cloud computing platforms.

BabylonJS

BabylonJS is a web 3D graphics engine that uses JavaScript to display graphics on HTML canvas elements.

FCurve

a FCurve is a function or curve that is used to define the animation of an object. This is done by setting points along a graph.

FeathersJS

FeathersJS is a backend web framework for NodeJS used to develop REST APIs.

Frames Per Second (FPS)

FPS is a computer graphics performance metric that captures the frequency at which images are displayed.

Frequency Spectrum

The frequency spectrum is audio data pertaining to the distribution of amplitudes over each frequency component (20hz-20kHz).

Heroku

Heroku is a platform cloud service provider that enables people to build and run applications in the cloud.

LaTeX

LaTeX is a document markup language and document preparation systems for the TeX typesetting program.

MediaRecorder

The MediaRecorder interface is a part of the web MediaStream Recording API that is used to record videos on the web.

Mesh Object

A Mesh Object is a 3D shape represented as a set of vertices and triangles.

MongoDB

MongoDB is a document-based NoSQL database system, capable of data persistence.

MongoDB Atlas

MongoDB Atlas is a cloud based database web service that fully manages the deployment, scaling, and maintenance of a MongoDB instance.

Netlify

Netlify is a cloud service provider that offers hosting of static web sites and serverless functions.

NodeJS

NodeJS is a backend JavaScript engine, capable of running JavaScript code outside of a browser.

OAuth

OAuth is a standardized authorization protocol used on the web for applications to grant access to information.

ReactJS

ReactJS is a frontend JavaScript library, capable of building modular, component-based user interfaces.

React Native

React Native is framework that utilizes ReactJS to build mobile applications on both Android and IOS in a single codebase.

REST API

A REST API is an application programming interface that conforms to the representational state transfer architectural style.

WebGL

WebGL is a web graphics API that enables JavaScript to run instructions on graphic processing hardware.

WebView

WebView is a React Native component used to display web pages inside mobile applications.

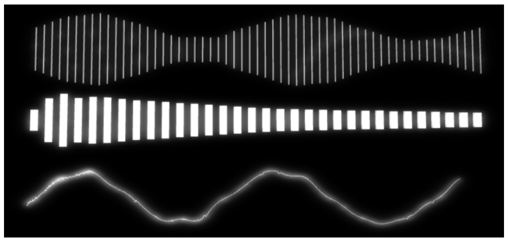
1. Introduction

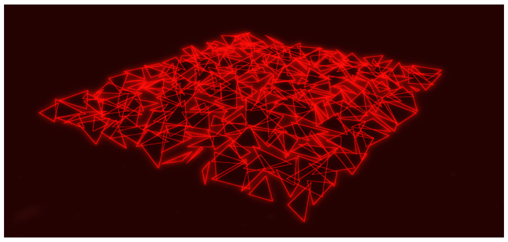
1.1. Overview

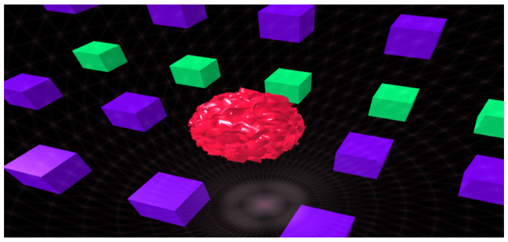
This project aims to create a software system that enables users to produce, manage, share, and export graphical visualizations of audio streams. Audio visualizations are computer generated animated graphics that are rendered in a way that synchronizes to audio. These visualizations can range from simple animated graphs to complex morphing objects that move over time. These visualizations are rendered in real time by taking a live audio source as an input and applying a set of user-generated rendering rules to produce a dynamic image that is affected by characteristics of the audio stream. Hence, the graphics can look drastically different depending on the audio fed into it. Applications of such visualizations include music videos, livestreams, podcasts, advertisements, and other forms of visual media. Figure 1 shows four examples of audio visualizations that demonstrate the variety of renderings available.

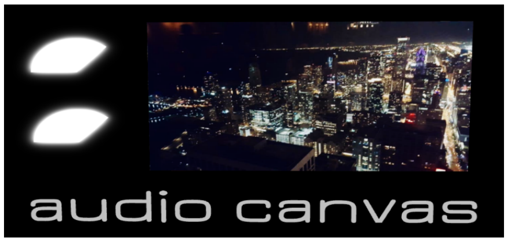
The visualizer supports rendering 2D and 3D geometric shapes, text, images, and patterns. Each graphical component has geometric characteristics that determine the position, rotation, and scale in either 2D or 3D space. Each component also has properties that affect the texture or material of the object. Users are thus able to control the color and texture of the animated visuals. Static characteristics are set to a constant value and remain fixed throughout the audio feed. Dynamic characteristics change with respect to the selected audio source or, alternatively, be programmatically changed over time.

The audio source can be an audio file or a microphone input. The audio stream's volume, frequency, and time domain signature can be connected to any of the various graphical properties that determine visualized rendering. These characteristics drive how the visualizer looks by changing a visual characteristic of the image. For example, we can extract the volume of an audio input and use it to determine the size of a visual component at a specific point in time.


(a) 2D Elements


(b) Mesh


(c) Objects



(d) Image and Text

Figure 1. Audio Visualizations

1.2. Background

A number of professional motion-effects software packages support the creating and management of audio-visual animations. Motion, Blender, and Adobe After Effects allow a user to create a multitude of different types of computer graphics including film making, 3D models, virtual reality, video games, and other use cases. In these tools, users are able to create mesh objects that are then animated through the use of FCurves. FCurves enable a user to map audio data to a particular mesh object property. FCurves accomplish this by specifying keyframes and interpolating the values between those keyframes over time. An audio stream can be baked to these FCurves to animate a property with respect to the audio stream provided [4]. Professional motion-effects software packages are computer resource intensive programs due to the way they render animations. Configuring audio-visuals in these tools can also be technical and require some expertise.

For users that do not have the computer resources or expertise necessary to make audio-visuals in professional motion effects software, applications such as Renderforest, Specterr, and STAELLA exist. These applications reduce the complexity of managing meshes and FCurves by providing preset animation routines. These presets already contain all the visual components and audio mappings necessary to create an audio visualizer. All the user needs to do is choose a preset visualizer and provide the audio source. These applications enable users to create audio-visuals for their content quickly and easily.

Professional motion-effects software packages have a steep learning curve, require hardware that is capable of extensive graphical processing, and are time consuming to use. Preset animation applications are easier to use and requires less capable hardware support but fails to provide the customizability that many users want. This project aims to strike a balance between these different methods of creating audio visualizations.

1.3. Goals

This software system is designed with simplicity in mind. An intuitive user interface should allow a user without expertise to create an image with motion-effects tied to music and/or speech.

This software system is also designed with portability in mind. This means making the system available on a wide range of computer devices. This system will allow a user to create an audio-visual project on one machine, save it, and pick up where they left off on another machine.

It is also important to note this software system is also designed with emphasis on collaboration and community building. This system will allow a user to connect with other users and groups, inspire one another, and create new audio-visuals. When starting a new creative project, learning from the examples of others can help from a creativity and technical perspective. This system will not only be capable of creating audio visualizations, but contain project management infrastructure needed to browse, share, view, and edit projects created in the system. A user will be able to share a project with a different user and allow that other user to work on the same project.

2. Software Development Process

2.1. Overview

I first decided to choose an appropriate software development life cycle model for project development. Software development life cycle models describe a series of phases of activities performed to construct a software system [6]. Each software development model has distinct characteristics that may or may not be suitable for a particular product. Several software development life cycle models were considered for this project. This section discusses these considerations and the decision to adopt a modified Scrum Model for this project.

2.2. Life Cycle Model Analysis

2.2.1. Waterfall Model

The Waterfall Model is a software development life cycle model that defines a straight sequence of activities [6]. Each phase must be complete before moving on to the next phase as subsequent phases are dependant on prior phases. This model starts with a requirement gathering phase where product requirements are captured. Once all requirements are captured and understood, the system is designed to meet those requirements. Once the design is fully understood, the system is then implemented. Once the system is fully implemented, it is tested. When testing is complete, the system is deployed and maintained.

The Waterfall Model was considered for use in this project but was not chosen due to its strict unidirectional flow through its phases. At project inception, I was unclear on various technical issues related to graphical rendering and these issues might cause me to rethink certain features. I was also unclear on the graphical performance I could expect from the various devices I was targeting for deployment. For instance, would the browser be capable of rendering graphics while simultaneously processing audio data? If so, what are the performance capabilities of such actions, and how will different hardware handle this? Since a proper architectural design depended to some degree on how these questions were to be answered, I required a more dynamic approach to software development.

2.2.2. Prototyping Model

The Prototyping Model is a software development life cycle model that uses the development of prototypes to better acquire and validate requirements [6]. This model is useful when product requirements are not fully known or understood. This model begins with an initial product requirement gathering phase, but does not require the product requirements to be fully understood at this point. Once that initial set of product requirements are known, the product is broken down into subsystems.

Prototypes are then built for each piece and evaluated by the customer. This process of prototype development followed by customer evaluation is repeated throughout the project until the customer is fully satisfied with all partial prototypes. This gives the customer the opportunity to further refine product requirements. Once the customer is satisfied, the product requirements are now fully known. The different prototypes are then integrated into the final product where it can be tested, deployed, and maintained.

The Prototyping Model was also considered for this project but was not chosen due to its customer evaluation and acceptance steps. The prototyping process would have been beneficial for this project to uncover some of the previously stated unknowns and fully capture project requirements. However, the customer evaluation step of this model is unhelpful for this project since the customer and developer are the same person. This model also requires customer acceptance before product integration begins. This could potentially introduce scope creep and extend product development. As the customer of the product, it would be difficult to determine at what point the prototypes are sufficient to progress. If for whatever reason I was not fully satisfied with a particular prototype, I could spend more time on its details rather then higher priority items. I could of recruited customers for testing purposes to make this model work, but opted for the Scrum model as it seemed to be a better project fit.

2.2.3. Spiral Model

The Spiral Model is a software development life cycle model that is known for its unique risk management feature [6]. This model incrementally builds product features in iterations known as spirals. Each spiral consists of four steps: identify objectives, risk analysis, development, and evaluation. When identifying objectives, precisely what is to be developed during the iteration is identified and understood. After the objectives have been identified, an in-depth risk analysis is performed on them. Risk analysis involves finding the potential risks of adding the objectives to the software system and mitigating them appropriately. This process includes the development of prototypes to better understand what risks exist and troubleshoot solutions. After risk analysis is complete, the objectives are developed and tested. Finally, the objectives are evaluated by the customer. This step gives the customer the chance to refine product requirements. The Spiral Model is very thorough in its approach to software engineering. This effort comes at a large cost in time. The risk assessment featured in the model is suitable for large scale, risk averse applications.

The Spiral Model was also considered for this project but was not chosen due to its cost. This risk analysis would have been beneficial for this project to determine and mitigate some risks of adding product features. For instance, an in depth analysis of how adding a particular feature would affect the overall performance of rendering process could of been helpful. However, this risk analysis can be unpredictable in regards to time estimation. Time management is important for this project to meet the project deadline. This project, being smaller in scale, can take on a little more risk building product features to improve throughput. The evaluation step of the spiral offers a check point with the customer to ensure the product is being developed appropriately. This step is not appropriate for this project as well due to what was discussed in the Prototyping Model consideration.

2.2.4. Scrum Model

The Scrum Model is a software development life cycle model that uses an agile approach for developing innovative products. The efforts are divided into three different roles: product owner, scrum master, and development team. The product owner is responsible for what is developed and in what order. The scrum master is responsible for guiding the team in following the scrum model. The development team is responsible for designing, building, and testing the product [9].

The Scrum Model defines several activities and artifacts. An initial set of project requirements are gathered and added to a product backlog. The product is then built in iterations known as sprints. Each sprint begins with sprint planning. During planning, a set of tasks are moved from the product backlog to the sprint backlog. The sprint is then executed. The sprint backlog tasks are designed, built, and tested at this time. Each day, a stand-up meeting takes place to help organize the team and provide status updates. At the end of the sprint, a sprint review with the customer takes place to evaluate what happened during the sprint. This offers a time for the customer to voice opinions about the work completed and future enhancements. It is ultimately up to the product owner to decide whether these opinions make it into the product requirements and their urgency. It is also up to the product owner to decided if the complete sprint work is releasable as is. This is an important distinction to that of Prototyping and the Spiral Model. The customer can not directly impact the release of increments. They can only influence future work and voice concerns. A sprint retrospective also takes place to evaluate what is going well and what needs improvement from a process standpoint. At the end of the sprint, the work completed is released as an increment. The last process of a sprint is product backlog grooming. This is where tasks are added, removed, and prioritized for upcoming sprints. These iterations are completed throughout the life of the product's development.

A modified Scrum Model was used for this project due to its incremental dynamic nature. This project required flexibility in regards to product requirements. The sprint review addressed this need by offering a time to revise those requirements. This model's incremental approach does this without sacrificing the ability to progressively release completed functionality. This project also required a model that was punctual in regards to time management. The ability to pull more or less tasks into each sprint proved to be necessary to complete the project in its given time frame. I have some prior experience with Agile software development which also played a role in the decision to follow this model. Dr. Kenny Hunt, the project advisor, played the role of ScrumMaster by ensuring scrum processes and artifacts were followed. I played the roles of the product owner, ScrumMaster, and developer. As such, minor adjustments to this model took place to accommodate this. For instance, there were no formal daily-standup meetings. The point of these meetings is to ensure the team is all on the same page with regards to the sprint work at hand. My project advisor and I also assumed the role of the customer for this project. As such, the sprint review served as a time to reflect on completed work and determine if it was fully capturing the goals for this project.

2.3. Overview of the Development Process

As the product owner, I gathered the initial set of product requirements. These requirements met my initial vision for the product and were captured as user stories. User stories are informal descriptions of requirements written from the perspective of the end user. Each of these user stories was also associated with a list of acceptance criteria along with an index to support traceability. These user stories served as a good starting place to help better understand what this software system was trying to achieve. Table 1 shows an example of how project management requirements were captured. A complete listing of all project requirements can be found in Appendix A.

Index	**2.1 — Create Project**
User Story	As a user, I would like to be able to create a project, so I can save my visualizer for future reference.
Acceptance Criteria	User provides project information – project is created User provides illegitimate information – Error message Notes: Project information - (name, visibility – public or private)
Index	**2.2 — Open Project**
User Story	As a user, I would like to be able to open a project, so I can view the visualizer.
Acceptance Criteria	User clicks open – project is opened
Index	**2.3 — Edit Project**
User Story	As a user, I would like to be able to edit a project, so I can change the name or visibility.
Acceptance Criteria	User edits project name or visibility – project is updated
Index	**2.4 — Delete Project**
User Story	As a user, I would like to be able to delete a project, so I can remove it from my list.
Acceptance Criteria	User clicks delete – project is deleted
Index	**2.5 — Share Project**
User Story	As a user, I would like to share my project with another user, so we can collaborate on the same project or enable the other user to view a private project.

Acceptance Criteria	User can enter in another user's email address to share "User not found" error message if user with email does not exist User can specify if other user has read or write privileges to the project Other user can view the project if they have read privileges. Other user can edit the project if they have write privileges. Notes: Project information - (name, visibility – public or private)

Table 1. Project Management Requirements

One of the key requirements of this system is the ability to manage audio-visual projects. As a result, a user needs to be able to create, view, edit, and delete projects. This system must also support collaboration, such that users must be able to share an audio-visual with one another. Also, a user needs to be able to determine whether the project is visible to all users or if only specified users can access it.

In order to make an audio-visual, the audio sources must be specified. The system therefore needs to enable users to manage their audio sources by creating, viewing, editing, and deleting audio inputs. The system needs to support both audio files and microphones as audio inputs. For audio file inputs, the system needs to support basic audio controls: play/pause and seek functionality.

The system needs to support creating and customizing graphs that map to data points collected from an audio input. The system needs to be able to create and customize a line chart: typically used for time domain waveform information. The system also needs to be able to create and customize a bar chart: typically used to illustrate the frequency spectrum.

The system needs to support maintaining mesh objects: cube, sphere, polyhedron, etc. As a result, a user needs to be able to create, edit, and delete these mesh objects. These mesh object primitives need to be able to be customized with relevant properties: width, height, material, etc. These properties also need to be able to be manipulated by audio data. These primitives need to be able to be grouped together to make increasingly more complex objects. These objects also need to be able to perform basic geometric transforms: position, rotation, and scale.

Any sort of video media would not be complete without images and text. The system needs to support adding and removing images and text from the visual.

It makes sense to be able to export the visual to a video. The system needs to support the ability to record the visual and export it to a file.

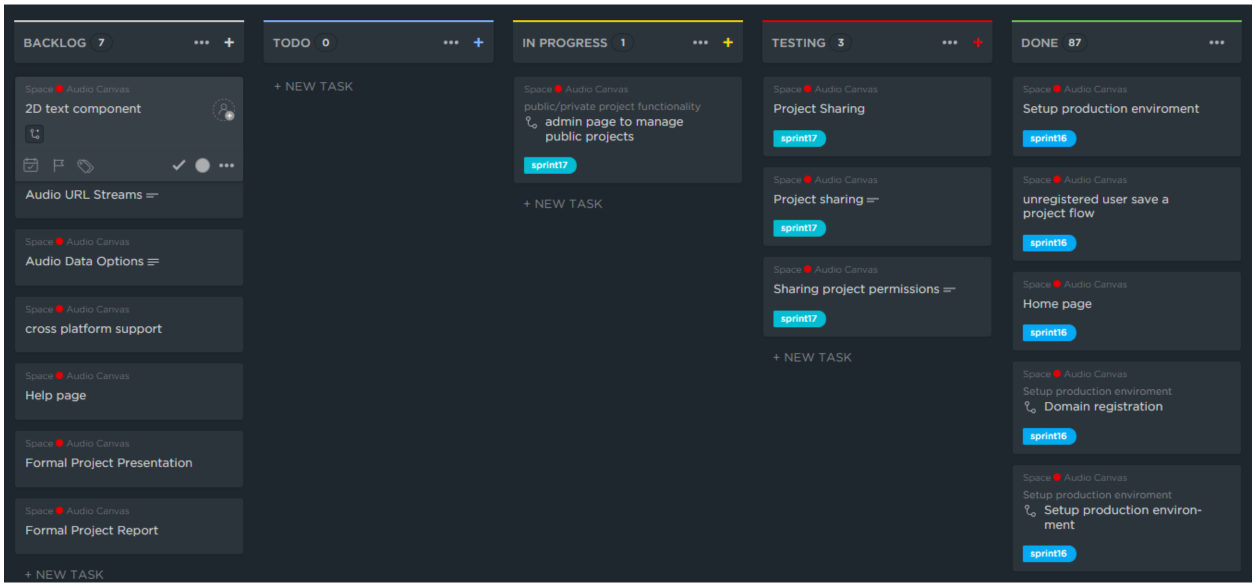
The product backlog is an emergent, ordered list of what is needed to improve the product [10]. This list was populated prior to the initial sprint with the initial product requirements. As the project was developed, new requirements were added to the project backlog. This occurred at the end of each sprint during product backlog grooming. This entailed adding, removing, and reordering tasks in order of priority. As the product owner, I determined the priority of each task. Emergent, critical product features were given higher priority. For example, tasks that involved in the management of users and projects were completed first. Then priority shifted towards creating the interface responsible for generating audio visualizations. More details on how the project work was prioritized will be discussed in

Section 4.

Project work was developed in iterations known as sprints. There were eighteen sprints in total to complete this project. Each sprint lasted two weeks with one exception. Sprint fourteen landed on J-Term and lasted a month. Sprints began with sprint planning and ended with a review and retrospective.

Sprint planning lays out the work to be performed for the upcoming sprint by assigning a set of tasks to the spring backlog [10]. As the development team, I needed to determine what tasks to pull. For this, I had to set a goal for the sprint based on the priority of the product backlog. For example, a goal of sprint ten was to add basic geometric shapes to the visualizer, and relevant tasks were selected to achieve this goal.

All project work was documented using a project management application known as ClickUp. I used the Kanban Board feature in ClickUp to manage the product and sprint-level backlogs. A Kanban Board is an agile project management tool designed to help visualize work [1]. Kanban Boards are made up of columns that indicate a status. Each backlog item is displayed on the board and has a status that determines what column contains that item. As shown in Figure 2, our Kanban board had five statuses: BACKLOG, TODO, IN PROGRESS, TESTING, and DONE. The BACKLOG column contained the prioritized product backlog. During sprint planning, the sprint backlog was pulled into the TODO column. As tasks were worked on, they were pulled into the IN PROGRESS column. When tasks were completed and ready for testing, they were pulled into the TESTING column. At this point, tasks were tested and evaluated. Once satisfied, the task was pulled into the DONE column. For documentation purposes, each task was assigned a label denoting what sprint they were completed in. Tasks only moved from left to right on this board, with the exception of the IN PROGRESS and TESTING columns. If testing failed, tasks were allowed to move back into the IN PROGRESS column. If new features or bug fixes were added to future sprints, tasks were created and added to the BACKLOG.


BACKLOG 7
- Space • Audio Canvas — 2D text component
- Audio URL Streams
- Space • Audio Canvas — Audio Data Options
- Space • Audio Canvas — cross platform support
- Space • Audio Canvas — Help page
- Space • Audio Canvas — Formal Project Presentation
- Space • Audio Canvas — Formal Project Report
- + NEW TASK

TODO 0
- + NEW TASK

IN PROGRESS 1
- Space • Audio Canvas — public/private project functionality — admin page to manage public projects — sprint17
- + NEW TASK

TESTING 3
- Space • Audio Canvas — Project Sharing — sprint17
- Space • Audio Canvas — Project sharing — sprint17
- Space • Audio Canvas — Sharing project permissions — sprint17
- + NEW TASK

DONE 87
- Space • Audio Canvas — Setup production enviroment — sprint16
- Space • Audio Canvas — unregistered user save a project flow — sprint16
- Space • Audio Canvas — Home page — sprint16
- Space • Audio Canvas — Setup production enviroment — Domain registration — sprint16
- Space • Audio Canvas — Setup production enviroment — Setup production environment — sprint16

Figure 2. Kanban Board

At the end of each sprint, a review took place to evaluate the work completed during the sprint, collect feedback, and prioritize tasks. The scrum team and any internal or external stakeholders typically participates in the sprint review. For this project, my project advisor and I participated in the sprint reviews. The retrospective offered valuable time to reflect on what was working well and what needed improvement. All reviews and retrospectives were documented and saved for future reference. Table 2 shows an example sprint review and retrospective from sprint four.

Sprint 4 Review and Retrospective	
What Happened?	Basic functionality of the "My Projects" page was completed. A basic user can view their projects, create, edit, and delete. A basic user is also able to duplicate a project. They are also able to launch any project in the main app. There was also a button added to share a project which will be implemented in a future sprint. Discovered a new feature in ClickUp application (used to track sprint progress) for tagging sprint tasks. Each sprint task as they are pulled into a sprint will be tagged with that sprint ex) sprint1. This way we can track which items came through in each sprint.
What went well?	My projects page functionality came together well. ClickUp tagging will be useful for tracking which tasks were completed in each sprint.
What needs improvement?	Sprint task naming needs to be more descriptive (thinking about keeping the title the same to fit on the board but adding a more descriptive description when someone opens the task).
Action Items	Main app menu GUI (left hand panel) Audio input module Audio input controller Audio input type setting in application settings
Notes	Paginate user management / bootstrap container User management is active should be checkbox

Table 2. Sprint Retrospective Example

During the sprint review, the work completed during the sprint was evaluated. Any work completed was released as an increment if it met the definition of done. The definition of done is a checklist of work that the team is expected to complete before work can be marked as releasable [9]. This list ensures the requirements pass all the necessary quality and completeness expectations. Code quality, acceptance criteria, and traditional nonfunctional requirements are expressed in this definition. The sprint review also offered time to make changes product requirements. During review, it was not uncommon to hear the phrase "it would be cool if ..." and for that cool new feature to be added to the backlog. For example, multiple audio inputs were not a part of the original functional requirements for this project. During one of the reviews, it was brought up that it would be a good idea to be able to specify multiple inputs and map them to different visual objects. This would be useful for podcasts when you want a different visual for each person talking. Making changes to functional requirements like this would not have been as easily supported through the use of other software development models. The sprint review also served as a checkpoint on where the project was at in its entirety. This meant reviewing the progress and timeline of the project as well as revising and prioritizing the product backlog.

Sprint retrospectives provided a time to reflect on the development process and allow for changes to improve quality and effectiveness [10]. This meant asking questions like - What went well in the sprint? What could be improved? What are some action items for future sprints? It is the responsibility of the scrum master to encourage these discussion points and ensure actions are taken to improve upon what is discussed. For example, In Table 2 the sprint task naming was something to be improved upon in future sprints.

3. Design

3.1. Overview

This section describes the application design. The goal of this effort was to take application requirements and transform them into appropriate data structures, modules, interfaces, and relationships to achieve those requirements. This section describes the UML Class Diagrams, UML Interface Diagrams, Database Entity-Relationship Diagrams, and User Interface Mockups that were used to achieve this goal.

3.2. UML Class Diagram

The UML class diagrams shown in Figures 3 and 4 describe the structure of the system by modeling object-oriented classes and their relationships with one another. The most important classes in this system are Users and Projects. This subsection discusses these two classes in more detail.

The user class is the entity that interacts with the system. The user uses the system to manage audio-visual projects. There are two types of users denoted by the userRole attribute. Admins can manage both users and projects. If a user or project needs to be edited, deactivated, or reactivated, admins can do so. Basic users can manage their own projects.

The Project class models a audio-visual project and all the metadata associated with it. Projects contain the ProjectAccess class which determines who has access to the particular project and their permissions associated with it. Projects also contain the ProjectData class which holds the data used to generate the audio-visual. In order to generate an audio-visual, several pieces of data are needed. ProjectData contains an Input class which models the audio inputs used in the audio-visual. ProjectData also contains the SharedData class which maintains some global properties needed to generate the audio-visual. The visual primitives used to generate the visualizer are also contained in the ProjectData class. The Component class models the different visual components that make up the visualizer. There are several subclasses of the Component class which describe the visual primitives available.

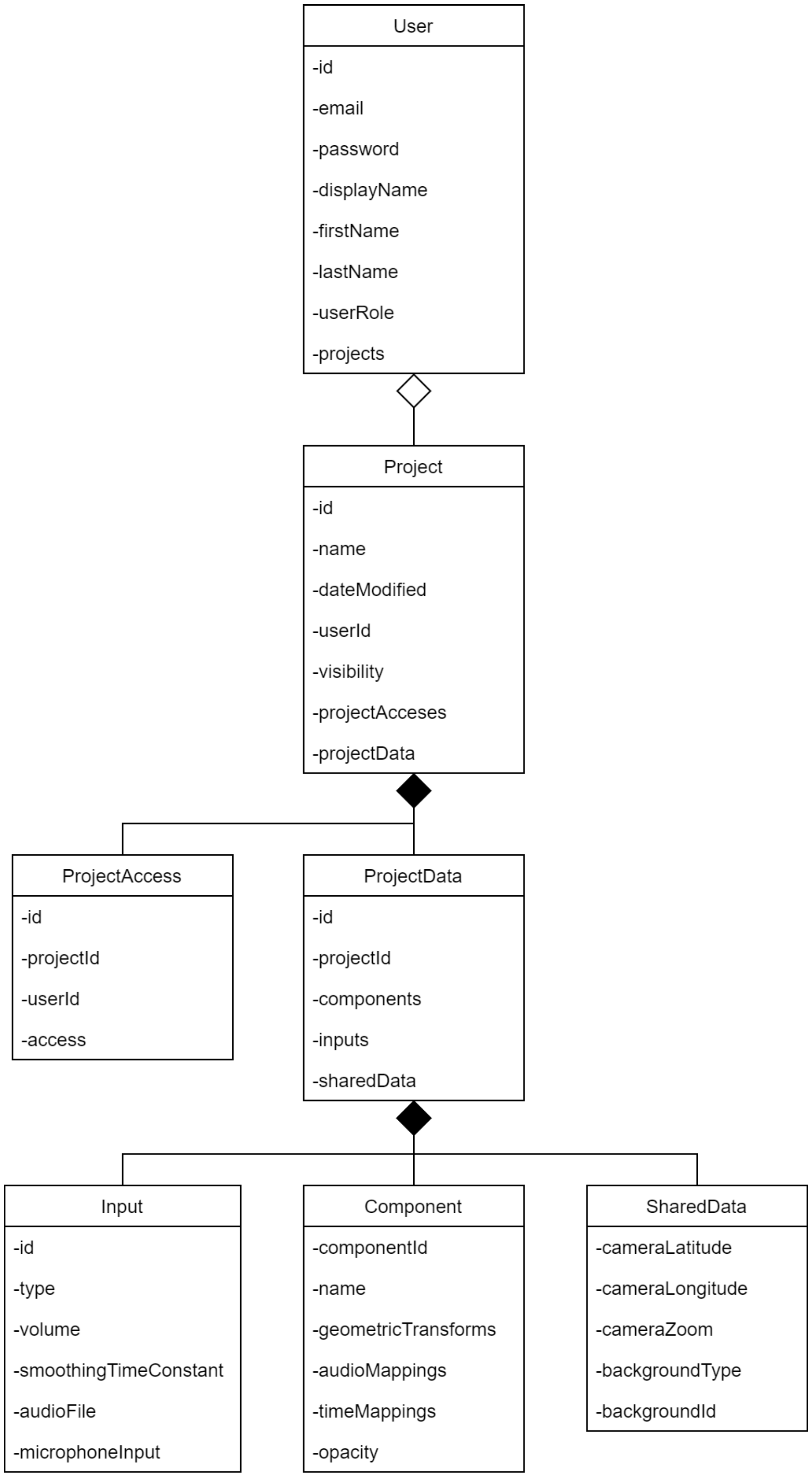


Figure 3. UML Class Diagram

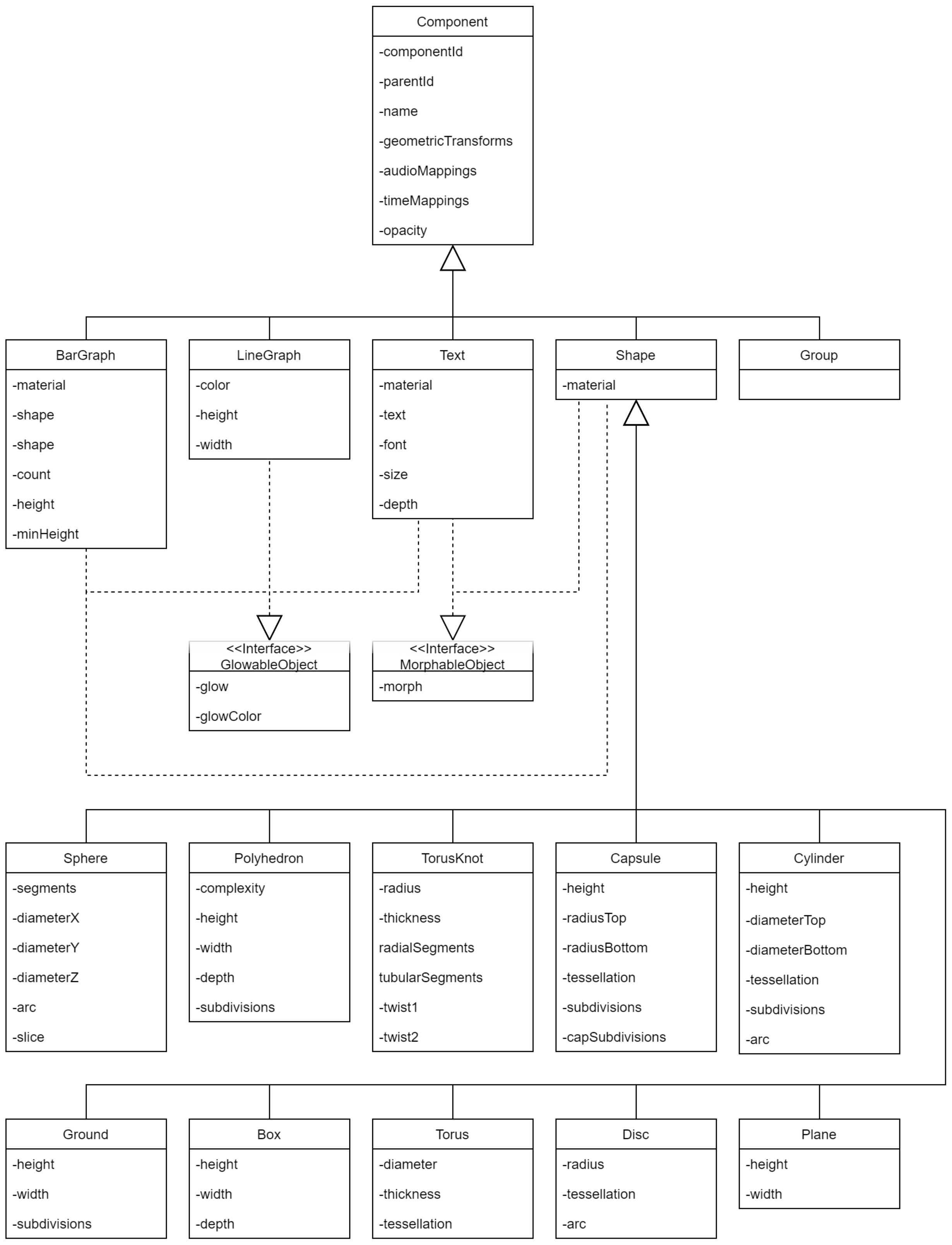


Figure 4. UML Class Diagram Continued

3.3. Application Interface

The UML Interface Diagram shown in Figure 5 describes the components, operations, and interactions in the application portion of the system. This interface is responsible for generating an audio-visual. The user interacts with several different editors to achieve this.

In order to make an audio-visual, the user needs to be able to add visual primitives to the project. This is done using the LayerEditor class and the ComponentEditor class. The LayerEditor class models the interface used to manage the different layers of the visual and their elements. This class is also responsible for the order of layers and how they are rendered in relation to one another. The ComponentEditor class models the interface used to manage a particular visual primitive. As such, the ComponentEditor class will allow the user to change various properties about a visual primitive. The Component class models the visual primitives to be generated in the visual.

In order to make an audio-visual, the user needs to be able to add audio inputs to the project. The AudioInputsEditor class models the interface used to manage the audio inputs. The AudioPlayer class models the interface used to manage a particular audio input. The AudioInput class models the audio inputs used in the visual.

In order to make an audio-visual, the user needs to be able to manage basic settings associated with the project. The VisualSettingEditor class models the interface used to manage global properties. The VisualSetting class models these global properties. The AppSettingsEditor class models the interface used to manage application level settings. The AppSetting class models these application level settings.

The rendering of the audio-visual happens in the Canvas class. This class takes in the components, audio inputs, and settings.

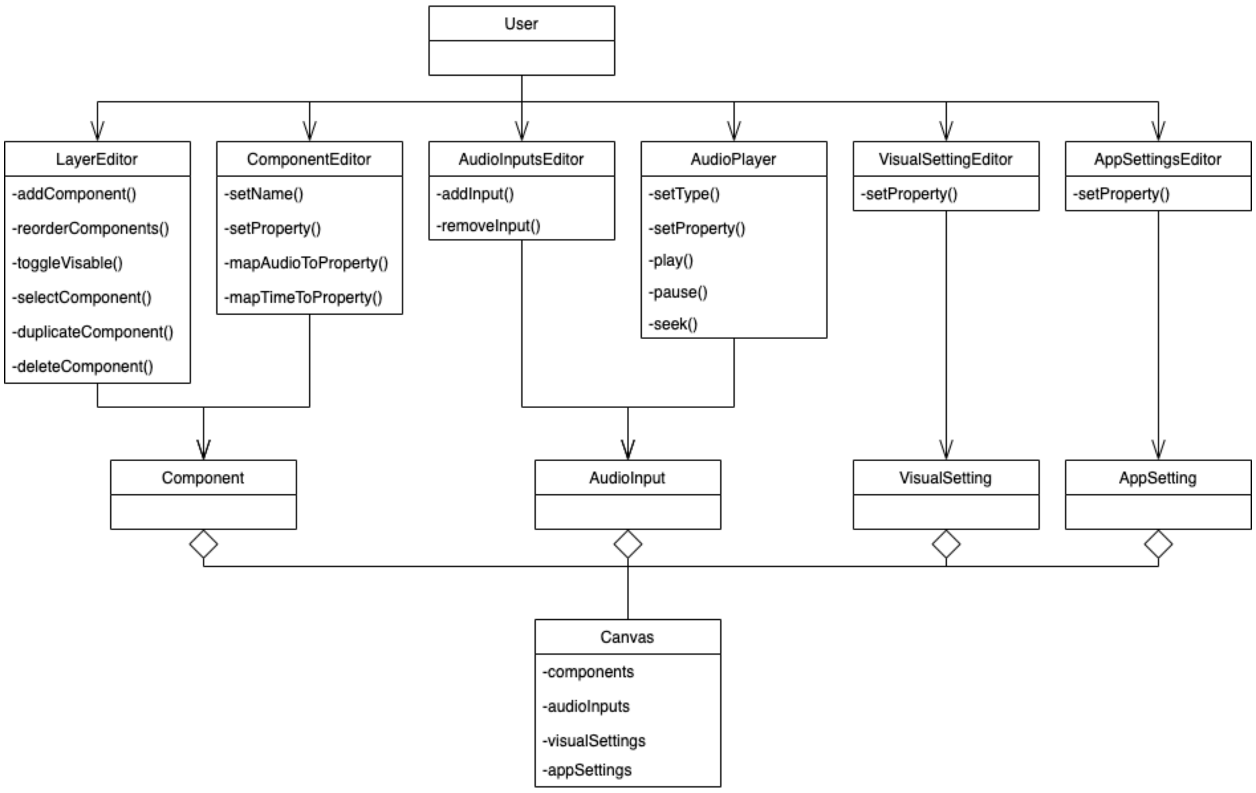


Figure 5. UML Interface Diagram

3.4. Database

The Database Entity-Relationship Diagram shown in Figure 6 models the different objects and their relationships with one another from a database design perspective. Similar to the UML Class Diagram section, the central entities are users and projects.

The user entity houses all the data that is important to store in regards to people interacting with the system. The email and password are used for authenticated purposes. The user's firstName, lastName, and displayName are user friendly names used to describe who the user is to other users in the system. The userRole property is used to describe what the user's role is in the system: admin or basic. Users have one relationship with the project class: a user manages zero or many projects.

The project entity houses all the important data related to an audio-visual project. The name property is a user defined friendly name associated with the project. The dateModified property states when the project was lasted edited. The visibility property describes whither this project is public, for anyone can view, or private, only the owner can view. The project has one relationship with the projectData class: a project has projectData. The projectData entity houses all the data needed to generated the audio-visual in the system: components, inputs, and shared data. The project also has one relationship with projectAccess: a project may have zero or more projectAccesses. The projectAccess entity houses all the data needed to share a project with another user including the permissions of the access.

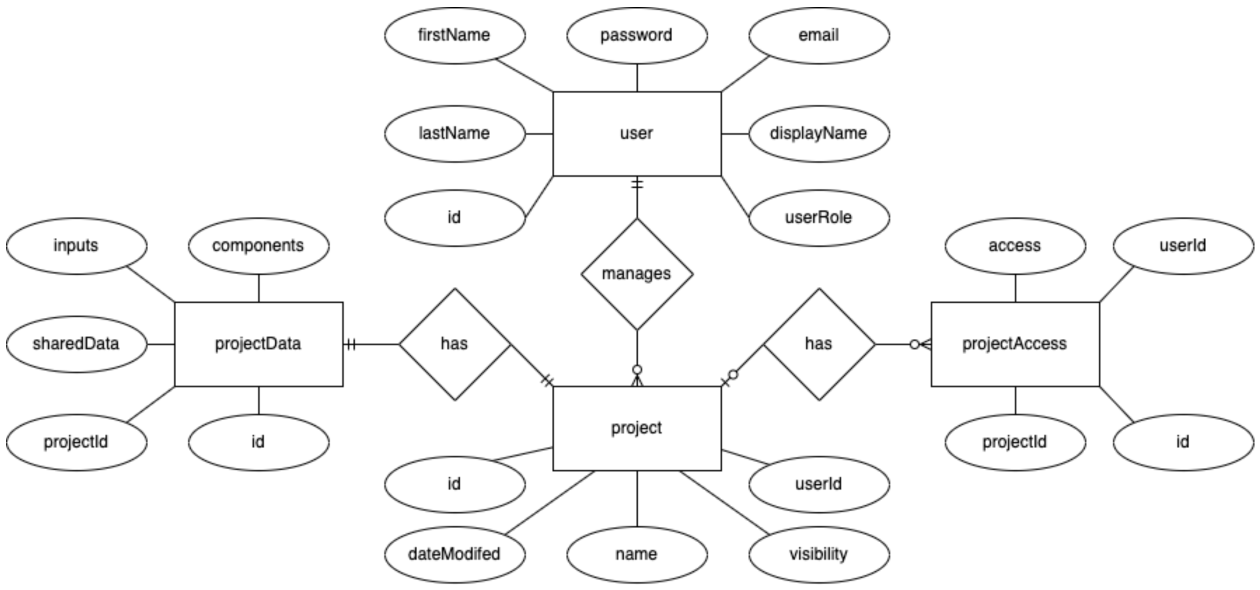


Figure 6. ER Diagram

3.5. User Interface Mockups

One of the important steps of the design process was designing the look and feel of the application. User interface mockups were developed for each page of the application in Figma, an online tool for creating UI mockups. These wireframe mockups helped guide the development of the client application and sparked ideas on how to make the interface more intuitive and user friendly. These mockups helped discover what web pages were needed and how the navigation between them would work. These mockups also helped guide the usability of the application's menu system: the project level configuration menu and the component level configuration menu. These menus were designed to open and close such that the user could view the audio-visual in its entirety. Usability is one of the main cornerstones of this project per the requirements, so it was important to have the application's user interface designed well. After these mockups were developed, they were presented to peers to ensure they were going to enable the end user to build audio visualizations in an intuitive way. These mockups were developed in the initial stages of the project development during sprint one.

The application mockup of Figure 7 shows a rough sketch of what I initially designed the user interface for the application portion of the project to look like. The menu on the left hand side of the image shows the project level configuration editor. This editor has the inputs necessary to specify the visual layers, the audio inputs, and global settings. The menu on the right hand side of the image shows the component editor which has all the component specific inputs necessary to customize a particular visual layer. For example, if a user were configuring a cube, they would be presented with inputs for width, height, depth, color, and a variety of other cube-related properties. In the center of the screen, where the Audio Canvas logo is present, would contain the live video playback of the visual being worked on. At the bottom of the image, if a user added an audio file as an input to the visualizer, a audio controller would be present to play/pause and seek through the file.

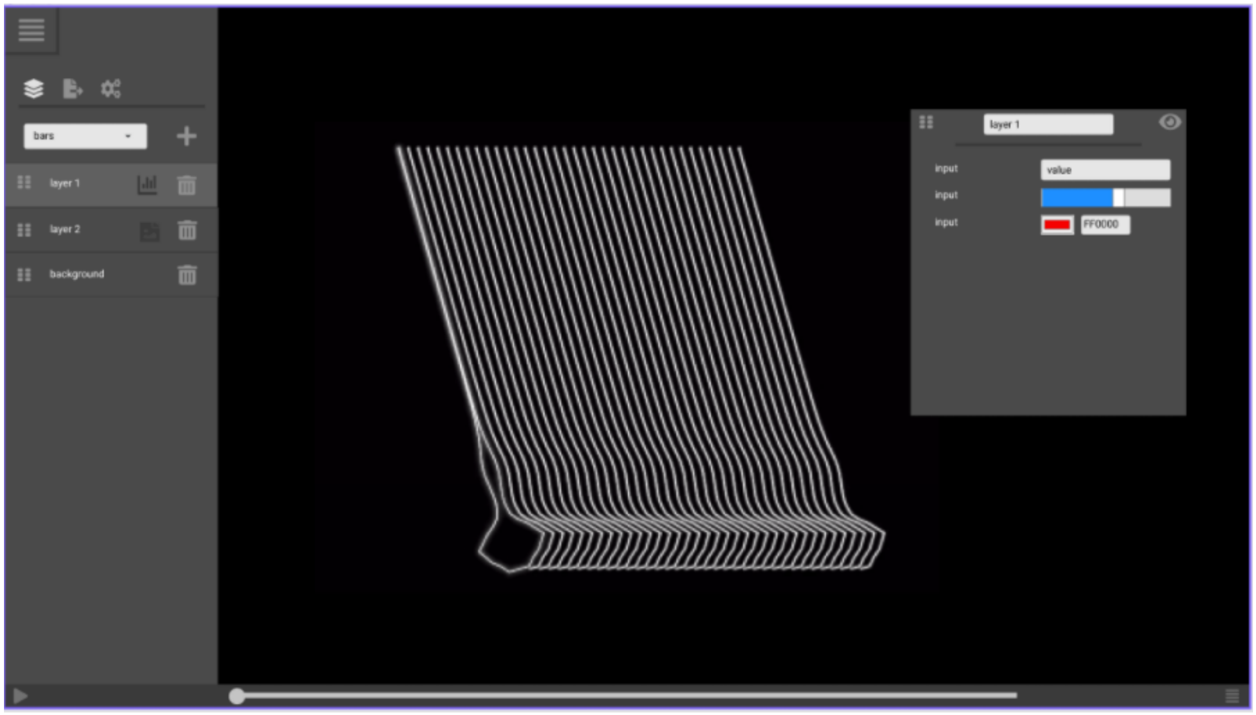
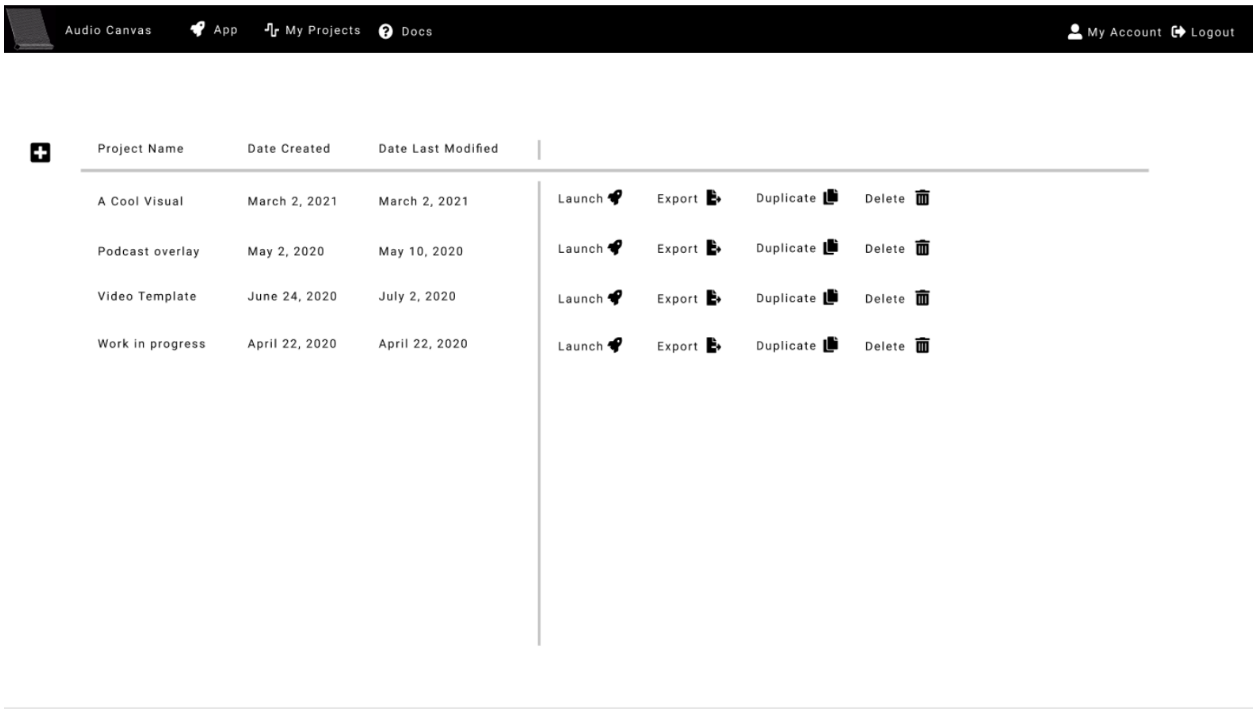


Figure 7. User Interface UI Mockup

The mockup of Figure 8 shows a rough sketch of what the user interface for the project management portion of the project would look like. This page was designed to allow the user to browse, create, edit, and delete their projects. The left side of the table gives a brief overview of the project. The right side of the table presents the operations the user can perform on the project. The launch operation opens the project up for viewing or editing in the application. The export operation exports the project to a file for sharing. The duplicate operation, duplicates the project. The delete operation removes the project from the system.



Audio Canvas | App | My Projects | Docs | My Account | Logout

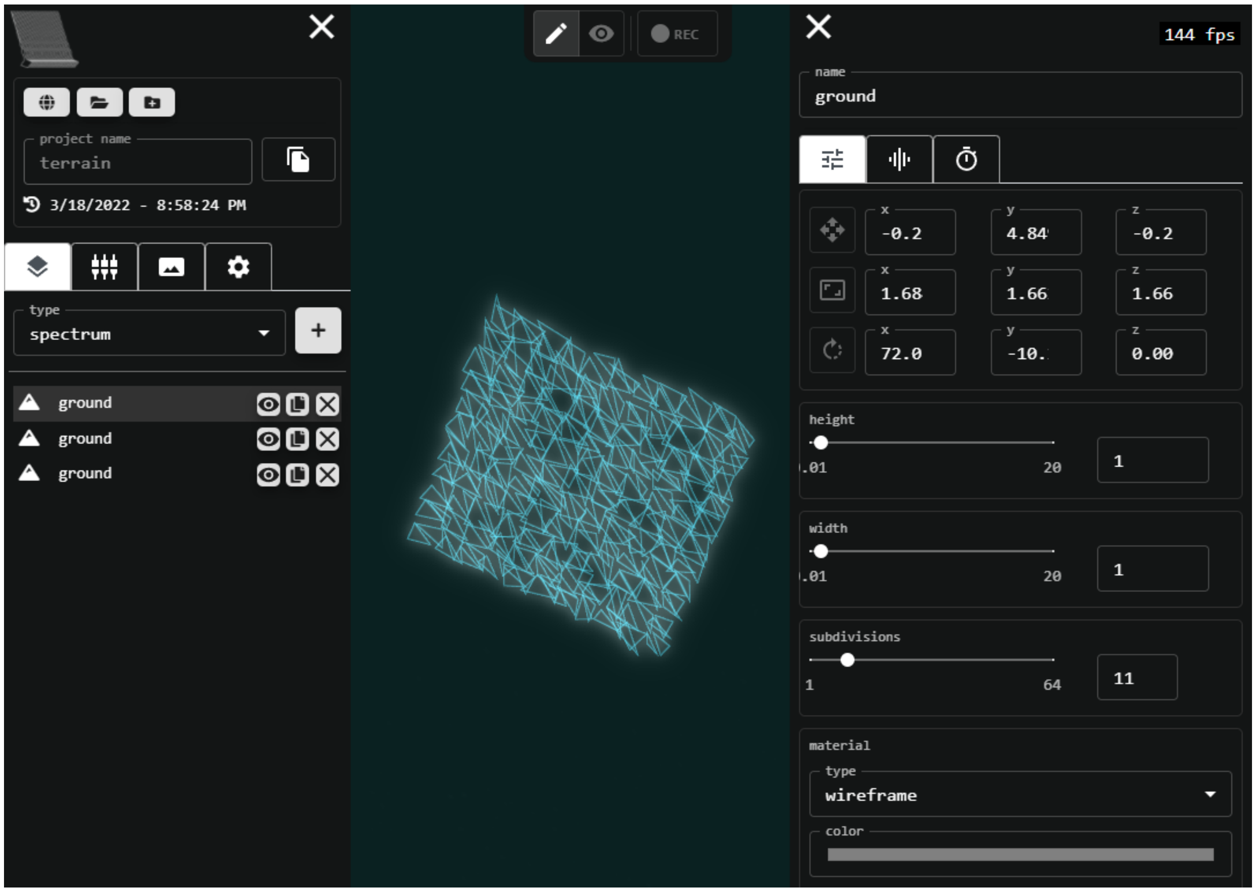
Project Name	Date Created	Date Last Modified				
A Cool Visual	March 2, 2021	March 2, 2021	Launch	Export	Duplicate	Delete
Podcast overlay	May 2, 2020	May 10, 2020	Launch	Export	Duplicate	Delete
Video Template	June 24, 2020	July 2, 2020	Launch	Export	Duplicate	Delete
Work in progress	April 22, 2020	April 22, 2020	Launch	Export	Duplicate	Delete

Figure 8. Project Management UI Mockup

3.6. Final User Interface

The final user interface used in this system was based on the initial mockups but provides many additional features that were discovered throughout the project development phase. Figures showing the final user interface can be found in Appendix C.

Figure 9 shows what the user interface for the application looked like after completion. This UI went through drastic changes to accommodate different product feature additions and changes. However, the presence of the two main editors remained constant - one for project level editing, and one for layer level editing. The project level editor added tabs for audio input management and background/camera management. The ability to add multiple audio inputs was not a part of the original project requirements and was added during one of the sprint reviews. The audio controller at the bottom of the screen in the mockup was moved to the audio input tab to accommodate this change. The background/camera management tab was something that was not originally thought of when the mockup was developed. The ability to control the background and camera such as position, rotation, and zoom of the visualizer was needed. As for the component editor, the various types of inputs to control the component remained constant. However, the presence of audio and time mapping tabs were added. Controlling audio mapping and time mapping was originally going to be controlled in one view with the component inputs. This functionality was separated out into different tabs for usability. When the functionality was being developed in one view, the UI begin to look and feel cluttered. The last addition to the application UI was the presence of the view editor at the top of the screen. This editor was added for usability to easily hide all editors and view the visualizer. This editor also housed the functionality to record and export the audio-visual as a video file.



project name: terrain

3/18/2022 - 8:58:24 PM

type: spectrum

ground
ground
ground

name: ground

x -0.2 y 4.84 z -0.2
x 1.68 y 1.66 z 1.66
x 72.0 y -10. z 0.00

height .01 20 1

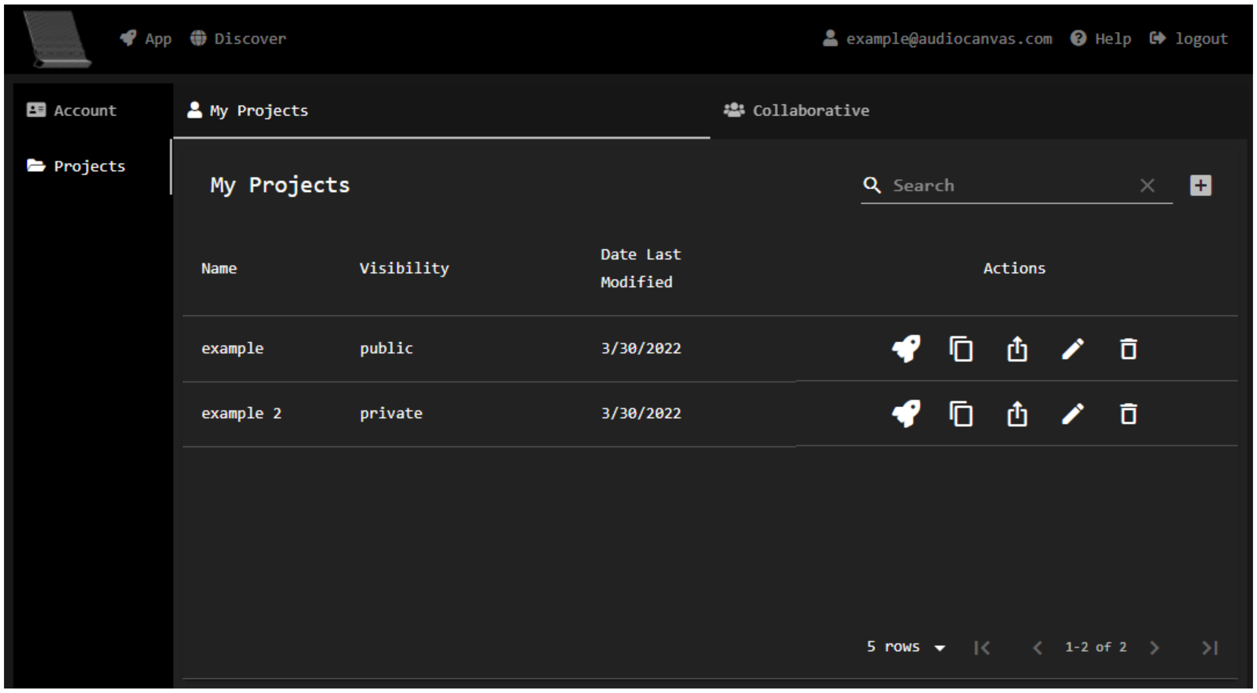
width .01 20 1

subdivisions 1 64 11

material
type: wireframe
color

Figure 9. User Interface UI Mockup

Figure 10 shows what the user interface for project management looked like after completion. The main differences between the the final UI and the original mockup is the absence of an export feature in the mockup, and an introduction of project sharing and visibility. Project sharing is the ability to share a project with another user. This feature was originally going to be accomplished by enabling projects to be exported as a file. This would allow users to send that project file to another user, and the other user could import the project into the system. During one of the sprint reviews, it was decided that keeping project sharing workflow internal to the system would make more sense from a usability perspective. This was accomplished by enabling users to control who has access to their projects. Users can look up another user in the system and grant read/write access to their project. This also enabled the functionality of collaboration: the ability for multiple user to contribute to the same audio visualization project. The visibility feature was another addition to the system that enabled users to specify whither the project was public or private. Other then these two changes, most of the UI remained similar with some slight adjustment to styling to match the rest of the application UI.



Name	Visibility	Date Last Modified	Actions
example	public	3/30/2022	
example 2	private	3/30/2022	

Figure 10. Project Management UI Mockup

4. Implementation

4.1. Technologies Used

The Audio Canvas application was developed using web technologies utilizing a client-server design pattern. Accessing the application through the browser provided a more portable interface. There are three different components: client, server, and database. This section describes the technologies these three components utilized and why they were a good fit for this software system.

4.1.1. Client

The client application contained most of the project weight. This component is responsible for providing an intuitive user interface capable of managing audio-visual projects. The views, routing, and forms of client application used the ReactJS web library. This library was a desirable choice because the project required complex dynamic web pages to be generated and manipulated by user input. ReactJS handles this well utilizing JSX to dynamically manipulate web pages. JSX is a syntax extension of JavaScript that allows HTML-like embedded documents to exist in JavaScript functions [8]. Figure 11 demonstrates how JSX was used to dynamically generated menus and editors in this project. Another popular web framework AngularJS would have been a good option as well as it uses similar ideas to build dynamic web pages. However, ReactJS seems to be more appealing as it is slightly more performant in the way it handles rendering the DOM.

```
{this.props.appSettings.view === "edit"
    ? <MainMenu canvasInterface={canvasInterface} />
    : ""}
{this.props.appSettings.view === "edit"
    ? <ComponentEditor />
    : ""}
<ViewEditor />
<Canvas />
```

Figure 11. Using JSX to Manage Menus and Editors

The main application's canvas utilizes two different popular web technologies: Web Audio API and BabylonJS. These technologies are supported by a vast majority of modern browsers, making the application available to a sizable portion of web users. Using these two technologies provided performant live feedback of the audio-visual.

The Web Audio API is responsible for deconstructing audio streams into frequency spectrum data and time domain data. Using this data, we can cast it upon visual properties to make objects visually move with audio. There is not much of any alternative to the Web Audio API in web technologies, so this was an obvious decision [11]. Figure 12 shows how the Web Audio API was used to extract frequency and time domain information from an audio analyser.

```
for (let key of Object.keys(_this.audio)) {
    let index = _this.props.audioInputs.findIndex(a => a.id == key)
    frequencyData[index] = _this.audio[key].audioAnalyser.getByteFrequencyData();
    timeDomainData[index] = _this.audio[key].audioAnalyser.getByteTimeDomainData();
}
```

Figure 12. Using the Web Audio API to Extract Frequency and Time Domain Audio Data

BabylonJS is responsible for rendering the user defined visual components to a canvas. BabylonJS is a WebGL abstraction and a scene graph system. WebGL is the web's 3D graphics standard: it is well integrated into browsers, well documented, and performant. WebGL can utilize hardware accelerated graphics, taking advantage of the powerful GPUs we have today [2]. Other popular graphic web technologies, Canvas API and SVG Graphics, would have been decent options for the rendering aspect of this project as well. However, the project aims to render 3D graphics which are not supported by these other technologies out of the box. Figure 13 shows how BabylonJS was used to create and initialize a scene from a canvas element.

```
const canvas = document.getElementById("renderCanvas");
_this.engine = new BABYLON.Engine(canvas, true);
_this.scene = new BABYLON.Scene(this.engine);
```

Figure 13. Using the BabylonJS Engine to Initialize the Scene

4.1.2. Server

The server application is responsible for handling the authentication, authorization, user management, and project management. The server uses FeathersJS framework to manage data services following the REST API architectural style. This framework fit well with this project as the project did not require an extensive backend system. FeathersJS performs all the necessary functionalities in a lightweight package. The server uses OAuth Standard to serve JWT tokens that users must provide with their requests to access their resources. The server also supports a user management system, where admin users can control user access. For example, an admin can look up a user and deactivating that account. The system also supports users performing create, read, update, and delete (CRUD) operations on their projects. The product service endpoints shown in Table 3 demonstrates these operations. A detailed listing of these API services can be found in Appendix B.

Endpoint	**GET /projects**
Description	Query projects
Inputs	Search projects by attributes via query parameters
Responses	200 Ok – list of projects is returned 401 Not Authorized
Access Controls	admins have access to all projects users have access to all public projects and their own private projects
Endpoint	**POST /projects**
Description	Create a project
Inputs	name: string visibility: public/private
Responses	201 Created – project is returned 400 Bad Request – invalid parameters 401 Not Authorized
Notes	project parameters are validated
Endpoint	**PATCH /projects/id**
Description	Update some project information
Inputs	name: string visibility: public/private
Responses	200 Ok – updated project is returned 404 Not Found 401 Not Authorized
Access Controls	admins have access to all projects users have access to their own projects and projects they have edit access
Notes	project parameters are validated
Endpoint	**DELETE /projects/id**
Description	Delete a project
Responses	200 Ok – project is deleted 404 Not Found 401 Not Authorized
Access Controls	users can delete their own projects

Table 3. Projects Service Example

The server exposes FeathersJS services for each of the data structures that require persistence: users, projects, projectData, and projectAccess. These services offer a uniform way of interacting with different data structures [5]. The FeathersJS class registrations create the REST API endpoints and connect them to a database adapter. The project service registration shown in Figure 14 demonstrates how these services are setup following the FeathersJS framework. This code can be interpreted as the "Projects" service is a MongoDB service that uses the "projects" collection to store data and this service uses the "/projects" path.

```
class Projects extends Service {
  constructor(options, app) {
    super(options);
    app.get('mongoClient').then(db => {
      this.Model = db.collection('projects');
    });
  }
};

const options = { paginate: app.get('paginate')};
app.use('/projects', new Projects(options, app));
const service = app.service('projects');
service.hooks(hooks);
```

Figure 14. Project Service Registration

These services are controlled by hooks. Hooks are middleware functions that can register before, after, or on errors in a service [5]. These hooks are used for authorization, validation, data manipulation, error logging, and other use cases. The project service hooks as shown in 15 demonstrates how these hooks are registered. The hooks are executed in order from left to right. During the create operation, the first hook ensures the request is authenticated. If the request is not authenticated, the hook will immediately return a 401 - Not Authorized response. The second hook validates the request data. This ensures the data passed to the create function is in fact a project object. If the data does not pass this validation, a 400 Bad Request response is returned.

```
before: {
  all: [],
  find: [],
  get: [],
  create: [authenticate('jwt'), validate],
  update: [authenticate('jwt'), hasWriteAcccess, validate],
  patch: [authenticate('jwt'), hasWriteAcccess, validate],
  remove: [authenticate('jwt'), hasWriteAcccess, validate]
},

after: {
  all: [],
  find: [hasReadAccess],
  get: [hasReadAccess],
  create: [],
  update: [],
  patch: [],
  remove: []
},

error: {
  all: [context => console.log(context.error)],
  find: [],
  get: [],
  create: [],
  update: [],
  patch: [],
  remove: []
}
```

Figure 15. Project Service Hooks

4.1.3. Database

The database system is responsible for the persistence of user and project data. User data contains basic account information. Projects have nested component data to represent the different layers of the image. These different components requires the database to be flexible with respect to type checking. Since this data is hierarchical in structure, a document-based database was a good fit. The database system uses MongoDB to save this data for this reason. MongoDB provides a json-like document-based storage system that can handle this flexibility.

4.2. Development

The application was developed in eighteen sprints. Sprints one through four contained all the requirements pertaining to authentication flow and the user/project management. Sprints four through sixteen contained all requirements regarding the interface that was responsible for creating and customizing audio-visuals.

4.2.1. Data Management System

The data management portion of this system was built first. This included setting up the database, server, and the client web pages responsible for user and project management. This allowed for the create, read, update, and delete operations on users and projects. The entire authentication flow was built in this portion of the project as well. This portion of the project was built first because I wanted to have a solid backend system in place before the application work of the project was started.

4.2.2. Application

The application portion of this project was built second. This included the user interface required to customize and build audio-visuals. This portion of the project was built by developing basic visuals first, then working towards more complex visuals. First, we had to get some geometric primitives on the canvas and allow the user to edit them: cube, sphere, polyhedron, etc. This was done by allowing the user to select from a list of supported primitives and exposing parameters about those primitives to customize them. With these parameters defined, the canvas can call BabylonJS to render the object. Once we had those geometric primitives defined, we built the functionality to enable the user to manipulate those geometric primitive parameters with audio data. Figure 16 shows the different primitives available in the system and how a user can manipulate a particular primitive's parameters. Figure 17 shows an example of how the structure of inputs are defined for the "Box" type.

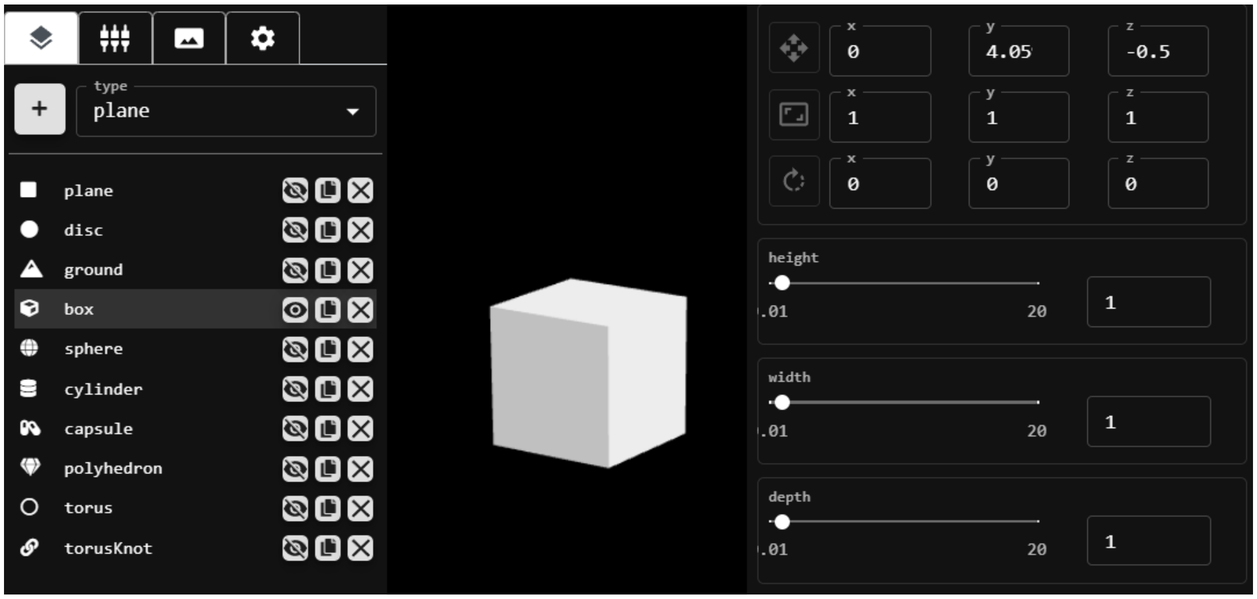


Figure 16. Geometric Primitives

```
case BOX_TYPE: {
    return [
        ...defaultFields,
        ...geometricTransformFields,
        { name: "audio", type: "audio", isConstant: true },
        { name: "height", type: "range", min: .01, max: 20, step: .01 },
        { name: "width", type: "range", min: .01, max: 20, step: .01 },
        { name: "depth", type: "range", min: .01, max: 20, step: .01 },
        ...shapeFields,
    ]
}
```

Figure 17. Box Primitive Input Structure

This introduced the concept of audio inputs and the data processing around its frequency and time domain data. This process of manipulating visual objects with audio data is performed in the following manner: The user finds a parameter they want to move with respect to an audio input. They specify a range over which this parameter will move within. They select the particular audio data that they want captured: frequency spectrum, time domain. They apply a filter on that data to specify what frequency range or what time domain window they want passed through. The root mean square (RMS) is calculated for the audio data generated. This value is normalized to determine at what point across the parameter range the value lies at a given point in time [3]. Figure 18 demonstrates how audio inputs are defined and how an input can be mapped to a particular parameter. Figure 19 shows how the root mean square (RMS) is calculated for this mapping process.

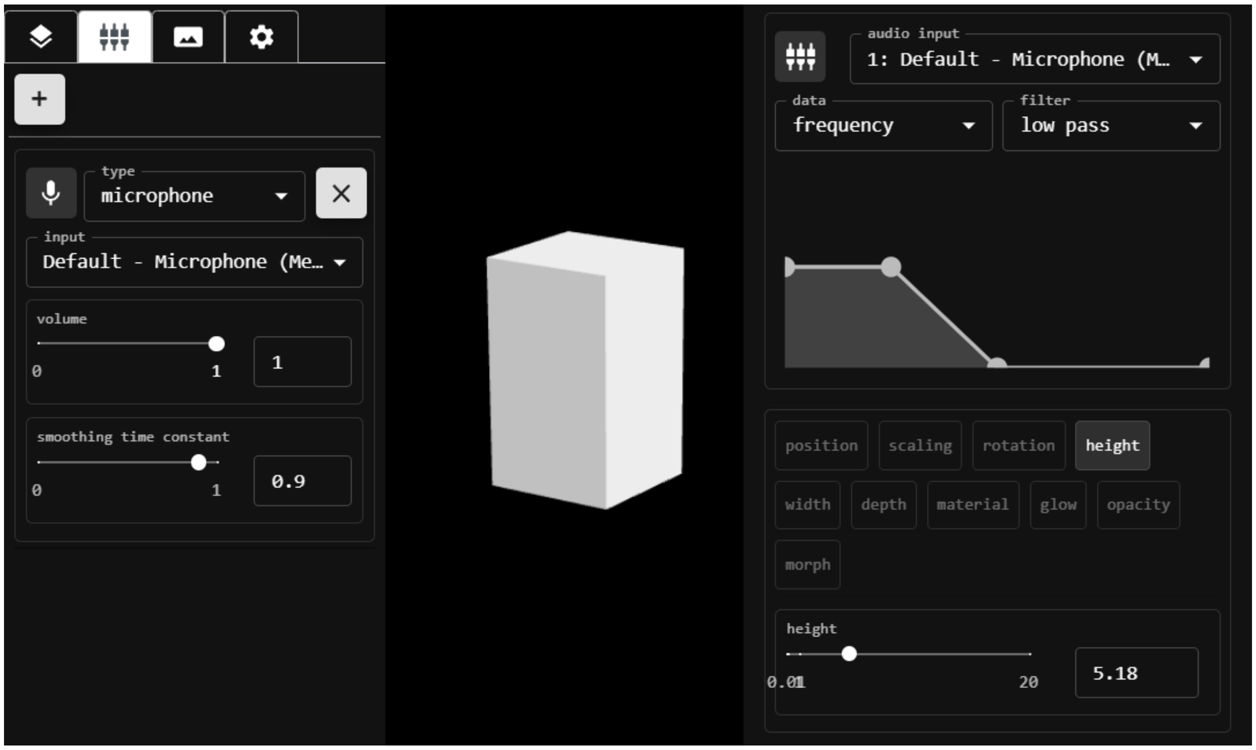

type: microphone
input: Default - Microphone (Me…
volume: 1
smoothing time constant: 0.9
audio input: 1: Default - Microphone (M…
data: frequency
filter: low pass
height: 5.18

Figure 18. Audio Mapping

```
export const rmsFrequencyDomain = (audioData) => {
    let avgPowerDecibels = 0;
    if (audioData && audioData.length){
       let sumOfSquares = 0;
       for (let i = 0; i < audioData.length; i++) {
         sumOfSquares += (Math.abs(audioData[i])) ** 2;
       }
    avgPowerDecibels = Math.sqrt(sumOfSquares / audioData.length);
    }
    return avgPowerDecibels.toFixed(2);
}
```

Figure 19. Root Mean Square Calculation (RMS)

Once geometric primitives were defined and their properties were able to be manipulated by audio data, more complex visual object ideas were developed: grouping and time animations. Grouping takes individual visual components and orders them in a hierarchical fashion. Time animations take component parameters and move them over time. Time animations are configured in the following manner: The user finds a parameter they want to move with respect to time. They specify the range over which this parameter will move within. The choose how long this loop goes on for. They specify the path for which this parameter travels over via a low frequency oscillator (LFO). With these tools in place the user is now at the point where they can develop complex visuals that can move in and out over time. Figure 20 demonstrates how components can be grouped and their parameters can be mapped to time. Figure 21 shows how the current time determines the position in a LFO and how that is calculated.

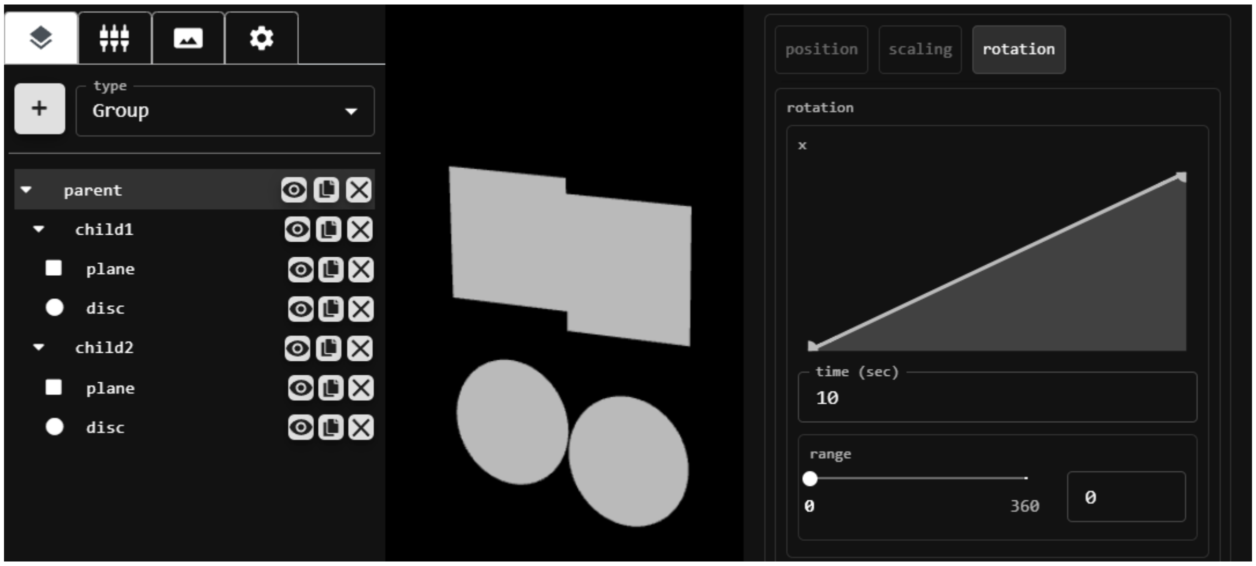


Figure 20. Grouping and Time Animations

```
const getTimeValue = (currentTime, loop, lfo, upperBound, lowerBound) => {
    let positionInLFO = (currentTime % (loop * 1000)) / 1000 / loop * 100;
    let highBoundIndex = lfo.findIndex(e => e.x > positionInLFO);
    let lowBoundIndex = highBoundIndex - 1;
    let rise = lfo[highBoundIndex].y - lfo[lowBoundIndex].y;
    let run = lfo[highBoundIndex].x - lfo[lowBoundIndex].x;
    let ratio = rise / run;
    let value = (positionInLFO - lfo[lowBoundIndex].x) * ratio;
    value += lfo[lowBoundIndex].y;
    let factor = value / 100;
    return (upperBound - lowerBound) * factor;
}
```

Figure 21. Time in LFO Calculation

5. Testing

5.1. Overview

Testing this software system verifies we were developing the system we set out to achieve, and validates it was performing as we expect it to. The testing aspect of this project was performed throughout development during the “TESTING” status of each sprint task. The verification and validation of this system is discussed in this section.

5.2. Verification

During the design and implementation of this software system it was crucial to ensure the system was developed to meet our main project goals: simplicity, portability, and collaboration. When it came time to decide on what technologies would be used to design this system, these goals were at mind. Our portability goal played a key role in determining many of the technologies used. For example, using web based system opens many possibilities for cross platform support and connectivity through the internet. Each functional component of the project pulled into sprints was developed with these goals in mind as well. Simplicity also played a key role in the development of many of the user interface interactions. For example, when developing how color would change with respect to time. It was determined that a color gradient editor would be the most straightforward way to convey this, so the user could specify what color an object was at any given point in time. Each sprint review provided valuable time to reflect on the functional components developed during the sprint and ask the question: Are we building the product right?

5.3. Validation

During the design and implementation of this software system it was important to ensure the system was performing as we expected it to. Each sprint provided a time to test the functional components built during that sprint. A few different validation testing techniques were performed at this time.

User input testing was important for this project due to the large amount of inputs necessary to customize an audio-visual project. The audio-visual editor application also has numerous types of inputs: text, number, range, color, and others. These input fields were tested using equivalence partitioning and boundary value analysis. Equivalence partitioning breaks input domain data down into smaller more testable data classes. Boundary value analysis targets input min/max values to find errors in an input’s range. Table 4 shows an example of how test cases were constructed by targeting the box component class and testing its height input boundaries.

Box Height Test Cases				
	Inputs			
#	Height	Width	Depth	Expected Output
1	-1	10	10	height set to .01
2	.01	10	10	height set to .01
3	.02	10	10	height set to .02
4	10	10	10	height set to 10
5	19.99	10	10	height set to 19.99
6	20	10	10	height set to 20
7	20.01	10	10	height set to 20

Table 4. Box Height Test Cases Example

Performance testing was also important for this project due to the nature of graphic processing in a web environment. In an effort to ensure a quality and professional render, the system needed to make sure the video playback of audio-visuals were reasonably performant. To capture this, we used the frames per second (FPS) metric. FPS is a computer graphics performance metric that captures the frequency at which images are displayed. For this application, we determined 30fps as reasonably performant. 30fps will ensure the visual is smoothly rendered with no flickering or skipping. Anything greater than 30fps will be functioning as expected. This was validated through integration testing. The FPS performance metric was provided by the BabylonJS graphics engine. This metric was tested against a variety of different complex visuals in the system. This was done to stress test the graphical processing capabilities. At various points in the project development, these stress tests failed. At this point, it was deemed necessary to rewrite processes in a more performant manner. Overall, these performance tests pushed development towards outputting higher quality video playback, and they were important in maintaining these standards.

Cross-Platform testing was another important testing aspect of this project. In order to achieve our portability goal, this application was required to run on a variety of different hardware and software systems. This testing was achieved by performing high-level project workflows on a variety of devices and browsers. For example, the process of creating, editing, and exporting and audio-visual was tested on Safari, Firefox, Google Chrome, Microsoft Edge, IOS, and Android.

6. Security

6.1. Overview

The security expectations of this project is to protect the confidentiality and integrity of the audio-visuals created in the system. It is also important to use best web security practices to reduce the risk of potential software attacks. The security features used to accomplish theses goals are discussed in this section.

6.2. Authentication and Authorization

This software system relys on a local authentication service to authorize user request to access projects. This authorization service uses OAuth protocol to achieve this. When a user signs into the system, a JSON Web Token (JWT) is generated and passed back to the client. This token is used for all subsequent requests to access resources. Figure 22 demonstrates this authorization flow. Since this system relys on local authentication, it is the systems responsibility to maintain user credentials. All passwords are hashed with a bcrypt before saving to the database system.

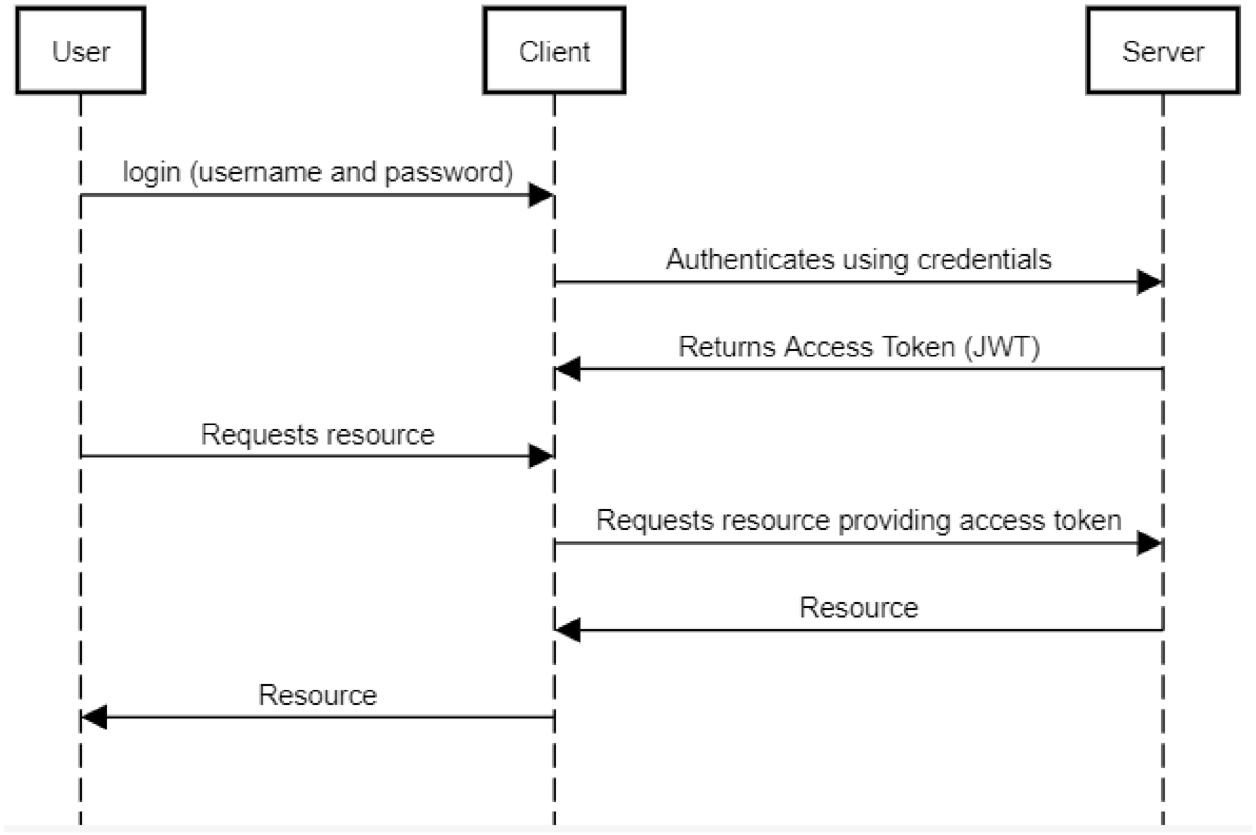

User
Client
Server
login (username and password)
Authenticates using credentials
Returns Access Token (JWT)
Requests resource
Requests resource providing access token
Resource
Resource

Figure 22. Authorization - Accessing Resources

The client only enables the user to access their own resources through the UI, however it is the server responsibility to enforce this. Each endpoint on the server has a hook before data is saved to the database to ensure the user has the necessary access controls to save the data that was requested. Figure 23 shows an example of these access controls where a user is not allowed to update another user's project data.

```
const hasWriteAcccess = async (context) => {
  let hasAccess = context.params.user.userRole === "admin";
  if (!hasAccess) {
    let project = await context.app.service('projects').get(context.id);
    hasAccess = context.params.user._id == project.userId;
  }
  if (!hasAccess) {
    throw new Forbidden();
  }
}
```

Figure 23. Server API Access Controls

6.3. Web Security

It is important to ensure web applications are up to date with current web security practices to reduce the risk of software attacks. The following methods were used to help mitigate threats in this system.

The software system uses HTTPS to provide security through the Transport Layer Security (TLS) protocol. HTTPS ensures that HTTP traffic between the client and server are encrypted. This encryption helps protect private data used in the system. HTTPS also ensures the integrity of the data used in the system. HTTPS provides a digital signature with each message sent over the internet. This enables the recipient to verify who sent the message and ensure that it has not been tampered with.

This software system is heavily focused on user generated content. As such, the threat of any potential malicious data entered into inputs on the client can enable cross-site scripting (XSS) attacks. These types of attacks were mitigated through ReactJS's use of JSX and input sanitization. ReactJS helps prevent XSS attacks by escaping embedded values. This ensures anything not directly written in JSX can not be rendered as HTML and is converted to a string [8]. Input sanitation was also used to help mitigate the threat of Cross-Site Scripting attacks. All inputs used to gather information from the user are filtered. For example, text inputs only allow the necessary characters needed for the parameter.

7. Deployment

7.1. Overview

The application is available through a web browser. This was possible by deploying the individual components of the system to different service providers. This section describes how each individual component of the software system is hosted and deployed.

The client application is hosted using the Netlify web service. This portion of software system only contains static web content. A modern approach to serving static content is to use serverless computing, which is what Netlify offers. Netlify web service sets up the necessary Amazon Web Services (AWS) under the hood to store and serve this content. Netlify also aids in continuous deployment process by auto building and deploying commits pushed to a git repository.

The server application is hosted on the Heroku web service platform. This portion of the software system is containerized in Heroku. Heroku builds, runs, and operates the server application. Heroku also aids in the continuous deployment process by building and deploying commits pushed to a git repository.

The Database is hosted on the MongoDB Atlas web service. This portion of the software system stores the user generated content in a cloud database service. Behind the scenes MongoDB Atlas sets up the necessary Amazon Web Services (AWS) to store this data, as well as allow read and write operations to it.

The goal of this deployment process was to provide an available proof of concept. The current deployment methods utilized are not setup in a robust fashion to handle extensive traffic. If this application was to be used in a more production friendly environment, additional items would have to be considered such as monitoring, error logging, auto scaling, load balancing, pricing, and others.

7.2. Mobile Application

This software system is also available through a mobile application. In some mobile workflows, it is beneficial to work from within a app rather then a web browser on the device. Mobile web browsers can be clunky and include unnecessary overhead. A small React Native mobile application was developed to serve this web application through a WebView. This makes it possible to have the web application available on IOS and Android platforms via a mobile application.

8. Conclusion

8.1. Overview

This software system makes the process of creating an audio-visual more simple, portable, and collaborative. This was done by designing and building a web application that enables users to manage, edit, and share audio-visual projects. These projects can be configured by managing audio inputs, adding visual elements to a canvas, and manipulate these elements with respect to the audio inputs specified.

8.2. Challenges

One of the main obstacles of this project was ensuring the quality of the visual render. When dealing with an application that uses graphical rendering, it is always is a little tricky to achieve a quality render. The project development required constant attention to how the various interactions with inputs were transformed into the rendered visual. For example, when initially setting up the geometric primitive inputs to control an object on the canvas, every time the user would move the range input, the render would see a large FPS drop (30-60fps). This was happening because the input changes were reflected on the canvas live, causing the canvas to rerender the image repeatedly. To overcome this, now the canvas is only rerendered once the input is fully set.

Another challenging aspect of this project was ensuring multi-platform support. It was a constant battle of implementing new features, then checking them against different browsers and ensuring everything still worked. For example, when using the MediaRecorder (a part of the MediaStream Recording API) to record the audio-visual, none of the main browsers supported the same output file type. In order to get the MediaRecorder to work cross-platform, there needs to be logic in place to recognize what environment the application is running in to determine what output file type to use [7].

8.3. Learning Experience

If nothing else, this project was a great learning experience for myself. Throughout the project, I had the opportunity to wear many hats: product owner, scrum master, developer, and tester. I also had the opportunity to work with many different technologies that I had no prior experience with: ReactJS, BabylonJS, Web Audio API, Netlify, Heroku, and many others. I thoroughly enjoyed taking ownership of the project and seeing it all the way through.

8.4. Future Work

I have received a lot of feedback from peers on improvements and new functionality to improve the application. I plan to explore these ideas further. Some examples include authenticating with OpenID Connect (Google, Facebook, Twitter, etc.), importing 3D Models (.stl, .obj, etc.), some sort of mixing feature (blend multiple projects together), and many more. At least for the time being, this application will be used by a small group of individuals to create and share audio visualizations. There are future plans on potentially making the application public for people to enjoy.

9. Bibliography

[1] Atlassian — Agile Coach. `https://www.atlassian.com/`, 2022.

[2] Babylon.js Documentation. `https://doc.babylonjs.com`, 2021.

[3] Wikipedia Contributors. Root Mean Square. `https://en.wikipedia.org/wiki/Root_mean_square`, 2021.

[4] Wikipedia Contributors. FCurve. `https://en.wikipedia.org/wiki/FCurve`, 2022.

[5] Feathers — a Framework for Real-Time Applications and REST APIs. `https://docs.feathersjs.com`, 2021.

[6] David C. Kung. *Object-Oriented Software Engineering*. McGraw-Hill Higher Education, 2013.

[7] MediaStream Recording API — Web API — MDN. `https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API`, 2021.

[8] Getting Started – React. `https://reactjs.org/docs/getting-started.html`, 2021.

[9] Kenneth S. Rubin. *Essential Scrum*. Addison-Wesley Professional, 2012.

[10] Scrum.org — The Home of Scrum. `https://www.scrum.org/`, 2022.

[11] Web Audio API. MDN Web Docs. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API`, 2021.

10. Appendices

10.1. Appendix A: Project Requirements

Index	**1.1 — Login**
User Story	As a user, I would like to be able to login to my account.
Acceptance Criteria	User provides appropriate credentials – Logs into system User provides inappropriate credentials – Error Message Notes: Credentials - (email, password)

Index	**1.2 — Logout**
User Story	As a user, I would like to be able to log out of the system.
Acceptance Criteria	User clicks logout – Logs out of system provides inappropriate credentials – Error Message

Index	**1.3 — Register**
User Story	As a user, I would like to be able to register for a new account.
Acceptance Criteria	User provides registration information – account is created User provides illegitimate registration information – Error message Notes: registration information - (email, password, display name, first name, last name)

Table 5. Authentication Requirements

Index	**2.1 — Create Project**
User Story	As a user, I would like to be able to create a project, so I can save my visualizer for future reference.
Acceptance Criteria	User provides project information – project is created User provides illegitimate information – Error message Notes: Project information - (name, visibility – public or private)

Index	**2.2 — Open Project**
User Story	As a user, I would like to be able to open a project, so I can view the visualizer.
Acceptance Criteria	User clicks open – project is opened

Index	**2.3 — Edit Project**
User Story	As a user, I would like to be able to edit a project, so I can change the name or visibility.
Acceptance Criteria	User edits project name or visibility – project is updated

Index	**2.4 — Delete Project**
User Story	As a user, I would like to be able to delete a project, so I can remove it from my list.
Acceptance Criteria	User clicks delete – project is deleted

Index	**2.5 — Share Project**
User Story	As a user, I would like to share my project with another user, so we can collaborate on the same project or enable the other user to view a private project.
Acceptance Criteria	User can enter in another user's email address to share "User not found" error message if user with email does not exist User can specify if other user has read or write privileges to the project Other user can view the project if they have read privileges. Other user can edit the project if they have write privileges. Notes: Project information - (name, visibility – public or private)

Table 6. Project Management Requirements

Index	**3.1 — Admin View Users**
User Story	As an admin, I would like to be able to view all users in the system, so I can manage their access and permissions.
Acceptance Criteria	Admins are presented with user screen to search through

Index	**3.2 — Admin Edit User Permissions**
User Story	As an admin, I would like to be able to edit a user's account, so I can manage their access and permissions.
Acceptance Criteria	Admins can update "is active" and "user role" properties

Table 7. Admin Requirements

Index	**4.1 — Multiple Audio Inputs**
User Story	As a user, I would like to be able to specify multiple audio inputs, so I can map my visualization components to different inputs.
Acceptance Criteria	User can add and remove audio inputs from visualizer
Index	**4.2 — File Input**
User Story	As a user, I would like to be able to upload an audio file for my visualization, so I can map my visualizer to it.
Acceptance Criteria	User can upload an upload an audio file User uploads file that is not an audio file – Error message
Index	**4.3 — File Input Controls**
User Story	As a user, I would like to be able to control my audio file inputs (play, pause, seek, volume)
Acceptance Criteria	User can play/pause User can seek in track User can specify volume
Index	**4.4 — Microphone Input**
User Story	As a user, I would like to be able to use a microphone input for my visualization, so my visualizer can sync with live audio recordings.
Acceptance Criteria	User can specify what microphone input to use User can specify volume on input feed
Index	**4.5 — Audio Data**
User Story	As a user, I would like to be able to specify if my components are mapped to frequency data or time domain data, so my visualizer can sync to both.
Acceptance Criteria	User can select between using frequency data or time domain data
Index	**4.6 — Audio Filter**
User Story	As a user, I would like to be able to filter the audio data passed to my visualizer, so I can tune what frequency range or time domain window is displayed.
Acceptance Criteria	User can apply a low pass, high pass, band pass, or custom filter Notes: When creating a custom filter, the user can specify breakpoints.

Table 8. Audio Input Requirements

Index	5.1 — Manage Layers
User Story	As a user, I would like to be able to manage my component layers, so I can group my components, determine the order of the layers, and if they are visible or not.
Acceptance Criteria	User can choose from a list of supported components and add it as a layer User can remove a layer User can duplicate a layer User can group layers together User can add and remove layers from a group User can reorder layers User can toggle visibility of layer

Index	5.2 — Edit Component
User Story	As a user, I would like to be able to edit my component layers, so I can customize how they appear.
Acceptance Criteria	User can select/deselect component User is shown component properties and can edit them

Table 9. Layer Requirements

Index	**6.1 — Bar Graph**
User Story	As a user, I would like to be able to add a custom bar graph to my visualization, so I can view the frequency spectrum.
Acceptance Criteria	User can add a bar chart to visualizer User can customize properties on that bar chart Notes: properties - (shape of bars, opacity, number of bars, distance between bars, minimum height of bars, maximum height of bars, width of bars, material)

Index	**6.2 — Line Graph**
User Story	As a user, I would like to be able to add a custom line graph to my visualization, so I can view the time domain audio data.
Acceptance Criteria	User can add a line chart to visualizer User can customize properties on that line chart

Index	**6.3 — Primitive Meshes**
User Story	As a user, I would like to be able to add custom primitive meshes to my visualization, so I can build a 3D visualizer.
Acceptance Criteria	User can add primitive meshes to visualizer User can customize properties on those primitive meshes Notes: primitives - (plane, disc, ground, box, sphere, cylinder, capsule, polyhedron, torus, torus knot) Primitives will have relevant properties that should be editable

Index	**6.4 — Text**
User Story	As a user, I would like to be able to add text to my visualizer, so I can add titles and labels.
Acceptance Criteria	User can add text to visualizer User can customize properties on that text meshes Notes: properties - (text, font, size, depth)

Table 10. Component Requirements

Index	7.1 — Geometric Transforms
User Story	As a user, I would like to be able to perform geometric transforms on my components (position, rotation, scaling)
Acceptance Criteria	User can position a component in the scene User can rotate a component in the scene User can scale a component in the scene
Index	**7.2 — Images**
User Story	As a user, I would like to be able to use images in my visualizer, so I can take an existing image and include it in the visualizer
Acceptance Criteria	User can map an image to a primitive mesh's material User can use an image as a static background
Index	**7.3 — Webcam**
User Story	As a user, I would like to be able to use my webcam in my visualizer, so I can add live video recording to my visualizer
Acceptance Criteria	User can map webcam to a primitive mesh's material User can use a webcam as background
Index	**7.4 — Color Gradient**
User Story	As a user, I would like to control the colors of my visualization with a color gradient, so I can specify multiple colors and control how the visualizer progresses through them.
Acceptance Criteria	User can define a color gradient User can add as many colors as they want User can specify the color gradient's breakpoints

Table 11. Special Input Requirements

Index	**8.1 — Background**
User Story	As a user, I would like to be able to control the background of my visualization, so I specify what is in the background.
Acceptance Criteria	User can specify color of background User can specify image as background User can specify webcam as background
Index	**8.2 — Camera**
User Story	As a user, I would like to be able to control the camera on my visualization, so I can control at what angle and zoom I am viewing the scene.
Acceptance Criteria	User can set camera properties Notes: Camera properties - (zoom, latitude, longitude, position)

Table 12. Scene Requirements

Index	**9.1 — Audio Mapping**
User Story	As a user, I would like to be able to move components with the audio inputs I specified, so I can sync my visualizer with audio.
Acceptance Criteria	User chooses an audio input, data, and filter User can turn on/off parameters User specifies a range in which the component's parameter can move within based on the data fed into it
Index	**9.2 — Time Mapping**
User Story	As a user, I would like to be able to move components with respect to time, so I can have my visualizer change over time.
Acceptance Criteria	User chooses parameter to move with time User can create a parameter changing loop by specifying (loop length, parameter range, and LFO)

Table 13. Mapping Requirements

Index	**10.1 — Record**
User Story	As a user, I would like to be able to record my visualization, so I can save it as a video file.
Acceptance Criteria	User hits record button and audio visualizer recording starts. User hits record button again an audio visualizer recording stops, and video file is presented to user. Notes: Recording should only capture what is on canvas

Table 14. Export Requirements

10.2. Appendix B: API Service Details

Endpoint	**POST /authentication**
Description	Retrieve an access token
Inputs	username: string password: string strategy: "jwt" OR accessToken: string strategy: "jwt" Notes: Credentials - (email, password)
Responses	201 Created – jwt access token is returned 401 Not Authorized – Invalid Login
Endpoint	**DELETE /authentication**
Description	Purge access token
Inputs	accessToken: string
Responses	200 Ok – accessToken deleted 401 Not Authorized – Invalid Token

Table 15. Authentication Service

Endpoint	**GET /users**
Description	Query users
Inputs	Search user properties via query parameters limit: int skip: int
Responses	200 Ok – list of users is returned 401 Not Authorized
Access Controls	only admins have access to this endpoint
Notes	password is removed from response
Endpoint	**POST /users**
Description	Register a user
Inputs	firstName: string lastName: string displayName: string email: string password: string
Responses	201 Created – user is returned 400 Bad Request – email already in use 400 Bad Request – invalid parameters
Notes	user parameters are validated password is hashed password is removed from response
Endpoint	**GET /users/id**
Description	Retrieve a user
Responses	200 Ok – user is returned 404 Not Found 401 Not Authorized
Access Controls	admins can get all user information users can retrieve their own information
Notes	password is removed from response
Endpoint	**PATCH /users/id**
Description	Update some user information

Inputs	firstName: string lastName: string displayName: string email: string password: string userRole: string
Responses	200 Ok – updated user is returned 404 Not Found 401 Not Authorized
Access Controls	Only admins are allowed to update user information
Notes	user parameters are validated password is hashed password is removed from response

Table 16. Users Service

Endpoint	**GET /projects**
Description	Query projects
Inputs	Search projects via query parameters limit: int skip: int
Responses	200 Ok – list of projects is returned 401 Not Authorized
Access Controls	admins have access to all projects users have access to all public projects and their own private projects
Endpoint	**POST /projects**
Description	Create a project
Inputs	name: string visibility: public — private
Responses	201 Created – project is returned 400 Bad Request – invalid parameters 401 Not Authorized
Notes	project parameters are validated
Endpoint	**GET /projects/id**
Description	Retrieve a project
Responses	200 Ok – project is returned 404 Not Found 401 Not Authorized
Access Controls	admins have access to all projects users have access to all public projects and their own private projects
Endpoint	**PATCH /projects/id**
Description	Update some project information
Inputs	name: string visibility: public — private
Responses	200 Ok – updated project is returned 404 Not Found 401 Not Authorized

Access Controls	admins have access to all projects users have access to their own projects and projects they have edit access to
Notes	project parameters are validated
Endpoint	**DELETE /projects/id**
Description	Delete a project
Responses	200 Ok – project is deleted 404 Not Found 401 Not Authorized
Access Controls	users can delete their own projects

Table 17. Projects Service

Endpoint	**GET/project-data**
Description	Retrieve project data
Inputs	Search project data properties via query parameters limit: int skip: int
Responses	200 Ok – list of project data is returned 404 Not Found 401 Not Authorized
Access Controls	admins have access to all projects users have access to all public projects and their own private projects
Endpoint	**POST /project-data**
Description	Create project data
Inputs	projectId: string components: Array¡Component¿ inputs: Array¡Input¿ sharedData: SharedData
Responses	201 Created – project is returned 400 Bad Request – invalid parameters 401 Not Authorized
Access Controls	admins have access to all projects users have access to their own projects and projects they have edit access to
Notes	project data parameters are validated
Endpoint	**PATCH /project-data/id**
Description	Update some project data information
Inputs	projectId: string components: Array¡Component¿ inputs: Array¡Input¿ sharedData: SharedData
Responses	200 Ok – updated project data is returned 404 Not Found 401 Not Authorized
Access Controls	admins have access to all projects users have access to their own projects and projects they have edit access to

Notes	project data parameters are validated
Endpoint	**DELETE /project-data/id**
Description	Delete project data
Responses	200 Ok – project data is deleted 404 Not Found 401 Not Authorized
Access Controls	users can delete from their own projects

Table 18. Project Data Service

Endpoint	**GET/project-access**
Description	Retrieve project accesses
Inputs	Search project access properties via query parameters limit: int skip: int
Responses	200 Ok – list of project access is returned 404 Not Found 401 Not Authorized
Access Controls	users have access to their own projects
Endpoint	**POST /project-access**
Description	Create project access
Inputs	projectId: string userId: string access: read — write
Responses	201 Created – project access is returned 400 Bad Request – invalid parameters 401 Not Authorized
Access Controls	users have access to their own projects
Notes	project access parameters are validated
Endpoint	**PATCH /project-access/id**
Description	Update some project access information
Inputs	userId: string access: read — write
Responses	200 Ok – updated project access is returned 404 Not Found 401 Not Authorized
Access Controls	users have access to their own projects
Notes	project access parameters are validated
Endpoint	**DELETE /project-access/id**
Description	Delete project access
Responses	200 Ok – project access is deleted 404 Not Found 401 Not Authorized

Access Controls	users can delete from their own projects

Table 19. Project Access Service

10.3. Appendix C: Graphical User Interface

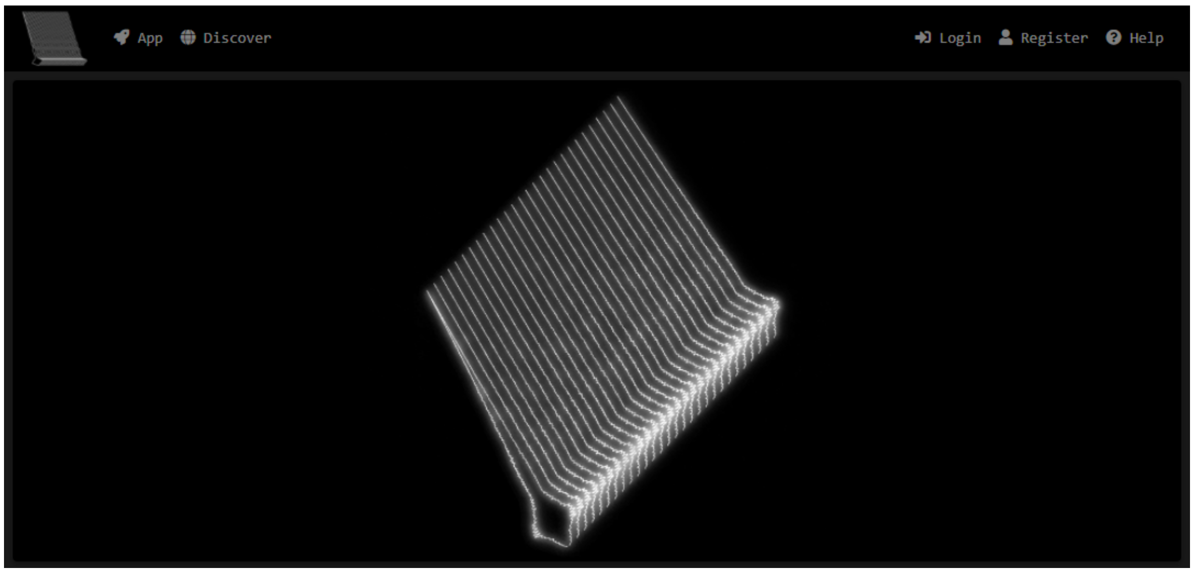


Figure 24. Home Page

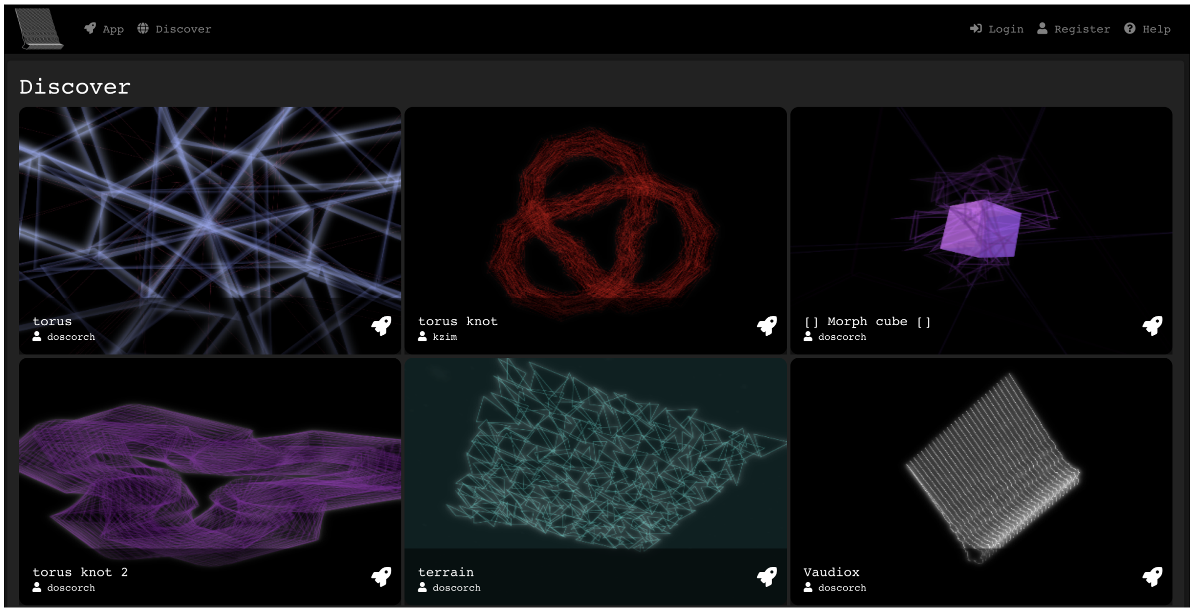


Figure 25. Discover Page

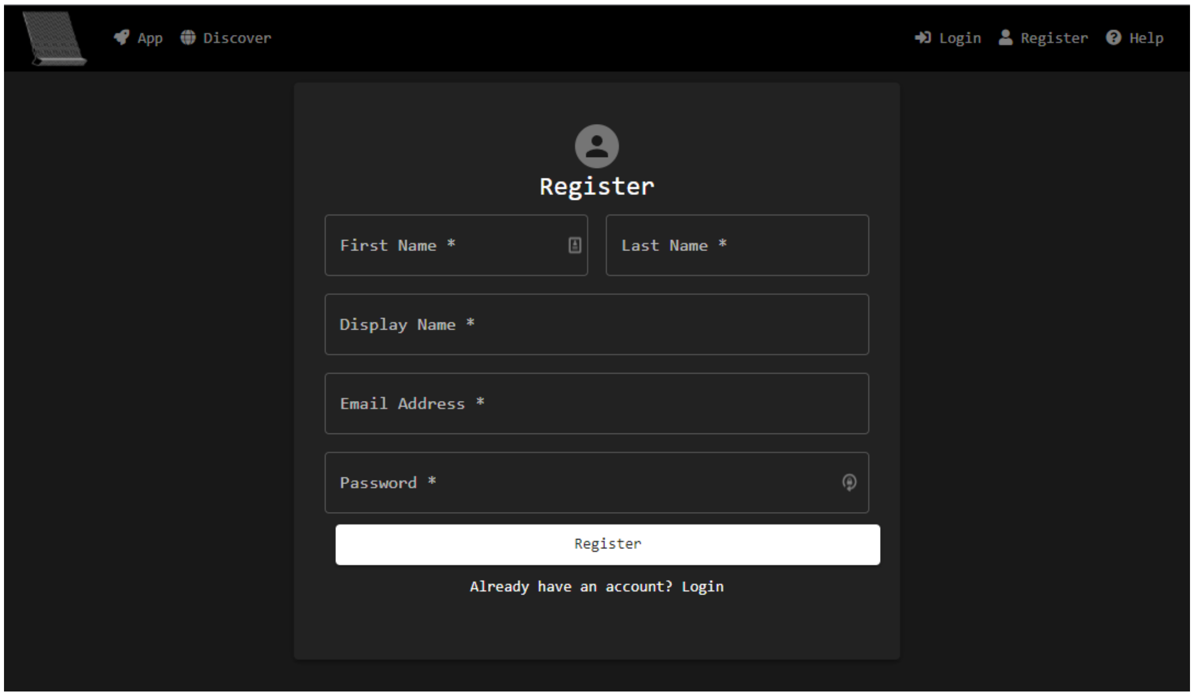


Figure 26. Register Page

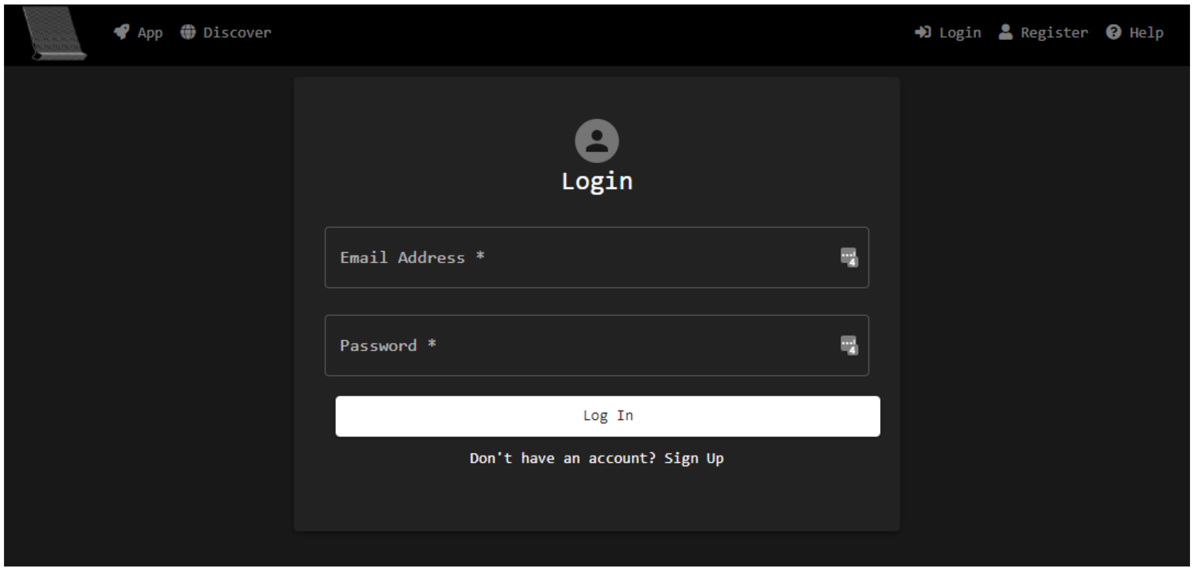
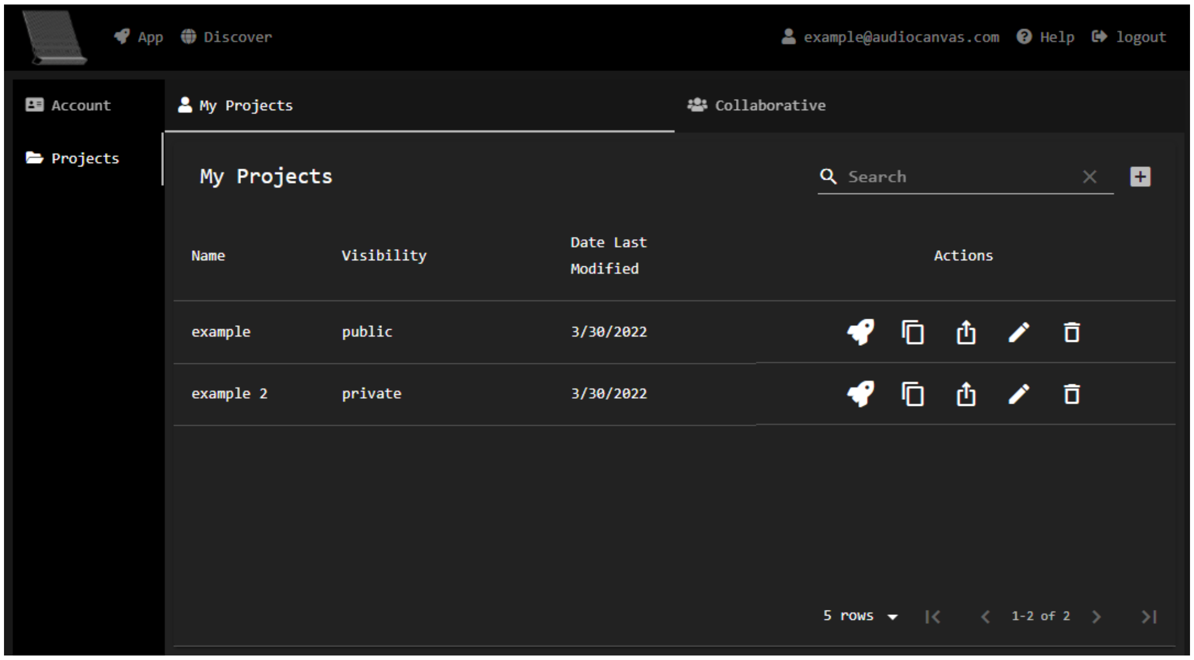


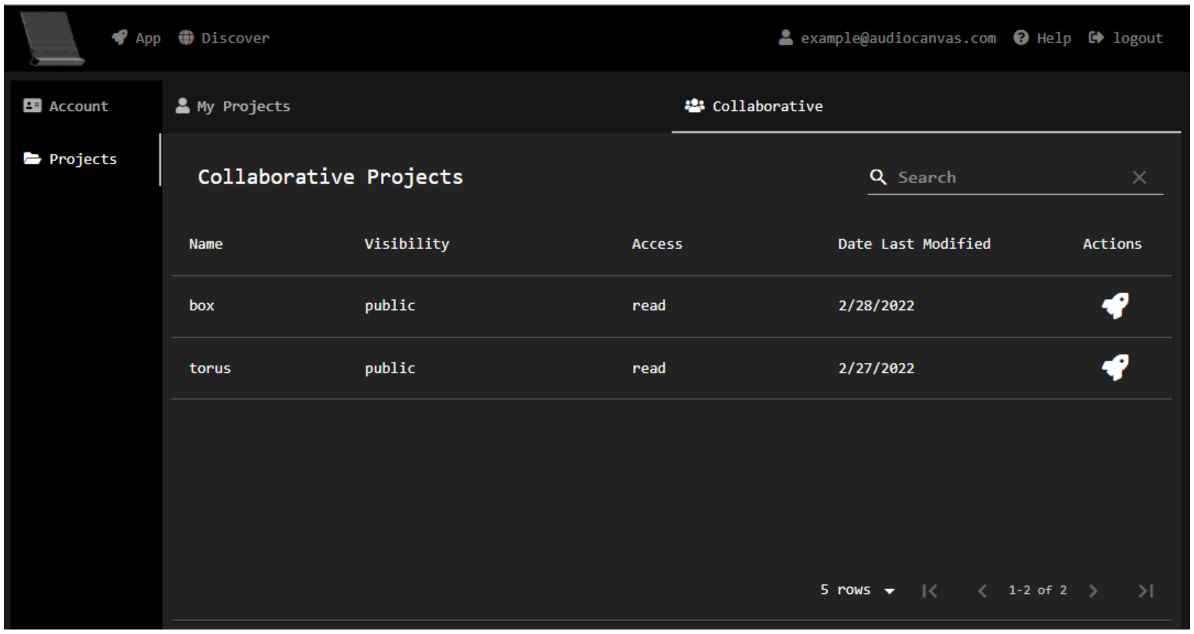
Figure 27. Login Page



My Projects

Name	Visibility	Date Last Modified	Actions
example	public	3/30/2022	
example 2	private	3/30/2022	

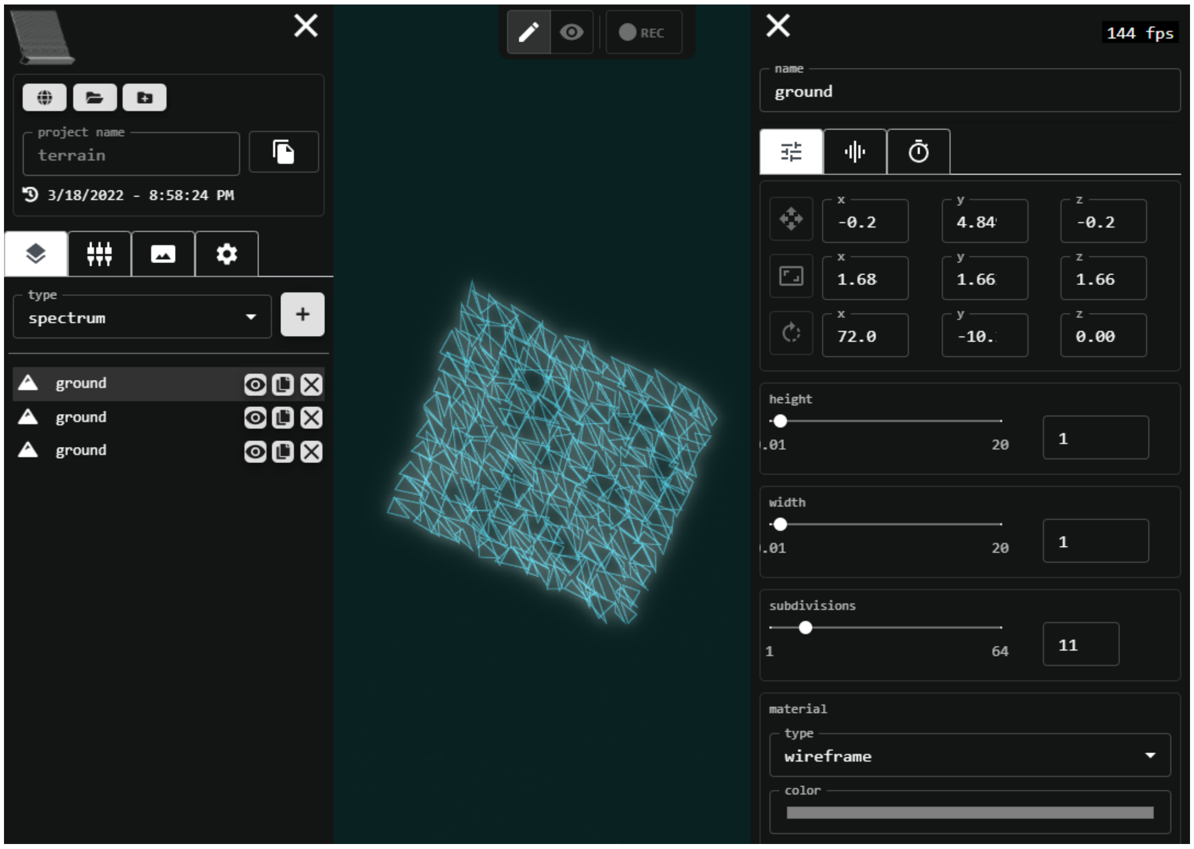
Figure 28. My Projects Page



Collaborative Projects

Name	Visibility	Access	Date Last Modified	Actions
box	public	read	2/28/2022	
torus	public	read	2/27/2022	

Figure 29. Collaborative Projects Page



project name: terrain
3/18/2022 - 8:58:24 PM
type: spectrum

ground
ground
ground

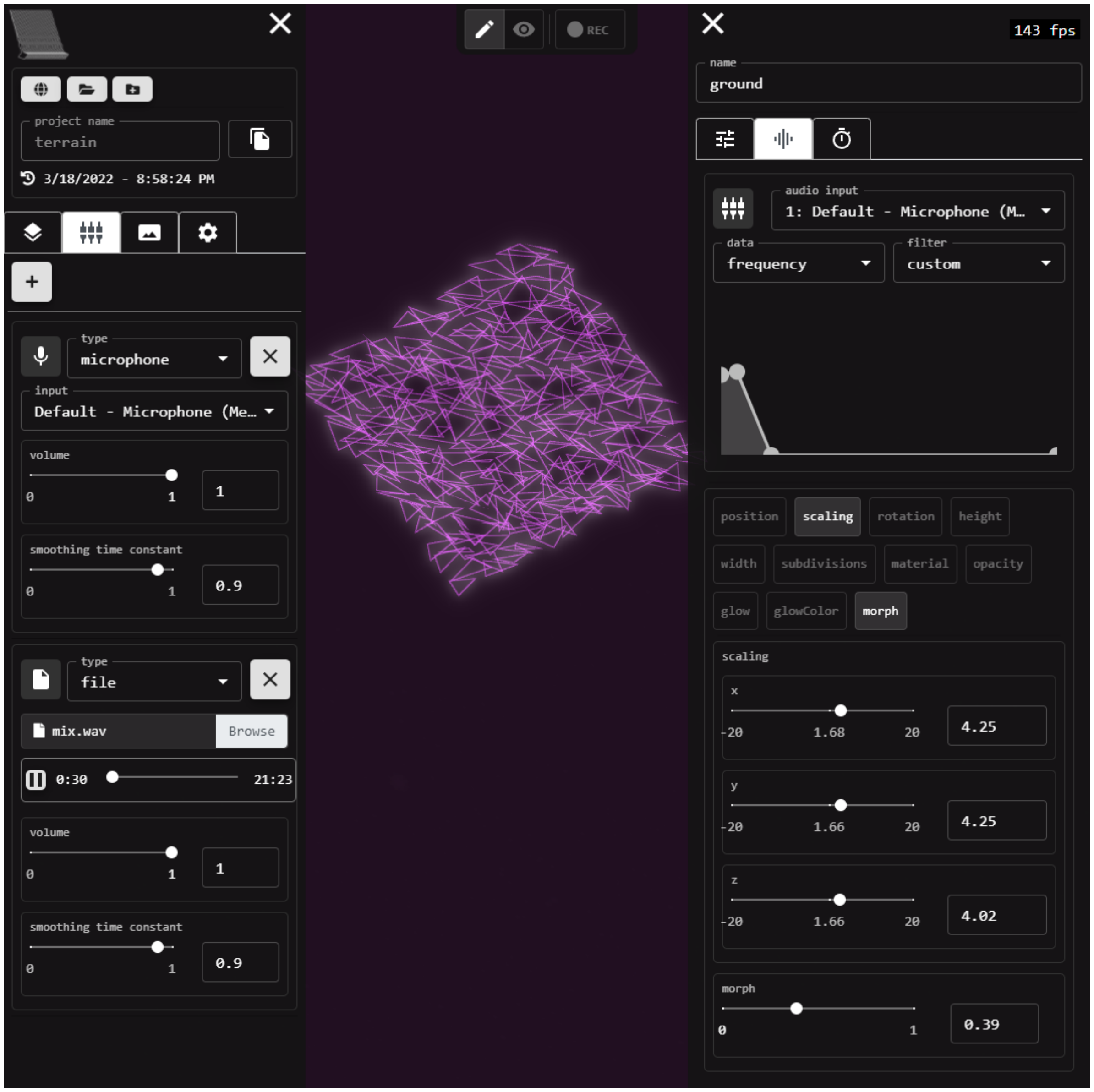
name: ground

	x	y	z
	-0.2	4.84	-0.2
	1.68	1.66	1.66
	72.0	-10.	0.00

height: .01 – 20: 1
width: .01 – 20: 1
subdivisions: 1 – 64: 11

material
type: wireframe
color

Figure 30. Application Page - Components



project name
terrain
3/18/2022 - 8:58:24 PM

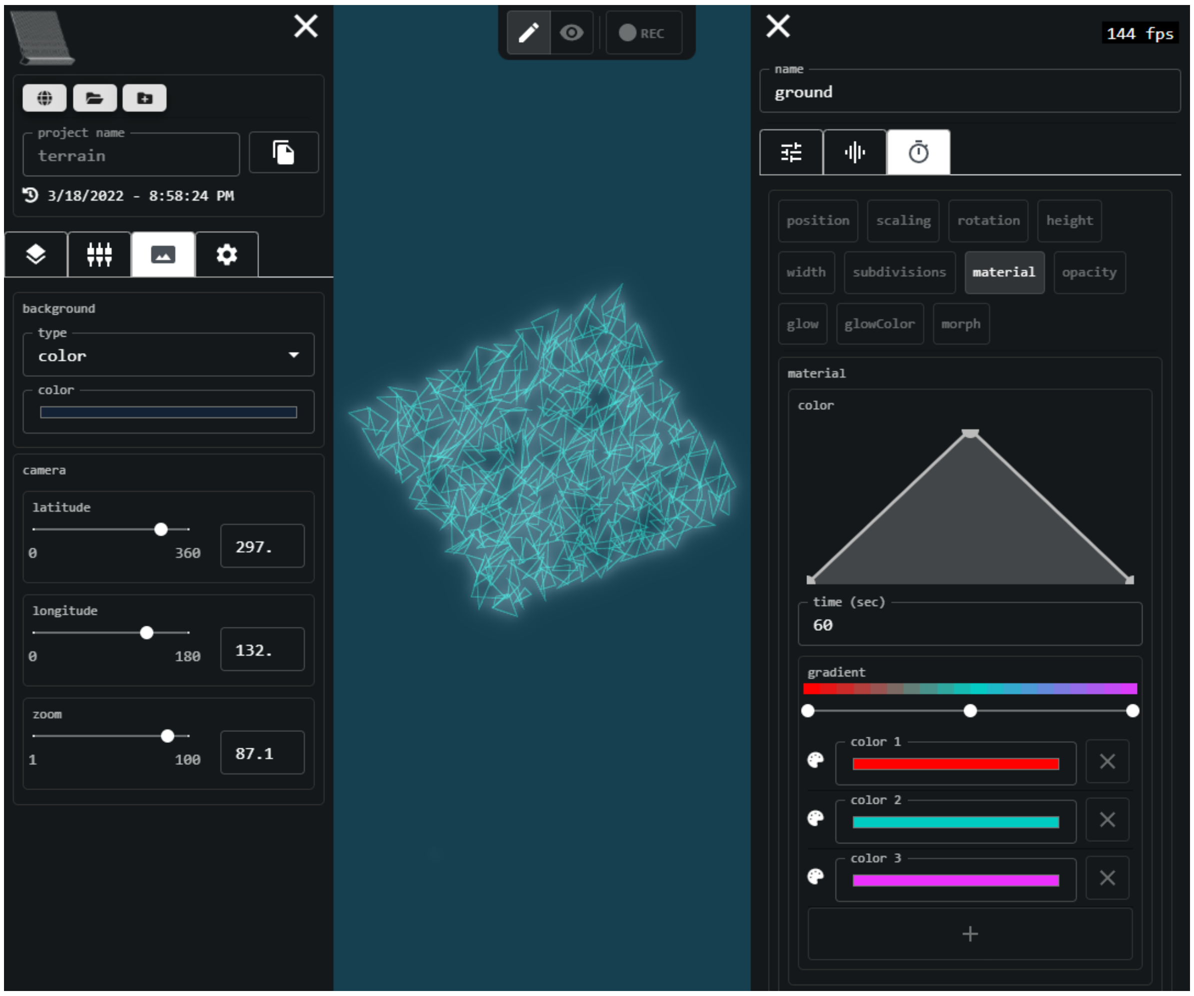
type
microphone
input
Default - Microphone (Me…
volume
0 1
1
smoothing time constant
0 1
0.9

type
file
mix.wav
0:30 21:23
volume
0 1
1
smoothing time constant
0 1
0.9

name
ground
audio input
1: Default - Microphone (M…
data
frequency
filter
custom

scaling
x
-20 1.68 20
4.25
y
-20 1.66 20
4.25
z
-20 1.66 20
4.02
morph
0 1
0.39

Figure 31. Application Page - Audio Mapping



project name
terrain

3/18/2022 - 8:58:24 PM

background
type
color
color

camera
latitude 0 360 297.
longitude 0 180 132.
zoom 1 100 87.1

144 fps

name
ground

material
color
time (sec)
60
gradient
color 1
color 2
color 3

Figure 32. Application Page - Shared Settings and Time Mapping