

Project Report

Department of Computer Science

CPSC 490 Undergraduate Seminar in Computer Science - **Proposal for Capstone Project**

Project title	Personal Audio Mixer Scene Manager		
Semester:	Fall, 2022	Proposal date:	December 17, 2022
Team name:	WeDev		
Member name:	Cooper Santillan	Email	CooperSan@csu.fullerton.edu
Member name:	Marius Iacob	Email:	mariusiacob@csu.fullerton.edu
Member name:	Christopher Melgar Morales	Email:	cmelgarmorales@csu.fullerton.edu
Member name:	Brennon Hahs	Email:	brennonhahs@csu.fullerton.edu
Member name:	Brian Miller	Email:	bcmiller@csu.fullerton.edu
Faculty advisor:	Kyoung Shin	Reviewer (optional)	

Abstract

Our group is partnering with a live entertainment production venue to solve a weekly frustration regarding their audio distribution during performance sets. In the venue, an audio engineer is responsible for distributing at least 12 different stereo audio mixes to various musicians, all unique levels and pans per channel at multiple settings. Our software will be able to capture the current state of the professional audio mixer, save the necessary values, and store captured data for later use. Additionally, the software will be able to retrieve the data previously stored and recall the state to the external professional audio console with only the specified values.

The company we are working with is Eastside Christian Church in Anaheim. They are encountering a problem with their musicians being unable to save their personal mixes per week. Most of their musicians do not play consecutively, but some will be there every week, some come back every other week, and some once a month. Because of their lack of attendance and the inconsistency of musicians using the same mix number, an audio engineer spends a significant amount of time trying to replicate a good level adjustment on their mix that was achieved at the end of a previous week's service. This application will allow the audio engineer to spend less time doing repetitive work and free more time doing other duties.

1. Introduction

Throughout the entire live music industry, there is a constant challenge when it comes to a rotating set of musicians from one venue to the next. Our software will allow musicians to save their mix, levels, and panning, from one audio console/venue to another venue. This will allow less headache for musicians in the back and forward communication to the audio engineer and enable the engineer to focus on audio processing, such as equalization and compression. Currently, no applications are accessible for musicians to do this under any interface.

There must be an understanding of sending UDP packets under the format of a network entertainment standard known as OSC to participate in this project. The documentation for this professional audio console we are using is a Yamaha CL5. This console has no official documentation to interfacing with it; however, there is unofficial API documentation from various Github projects recording known commands. To enable our understanding of proper communication between the professional audio console and the software we will implement, we will utilize WireShark for all UDP networks.

Related Work

Describe the related or existing work in detail. This section is like a survey on the selected problem or topic.

<https://mixingstation.app> : Currently, there is one similar application that is capable of moving personal musician mix settings from one mix to another. However, this application does not support the audio console we are trying to work with.

https://cz.yamaha.com/files/download/other_assets/5/1407565/RIVAGE_PM_osc_specs_v102_en.pdf: This is the official documentation for the company's flagship professional audio console.

There is no official documentation that the flagship network commands will maintain compatibility with the audio console will be solving. However, there is virtually complete compatibility upon utilization with others.

https://github.com/btendrich/yamaha_osc_translator : This is an extension of the official flagship models documentation with founded more nuance specifications not directly explained by the company.

Problem Statements

Musicians cannot keep their preferred levels and pans on audio consoles from venue to venue.

Typically this requires the band to arrive early at the venue and spend time sound checking and trying to tune the mixing station to the band's requirements.

2. Goals and Objectives

Our first goal is to capture, in mass, a large amount and, without any interruption, real-time data for the production of the show and then to store it in a database specified per musician within the requested mix. The first objective is to understand the network communication from an audio console in order to receive the desired data. For this project, it would be the audio levels and panning. The second objective is for us to parse that network response from the audio console into a non-relational database structure. Thirdly, we must use the database later to recall that data onto the same or different connected console.

Another goal is for the software to be able to load data from one audio console brand, such as Yamaha, onto another audio console brand, such as Allen & Heath. The first objective is to complete the first goal but in a structure that is abstract enough to communicate data from different brands of consoles.

Our final goal is for our application to be synchronized through interactions with a web server or cloud provider to enable synchronization of console configurations when stored and accessed from a created account.

3. Proposed Approaches

Some possible strategies are to use a low-powered barebone computer to host a local website that interacts within itself with a database of the various musicians' preferred audio mixes; The advantage is availability in the environment, but the disadvantage is the requirement for clients to introduce external devices and peripherals to utilize the system. Another strategy is to host the application on a mobile device (android or iPad) and store a database of various musicians' unique mixes on the local device; the advantage is that there is no external hardware needed to maintain it. Another approach would be hosting the application exclusively as a web application, allowing any user with access to a web browser to get and set the configuration of their preferred audio controller, which also has the advantage of needing only the initial request for the website and potential API calls for profile data stored on the online server.

Proving point of concept with Terminal commands

TERMINAL COMMAND: nc 172.20.10.143 49280

- The *nc* command is an arbitrary TCP and UDP connection program interfaced using the terminal/command line. We first provide the IP address of the host console (172.20.10.143) and the port (49280), and the program creates a simple TCP connection that we can type ASCII text on the blank newline following entering this command. To end the connection, we would just need to press the **Ctrl+C** keystroke to end. Below are examples of what happens when entering certain strings to the Yamaha CL5 console and the responses it gives back to us, the user.

Notes overall: Here are some references of interest in knowing the structure of how the console interprets data.

- last three numbers (channel number starting index 0) (Mix number; each mix is stereo, so need to check two) (if setting: supply number to set to)
- if changing one mix (left or right side), it automatically changes both sides

Get the levels of the mix (one channel at a time)

- From channel 35 to mix 2, the level is -25

[USER INPUT]: get MIXER:Current/InCh/ToMix/Level 34 1

[SERVER RESPONSE]: OK get MIXER:Current/InCh/ToMix/Level 34 1 -25

Get the panning from the mix of the channel

- Get the panning for channel 12, for mix 3
- A negative number refers to the left ear (0 is in the middle), and positive is right ear

[USER INPUT]: get MIXER:Current/InCh/ToMix/Pan 12 2

[SERVER RESPONSE]: OK get MIXER:Current/InCh/ToMix/Pan 12 2 -5 "-5"

Get whether the channel is on or off from the mix

- Request if channel 13 of mix 13 is on

[USER INPUT]: get MIXER:Current/InCh/ToMix/On 12 12

[SERVER RESPONSE]: OK get set MIXER:Current/InCh/ToMix/On 12 12 1 "ON"

When the channel is off on the mix

[USER INPUT]: get MIXER:Current/InCh/ToMix/On 12 12

[SERVER RESPONSE]: set MIXER:Current/InCh/ToMix/On 12 12 0 "OFF"

Get the name of that channel

- Requesting the channel name for channel 8

[USER INPUT]: get MIXER:Current/InCh/Label/Name 7 0

[SERVER RESPONSE]: OK get MIXER:Current/InCh/Label/Name 7 0 "Tom 2"

4.Required Environment, Resources, and Planned Activities

Required Functions for Application Diagrams

Get Profile Data

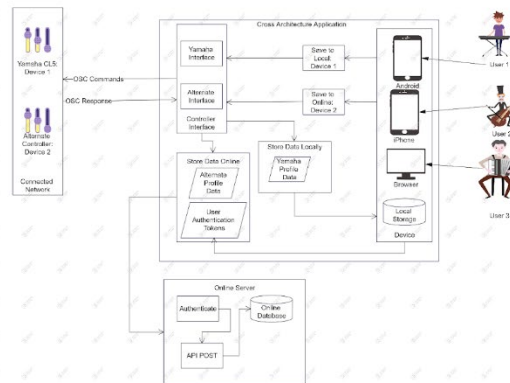


Fig 1: Logic for getting profile data from different platforms, and saving locally or online

The application itself must follow the above diagram to allow saving of an audio controller's configuration from a variety of end-user devices (Android, iOS and PC through browser). A user will use a device that initiates communication to an audio controller connected to the same network as the user device, which allows multiple OSC commands to be sent in order to retrieve the current configuration of the audio controller. The configuration will then be saved into the local storage of the end user's device for later use. The diagram includes the logic needed to get the configuration of audio controllers of both our chosen controller for prototyping (the Yamaha CL5) and other audio controllers, as well as the logic needed to store the configuration data on an online server. For the online storage of audio controller configurations, we will need to implement authentication measures to ensure profiles between users remain separate and unmodified without proper credentials.

Set Profile Data

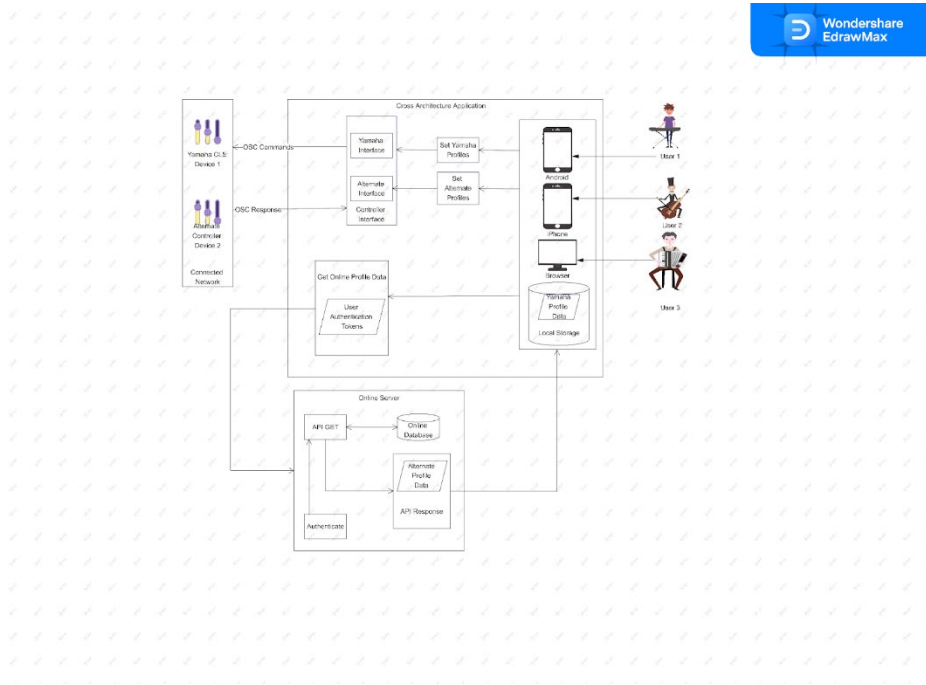


Fig 2: Logic for setting profile data from different platforms, from local or online save

The application must follow the above diagram to allow setting of an audio controller's configuration from a variety of end-user devices (Android, iOS, and PC through a browser). A user will use a device that initiates communication to an audio controller connected to the same network as the user device, which allows multiple OSC commands to be sent, that set the values of the different mixes and channels for an audio controller given a previously saved configuration. The diagram includes the logic needed to set the configuration of an audio controller for both our chosen controller for prototyping (the Yamaha CL5) and different audio controllers, as well as the logic needed to retrieve the configuration data from local storage and potentially an online server. For the online retrieval of audio controller configurations, it is not a severe security issue if users can read other users' controller information. However, separating

the retrieved profiles by authenticating first is necessary to reduce the overall data sent between the online server and the end-user device.

Required Environments and Tools for Testing and Building the Application

GitHub and Git

To ensure easy management of different versions of our source code for the application,

GitHub and Git will be needed. We will have to host our files on GitHub, so that synchronization between builds and developers in our group remains consistent.

React Native

Since we plan to allow our application to run between architectures like x86_64 for PCs running Linux, Windows, and macOS. ARM will be used for mobile devices running Android or iOS since we need a framework that allows for cross-architecture development. React Native, as described by Wikipedia, is an “open source UI software framework” which can run the same source code natively on different devices, especially for mobile platforms like iOS and Android. For React Native to run correctly on a development environment for our purpose of sending OSC commands, we will need *Node.js*, *NPM and its Command Line Interface*, and *Expo and its Command Line Interface*.

Node.js

React Native requires a runtime environment for JavaScript to run correctly, so we will be using the java script runtime environment, Node.js, for our application stack.

NPM and its Command Line Interface

NPM is "the world's largest software registry," according to their own website at <https://docs.npmjs.com/about-npm> . There are a large number of packages hosted on the site for retrieval and use in React and React Native applications. Will be importing packages from NPM to reduce the time needed to develop the application, such as with packages like NPM-OSC, which is a library for interacting with devices through the OSC protocol.

Expo and its Command Line Interface

In order to run the development version of our application, the open-source framework Expo will be needed. React Native applications can normally be built using tools provided by the IDE and SDK for Android and iOS, like Android Studio and Xcode. However, minimizing the tools needed to develop for all platforms will allow us to code and build fast without getting used to different development tools and their virtual environments. Expo allows for a React Native application to be hosted and viewed in any browser(for PC development and potential production use) and as an application for Android or iOS. It is unknown to us whether we can create a standalone build for PC without the prior development environment. However, Expo does allow for Android and iOS applications to be built in their native application format, like *.aab* for Android and *.ipa* for iOS.

Flask

During our research into creating an environment that can send TCP commands, we discovered that when using React Native as a front end, it was unlikely that we would be able to send data in the way the audio console expects. While React Native has methods to transmit information using TCP methods, there is extraneous information that should be excluded, as the console will not recognize the commands. Thus we need to move the sending of TCP commands to a system that works similarly to our proof of concept with the Netcat executable. Flask is a micro service framework that can receive API calls and execute the needed commands in a python script. Python has packages for sending TCP data, which means that our application will use a back end powered by Rest API through Python.

Docker and Docker Desktop

Since it will be difficult to synchronize the dependencies and development environment between our group members, as we all have our own devices, using Docker and Docker Desktop will be the simplest way to ensure everyone on the team can develop our application without errors due to dependencies or different operating systems. Docker is a method for building virtual environments that can be easily executed between devices and is well known for allowing instances of programs and applications to start and shut down to meet the necessary computation load. For our application, it will be essential to use these Docker containers to synchronize development between our different group mates operating systems and architecture. As long as our members build the configured container on their local operating system and have

virtualization enabled, they will have immediate access to the same tools and dependencies as every other group member.

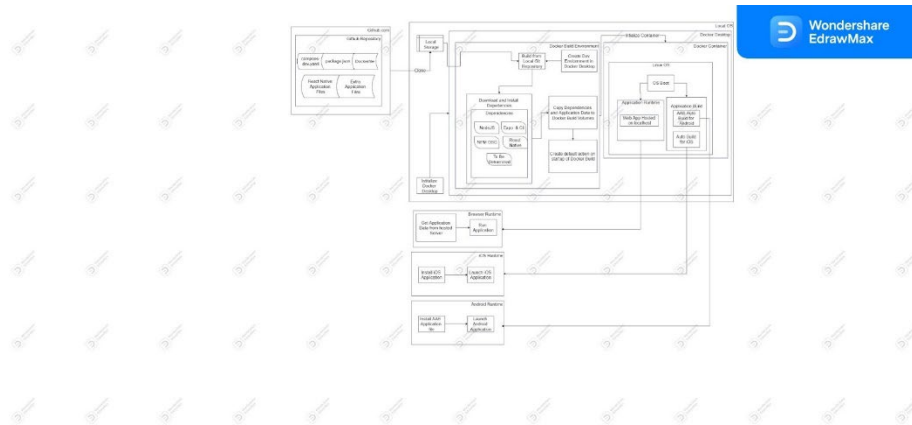


Fig 3: Logic for creating a synchronized development environment and runtime builds of the application across platforms.

The diagram above is the event chain that we will need to implement in order to have synchronized development and production builds between our group members. It will be necessary for our developers to clone a GitHub repository that contains different configuration files, such as `compose-dev.yaml` and `Dockerfile` for building our Docker build and container, as well as `package.json` for installing the required NPM packages needed to run our React Native Application. This will allow any one of our developers to open up Docker Desktop on their Local Operating System and choose to build our development environment from the Github repository and run it in a Linux Virtual Machine for easy hosting of the Web application and Auto Building of the Android and iOS application builds, as shown in the diagram above.

Required Resources for Storing Application Data:

As of writing this proposal, we have currently settled on storing the configurations of audio controllers in JSON files in the format as follows

```
{
  "Kendalls Mix":[
    "1": {"Name": "Bass", "Level":-4, "Pan":35, "On":false},
    "2": {"Name": "Kick", "Level":-24, "Pan":0, "On":true},
    "3": {"Name": "AG", "Level":0, "Pan":-25, "On":true},
    ...,
    "72": {"Name": "Mic 4", "Level":3, "Pan":5, "On":true}
  ],
  "Toms Mix":[
    "1": {"Name": "Bass", "Level":0, "Pan":0, "On":true},
    "2": {"Name": "Kick", "Level":-37850, "Pan":-20, "On":true},
    "3": {"Name": "AG", "Level":-3, "Pan":5, "On":true},
    ...,
    "72": {"Name": "Mic 4", "Level":-10, "Pan":5, "On":false}
  ]
}
```

Fig 4: Proposed JSON format for storing configuration data from Yamaha CL5

Local Storage

While React Native allows for applications to be run on multiple architectures seamlessly, it may not be possible to use the same method to store data locally between PC, Androids, and iOS. The simplest method to use for storing said JSON data would be to use AsyncStorage, an NPM package for storing data on React Native applications built for the Web, Android, and iOS. According to the GitHub page for AsyncStorage[<#>], the package does support web development, but it may be necessary to use a database such as SQLite for storing controller profiles when our application executes on a PC. That means it will be necessary to plan and implement a database model that either tracks the JSON files for our profile data or a model that supersedes the JSON format and stores the values directly into the database.

Online Storage

For saving and retrieval of profile data to an online web server, requiring a cloud provider like Amazon Web Service could help to streamline the process of both authentication and data retrieval. AWS has tools and documentation for implementing Authentication protocols from creating, verifying, and logging into user accounts that work with React Native. If we were to work with AWS for storing our user profile data online, we would also need to use a service with easy compatibility with our existing JSON format, which is supported by Amazon DocumentDB.

As shown in the diagrams above, our application must be able to accommodate users

5. Project Outcomes

At the very least, we will deliver a program that takes audio levels and settings from an audio interface and stores them in a database for later use. Ideally, we would like to deliver a program that can store data from one type/brand of audio interface and recall that data on a different type/brand. Additionally, a program that could synchronize these audio interfaces through a web server based on a user's account would be fantastic.

As we work on the project, we will jot down the process that a user needs to follow in order to get their desired outcome. In essence, this would be a user manual of sorts, potentially written as a PDF document, detailed in a Wiki format on Github, or implemented directly into our application itself.

If needed, we can deliver all of our source code through our GitHub repository or another file-sharing platform. Since this project is focused on cross-platform development, our goal is for the source code for the React Native application to remain almost identical between versions. From our source code, a user of our application should be able to set up local versions of the application on their Android or iOS device and set up a production build hosted either on their hardware or hardware through a cloud provider. We also plan on delivering prebuilt software packages for users inexperienced with building applications from source code to ensure their accessibility.

For the data and database of our application, we plan on setting up instructions on setting up a database for the cloud/web server build of our product and providing a configuration file for each

console we support that interacts with our interfaces for storing and setting console data, to simplify the process of supporting additional consoles, whether it be from our tested and configured consoles, or other consoles that are not yet supported.

6. UI Sketch

+ New Mix + Modify Mix	Kendall's Mix					
Jamie's Mix Dante's Mix Test Check Concerto Kendall's Mix Wacky Wham Tribute Mix						
	Channel 1					
	Channel 2					
	Channel 2					
	Channel 3					
	Channel 4					
	Channel 5					
	Channel 6					
	Channel 7					
	Channel 8					

Fig 5: Desktop UI Sketch

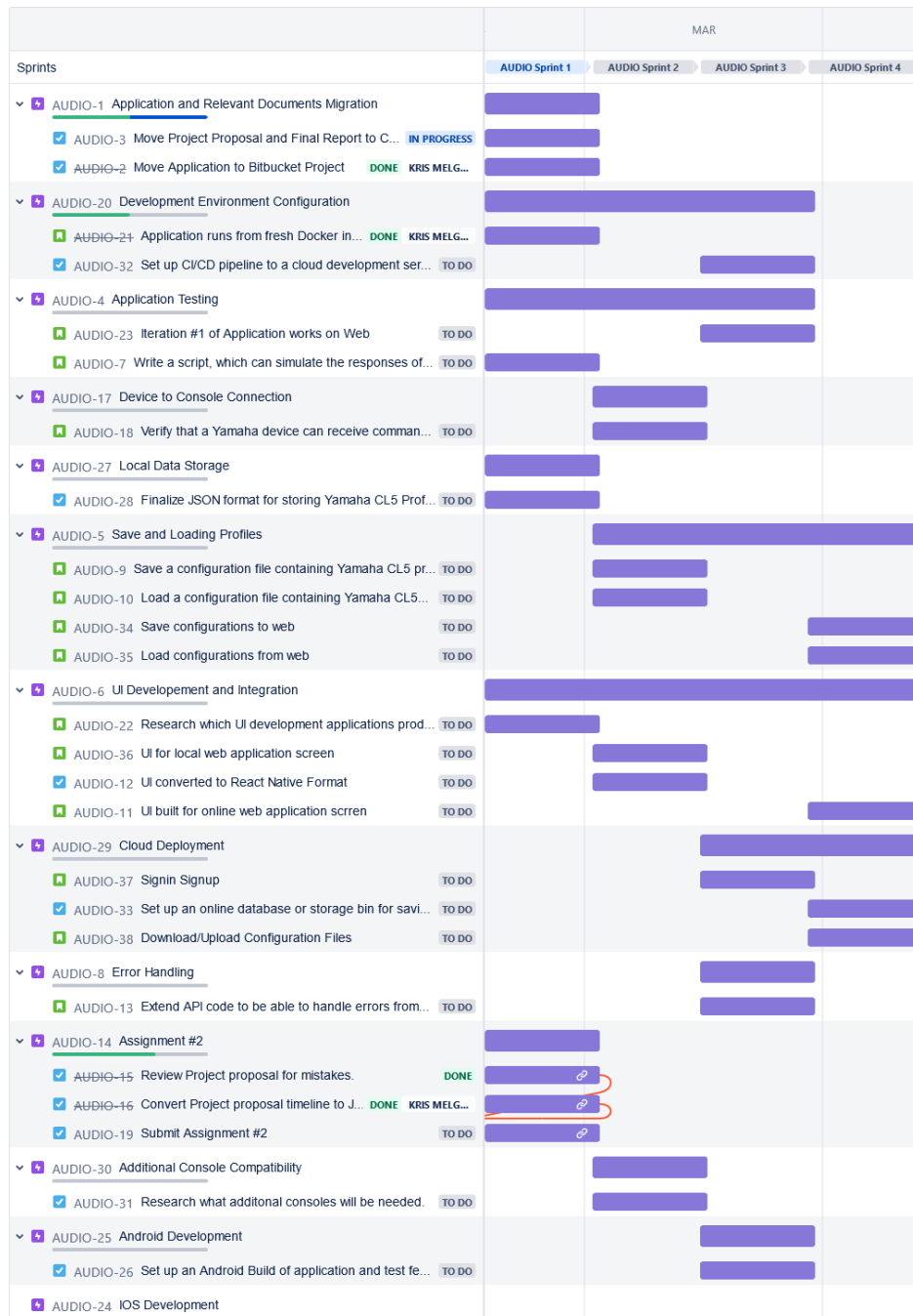
7. Project Timeline-Fall 2022



Fig 6: Original proposed project stages for application from prototype to final local and online production build

In our initial project proposal we had written task to for each stage of development for the project. The format for these task was not consistent with methods used by professional companies, so these tasks have been migrated to the Jira software stack, as tasks and stories.

These new task and stories have been distributed across the 8 weeks for the following 4 sprint schedules, with descriptions and points assigned to each issue.



In sprint 1, we need to migrate to the services provided by Atlassian with Confluence, Jira, and Bitbucket, while starting to work on testing software. For further development, having a

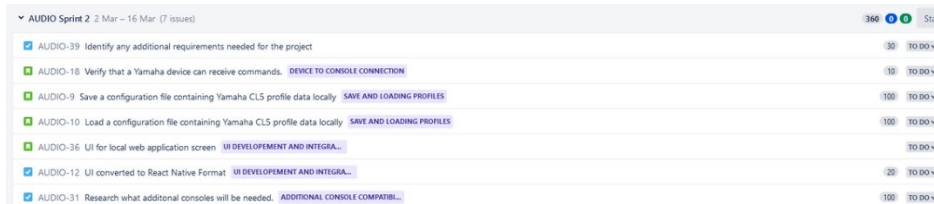
simulated console will allow our developers to write programs for the Yamaha CL5 without access to the console at the church. These weeks are mainly for setting up our Agile workflow, and and preparing to work on key features.



A screenshot of a Jira sprint backlog for 'AUDIO Sprint 1' (16 Feb - 2 Mar, 9 issues). The sprint has 160 points. The backlog items are as follows:

Issue ID	Summary	Category	Points	Status
AUDIO-3	Move Project Proposal and Final Report to Confluence	APPLICATION AND RELEVANT DOC...	30	IN PROGRESS
AUDIO-8	Move Application to Bitbucket Project	APPLICATION AND RELEVANT DOC...	30	DONE
AUDIO-24	Application runs from fresh Docker initialization	DEVELOPMENT ENVIRONMENT CO...	30	DONE
AUDIO-28	Finalize JSON format for storing Yamaha CL5 Profile data	LOCAL DATA STORAGE	50	TO DO
AUDIO-7	Write a script, which can simulate the responses of a Yamaha CL5.	APPLICATION TESTING	50	TO DO
AUDIO-22	Research which UI development applications produce React Native code	UI DEVELOPEMENT AND INTEGRA...	50	TO DO
AUDIO-15	Review Project proposal for mistakes.	ASSIGNMENT #2	40	DONE
AUDIO-16	Convert Project proposal timeline to Jira issues.	ASSIGNMENT #2	70	DONE
AUDIO-19	Submit Assignment #2	ASSIGNMENT #2	10	TO DO

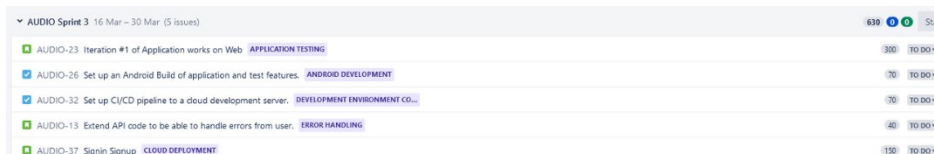
In sprint 2, we will work on the critical feature of the application, and develop the UI to have a polished prototype.



A screenshot of a Jira sprint backlog for 'AUDIO Sprint 2' (2 Mar - 16 Mar, 7 issues). The sprint has 360 points. The backlog items are as follows:

Issue ID	Summary	Category	Points	Status
AUDIO-39	Identify any additional requirements needed for the project		30	TO DO
AUDIO-18	Verify that a Yamaha device can receive commands.	DEVICE TO CONSOLE CONNECTION	10	TO DO
AUDIO-9	Save a configuration file containing Yamaha CL5 profile data locally	SAVE AND LOADING PROFILES	100	TO DO
AUDIO-10	Load a configuration file containing Yamaha CL5 profile data locally	SAVE AND LOADING PROFILES	100	TO DO
AUDIO-36	UI for local web application screen	UI DEVELOPEMENT AND INTEGRA...		TO DO
AUDIO-12	UI converted to React Native Format	UI DEVELOPEMENT AND INTEGRA...	20	TO DO
AUDIO-31	Research what additional consoles will be needed.	ADDITIONAL CONSOLE COMPATIBL...	100	TO DO



In sprint 3, our application should be complete for the local web version, and we will explore and setup our software for running on Android and through a cloud provider on the web.





A screenshot of a Jira sprint backlog for 'AUDIO Sprint 3' (16 Mar - 30 Mar, 5 issues). The sprint has 630 points. The backlog items are as follows:

Issue ID	Summary	Category	Points	Status
AUDIO-23	Iteration #1 of Application works on Web	APPLICATION TESTING	300	TO DO
AUDIO-26	Set up an Android Build of application and test features.	ANDROID DEVELOPMENT	70	TO DO
AUDIO-32	Set up CI/CD pipeline to a cloud development server.	DEVELOPMENT ENVIRONMENT CO...	70	TO DO
AUDIO-13	Extend API code to be able to handle errors from user.	ERROR HANDLING	40	TO DO
AUDIO-37	Signin Signup	CLOUD DEPLOYMENT	150	TO DO

In sprint 4, we will finalize the features for the online web version, and prepare additional UI elements for features that aren't part of the local version.

▼ AUDIO Sprint 4 30 Mar – 13 Apr (5 issues)			389			Star
<input checked="" type="checkbox"/>	AUDIO-33 Set up an online database or storage bin for saving user data.	CLOUD DEPLOYMENT	80	TO DO	▼	
<input checked="" type="checkbox"/>	AUDIO-38 Download/Upload Configuration Files	CLOUD DEPLOYMENT	50	TO DO	▼	
<input checked="" type="checkbox"/>	AUDIO-34 Save configurations to web	SAVE AND LOADING PROFILES	100	TO DO	▼	
<input checked="" type="checkbox"/>	AUDIO-35 Load configurations from web	SAVE AND LOADING PROFILES	100	TO DO	▼	
<input checked="" type="checkbox"/>	AUDIO-11 UI built for online web application screen	UI DEVELOPMENT AND INTEGRA...	50	TO DO	▼	

#.Functional Requirements

Target release	17 Mar 2023
Epic	 AUDIO-56 - Script returns profile data when asking for state of Yamaha CL5 in code review  AUDIO-57 - Script accepts profile data when uploading new state to Yamaha CL5 In Progress
Document status	DRAFT
Document owner	Kris Melgar Morales
Designer	Kris Melgar Morales
Tech lead	Kris Melgar Morales
Technical writers	Kris Melgar Morales
QA	


Objective

Create a python script that allows a user to spin up a fake Yamaha CL5 consoles that can respond the commands sent over TCP.

🧐 Assumptions

User already has access to a python interpreter. User has knowledge on how use the python scripting language to import the emulator into their code wherever needed.

📋 Requirements

Requirement	Importance	Jira Issue	Notes
The emulator must be able to receive TCP messages from all IP address, and it must return the data request from the state of the emulator as a TCP message.	HIGH	 AUDIO-56 - Script returns profile data when asking for state of Yamaha CL5 in code review	

🎨 User interaction and design

```
# get the levels of the mix (one channel at a time)
--> from channel 35 to mix 2, the level is -25
get MIXER:Current/InCh/ToMix/Level 34 1
OK get MIXER:Current/InCh/ToMix/Level 34 1 -25

# get the panning from the mix of the channel
--> get the panning for channel 12, for mix 3
get MIXER:Current/InCh/ToMix/Pan 12 2
OK get MIXER:Current/InCh/ToMix/Pan 12 2 -5 "-5" --> it is 5 to the left (0 is in the middle)


# get whether the channel is on or off from the mix
--> request if channel 13 of mix 13 is on
get MIXER:Current/InCh/ToMix/On 12 12
OK get set MIXER:Current/InCh/ToMix/On 12 12 1 "ON"
--> when the channel is off on the mix
get MIXER:Current/InCh/ToMix/On 12 12
set MIXER:Current/InCh/ToMix/On 12 12 0 "OFF"

# get the name of that channel
--> requesting the channel name for channel 8
get MIXER:Current/InCh/Label/Name 7 0
OK get MIXER:Current/InCh/Label/Name 7 0 "Tom 2"

# Notes overall
last 3 numbers (channel number starting index 0) (Mix number ; each mix is stereo so need to check 2) (if setting: supply number to set to)
if changing one mix (left or right side) it automatically change both sides
```

❓ Open Questions

Question	Answer	Date Answered
Why is there a merge into master for loading data into the emulator while its still incomplete?		

Target release	23 Mar 2023
Epic	 AUDIO-40 - Write API call to gather data from Yamaha Console. in code review  AUDIO-46 - Write API call to load data to a Yamaha Console. in code review
Document status	DRAFT
Document owner	Kris Melgar Morales
Designer	Kris Melgar Morales
Tech lead	Kris Melgar Morales
Technical writers	Kris Melgar Morales
QA	


Objective

Create two API calls within Flask and Python that can communicate with a Yamaha CL5 to retrieve all information about its state or load new information to its state.

Assumptions

User has access to Frontend and Backend containers of React Native application and they are running.

Requirements

Requirement	Importance	Jira Issue	Notes
User must be able to call a function that sends an API request to the backend, to gather and return the state of the Yamaha CL5 as a json file.	HIGH	 AUDIO-40 - Write API call to gather data from Yamaha Console. in code review	Preferably JSON data is saved/downloaded to local storage and works with all platforms

User interaction and design

File should be save in format decided in  [AUDIO-28](#) - Finalize JSON format for storing Yamaha CL5 Profile data Done .


```
{
  "filename": "CL5.json",
  "version": "0.1",
  "user": "a5e135c",
  "mixes": [
    {
      "1": {
        "1": {
          "Name": "Bass",
          "Level": -32768,
          "Pan": 35,
          "On": false
        },
        "2": {
          "Name": "Guitar",
          "Level": 10,
          "Pan": 35,
          "On": true
        },
        "3": {
          "Name": "Piano",
          "Level": 100,
          "Pan": 35,
          "On": false
        },
        "4": {
          "Name": "Mic",
          "Level": 50,
          "Pan": 35,
          "On": false
        }
      }
    }
  ]
}
```

Open Questions

Question	Answer	Date Answered


#.Implementation:

Yamaha CL5 Emulator:


Due to the team not having our own personal Yamaha CL5 to test with, there need to be a way to test our code according to the response of the CL5.As a result we created issue  [AUDIO-7](#) -

Write a script, which can simulate the responses of a Yamaha CL5. In Progress so that our developers could test their code against the expected responses.

A module called **emulated.py** was made, that would initialize a server that pretends to be a audio Console, based on a the configuration of the given Host, Port, Maximum Input Channels, and Maximum [Mixes](#). The function called **echoServer()** will take the parameters and host an emulator that responds based on the values of **dummy.json** in the same folder.

As required by  [AUDIO-56](#) - Script returns profile data when asking for state of Yamaha CL5 in code review , the emulator responds to the get commands a Yamaha CL5 console would respond with. Currently there is no valid handling of incorrect inputs, as the emulator expects commands in the correct format within the bounds of each type of argument.

Save and Downloading Yamaha Profile Data:

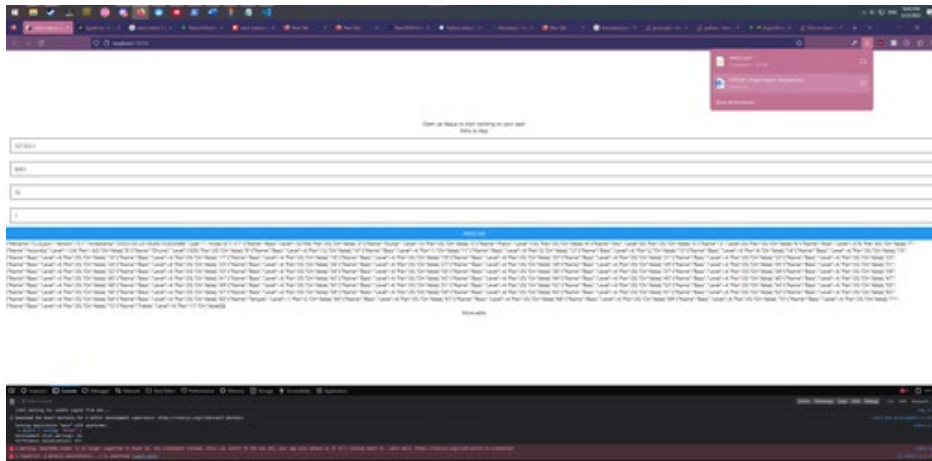
In the React Native web app, a user must be able to download the JSON data format of a Yamaha console to their device with an API call as describe in  [AUDIO-40](#) - Write API call to gather data from Yamaha Console. in code review . An API call was added to the Flask backend called **getConfigProfile** which will send messages to the Yamaha emulator or actual console

depending on the arguments passed in the JSON detailed in the API documentation. The flask API requests the information from the device and returns the data formatted back to the API caller.

A matching file and function called **getConfigProfile** to fulfill [🔊AUDIO-45](#) - Attach API call to react native application In Progress and was added to the React native workspace to fetch the JSON data from the Flask API and then return it to the React Native frontend.

In the case of users needing to move their configuration file data manually, we have implemented a Java script function that enables any JSON data to be downloaded to the local machine the web app is running on. This feature has been committed in [🔊AUDIO-54](#) - Download Console configuration to Device in code review as the function **saveData()**, which takes a JSON files and saves it locally.


To ensure cross compatibility, two version of each function definition were made, one targeting React Native for web using **File reader** object in javascript, and API calls to the backend in **functionsWeb.js**, and one targeting React Native for Android and IOS using **react-native-fs** for downloading files and **react-native-tcp-socket** for sending TCP message to the audio console which exist in the file **functionsMobile.js**.




Data Loading to Yamaha CL5


Loading data to a CL5 requires a specially formatted ‘set’ command. It goes something like ‘set <attribute> <mix> <channel> <value>’. Basically what this is saying is that it will load a value into a certain attribute of a channel inside of a mix that is in the CL5. For example loading a ‘pan’ value of 35 to mix 1, channel 1 would look like this:

set MIXER:Current/InCh/ToMix/Pan 1 1 35

A client was developed in order to send these kinds of commands  [AUDIO-10](#) - Load a configuration file containing Yamaha CL5 profile data locally In Progress to our emulated CL5. The client concatenates the specific profile data to their respective set commands and then sends them to the CL5.

Downloading JSON Files

In the case of users needing to move their configuration file data manually, we have implemented a java script function that enables any JSON data to be downloaded to the local machine the web app is running on. This feature has been committed in  [AUDIO-54](#) -

Download Console configuration to Device in code review with it tested through the use of the
getDummyProfile api call created in  [AUDIO-40](#) - Write API call to gather data from Yamaha
Console. in code review

#.Documentation:

Flask API Endpoints

App.getConfigProfile

Verb: GET

Path: /App/getConfigProfile

	JSON Parameter	type	Description
1	host	string	IP address of Audio Console
2	port	string number	port console is listening on
3	channel	string number	Max channel size of console
4	mixes	string number	Max mixes on console

5	isDummy	bool	Controls whether emulator of audio console is created when API is called
---	---------	------	--

Response

Content Type: application/json

Type: object

Object Properties

Response

Returns a JSON file containing information about Yamaha Console with respect to each channel and mix, and there values for the following paths.


```
validInfix = ['MIXER:Current/InCh/Label/Name',
              'MIXER:Current/InCh/ToMix/Level',
              'MIXER:Current/InCh/ToMix/Pan',
              'MIXER:Current/InCh/ToMix/On']
```

JSON Response should appear in this format

```
jsonFormat = {
    "filename": "CL5.json",
    "version": "0.1",
    "timestamp": 'temp',
    "user": "",
    "mixes": []}
```

#.Deployment and Testing


Known Issues:

Jira Issue	Description	Resolution
 AUDIO-63 - Backend and Frontend do not launch on Docker Intialization Done	When docker containers startup from a fresh build, both containers are unable to properly startup	Ensure line endings for files in /backend and /frontend are LF

Test and Merge:


Yamaha CL5 Emulator

#WRITE HERE ABOUT TESTING AND MERGING TO MASTER BRANCH OF COMMITS
RELATED TO

 [AUDIO-7](#) - Write a script, which can simulate the responses of a Yamaha CL5. In Progress

Save and Load API

#WRITE HERE ABOUT TESTING AND MERGING TO MASTER BRANCH OF COMMITS
RELATED TO

 [AUDIO-45](#) - Attach API call to react native application In Progress

#.References

- [1] Burges, C. J. C. Tutorial on Support Vector Machines for Pattern Recognition. Kluwer Academic Publishers, 1998.
- [2] Chen, P., Fan, R., and Lin, C. A study on SMO-type decomposition methods for support vector machines. IEEE Transactions on Neural Networks, 2006.
- [3] https://en.wikipedia.org/wiki/React_Native
- [4] <https://docs.npmjs.com/about-npm> , 11/22/2022
- [5] <https://github.com/react-native-async-storage/async-storage> , 11/22/2022