SYSC 3010 A

Computer Systems Development Project

iJukebox

Final Report

Group L3-G6

Flynn Graham, 101107831

Corbin Garlough, 101101493

Yunas Magsi, 101115159

TA: Roger Selzler

April 12th 2022

Table of Contents

Project Description **4**

Final Design Solution **4**

Deployment Diagram 5

Message Protocol Table 6

Sequence Diagrams 6

Discussion of Final Design **8**

Contributions **9**

Authors of all code pieces 9

Authors of each section of this document 11

Reflections **11**

Appendix A **14**

References **15**

Project Description

The Jukebox, a popular automated music-playing device most commonly used in the 1950s[1] has become outdated. Technologies such as smartphones, CDs, MP3 music players, etc have made them obsolete. Computer Systems Engineering students (Flynn Graham, Corbin Garlough, and Yunas Magsi) at Carleton University decided that now is the time that the Jukebox makes a comeback by introducing the iJukebox.

The iJukebox is a miniature Jukebox that sits in a common area, its main function is to read NFC cards and play songs while displaying the cover art of the song being played. These NFC cards are decorated with the artist's album cover and store values for albums from a specific artist. When a user picks an album they like, they will insert their card in the iJukebox and it will scan the NFC card to place a request for that song to play from a speaker.

Playing music is quick and convenient for the user using NFC technology. The user will still enjoy the aesthetic of having a record collection which they can play on a colorful Jukebox case. There is something rewarding about having tangible representations of music, and our project brings that aspect of music back into the limelight.

Final Design Solution

Figure 1 shows the deployment diagram for all the components used in the makings of the iJukebox. A further diagram in figure 2 shows how the majority of the Arduino wiring with multiple hardware components. The iJukebox has a NFC hat which can be viewed in more detail in the Github Readme.md file. Along with instructions on setting up the GUI interface the Speaker Pi, the iJukebox Pi and the Arduino.

Deployment Diagram

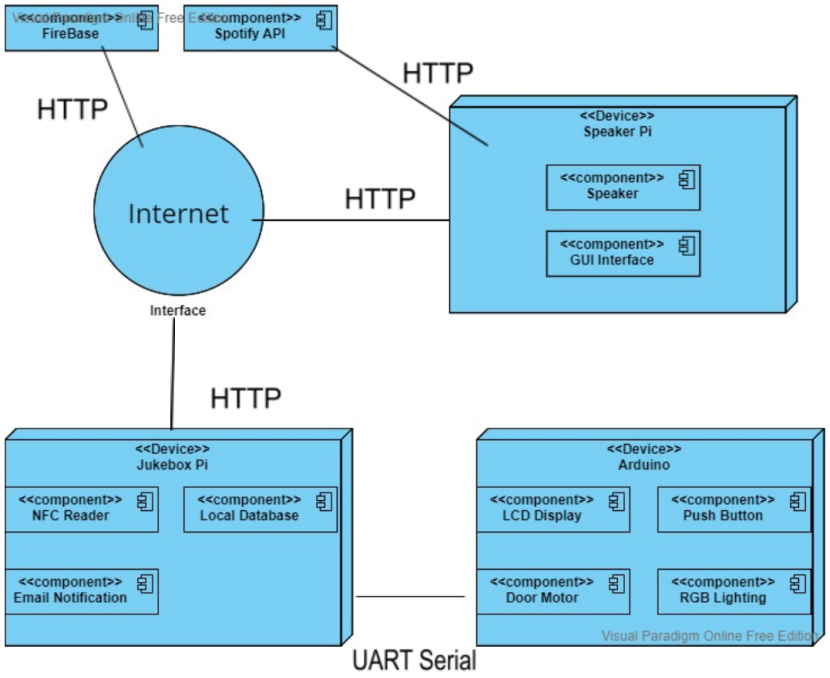

Figure 1: Final Deployment Diagram of the iJukebox System

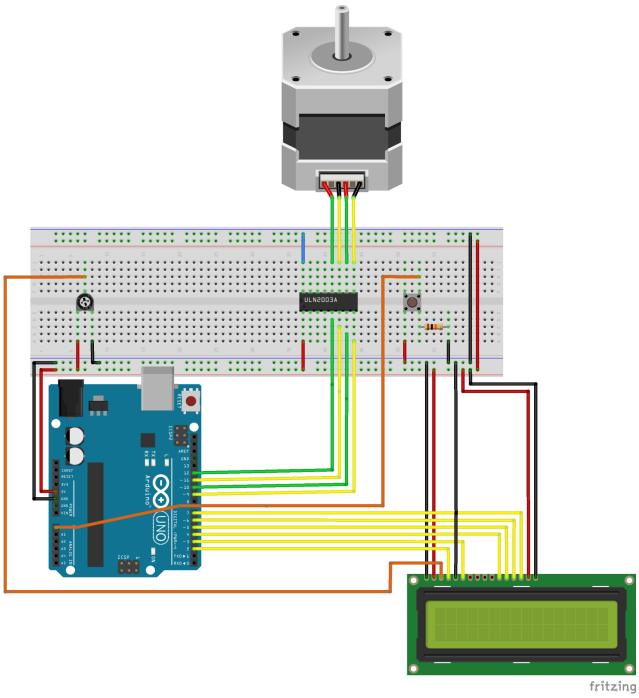

Figure 2: Wiring Diagram for the iJukebox System

Message Protocol Table

Sender	Receiver	Message	Data format
Jukebox Pi	Arduino	updateScreen	“Song name,artist name”
Jukebox Pi	Firebase Database	findTagID	int tagID
Jukebox Pi	Firebase Database	logRequest	“Song name, artist name, date, time”
Jukebox Pi	Speaker Pi	playSong	“songID”

Figure 3: Communication Protocol Table

Sequence Diagrams

Message Sequence Diagram 1: Pi to Arduino to update screen

Figure 4 illustrates when a new song is played on the Jukebox. When the song starts, the song name is sent to the Arduino so it can be displayed using the LCD display on the front of the Jukebox case.

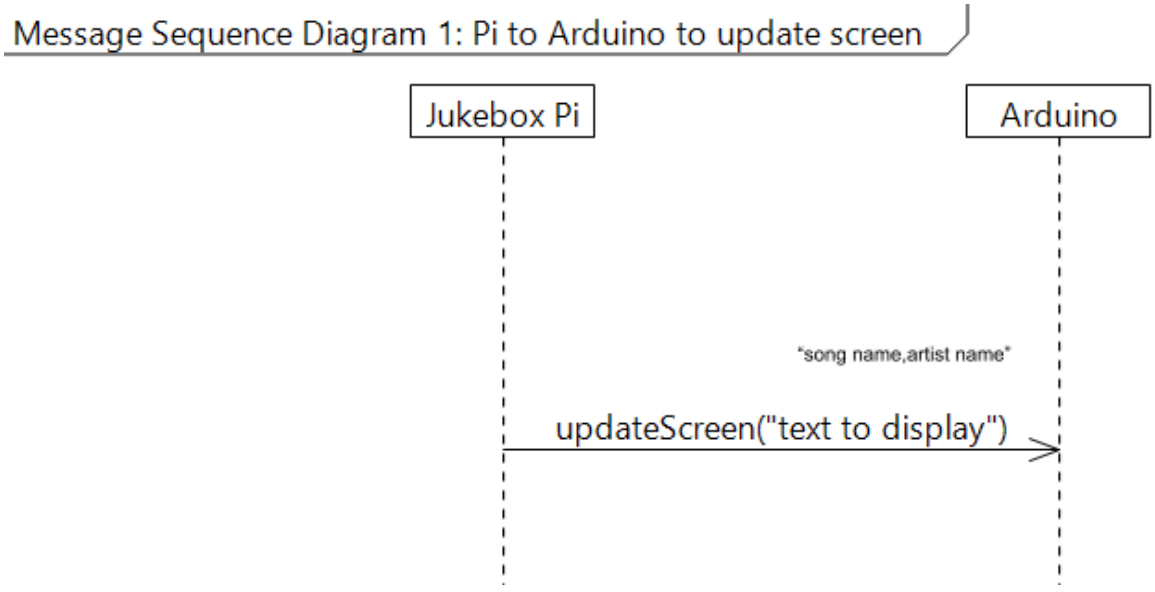

Figure 4: Message Sequence Diagram 1: Pi to Arduino to update screen

Message Sequence Diagram 2: Pi to Firebase database to find song ID from an NFC tag ID

Figure 5 illustrates when an NFC tag is scanned by the Pi and the tag ID is sent to the database so that the corresponding song ID can be found.

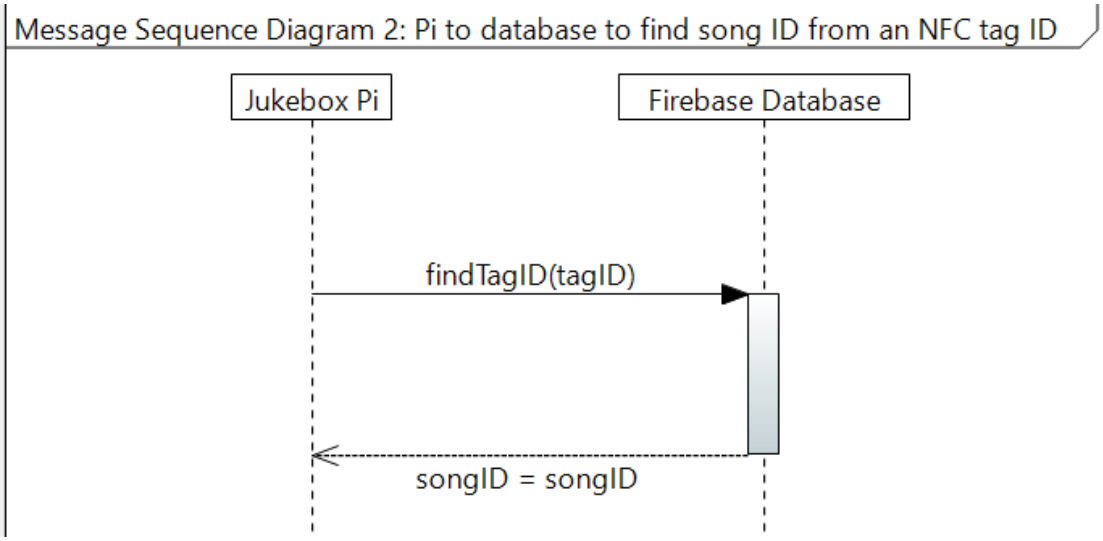

Figure 5: Message Sequence Diagram 2: Pi to database to find song ID from an NFC tag ID

Message Sequence Diagram 3: Pi to Firebase database to log song play request

Figure 6 illustrates when a song is played and the Jukebox sends a message to the database to log the request.

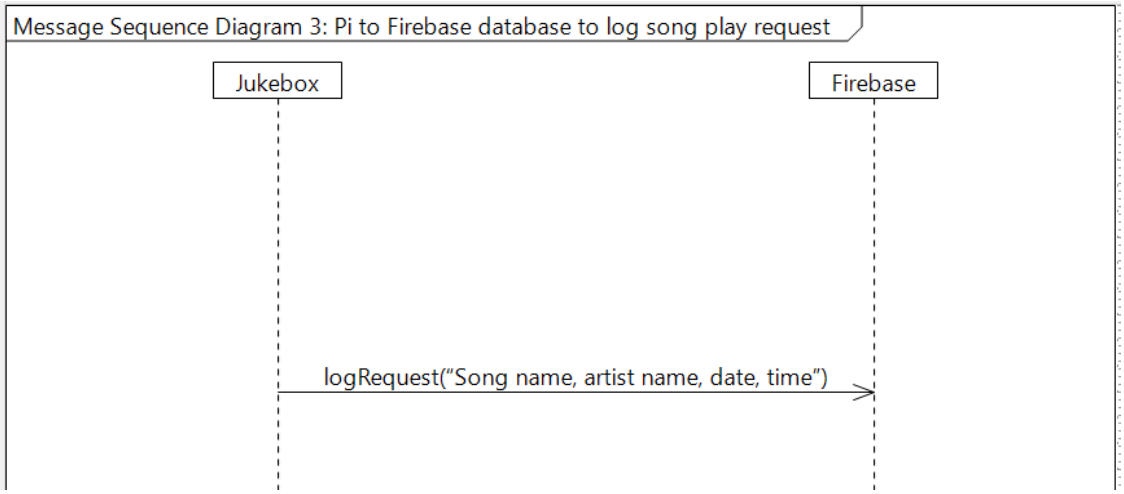

Figure 6: Message Sequence Diagram 3: Pi to Firebase database to log song play request

Message Sequence Diagram 4: Pi to Speaker Pi to play a song from Spotify

Figure 7 illustrates when a card is inserted into the Jukebox to play a song. After fetching the songID from the database, the Jukebox Pi sends a POST request to the server hosted on the Speaker Pi with the songID of the song to be played. The Speaker Pi then will pass on the POST request to the Spotify API which will actually play the song. If the songID is valid and the request was successful, the API will respond with an HTTP code 200 (200 means the request was successful) to the speakerPi, the Speaker Pi will then pass on that code to the Jukebox to acknowledge everything worked.

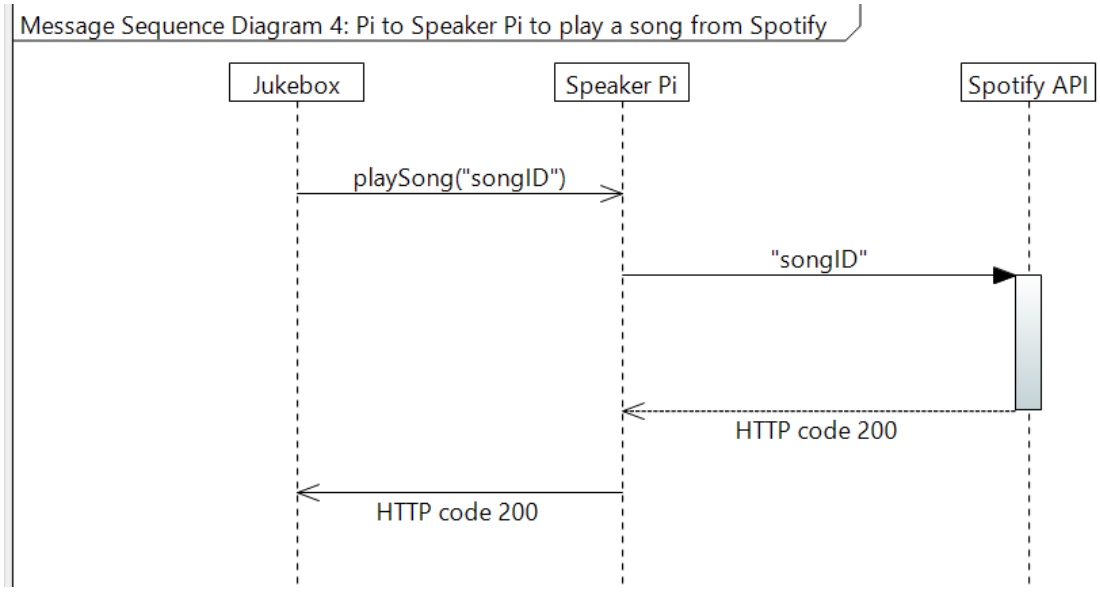

Figure 7: Message Sequence Diagram 4: Pi to Speaker Pi to play a song from Spotify

Discussion of Final Design

In our project we set out to build a Jukebox that is able to read NFC cards, fetch a corresponding song's info from a database based on an ID read from the NFC tag, update a display with the song info, have the song play on a separate speaker, and have sound controlled RGB LEDs on the outside on the Jukebox case that pulse with the music. We were able to accomplish all goals except the sound controlled LEDs. We were unable to implement the sound control because the sound detector we purchased did not produce the results we desired for the appropriate sound ranges. There was an audio channel on the sound detector,

however with all the components we were running on a singular arduino, it would not be able to sample it properly, thus we ended up putting the microphone on the backburner. We also had to swap the touchscreen display we had originally desired for a regular LCD screen as the Jukebox Pi had the NFC reader attached to its GPIO pins, and there were not enough pins available for connecting the touchscreen display. Another small change we made was instead of hosting the GUI on a smartphone app, we now have a server being hosted on the Speaker Pi and it can be accessed from any device on the same network to control the GUI. This server handles connecting to the Spotify API and services any requests from the Jukebox Pi as well as hosts the frontend GUI. The last change we made from the original proposal is moving the card display slot controls from the Jukebox Pi to the Arduino to eliminate the extra message passing needed to send the signals to open and close the door to the Arduino.

Contributions

Authors of all code pieces

Code Author	**File Names**
Corbin	- jukeboxMain.py - piToArduinoTest.ino - NFCtoPiTest.py (end to end test) - piToArduinoTest.py - piToFirebaseTest.py (end to end test) - NFCtoPiTest.py (unit test) - PitoFirebaseTest.py (unit test) - localDBTest.py

Flynn	- Web app folder which includes
	  - Src
	  - App.css
	  - App.js
	  - Login.js
	  - WebPlayback.jsx
	  - Index.css
	  - Index.js
	  - setupProxy.js
	  - Server

	- Index.js - Tests - appServerTest.py
Yunas	- Button_Screen_Motor.ino (used in final project) - Button_Motor_Mic_Led_Test.ino (used for Unit Testing) - openWebBrowser.py (extra feature) - audioDetector.ino (dropped feature) - LCD_unit_test.ino (used for Unit Testing) - lcdControl.py (display script to update display with python) - test_lcdControl.py (used for Unit Testing)

Figure 7: Authors of all code pieces

Authors of each section of this document

Proposal sections	**Authors**
Project description	All
Final Design Solution: Deployment Diagram	Yunas
Final Design Solution: Message Protocol Table	Corbin
Final Design Solution: Sequence Diagrams	Corbin, Flynn
Discussion of Final Design	Corbin, Yunas
Contributions	All
Reflections	All
Appendix A	All

Figure 8: Authors of each section of this document

Reflections

Corbin:

I think that our project went relatively well. We definitely had some hiccups with needing to change the plan as we encountered unexpected issues, but we were able to work as a team and have a decent final product. We did have some misunderstandings about the requirements of the end to end test demo, and so we did not produce exactly what was expected of us, but we learned from it and made up for it by performing the unit test demo successfully.

This project helped me to see the importance of communication and planning in a team, especially during COVID where we were remote. In our team we are all very busy with other classes and jobs so we made sure that we are consistently messaging in the Slack and Discord server. This allowed us to divide tasks into portions that each of us are able to handle with our other responsibilities, and make plans for when we are able to work all at once on important deliverables. There were a few times in the beginning of the term where we didn't message as often as we should, and got a bit behind on milestones, but once we got in the habit of frequently messaging the chat to say when we could and couldn't work those issues did not arise again. Also, our team used 3D printing to create parts of our system, and due to printing

issues and failed prints, it was down to the wire to have our demo video filmed. Now I know to leave almost double the expected time to print to account for any issues that need to be reprinted.

If given more time we would have liked to enclose the Jukebox case so that we don't have wires and breadboards laying out behind it, this will be a matter of condensing all the wiring so that the big breadboards are not needed. Also, we would like to make the system more scalable so that multiple iJukeboxes can be used in the same home to play from anywhere in the house.

Yunas:

The project performed exactly as we expected it to do for the primary task, which is why I believe it went perfectly. The primary task that was initially discussed was to have something that looks like a jukebox and can play music on spotify when an nfc tag is taped upon it. For that it is perfect. The remaining stuff were all extra features to make the jukebox turn into an iJukebox.

One part I wish that we had working that we had to drop last minute was the adaptive LED based on sound. Unfortunately, the microphone we had was more like a sound detector rather than a microphone. We were not able to use the microphone to create adaptive lighting. However, the lighting was ambient and would move at a steady pace and looked good with the jukebox. It was a lot of hard work, sweat, and tears put into the project, with that, I was able to learn a little bit about how some API's work(Spotify), learnt a cheat to threading on a microcontroller, and also got learn about debouncing and using it in the project(was a big help in avoiding input that would otherwise mess with the timing for different components in the project).

Given an extending time, we would incorporate the microphone into it, and using better quality leds, we had addressable led strips. However they were cheap quality and would flicker, therefore we ended up going with regular rgb leds, which held up good and were a bit simpler to handle and no flicker. Asides from that, a plate of spaghetti looks less messy than the wiring in the back of a jukebox. If we were to develop it further, I would incorporate everything on a PCB Board and possibly use a NodeMCU, which means that instead of using Arduino Serial communication, I could possibly host the communication over wifi instead. Overall I am very happy with the results and the small touch ups along the way that made our Jukebox an iJukebox.

Flynn:

The project succeeded in achieving our goal of having a 3D printed jukebox that we could use to play music through NFC tags. The project did that and looked rather good doing it, so I was in the end proud of what we made. However, whilst the project went well at achieving our goal, I think it definitely would have been doable to further extend the project to tack on different features and extensions. Some things I learned along the way were how to interact with API's, how to communicate over HTTP, how to develop a GUI and how to code in javascript.

It was definitely an interesting experience as it was like no other class I have taken before. There was a lot of learning on the job, and I definitely had some struggles with self motivation for this project. It was very open ended and I think that I had some trouble with that. My whole academic life has mainly been a series of very defined tasks which I would then complete to the degree I knew was expected of me. This project however fell into a strange new category of being very open ended, but still needing marking criteria. As it was a schooling project with rigid marking schemes I felt that I never really allowed myself to let loose and actually enjoy the project. As a result I think I definitely spent more time worrying about how the TA would mark us than I did actually pouring any sort of passion into the project itself. If I learned one thing throughout this experience it is that I need to work on self motivation for more open ended tasks and get better at finding ways to make them fun for myself

As for extensions to the project, I think It would have been interesting to add a couple of extra features to make the project more easy to use. Adding things like a way to register new NFC tags with albums or adding a selector to pick which device to play the music on would have made it more approachable for non technical users.

Appendix A

Found in this appendix is the top level readme folder from our projects github.

iJukebox

SYSC 3010 A

Group L3-G6: Corbin Garlough, Yunas Magsi, Flynn Graham

TA: Roger Selzler

Project Summary

The iJukebox is a miniature Jukebox that sits in a common area, its main function is to read NFC cards and play songs while displaying the cover art of the song being played. These NFC cards are decorated with the artist's album cover and store values for songs from a specific artist. When a user picks a song they like, they will insert their card in the iJukebox and it will scan the NFC card to place a request for that song to play from Spotify. The iJukebox reads NFC cards, fetches a corresponding song's info from a Firebase database based on an ID read from the NFC tag, updates a display with the song info, and plays the song on a separate speaker.

Repo Description

Our project repo is made up of directories for the Python, Arduino, and web app code that is required to make the iJukebox work. We also have a directory for our WIPURs, and an image of the final iJukebox product. In the PythonWork directory you will find folders for the end to end, and unit tests for the system, the support files needed for the PN 532 NFC reader, the local database that we use to store play requests (iJukeboxDB), and the main program to run on the jukebox Pi (jukeboxMain.py). (YUNAS FINISH SUMMARY OF ARDUINO FOLDER).

In the web app folder there are four sub folders, src, server, public and tests. The src contains all the code to create the frontend GUI (App.js, index.js, login.js, setupproxy.js and all the corresponding css files) and embed a spotify player instance into the web browser (WebPlayback.jsx). The server folder hosts the server that interacts with the spotify API. The tests folder contains tests for ensuring the web app functions properly. The public file just includes some images and things used by React.

Installation Instructions & How to Run the System

Start by cloning this repo onto the Jukebox Pi and Speaker Pi, then proceed with the following setup.

In order to use the project the user is required to create an account with Spotify. Once an account is created the user must register their app with spotify for developers to gain permission to use the API.

Jukebox Pi

To get the jukebox Pi part of the system operational, attach the PN 532 NFC reader to the GPIO bank on the Pi while the power is disconnected, and plug in the printer cable to the USB port on the Pi and corresponding port on the Arduino. Then open the jukeboxMain.py to make a few edits based on your specific information. First, on line 30 enter your email account you would like notifications to be sent to (right now the notifications only work with Gmail accounts). Next on line 307, change the IP address used in the post request to that of the Speaker Pi you are using. Now you can run this program, and once the other components are started, the iJukebox Pi is ready to rock out!

Speaker Pi

To run the speaker Pi there are two things you must do to set it up. Firstly in the spotify for developers dashboard the user must create a callback by editing the app settings. This callback should be for https://'speakerIP':3000/auth/callback. Once that is done all that is left to do is ensure that the .env variables are properly set, the user must input their spotify client ID and secret that they got from registering their app as well as the ip for the PI. Once that is done, simply type "npm run dev" in the web app folder to run the server and GUI.

Arduinos

To run the arduino portion of the iJukebox these are the ingredients required: an Arduino Uno, 16x2 LCD display, Stepper Motor, a Push Button, 10kOhm resistor, 10KOhm potentiometer, and a lot of wires. A bonus may include a 5v power supply to help offset the load on the Arduino. Once those components are gathered. Please look over the wiring schematic on the final report and set up the components to that spec. After that is all set, in the Arduinowork directory, there is an Arduino file called "Buttons_Screen_Motors.ino". Simply copy that code and ensure that the wiring matches with the variable setting. Compile and Run it. Now you're all set for the Arduino

How do you know the system is working properly?

To ensure the system is working correctly, you can run the tests found in the end to end communication, and unit test directories and observe their output. If all tests pass then when the main program for the system is run, inserting a card into the display slot and closing the door will start the song playing on the Speaker Pi, the song info will be displayed on the LCD screen, a request log will be seen in both the Firebase and local databases, and you will receive an email notification with info about what song was played.

References

- *[1] A short history of how jukeboxes changed the world*. Rock. (2019, August 14). Retrieved February 10, 2022, from https://www.rock-ola.com/blogs/news/a-short-history-of-how-jukeboxes-changed-the-world

-