CS 207 Final Project: Arduino HID Controller for Spotify

Nathan Chay (200403221)

Department of Computer Science, University of Regina

CS 207: Building Interactive Gadgets

December 8, 2020

Table of Contents

I. Introduction

This project is an Arduino HID controller for controlling the user's Spotify account. The controls consist of five buttons, for toggling shuffle, changing loop mode, going to the previous track, going to the next track, and playing/pausing the current song, and a potentiometer for changing the volume. An LCD screen is used to display the name of the current song and the artist of said song. The LCD screen also displays the volume when changed, and when a button is pressed, the name of that button.

When an input is detected by the Arduino, the necessary information about that input is sent over serial to a Java program that handles the connection to the user's Spotify account. Prior to use, this Java program must be authorized to access the Spotify Web API [1] on behalf of the user. The Java program sends requests and receives data via this API, and the necessary data is then sent back over to the Arduino via serial. Finally, the Arduino displays the necessary data using the LCD screen.

II. Background

This project was very loosely based on the work of Eward Hage, who build an Arduino HID controller to simply send keypresses to a PC over USB [2]. Upon discovering Hage's project, I became interested in controlling some sort of function on my PC via Arduino. I eventually settled on music, since I spend a lot of time listening to music on my PC while using it for other tasks, such as doing assignments, playing video games, or just browsing the internet. I felt that a simple button interface would be perfect for a project like this, since the functions I wanted to be able to control only required a binary input (play/pause, skip to next/prev track, etc.). Additionally, Hage's project required flashing the Arduino's firmware to allow it to directly send the keypresses. Since I didn't want to have to re-install the original firmware to be able to complete the labs, I decided I would need to find a different way to read the button presses from the Arduino.

Although Hage's project was helpful in inspiring my final project, it lacked the necessary features to achieve my goals for this project without significant modification. I was not going to be able to communicate with my Spotify account through the Arduino alone – I needed some sort of "middle-man" program to send and receive requests to/from Spotify while simultaneously communicating with the Arduino. After some research, I stumbled upon a project by Tanmay Desukar, who created a video game controller using an Arduino [3]. What intrigued me about Desukar's project was that, unlike Hage's project where he sent keypresses directly through USB, Desukar used a Java program to receive the button presses from the Arduino and turn them into keypresses that the PC recognizes. Desukar's Java program seemed like the type of "middle-man" program that I was looking for, and with these two projects as reference, I began work on my own project.

III. Build

The build for this project was relatively simple. I begun by creating a circuit of 6 pushbuttons in a pull-up arrangement. I decided on a pull-up arrangement due to my experience creating the 8-button controller for an earlier assignment in this class, where I found that using a pull-down arrangement for buttons caused some inaccuracy and interference between buttons. I organized the buttons in an intuitive way – shuffle and loop are beside each other on the left side of the breadboard, previous track, play/pause, and next track are grouped in the middle of the breadboard, and the mute button was on the right side of the breadboard, alone. Originally, I planned to have the buttons arranged in a 2x3 grid, but I realized that pushbuttons on a breadboard can only be placed along the center line, to isolate each of the 4 pins.

My original proposal for this project included LEDs to indicate whether toggles like loop or mute were currently on. This idea was scrapped relatively early into the build process. The main reason was simply lack of breadboard space. The only intuitive place for these LEDs would be right above or below their corresponding button, yet I needed to use all the space above and below the buttons to connect them to voltage/ground/input. The only other solution would be to place them below the buttons on a separate breadboard, but I felt as though the device simply became too bulky to be useful.

Since the addition of indicator LEDs was one of my initial ideas for an improvement of the projects I was basing this project off of, I figured that I should think of a new improvement to add. Partway through the code design process I had the idea of swapping the mute button for a volume knob. This would allow me much more flexibility in controlling the volume of the music, and since I already had a potentiometer in my Arduino kit, I didn't need to order any extra parts. I simply replaced the mute button with the potentiometer and connected it to an analog pin instead of digital.

The final part of my build process was adding the LCD screen. Unfortunately, while ordering parts, I did not realize that the shield on the LCD screen requires jumper wires with female ends. I ordered some right away, but I was worried that if I didn't start coding the screen functionality before they got here I wouldn't finish the project on time. I decided to look for some temporary workaround to connect the LCD screen to the Arduino. Upon researching ways to convert a male wire to a female wire, I found a method that requires simply applying a heat shrink wrap around the end of the wire [4]. Since I didn't have any heat shrink wrap available, I attempted to produce the same effect by wrapping scotch tape around the end of the wires [Appendix A Fig. 1]. I was able to get the screen to power on; however, the connection was very unstable, and the constant disconnects made testing it near-impossible. At this point I gave up and decided to wait until the correct wires arrived.

Later on in the code design process another idea came to mind – plugging the LCD shield pins directly into a breadboard. Luckily, the pins were spaced apart by the perfect distance to align with the breadboard and were close enough to the edge of the LCD to allow me to plug them in without problem [Appendix A Fig. 2]. Once the correct wires finally arrived, I was able to complete the final build [Appendix A Fig. 3].

IV. Design

The code design process begun with the Arduino code – simply reading the button presses and printing to serial which button was pressed. Since this is almost exactly the same as the code I wrote for the 8-button Arduino controller assignment, I simply copied the code from that assignment and modified it slightly.

Once the buttons were functioning properly, the next logical step was to implement the Java program to connect to Spotify. Tanmay Desukar's project, as mentioned in the background section, used a Java IDE called Processing to implement these keypresses [3][5]. Processing has a built-in library that makes it extremely easy to send and receive input through serial, which is necessary for communicating with the Arduino; however, the Processing IDE has a multitude of problems that I encountered whilst in development.

Firstly, Processing was developed specifically to handle image processing via Java code – similarly how the Arduino IDE was developed to handle communication with Arduino via C code. It is fair to say that attempting to write a normal Java project in Processing is similar to attempting to write a normal C project in the Arduino IDE. After struggling to add external libraries to Processing, I decided to abandon it altogether in favor of a more traditional and flexible Java IDE.

After spending a few hours downloading software and figuring out how Java works outside of Processing, I was able to set up a useable Java workspace. Unfortunately, I was now missing a huge advantage that Processing provided – a built-in serial communication library. After doing some research, I stumbled upon a library called RXTX [6]. I was able to install the library and begin coding; however, after half a day of attempting to figure out how to write in Java as well as trying to figure out how to use this library, I became stuck. I simply could not figure out how to use this library to read inputs from the Arduino.

At this point, I decided to search for an alternative. I found another serial communication library called jSerialComm [7]. Some users on StackOverflow claimed it to be a much better alternative to RXTX, so I decided to give it a try. I immediately found that this library was not only much more streamlined than RXTX but had tutorials available specifically for connecting to Arduino. After a few hours of coding I was able to read input from the Arduino in my separate Java program.

The next step was to connect to my Spotify account. Prior to beginning the project, I had done a small amount of research, and found that Spotify has an API that allows control over a user's playback – specifically the functions that I needed for this project (play/pause, skip, etc.) [1]. I was able to find a Java library that handles connections to this API called spotify-web-api-java, and decided to give it a try [8].

The first step in accessing my Spotify account was to authorize my account with my program. Unfortunately, Spotify's authorization process is somewhat tedious – I first had to follow a link where I approved the connection to my account in my web browser. I was then given an authorization code, which I had to input into my program. Once the program ran, it traded the authentication code with Spotify for an access token (that was only valid for 1 hour), which could finally be used to access the API on my account's behalf. I had to do this process manually every time I wanted to test the program – I could have coded the entire authorization flow, but it would have involved me setting up a website to get the authorization code and overall would not have been worth the effort.

At this point, I was also having issues with the library that I was using. Similarly to the RXTX library, I could not get the spotify-web-api-java library to access my account, even with the correct authorization. While researching other methods of accessing Spotify's API, I was looking through the official documentation, and found that the API requests were just simple HTTP POST and GET requests, which could be sent from my Java program simply by using a built-in library. Spotify's official documentation also allows you to demonstrate each API request by sending it to your account and showing you the results, all from within your browser. I noticed that this feature also displays all information about the request that is sent – including the access token that I was having to painstakingly acquire manually earlier in development. Upon discovering this, I was able to insert the access token itself directly into my program and I would be able to test it for an hour.

Once I programmed all of the API requests and made sure they were working correctly, I began programming the LCD display. After some research, I found the hd44780 library [9], which allowed me to print characters to the display with one simple function. Sending data from the Java program over serial back to the Arduino proved more difficult than expected. After spending some time scouring the internet, I found that, for reasons that are unexplained, the program needs to wait roughly 2 seconds between connecting to the serial port and sending data to the Arduino. Displaying the user feedback messages for button presses and volume changes was able to be done right in the Arduino code and therefore was easy and simple. Once I polished up and commented the code, the project was finally completed.

Setup/Usage

Required Components:

1x	Arduino UNO
5x	Pushbutton
5x	10kΩ Resistor
1x	Potentiometer
1x	LCD Screen (Preferably [10])
4x	Male to Female Jumper Wires
~15x	Male to Male Jumper Wires

Build Instructions:

The breadboard diagram (LCD omitted) is included in Appendix B.

1. Wire the pushbuttons in a pull-up arrangement. The buttons should be connected to these pins:

Shuffle	-	Pin 7
Loop	-	Pin 8
Prev. Track	-	Pin 9
Play/Pause	-	Pin 10
Next Track	-	Pin 11

   These pins can optionally be changed if the code in the .ino file is reflected to match said changes.

2. Wire the potentiometer to pin A0 (or any other pin, as long as the .ino file is changed).

3. Wire the LCD shield's pins to the following:

GND on LCD	-	GND on Arduino
VCC on LCD	-	5V on Arduino
SDA on LCD	-	SDA on Arduino
SCL on LCD	-	SCL on Arduino

Note that different LCD screens with different shields will need to be wired differently. The hd44780 library has support for various LCDs and shields and may prove useful [9].

Usage Instructions

Using this device requires a Java workspace, preferably IntelliJ IDEA [12], since this is the IDE used to develop this project, and the source on GitHub [Appendix C] contains all necessary project files to compile the code smoothly. This project also uses Maven to handle library dependencies, so make sure that you choose an IDE that supports Maven and open the project in a Maven-enabled environment.

1. Upload the Arduino program, and make sure the serial port is set to COM3.
2. Obtain your Spotify access token by going to [11] and clicking the “GET TOKEN” button. Make sure all boxes are ticked. When you click “REQUEST TOKEN”, your access token should appear in the box labeled “OAuth Token”. This token needs to be added to the InputListener.java file on line 19 inside the quotation marks.
3. Run the Java program. Once you see “Started” in the output, the device should be ready for use.

Troubleshooting

The Java program will output the results of each API request to the console, so any problems with the API should be obvious – keep in mind that you will have to get a new access token every hour.

The *JSONObject["item"] is not a JSONObject* error occurs when the program attempts to get the current song’s information while you are not currently playing a song or, more often, halfway between switching songs. This can be solved by simply restarting the program.

Occasionally the LCD display will display nonsense/incorrect characters. This is caused either by non-UTF-8 characters in a song title/artist name (non-english characters, uncommon special characters), or an unusually long song title or artist name. This can usually be solved by switching to a different track.

Milestones

The original milestones for this project were as follows:

Oct. 28	-	Parts gathered
Nov. 4	-	Device partially assembled (no screen)
Nov. 11	-	Device assembled
Nov. 18	-	Arduino code to print button presses to serial completed
Nov. 25	-	Started learning Java
Dec. 2	-	Java code to convert serial output to keypresses completed
Dec. 8	-	Added mechanical key switches (stretch goal)

Here are the completion dates for comparison:

Oct. 23	-	Parts gathered
Nov. 30	-	Device partially assembled (no screen)
Nov. 30	-	Arduino code to print button presses to serial completed
Dec. 1	-	Started learning Java
Dec. 1	-	Java code to convert serial output to keypresses completed
Dec. 2	-	Device assembled
Incomplete	-	Added mechanical key switches (stretch goal)

The only milestones that were met was gathering parts and completion of the project. This is because I foolishly decided it would be a good idea to procrastinate the entire project until the last two weeks. Meeting the first deadline was rather easy since I ordered my parts through Amazon and they arrived within a few days. Meeting the last deadline, however, was extremely stressful. I was constantly worried about whether I would be able to finish in time, and I feel very lucky to say that I was able to pull it off. The rest of the deadlines being missed were a result of me being swamped with assignments and work and therefore not wanting to put any time into the project, even though I could have devoted the occasional small amount of time and even the reading week break (during which I did no work) to working on the project.

I also would have liked to have met my stretch goal; however, me failing to meet this was not entirely my own fault. The plate for the mechanical key switches I was planning on using [14] has been sold out since shortly after I submitted the proposal for this assignment, and I unfortunately have not been able to find anything similar to it.

In the future, I would like to polish up this project by making it more presentable and easier to use. I would like to build (3D print?) some sort of case to fit everything inside, add the mechanical key switches, and replace the potentiometer with one that has a bigger knob. I would also like to implement a full authorization flow so that anyone who wants to build/use it does not have to compile the code themselves.

Appendices

Appendix A – Build Photos

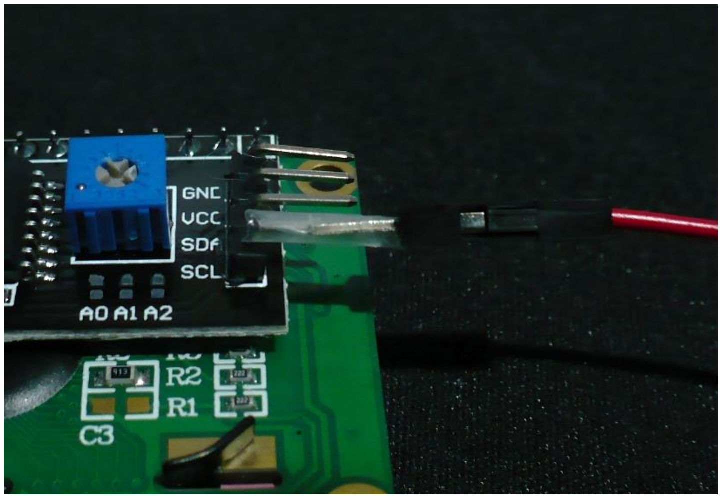


Fig. 1 – Attempting to tape two male wire ends together

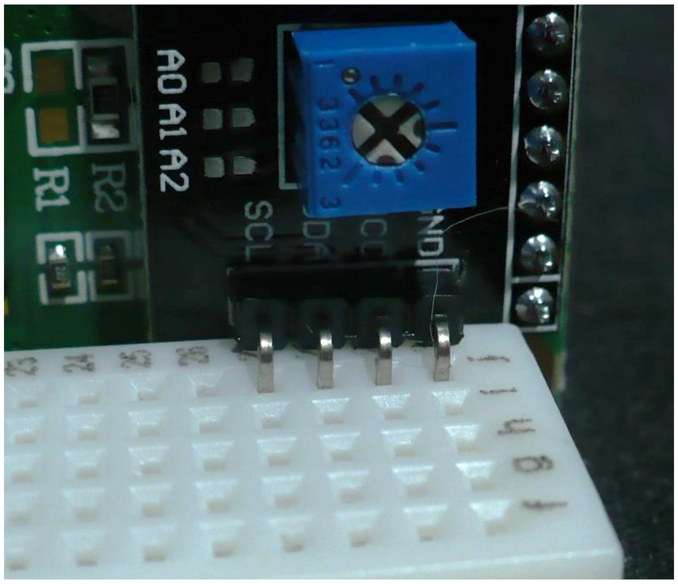


Fig. 2 – Temporary breadboard solution

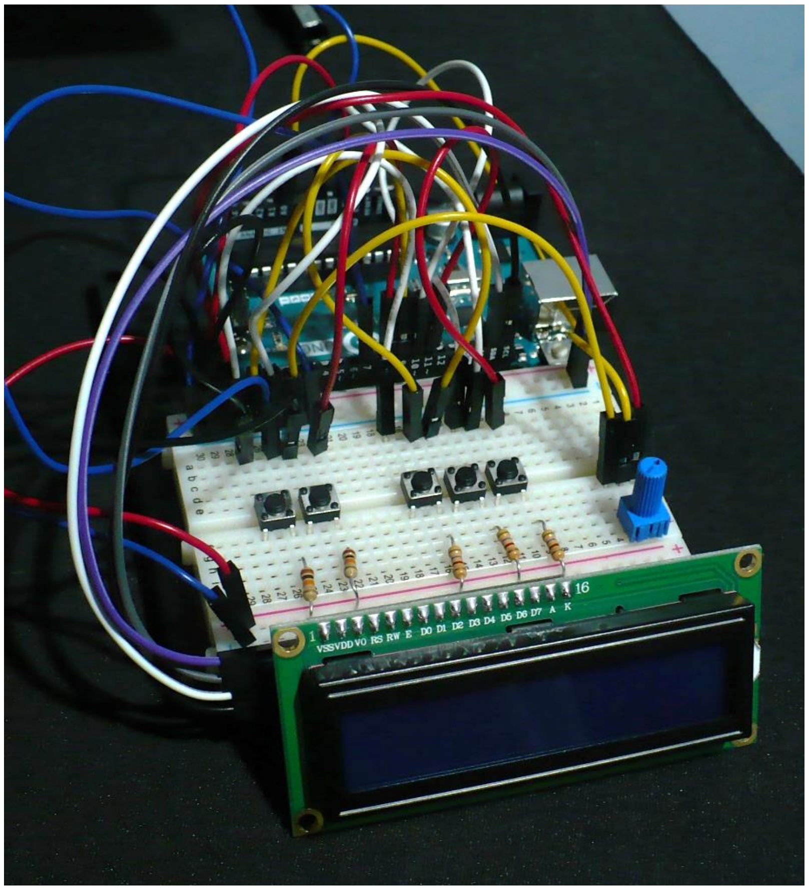
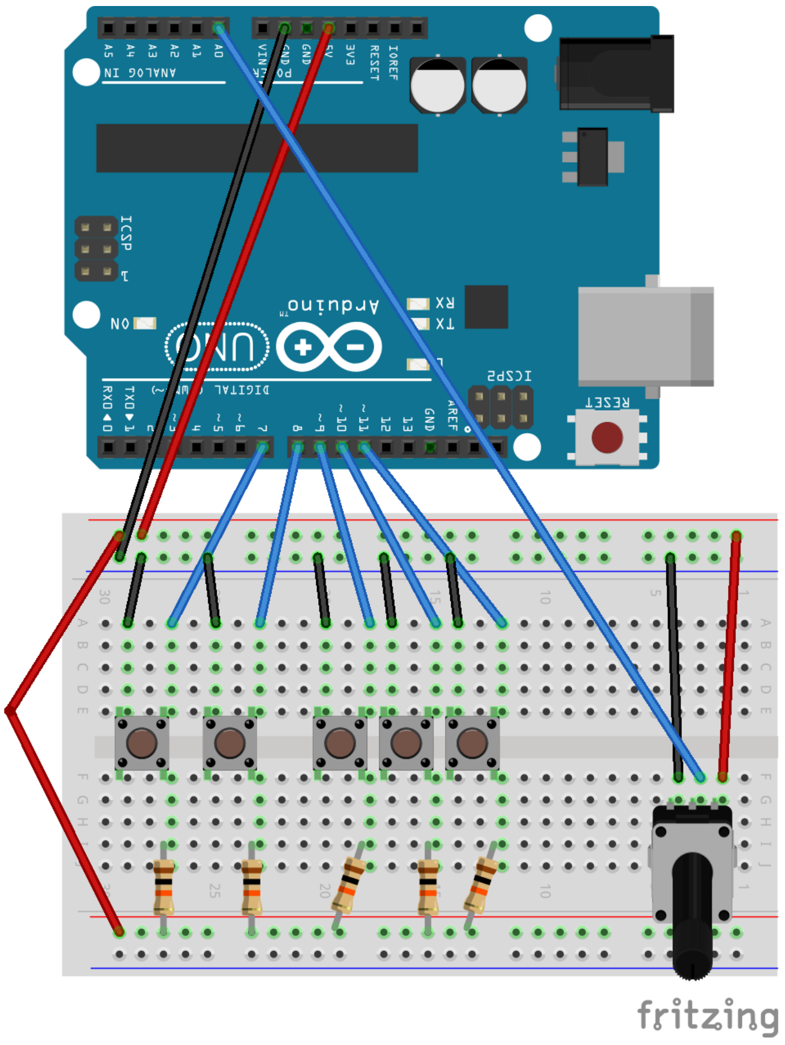


Fig. 3 – Final build completed!

Appendix B – Breadboard Diagram



Appendix C – Project Code

All project code can be found at [13].

References

[1] https://developer.spotify.com/documentation/web-api/

[2] https://www.instructables.com/How-to-Make-a-Arduino-HID-Keyboard/

[3] https://www.instructables.com/Video-Game-Controller-With-Arduino/

[4] https://www.instructables.com/DIY-Male-to-Female-Jumper-Wire/

[5] https://processing.org/

[6] http://fizzed.com/oss/rxtx-for-java

[7] https://fazecast.github.io/jSerialComm/

[8] https://github.com/thelinmichael/spotify-web-api-java

[9] https://www.arduino.cc/reference/en/libraries/hd44780/

[10] https://www.amazon.ca/gp/product/B019K5X53O/

[11] https://developer.spotify.com/console/put-pause/

[12] https://www.jetbrains.com/idea/

[13] https://github.com/natetheneet/arduino-spotify-controller

[14] https://techkeys.us/collections/accessories/products/sixkeyboard?variant=25290988040