Rhythmic Learning Tool Final Report

Submitted To

Dr. Jacob Abraham
Dr. Jim Wiley
Heng-Lu Chang

Prepared By

Briar Coker
Matthew Daumas
Kenneth Hall
Ryan Mallin
Blake Muir
Karime Saad

EE464 Senior Design Project
Electrical and Computer Engineering Department
University of Texas at Austin

Fall 2017

CONTENTS

TABLES

FIGURES

EXECUTIVE SUMMARY

This summary details the design, implementation, evaluation, and further considerations of our Rhythmic Learning Tool Project for EE 464 during Fall 2017 at the University of Texas. Our report begins with an explanation of drawbacks of current music teacher methods for younger students. Afterwards, we determine an ideal, cost efficient solution using well supported and available software and components such as the Raspberry Pi, a single-board computer. Our development focused on accessibility for public schools, safety and ease-of-use for students, and modularity to facilitate development and expansion of features such as different time signatures or note types.

Current methods for music education in primary and secondary schools are low-cost and give poor feedback, or high-cost with individualized feedback. Low-cost teaching methods, such as clapping along with a teacher, do not allow teachers to assess students individually or address difficult rhythm patterns. High-cost solutions, such as iPad applications or desktop programs, require have large startup and maintenance cost. Our solution requires accessible teacher and student interfaces. The teacher's interface must display lessons to students, allow for lesson creation, monitor student performance, and maintain minimal latency to the student interface. The student's interface must be capable of receiving student input, evaluate performance accuracy, send and receive data to the teacher's interface, and fit in a compact and wireless enclosure. Both modules must interact to provide student and teacher feedback while being cost efficient.

Our solution to find a low-cost and individualized teaching tool includes a handheld student controller and a lesson management webapp. The handheld focused on minimizing controller-to-database latency and ease of use for students. This was accomplished by preloading data where appropriate and designing a tactile and responsive handheld controller. The hardware implementation consists of open-source or low-cost parts such as the Raspberry Pi Zero W or the Adafruit Trellis Keypad. Similarly, the teacher's web interface featured low-cost and well documented modules: AngularJS, a JavaScript framework, was used to implement lesson playback, lesson creation, and student management. Firebase, a real-time, cloud-hosted database was chosen to store student performance and lesson data due to its accessible integration with AngularJS, both of which are Google supported services.

After researching and experimenting with different applications and components, our team was able to complete a design which met the above specifications. One of our first design choices was to use an Arduino Uno microcontroller, but the device did not meet our needs for a wireless solution. Ultimately, our team chose to use a Raspberry Pi Zero W, which not only met our wireless needs, but also had support for keypad drivers and database access. Our team initially investigated many webapp templates before deciding to use Yeoman. Some of the challenges in developing the webapp include learning HTML, CSS, and JavaScript. Lastly, our team chose to

implement the Firebase database, which had built in interactivity with several other modules we had in place.

Testing and evaluation of our design was organized into: hardware testing, software testing, and full-system testing. Hardware testing included measuring device metrics such as current drain and keypad timing accuracy. Additionally, our team measured the latency cost for accessing the Firebase database. Software testing included webapp accessibility, lesson creation inputs, and visual feedback for students. Unit tests were employed to facilitate quick testing of new sub-modules. Our team also tested the teacher's GUI for consistency and coherency with Google Chrome extensions that inspected individual elements autonomously. Afterwards, our team assessed accesses to the Firebase Database for accuracy and latency cost. Lastly, our team performed full-system testing with both ideal and complex rhythm patterns. These tests checked for total system coherence and were also used to calibrate acceptable keypress timings.

Although our team had no budget, we were able to complete a low-cost design within two semesters using free or open-source software, low-cost devices, components, and 3D printing. Initially, choosing low-cost modules and software added another layer of complexity to our first design decisions. However, working with low-cost components lowered our finished system's cost, and moved us towards our goal of an affordable system that public schools would be able to adopt. Finally, in order to mitigate unexpected time sinks towards the end of development, our team adopted an organized schedule and followed a set meeting agenda.

The primary safety and ethical provision for our system was ensuring that our handheld controller would never impose a safety hazard, even with rigorous use by young students. Additionally, our team researched the best methods that would keep our user's data and information and secure while they utilized the device's wireless features. Lastly, our team ensured our controller design would adhere to safe volume levels, feature tamper proof hardware, and a resilient controller enclosure.

Our recommendations for future development on the Rhythmic Learning Tool include support for additional note types, time signatures, and switching the hardware controller from a computer to a microcontroller. Our design supports scaling for new note types, such as dotted notes or eighth notes, and different time signatures; future implementation of new features will provide a more thorough learning experience with a low implementation cost. The Raspberry Pi Zero offered a straightforward solution to connection to the webapp through Wi-Fi, however programming in a Linux environment frustrated programming for real-time requirements. Lastly, our team believes providing educators with additional performance statistics, such as frequently missed notes or overall classroom performance, is important. Focusing development on properties that are intrinsic to our design will yield a richer learning experience for students.

Our team was able to meet all the key design requirements set forth, for the Rhythmic Learning Tool, which resulted in a working prototype that placed third at the Fall 2017 Senior Design Open House Showcase. Our team plans to advance new and current features so that the Rhythmic Learning Tool may one day be adopted by public schools. Developing the Rhythmic Learning Tool gave our team firsthand experience with researching, designing, and testing a new music education tool.

1.0 INTRODUCTION

This report is a comprehensive summary of the design, implementation, evaluation, and further considerations for our Rhythmic Learning Tool Project. The purpose of the project is to address the challenges of teaching rhythm concepts in elementary school classrooms through a web application and connected digital instruments. Currently, methods of teaching rhythmic structure face a tradeoff of cost of implementation versus how effective the method is to each individual student. Our design addresses the need for an inexpensive education tool to provide teachers with individual student performance data and students with an engaging, tactile instrument for learning rhythmic structure. The teacher will create or choose rhythmic patterns from the web application to be projected to the classroom and followed along by the students on their digital instruments. The performance of the students will be recorded and evaluated to provide the teacher with a summary of student performance. For EE 464, we created the teacher's web app with demo rhythmic patterns and one of the digital instruments a student would use.

The following sections elaborate on the challenges of teaching rhythm in classrooms, our design strategy and implementation to address these challenges, the testing and evaluation of our implementation, and finally, the time, cost, and safety considerations for our project. In the design problem section, we explore how rhythm is taught in the classroom and the specifications our tool should meet to provide a better learning experience for students. The design solution section describes the design strategy of our three modules, the teacher interface (web application), student interface (digital instrument) and an online database to enable communication between the two. The design implementation section details how we implement these modules through a web application written in AngularJS, a Raspberry Pi Zero based digital instrument, and Google's Firebase Real-Time Database. Following the design sections is a look into the testing and evaluation of our learning tool, from module testing to full system testing. Next, we examine the time and financial costs associated with our project as well as the safety and ethical considerations of an elementary educational tool. Finally, we conclude with recommendations for future implementations of our rhythmic learning tool.

2.0 DESIGN PROBLEM

In this section we will look at how rhythm and meter is currently taught in classrooms, the problems those methods have, and how we hope to solve those problems with our design. We will also discuss the requirements and specifications of our rhythmic learning tool's three modules: the teacher interface web application, the student interface digital instrument, and the real-time online database.

2.1 Problem Statement

Through our research and interviews with music educators, we have found that there are two different solutions to teaching rhythm and meter being used in classrooms today that fall onto different sides of a cost/effectiveness spectrum. The traditional solution for rhythmic education in elementary schools has students clap along with a teacher or following along with a pattern projected to the class. While this is an inexpensive way to teach the basics of rhythm, it is not effective since the teacher cannot easily track individual students' performance. The second solution is used more often in private practice, and uses a software application on a computer or tablet to provide the functionality to record and track each student's progress throughout lessons. However, computer solutions are often expensive and require each student to have their own computer. A tablet solution will be expensive if the school does not already own compatible tablets and would require tablets durable enough for rhythmic tapping by elementary children. Our solution is to meet both of these solutions halfway, creating our own web application accessed by the teacher's computer and our own durable hardware device that can be manufactured inexpensively relative to a tablet. The rhythmic learning tool contains two main components: a hardware device for student input and evaluation and a web application accessed on the teacher's computer to display lesson content and view student performance data. Both of these components communicate wirelessly through a shared real-time online database storing student input data and teacher created rhythmic patterns.

2.2 Specifications and Requirements

The specifications and requirements our rhythmic learning tool must meet can be divided between a teacher interface and student interface. Teachers need to create rhythmic patterns, display lessons to students within an acceptable latency threshold, and view student performance data. The student needs a hardware interface that connects wirelessly to the database to receive

lesson data and send performance data. The student interface records student input, compares it to the correct timing from the lesson, and sends this data back to the web application. The device also plays a sound and lights up an LED corresponding to the button pressed. The database must allow our web application and digital instrument to play rhythmic patterns online and record performance on the hardware in sync with respect to each other.

The specifications for the teacher interface include the ability to support the creation and playback of lessons containing 16-bar rhythmic patterns. These rhythmic patterns consist of quarter notes, half notes, and whole notes, at tempos ranging from 40-200 beats per minute (bpm) in either 3/4 or 4/4 time signatures. A complete lesson includes the lesson's name, at least one complete rhythmic pattern, the time signature for the lesson, and a numerical value for the beats per minute variable. The web application does not let the teacher input an incorrect format on a text field such as special characters, or letters where only numerical values are required.

The specifications for our student interface include evaluating recorded performance data in real time by comparing the student's performance to expected timings. This module allows music teachers to assess their student's learning quantitatively; grading is easier, more accurate, and gives teachers feedback into how effective their lessons are. The device also provides feedback to the student in the form of sound and LED responses to input. The student interface must also have at least 3 hours of battery life to last through at least two class periods of instruction and be durable enough to allow for elementary children's use.

3.0 DESIGN SOLUTION

Our design focused on a hardware interface for students to perform rhythmic patterns and a web interface for the teacher to view student performance data and create lessons. Figure 1, shown below, provides a high-level description of the input and outputs of our rhythmic learning tool with the student hardware interface and teacher web interface to be described further in the following subsections 3.1.1 Student Hardware Interface and 3.1.2 Teacher Web Interface. Our goal for the student interface was to include tactile inputs for student to press that will play a sound to mimic an instrument. As shown in Figure 1, our design solution accomplishes this with a Trellis button pad for input and LEDs and a speaker to deliver feedback to the students. Our

software interface required functionality to generate student performance reports for the teacher and display rhythmic patterns created on the software web application and concept summaries to the class to follow along with. The following subsections describe the student and teacher interfaces respectively.

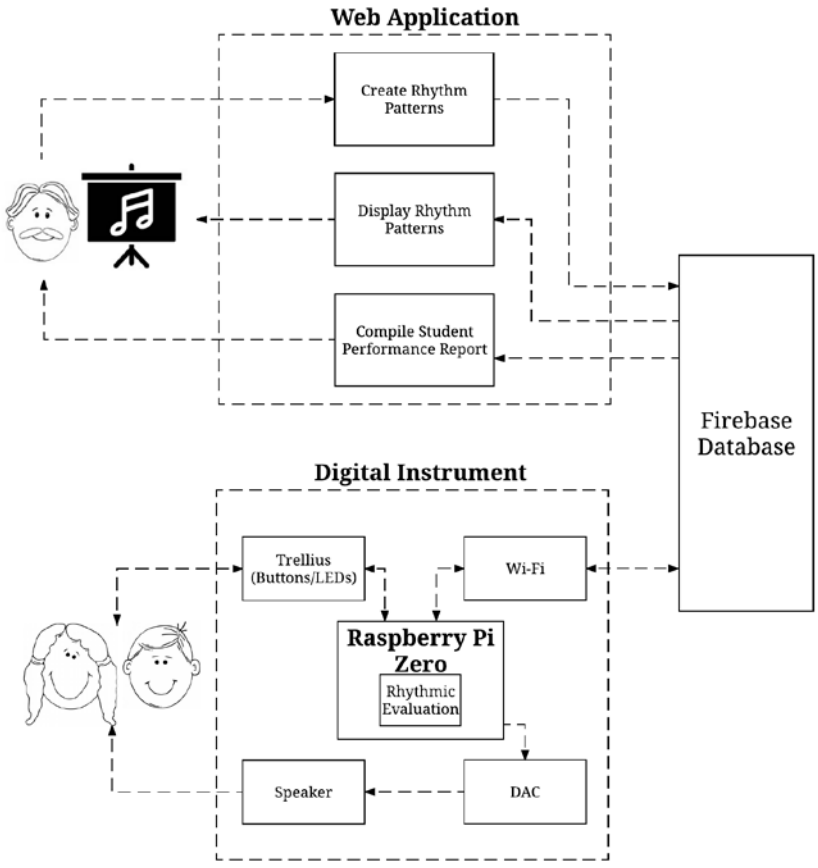

Figure 1. General System Input Output Diagram

3.1 Student Hardware Interface

The hardware interface incorporated an Adafruit Trellis 4x4 keypad to record student input, compared it to the correct timing from the lesson, and sent a grade back to the teacher's computer wirelessly. For each button pressed, the light corresponding to that button would turn on while pressed and a tone would be played. For the audio, each button played a different tone and the volume could be controlled by a knob. Figure 2 shows how each of these modules fit into the student interface design. To make the design portable, we used a portable power bank which would provide a 5V power supply for the entire system and a charging port. We designed and 3D

printed an enclosure for the student interface to ensure that only the keys, volume control, power switch and charging port were accessible to the user.

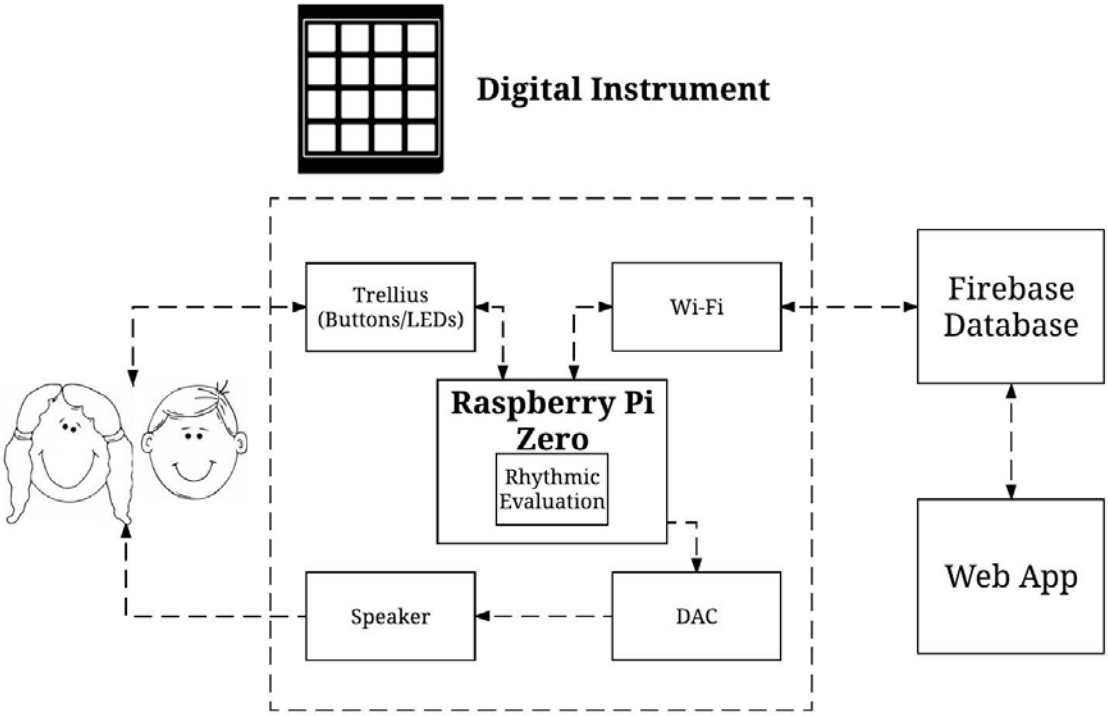

Figure 2. Student Interface Block Diagram

After experimenting with an Arduino Uno as the core of the student interface, we chose to instead implement a Raspberry Pi Zero W due to its smaller form factor, lower price, and built in Wi-Fi functionality. Our design requires wireless functionality to share data with the teacher's computer and synchronize clock timing. We chose Wi-Fi connectivity since it would allow an entire classroom of students to connect to the teacher's computer simultaneously. Student input is taken on an Adafruit Trellis 4x4 keypad, which can be connected to the Pi through its Inter-Integrated Circuit (I2C) interface. The Trellis contains all the necessary combinational logic to map 4x4 grids of both LEDs and keys separately while only requiring two pins of connection to the Pi. Students can follow along with a lesson by pressing specific keys on the Trellis in time to the lesson.

To implement sound, we used the pygame open source library to drive audio using the Raspberry Pi's onboard pulse width modulation digital-to-analog converter. This gave us an analog audio signal through an output pin on the Pi. However, the pulse width modulation created significant amounts of high frequency noise, which we needed to filter. We used a passive first-order resistor-capacitor (RC) circuit to create a low pass filter where the cutoff frequency is $\frac{1}{2\pi RC}$, which we wanted to be approximately 20 kHz at the maximum frequency of human hearing. We then used a first-order RC high pass filter with cutoff frequency at approximately 20 Hz. After finding standard value resistors and capacitors, we were able to make a bandpass filter that passes from 106 Hz to 17862 Hz. We used a TPA731 class-AB power amplifier chip to create a feedback circuit with variable resistance, which gave us volume control and accounted for the signal attenuation inherent in passive filters. The diagram for the audio circuit we implemented is shown in Figure 3, along with the frequency response bode plot. For feedback resistors R6 and R11, the diagram shows the maximum value of 100,000 ohms for the potentiometer which gives maximum gain. The potentiometer can be adjusted to approximately 100 ohms for 60 dB of attenuation and minimal volume.

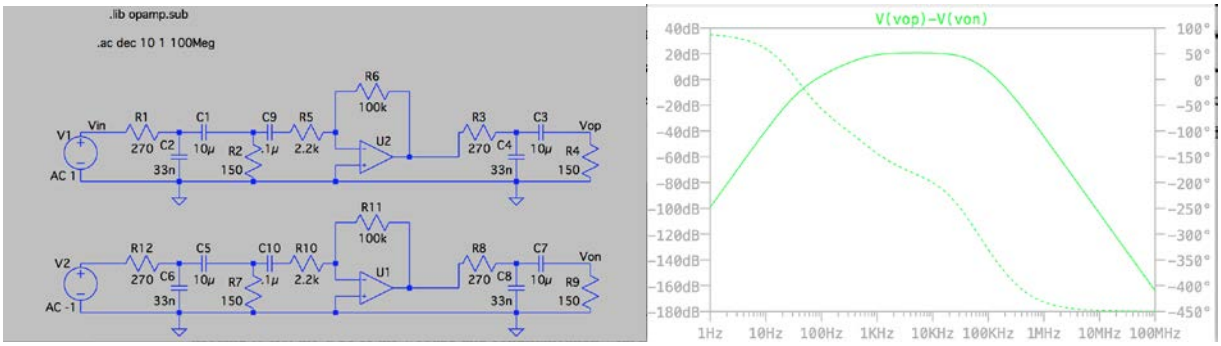

Figure 3. Audio Circuit Diagram and Simulated Frequency Response

To evaluate rhythmic correctness, we compared the timing of a student's button push and the expected timing according to the rhythmic pattern on the Raspberry Pi Zero. If the student's timing is with some threshold of the expected timing, the input would be evaluated as correct. The threshold will be determined by a variable which can be altered by the instructor to change the strictness of an acceptable performance. The lower limit of correctness is determined by

$$(Student\ Timing) - (100\% - Percentage\ Variable)(Desired\ Timing) \geq 0 \quad (1)$$

and the upper limit of correctness is determined by

$$(Student\ Timing) - (100\% - Percentage\ Variable)(Desired\ Timing) \leq 0 \quad (2)$$

If both timings are within the threshold, the button push would be marked as correct. For each lesson, the Raspberry Pi would connect to the firebase database to access the correct timing data.

We used 3D printing to construct an enclosure for the student interface. Using the 3D printing services available for free in the UT makerspace allowed us to draft an enclosure prototype with polyactic acid (PLA) filament for no cost. To create the final enclosure we took measurements of our final device and used Autodesk Inventor computer aided design software to draft an interlocking case. Our design for the case included four pieces that would screw together and hold the hardware securely in place. We made sure that only the volume control, power switch, keys, and charging port are accessible to the user. The schematics for the design is given in Appendix A, which shows multiple views of each piece and the complete assembly.

3.2 Teacher Web Interface

For the teacher web interface, we created a web application that allows the teacher to create lessons, edit lessons, display the lessons created to the students, and visualize student performance data. The web interface interacts with a database to store and retrieve information. The lessons created by the teacher are stored in the database, so they can be accessed anytime and modified if needed. The web application obtains the student performance data from the hardware controller through the database. A detailed representation of the teacher's web interface is shown in Figure 4.

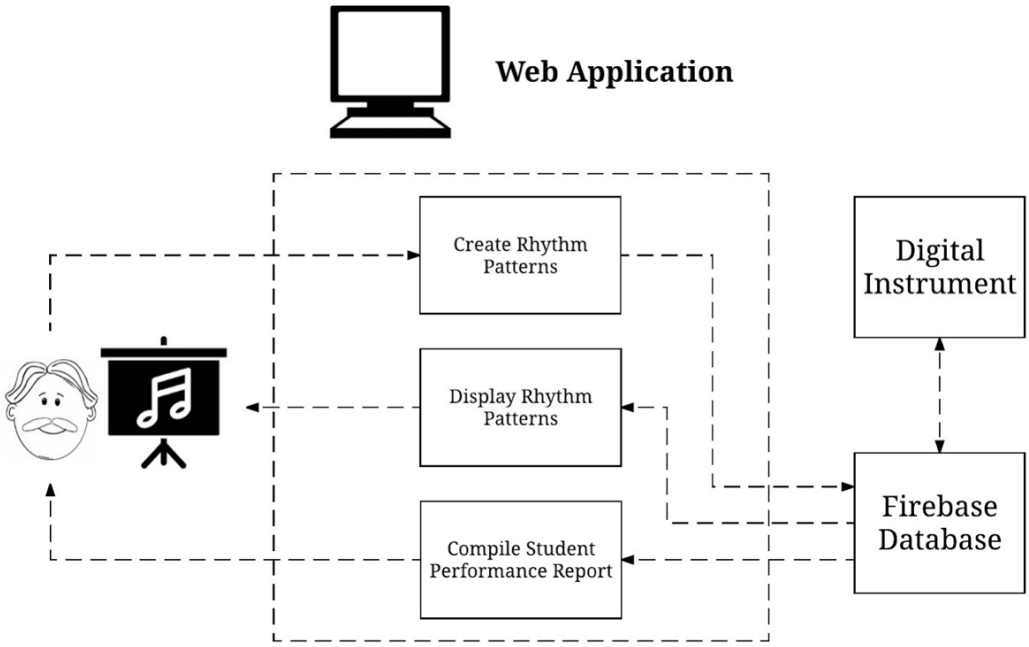

Figure 4: Teacher Interface Block Diagram

For the creation of our web application, we used a structural JavaScript framework for dynamic web applications called AngularJS, and for our database we used a real-time cloud-hosted database called Firebase. We decided to use AngularJS because it helps simplifying the process of building web applications by using a Model-View-Controller design pattern which helps make the code easy to understand, adaptable, and extensible to other libraries. AngularJS also has testing already enabled, which is helpful for doing unit and system tests of the application. Another reason why we decided to use AngularJS is because it was developed by Google, which allows an easy integration with other Google services, such as Firebase. We decided to use Firebase because, besides it being easily integrated with AngularJS, it is a real-time cloud-hosted database that allows for storing data in the cloud instead of locally on the teacher's computer. This means that the teachers will be able to access any data they want from any computer. Integrating AngularJS with Firebase will make the building process of our web application easier, faster, and more productive.

As shown in Figure 4, the web application uses Firebase to manage the lessons created by the teacher. The teacher will be able to create, edit, and delete rhythmic pattern lessons through the

UI of the web application. A lesson is composed by one or more different rhythmic patterns with each having their own beats per minute value. A rhythmic pattern consists of a 4 by 4 grid of buttons, were each button is assigned a note duration, such as quarter note, half note, and whole note. We differentiated each note duration using colors, where purple is used for whole notes, green is used for half notes, blue is used for quarter notes, and gray is used for rests. Figure 5 shown below is an example of a lesson being created or edited.

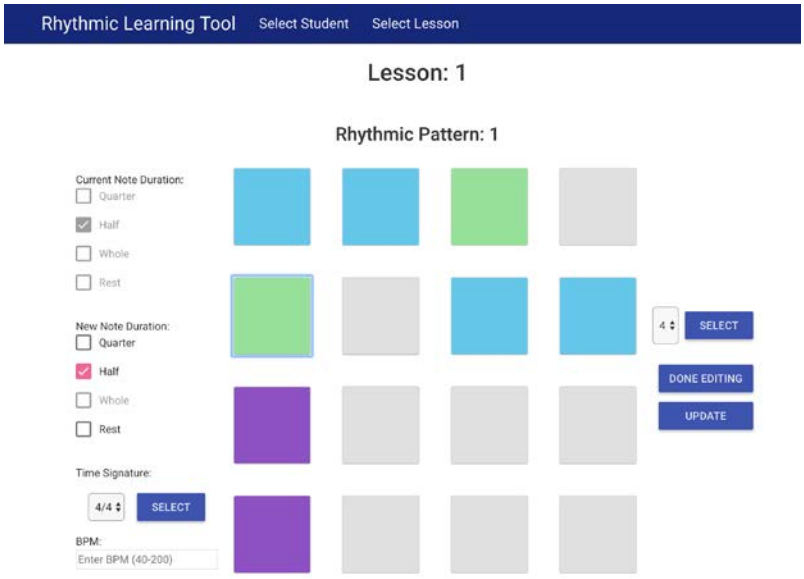

Figure 5: Edit Lessons Page

Once a lesson has been created, it is stored in Firebase. The web application, as shown in Figure 6, shows every lesson the teacher has created and lets the teacher modify them or delete them. When a teacher decides to edit a lesson, Firebase updates the old version of the edited lesson with the new edited version. Similarly, when a teacher decides to delete a lesson, the lesson is removed from Firebase. The teachers have full control on the lessons they create.

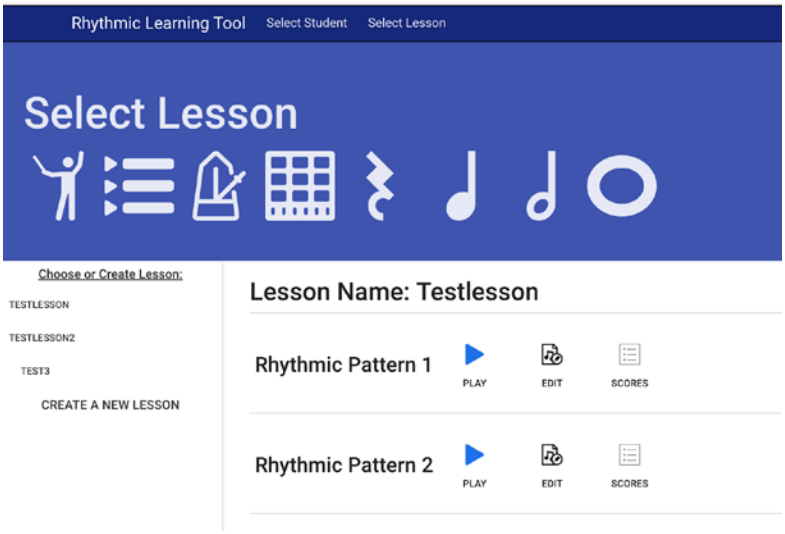

Figure 6: Select Lesson Page

The web application also has a module for presenting rhythmic pattern lessons to the students. The teacher can choose from a list of lessons stored in Firebase, the lesson or rhythmic pattern they want to display to the students. When the teacher selects a lesson, the web applications creates an animated version of the lesson showing which buttons should be pressed according to the beats per minute value stored in Firebase. The web application also produces a sound for every beat, like a metronome, to help guide the students. The goal for this module is for the teacher to use it to teach rhythmic patterns through visuals and audio. Figure 7 shows a screenshot of this module.

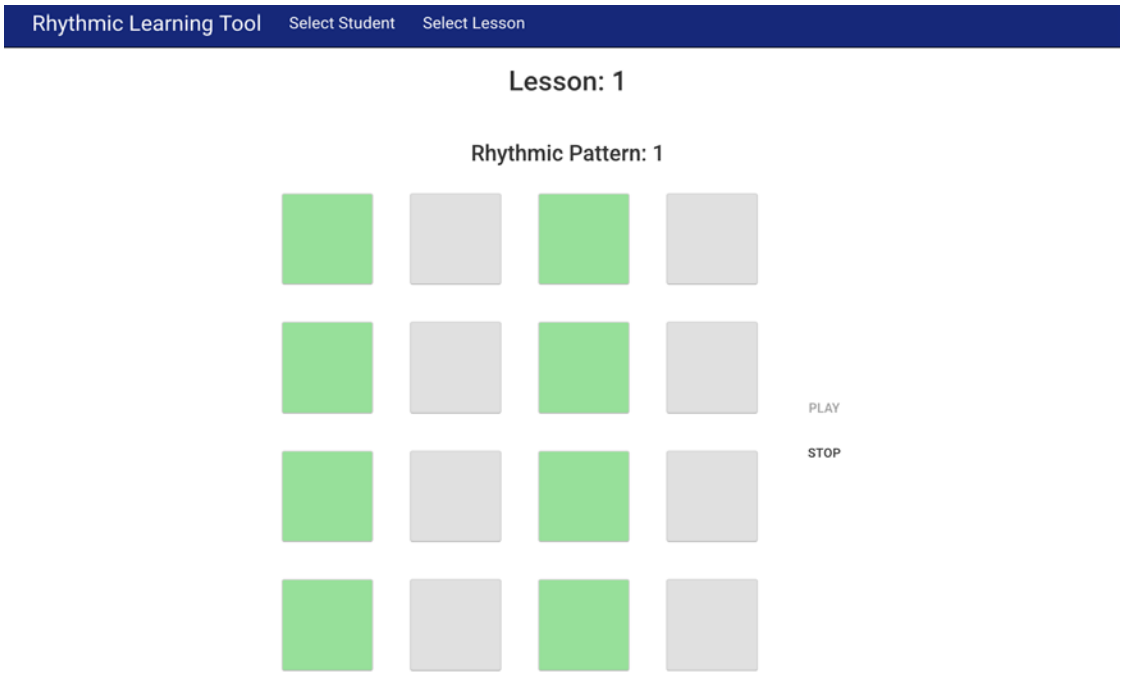

Figure 7: Play Lesson Page

The web application also allows the teachers to view their students' performance data from each lesson anytime. It obtains each student's performance data through Firebase and displays it on the UI. As described earlier, music teachers do not have a quantifiable way of grading students, or analyzing if the students are learning or not, especially in large classrooms with many students. This module will assist make grading easier and more accurate, and will let the teacher know how effective the lessons are. An example of this module is shown below in Figure 8.

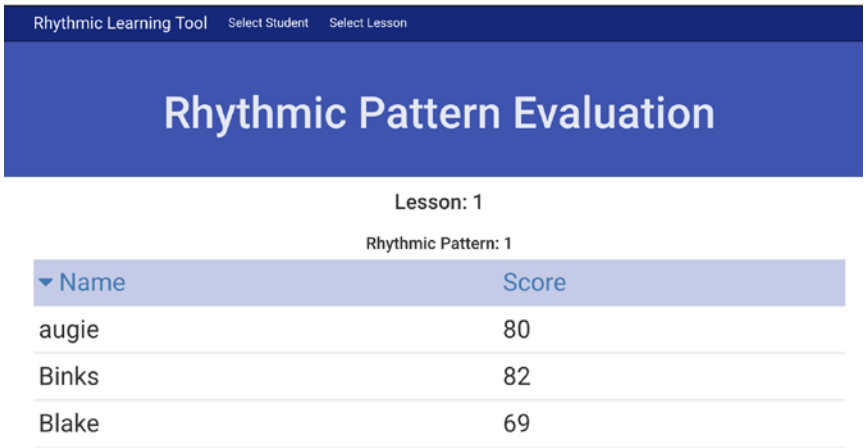

Figure 8: Rhythmic Pattern Evaluation Page

4.0 DESIGN IMPLEMENTATION

During our semester in EE 464, our team successfully completed our projected proposed in EE364D and described in Section 3.0. In this section we describe our implementation process for both the software interface and the hardware interface, and explain the issues we encountered, the design decisions we made, and the different technology tools we tried.

4.1 Student Hardware Interface

During the design process for the student interface, we changed the core component of our design from the Arduino Uno microprocessor to the Raspberry Pi Zero W. We initially encountered problems with connecting to the internet wirelessly through a Wi-Fi shield. The shield was set by default to the wrong baud rate to interface with the Uno, and, from searching the internet, we could not find the correct documentation for the firmware version of our shield. The team chose the Pi since it has built in Wi-Fi and significantly reduced the cost of our design. We encountered difficulty with enabling the Wi-Fi on the Pi on campus due to the WPA Enterprise protocol security, but were able to connect after installing the certificate authority (CA) from the UT website and including it in the login information. After connecting to the campus network, we could then begin to code on the Pi. Our first task was to ensure that the Adafruit Trellis would function on the Raspberry Pi using the python drivers. Afterwards, we used the python-firebase functionality built into the python on the operating system to begin sending and receiving information to and from the firebase database. During the same time, we also redesigned our audio implementation, since our original design utilized a digital-to-analog (DAC) chip to interface with the Arduino Uno through its synchronous serial interface (SSI). After switching to the Raspberry Pi, it was far more difficult to create audio using interrupts as originally planned, but the team found that we could use the built in pygame library to send audio to an output pin using the onboard pulse width modulation (PWM) DAC. This method was very noisy and we had to filter the audio output, amplify it, and filter it again. The circuit diagram can be found in section 3.1. Similar to the difficulty in manually enabling interrupts as we would on a microcontroller, we also had difficulty in accessing the hardware timer on the Raspberry Pi. We were able to access it in C instead of python, which also required running the function as an administrator in the terminal to allow the program to access low level hardware where the clock could be found. This also introduced the issue of integrating C functions into our

full python script. After learning to write python wrappers for C functions, we began working on the grader function, which was also in C. At this time, we had also combined the audio and timer functions into the main script and could record arrays of button presses with their corresponding timings while playing a sound for each. We used the python script to pull lesson timing data from firebase and append it to the array of recorded data from the key presses to pass into the grader function. The team had difficulty in establishing the best method for calculating a single grade from the data. Ultimately, we had to compile test data and account for latency manually while adjusting the threshold such that it would be easy for someone with no experience to earn a 100 after a few tries. This also led us to realize that it would be best to have the threshold for correctness be adjustable by the teacher in the future to accommodate for different levels in music education.

4.2 Teacher Web Interface

During the implementation process of the software interface we encountered a few challenges that made us try different and new technologies, however we were able to complete our original design with its core functionality. We started by initializing our web application with an Angular skeleton template. We tried multiple templates, such as Angular-Seed and Lineman, until we found Yeoman, whose features are perfect for our design requirements. Using a template helped us speed up the development process by not having to start the web application from scratch. The only issue with using a template is that each template has its own structure, coding pattern, and format, which ended up affecting the development process since it required the team to learn how to use it and to accommodate to it. The main challenge that our team experienced while working on the web application is our initial lack of experience in web development. We had to spend some time watching tutorials and learning the different languages and tools for web development, such as HTML, CSS, JavaScript, and AngularJS. Once we had a basic skeleton for our web application, we started looking into JavaScript libraries related to the features and functionality we had planned for our application. We found multiple audio and visual JavaScript libraries, like Tone.js, NexusJS, and Bounce.js, however, some of them were either very complicated to use or lacked some of the features we were looking for. Looking for the best libraries to use was very challenging and time consuming since it required learning the documentation for each library, and apply it to our web application. We ended up not using most

of the libraries we tried, and only using Bounce.js for just one of the modules. Another issue we faced during the implementation process was incorporating Firebase to AngularJS. When we started working on the web application, Firebase had just had an update, which made most of the documentation and tutorials online outdated. It was a challenge because we had to learn how to incorporate Firebase to our AngularJS application without up-to-date documentation or tutorials. Despite all these setbacks, we were able to finish our web application on time with all the features required in our original design plan.

5.0 TEST AND EVALUATION

Our testing and evaluation can be viewed as three different sections: hardware testing, software testing, and system testing. For our hardware testing, our team focused on testing the hardware controller electrical components, communication to firebase, and evaluation of the data gathered from firebase. Our software testing was relatively simple compared to the hardware testing, only needing to test the GUI of the webapp and communication with firebase. Finally, our system testing involved determining the ability of our system to integrate both the hardware and software components correctly.

5.1 Hardware Testing

Our hardware testing began with determining the functionality of our electronic components. We initially determined the current that we needed to provide the Raspberry Pi Zero W from the device battery. For our test, the Raspberry Pi was connected to a lab bench power supply to see what current is required for normal device operation. We also consulted the technical data sheets of the Raspberry Pi to determine that the device can handle a maximum current of 350 mA and has an average current of 230 mA. The bench power supply was set to 5V and the Raspberry Pi was run under different conditions. The Raspberry Pi current was measured during boot, idle, a poll to firebase using python script, and finally while executing a rhythmic pattern lesson. The results can be viewed in Table 1. The table shows that our worst case current consumption is 218 mA, which will give us 45.87 hours of battery life from a 10,000 mAh battery.

Table 1. Current Drawn Test Results (mA)

	Boot Sequence	Idle	Firebase Poll	Rhythmic Pattern
Maximum	208	130	218	148
Minimum	130	122	198	132
Average	169	126	208	140

In addition to the battery life test, we tested the Trellis keypad with LED's by creating a python program to read the timing and button press using terminal. The buttons were assigned unique names which recorded the press time and release time based on the internal Raspberry Pi clock. We were able to view that the buttons pressed were being recorded correctly and that the LED would light up.

Our next hardware test involved communication between the hardware device and Firebase. We tested the communication by creating several python scripts which pull and push data to and from Firebase to the Raspberry Pi. After the scripts were designed, we tested the edge cases by populating Firebase with 2 full rhythmic patterns each with 4 sequencers and 16 button presses per sequencer. We manipulated the Firebase database to test for different data scenarios while pulling data which can be viewed in Figure 9 and Table 2.

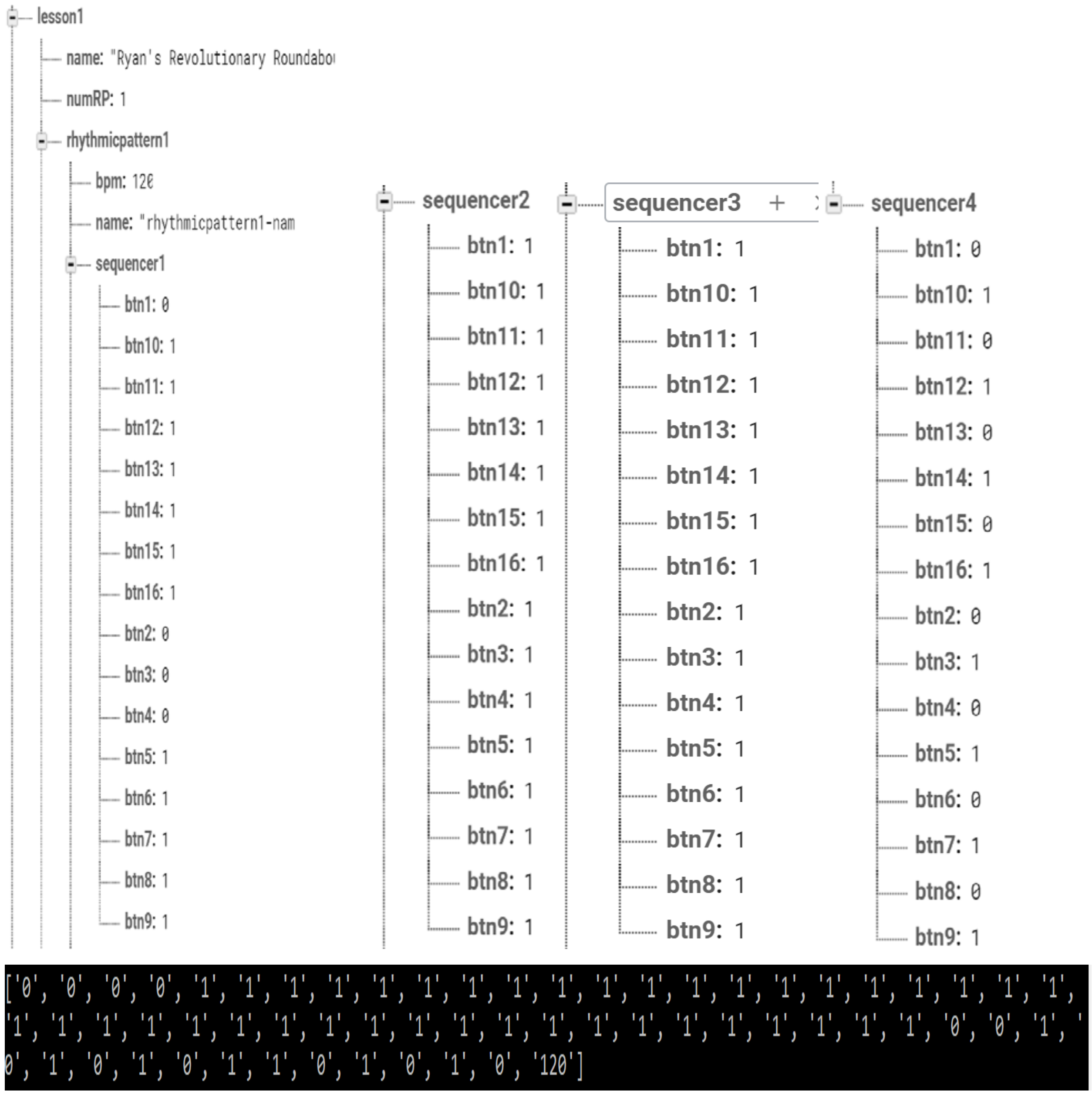

Figure 9. GetButtons.py Test

Table 2. GetButtons Function Test Results

Test Case (Firebase Data)	Result
Complete Rhythmic Pattern with 4 sequencers and a beats per minute value.	All data was stored and passed correctly.
Incomplete Rhythmic Pattern with 3 sequencers.	Error from trying to access the fourth sequencer when it doesn't exist.

Current lesson number does not exist.	Error from trying to access a lesson that doesn't exist.
Current rhythmic pattern does not exist.	Error from trying to access a pattern that doesn't exist.
Beats Per Minute is not set.	The function will pass 0 bpm to the grader, which will cause the duration of all buttons to equal 0. Therefore, no button presses will be correctly evaluated unless the duration is originally zero.
Current rhythmic pattern is not updated.	The function will retrieve old rhythmic pattern button presses, and will not be graded against the correct lessons.
Current Lesson is not updated.	The function will retrieve the current rhythmic pattern from the wrong lesson. The grading will not be correct because it will be compared against a different lesson.

The results from this test show that we can pull data from the database correctly, but we need to take caution when populating Firebase nodes. If a node is incorrectly named, or the additional data fields such as beats per minute, current rhythmic pattern, or current lesson are no updated, then the system will fail to read correctly. One result from this test was that pulling data from Firebase used a variable amount of time which would not meet our real-time specifications unless the lesson started with a variable delay to sync the hardware and software devices. We decided to preload the lesson to the Raspberry Pi before the lesson began to minimize the time spent making Firebase calls during operation. This helped us to reach our goal of becoming a real-time system.

Once we were able to receive data from Firebase, we needed to test our ability to send information to the database. Our FireSend python script was developed to gather the graded information and send it to the performance section of Firebase. Two students were created with separate grades and were pushed to Firebase. The results can be viewed in Figure 10 and Table 3.

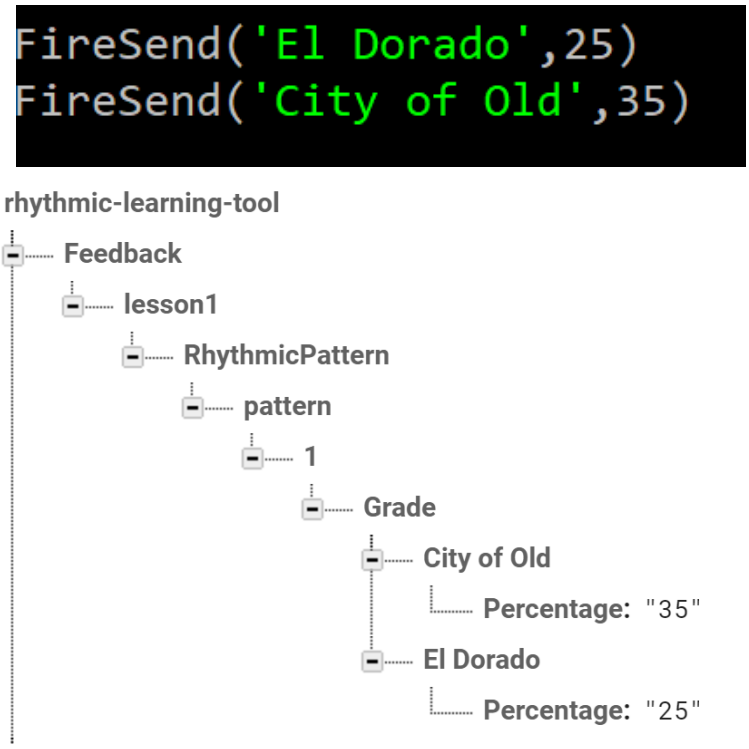

Figure 10. FireSend.py Test

Table 3. FireSend Function Test Results

Test Case	Result
Two students with different identifiers performing the same rhythmic pattern in the same lesson.	Both student performances were captured and nested correctly under the lesson and rhythmic pattern.
Two students with the same identifier performing the same rhythmic pattern in the same lesson.	The second student overwrote the data of the first student. Each student must have a unique identifier.
The current lesson was not updated.	The data will be pushed to an incorrect location which corresponds with the value of current lesson. We must ensure the lesson is updated to the desired lesson.
The current rhythmic pattern was not updated.	The data will be pushed to an incorrect location corresponding to the current rhythmic pattern under the current lesson. We must ensure that rhythmic pattern is updated.
A student performs a lesson and rhythmic pattern multiple times.	The data overwrote the previous lesson data.

The student grade is not a numeric argument.	The data will still be pushed as a string, regardless of format.
The student identifier is not a string.	The data will be pushed as a string regardless of format.

This results from this test allowed us to realize that we need to have unique identifiers for each student, and that if a student shares an identifier with another student, then every time a grade is sent, it will overwrite the previous grade. Additionally, since the data is pulled from the teacher web app interface, the data must be sent in the correct format and in the correct Firebase node locations, or the web app will not be able to recover the data. The reason that data correctness is a priority is so that it can provide feedback for the instructor, which was one of the features that we demanded from our system.

Our final hardware test was to take the information from Firebase and compare it to the data captured by the hardware device. By this point, we have already tested our electronics and database communication. To test the grader, we passed an array of button timing data from our Firebase function to the grader which can be viewed in Figure 11.

Figure 11. Note Duration Test .json

The values in the database stand for note duration and can be interpreted as a quarter note is “1”, a half note is “2”, dotted half note is “3”, and a whole note is “4”. Once the grader matched our database, we then began to test the button press timing against the Firebase values. We performed various testing on the edge cases which might occur. The results can be viewed in Table 4.

Table 4. Grading Test Cases and Test Type

	Test Type	**grader.cpp Sub-module Tested**
General Use	Perfect presses at perfect times, moderate speed	Evaluate
	Perfect presses at perfect times, fast speed	Evaluate
	Buttons pressed within “typical” time	Evaluate
	Simultaneous button presses	Evaluate, Interpreter
	Note duration combinations	Interpreter
Edge Cases	Student pressed 0 buttons	Evaluate, Parse
	Student pressed too many buttons	Interpreter, Parse
Evaluation	Button pressed early	Evaluate
	Incorrect Button pressed early	Evaluate
	Button pressed late	Evaluate
	Incorrect button pressed late	Evaluate
	Buttons not released when recording stops	Evaluate, Interpreter

The table shows that even under various conditions, the grader will still be able to evaluate the data even if the student is incorrect. This helps us to provide proper feedback to instructors since they need to know what a student does incorrectly. More results of this test can be viewed in Table 5.

Table 5. Initial Test Case Results

	Test Type	**Status**	**Reason Test Failed**
General Use	Perfect presses at perfect times, moderate speed	Pass	
	Perfect presses at perfect times, fast speed	Pass	
	Variable bpm	Pass	
	Buttons pressed within “typical” time	Pass	
	Simultaneous button presses	Pass	
	Note duration combinations	Pass	
Edge Cases	Student pressed 0 buttons	Pass	
	Student pressed too many buttons	Fail	Specified array was too small
Evaluation	Button pressed early	Pass	
	Button pressed 1 μs early	Pass	
	Button pressed late	Pass	
	Button pressed 1 μs late	Pass	

	Buttons not released when recording stops	Pass	

The results of this table are more extensive than in the previous table, and show that the grader will work under a variety of conditions such as different beats per minute or button press timings. With this test, the hardware tests were completed and met our expectations of having a real-time system which provides instructor feedback. While the hardware tasks were being complete, the software team tested their sections.

5.2 Software Testing

For our web application, we tested the completeness of lessons, text input, playback of rhythm patterns, and the consistency of font and color across web pages. The testing of lesson completeness and text input will check that our lessons uploaded to Firebase are correctly formatted. Our playback testing checks the correctness of rhythmic patterns consisting of quarter notes, half notes, and whole notes, at tempos within our defined beats per minute (bpm) range in a 4/4 time signature. Finally, the consistency of font and color across web pages will help provide a simple, uniform experience for the teacher.

We used unit tests to make sure our code does not let incomplete lessons be uploaded to the database. To create the unit tests, we decided to use Karma, a JavaScript test-runner, and Jasmine, a JavaScript testing framework because they are the preferred testing tools for AngularJS, which is the JavaScript framework our web application is built in [1]. To test for the completeness of a lesson, we created a unit test for each possible test case. The test cases we considered are: empty lesson, lesson with no name, lesson with no time signature, lesson with no rhythmic pattern, lesson with empty or incomplete rhythmic pattern, lesson with no beats per minute, and lesson with no empty fields. For each unit test we created a mock object of a lesson with the respective missing fields and tested for its completeness. The unit tests using mock objects of incomplete lessons must not upload them to the database, and the unit tests using mock objects of complete lessons must upload them to the database.

A test is considered successful if it contained an incomplete lesson and the lesson was not uploaded to the database, or it contained a complete lesson and the lesson was uploaded to the database. Table 6 shows the result of each test case specified in the method section above.

Table 6. Complete Lesson Test Results

Test Case	Result
Complete lesson is uploaded to the database	Passed
Empty lesson is not uploaded to the database	Passed
Lesson with no name is not uploaded to the database	Failed
Lesson with no time signature is not uploaded to the database	Passed
Lesson with no Rhythmic Pattern is not uploaded to the database	Passed
Lesson with an incomplete Rhythmic Pattern is not uploaded to the database	Passed
Lesson with no beats per minute is not uploaded to the database	Passed

From the results gathered after running each unit test, we found that 6 out of the 7 unit tests passed, and only 1 of them failed. Lessons with missing parameters such as time signature, complete and incomplete rhythmic patterns, and beats per minute are not being uploaded to the database, however, lessons with no name are being uploaded to the database. This means that our code is not checking if a name has been inputted for the lesson being created before uploading it to the database.

We used unit tests to ensure every lesson being created contains only the correct data with the correct format before the lesson is uploaded to the database. We decided to also use Karma and Jasmine for the creation of these unit tests. Each unit test account a different test case. The web application has two input fields that the user is required to fill, the lesson name and the beats per minute value. From these two input fields, we gathered the following test cases: the lesson name

is longer than 40 characters, the lesson name is less than or equal to 40 characters, “beats per minute” contains special characters, “beats per minute” contains only numbers, “beats per minute” is within the range of 40 and 200, and “beats per minute” is not within the range of 40 and 200 “beats per minute”. These unit tests will test different text validation functions that only allow the correct data and format for each field to be inputted.

A test case is considered successful it only allows the correct format and data to be inputted. The results of each test case are shown on the next page in Table 7.

Table 7. Input Validation Test Results

Test Case	Result
Lesson name is longer than 40 characters	Failed
Beats per minute has special characters	Passed
Beats per minute is outside the range required	Passed
Beats per minute is within the range required	Passed

From the results gathered after running each unit test, we found that 3 out of the 4 unit tests passed, and only 1 of them failed. The beats per minute field passes all the required tests, but the lesson name field does not. This means that our code fails in checking the length of the lesson name.

To check that our web application correctly played back quarter notes, half notes, and whole notes we created a .json file to upload to our database. The .json file contained a sample lesson with a rhythm pattern that used one sequencer for quarter notes, one for half notes, one for whole notes, and one with a combination of different note durations. We played the lesson first at 60 bpm and then again at 120 bpm and observed whether the playback matched our expectations as defined in the test file.

All quarter note, half note, and whole note were correctly played back at 60 bpm. All half note and whole note were correctly played back at 120 bpm, but the quarter note playback was incorrect when two quarter notes were positioned right after each other. The visual of the both quarter notes would be correct, but the sound was skipped in every second quarter note.
The results of the note duration test meet the specifications to play different note durations for quarter note, half note, and whole notes at 60 bpm but not at 120 bpm. The note durations played successfully for half notes and whole notes at 120 bpm, but missed the second quarter note if two were positioned back to back. We believe this is due to the JavaScript code responsible for the sound. There appears to be a limit to the beats per minute for the method we are using that we must overcome.

Our team did a consistency inspection to test the color and font consistency of our web application's user interface. For the consistency inspection we used the Google Chrome extensions WhatFont and ColorPick Eyedropper to check the font family and hex colors for the Navbar, Heading, Sidebar, and Background on each page. We enabled WhatFont and ColorPick and scrolled over each instance of text and color on the page. Once obtained, we recorded the font family and hex color code in the table below. Table 8 shows the results gathered from the consistency inspection.

Table 8. Consistency Inspection Test Results

Web Page	Font Family	Navbar Color (Hex)	Heading Color (Hex)	Sidebar Color (Hex)	Background Color (Hex)
Lessons	sans-serif	#1B277C	#4154B2	#FFFFFF	#FFFFFF
Student Performance Report	sans-serif	#1B277C	#4154B2	#FFFFFF	#FFFFFF
Rhythm Pattern Evaluation	sans-serif	#1B277C	#4154B2	N/A	#FFFFFF
Edit Rhythmic Pattern	sans-serif	#1B277C	#4154B2	N/A	#FFFFFF

Play Rhythmic Pattern	sans-serif	#1B277C	#4154B2	N/A	#FFFFFF

The results of the consistency inspection meet the specifications for a consistent user interface. All web pages used sans-serif as the font and had matching colors for the Navbar, Heading, Sidebar, and Background where applicable.

For our Firebase database testing, we evaluated our ability to send and receive data from Firebase using both the teacher web application and the student hardware interface. We began by testing our ability to create lessons with the teacher interface to make sure the database reflects the created lesson. Secondly, we tested our ability to retrieve the button presses with the student hardware interface. We had to make sure that the data being retrieved matched the data on Firebase, which was previously tested to match the instructor lesson data. Finally, we evaluated the ability to send the performance data back to Firebase after sending the button presses to the grader.

We created new lessons using the ‘create new lesson’ button and then inspect the Firebase database to ensure that the names of lessons are stored correctly. We also inspected the web application to verify that the lesson list is being pulled from the database correctly and giving the correct name of the lesson chosen at the top of the lesson list page. During this test we will also be inputting special characters into the dialog box to make sure we can have a wide variety of lesson names. We will be using a blank lesson name, names with punctuation and numerical values, and names with the ‘$’ character in them since AngularJS uses this character as a special symbol within its framework. We will also be running the same test on the ‘create new rhythmic pattern’ button. This test should be easier since rhythmic patterns will not have user inputted names, and instead will be stored in Firebase in the format of “Rhythmic Pattern X”, where X is pulled from the Firebase as a running index indicating which number rhythmic pattern is to be stored next. Lastly, this test will make sure that when we create a new lesson or attempt to play/edit a lesson that book keeping variables stored on the database are updated as well (including the variable that stores which rhythmic pattern is to be stored next in a given lesson).

All names were able to be stored without problems given the testing parameters. All symbols and punctuation worked. Even special characters such as '$'. We attempted to name lessons both "$scope.currLessons" and "{{lesson.name}}" which are global and local variables, respectively, to the lesson list page and functions and they still displayed properly. When populating the lesson list with the stored names, the names were displayed in with capital letters, however when a lesson is selected, the name appears as typed at the top of the page. There was also an issue with not being able to submit a lesson with a blank name or a name that contained nothing but spaces. When this is the case the create button for the dialog box to create a new lesson will be unclickable. To test what would happen if that were to somehow make it on the database we populated the database with a lesson containing a blank name. It still showed up on the lesson list as a button, but it appears as a blank space. However, it still functions as normal. The rhythmic patterns were also stored and displayed in order and in the correct lessons. The only issue was when we attempted to create a new rhythmic pattern when no lessons existed. Firebase will make a lesson that shows up on the database, but it will not populate to the lessons list. Continued presses of the 'create new rhythmic pattern' button will continue to populate this faux lesson with rhythmic patterns. However, when a lesson is created, it overwrites the non-working lesson in the database and all the created rhythmic patterns. Lastly, the counter for the number of lessons and number of rhythmic patterns were incremented correctly as we made lessons and lessonselected and rhythmicpatternselected, housekeeping variables that are used to determine which lessons the hardware devices pull from the database, were updated in the database as expected when the play button or the edit button was clicked.

Minor issues include the lesson list displaying as all capital letters and a blank lesson name not being able to be stored to the database. The capital letters issue can be fixed in the html code and we don't foresee any need for being able to submit a blank lesson name. The most worrying bug, while we haven't been able to get it to break the web application or Firebase yet, is the unusable lesson created when a rhythmic pattern is created while no lesson exists.

We then used a testing method that made use of the GetButtons function which serves to pull the current lesson, rhythmic pattern and button duration from Firebase for use in the Grader function. To test this function, we populated Firebase using .json data which contained 2 full rhythmic

patterns each featuring 4 sequencers which have 16 buttons each. Using the Raspberry Pi, we demonstrated that we could pull all database information, and create a passable array to the C++ grader. Table 9 details the results we gathered after testing the function GetButtons.

Table 9. GetButtons Function Test Results

Test Case (Firebase Data)	Result
Complete Rhythmic Pattern with 4 sequencers and 60 beats per minute value.	All data was stored and passed correctly.
Incomplete Rhythmic Pattern with 3 sequencers.	Error from trying to access the fourth sequencer when it doesn't exist.
Current lesson number does not exist.	Error from trying to access a lesson that doesn't exist.
Current rhythmic pattern does not exist.	Error from trying to access a pattern that doesn't exist.
Beats Per Minute is not set.	The function will pass 0 bpm to the grader, which will cause the duration of all buttons to equal 0. Therefore, no button presses will be correctly evaluated unless the duration is originally zero.
Current rhythmic pattern is not updated.	The function will retrieve old rhythmic pattern button presses, and will not be graded against the correct lessons.
Current Lesson is not updated.	The function will retrieve the current rhythmic pattern from the wrong lesson. The grading will not be correct because it will be compared against a different lesson.

Precaution should be taken to ensure that the current rhythmic pattern, current lesson plan, and beats per minute are updated correctly. If they are not, then the grader will be sent an incorrect lesson and rhythmic pattern, and all grades will be incorrect. Additionally, each lesson must contain at least 1 rhythmic pattern, and each rhythmic pattern must contain exactly 4 step sequencers with 16 button press duration values. If there is no lesson available, a hardware error will occur.

To test the reliability of our Firebase database for communication to the Raspberry Pi, we made use of our FireSend function which passes performance data to Firebase from the student interface. To test the function, we created dummy variables which included a student identifier, and a student performance score. These variables were then sent to the function to ensure that Firebase updated the correct information in the nested location corresponding to the lesson and rhythmic pattern. The following tests need to be performed so we can test the edge cases with both complete and incomplete data.

Table 10 shows the results we gathered after testing the function FireSend with both complete and incomplete data.

Table 10. FireSend Function Test Results

Test Case	Result
Two students with different identifiers performing the same rhythmic pattern in the same lesson.	Both student performances were captured and nested correctly under the lesson and rhythmic pattern.
Two students with the same identifier performing the same rhythmic pattern in the same lesson.	The second student overwrote the data of the first student. Each student must have a unique identifier.
The current lesson was not updated.	The data will be pushed to an incorrect location which corresponds with the value of current lesson. We must ensure the lesson is updated to the desired lesson.
The current rhythmic pattern was not updated.	The data will be pushed to an incorrect location corresponding to the current rhythmic pattern under the current lesson. We must ensure that rhythmic pattern is updated.
A student performs a lesson and rhythmic pattern multiple times.	The data overwrote the previous lesson data.
The student grade is not a numeric argument.	The data will still be pushed as a string, regardless of format.
The student identifier is not a string.	The data will be pushed as a string regardless of format.

From the test, it is apparent that we need to take special precautions when pushing data to the performance section of Firebase. If we push any data, and the field does not exist, then a new

field will be created in Firebase regardless of the data type pushed. Furthermore, if a field exists in the performance section of Firebase, then it will be overwritten if the same field is pushed again.

5.3 Integration Testing

To test our entire system, we populated Firebase with a full lesson using the web app, pulled the data to the hardware controller to be evaluate, performed the lesson, sent the data to Firebase from the Raspberry Pi, and displayed the graded results. This test built on the previous hardware and software tests and allowed us to test our full system with the already completed modules. With this test, we were able to see how the timing differed between the displayed button presses and the actual hardware controller button presses. To begin, we created three basic rhythmic patterns with one button duration per sequencer and one complex rhythmic pattern with multiple button durations in a sequencer, and example of which can be seen in Figure 12.

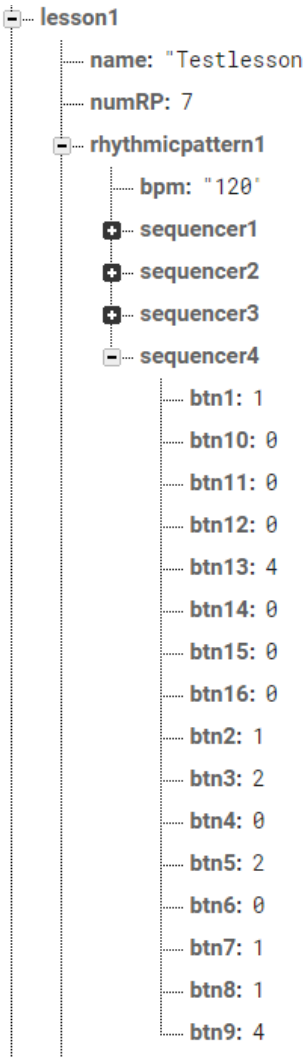

Figure 12. Complex Sequencer

We then used the Trellis keypad to play along with the rhythmic pattern and viewed the local button press timing using terminal to determine if the button presses were occurring at the correct time. If the buttons were pressed correctly locally, and they were labeled incorrect by the grader, we needed to adjust the threshold for correctness. We completed this calibration over multiple sessions using multiple users to adapt the system. Thresholds were adjusted based off of a percentage of note duration. This test proved that our system worked and allowed us to move on to additional features.

6.0 TIME AND COST CONSIDERATIONS

Overall, the project was completed in a timely manner for a very low cost. Our budget constraints were our most limiting factor at the beginning of the project, and towards the end of the project time was the limiting factor.

One of the largest obstacles that faced our team was the lack of a budget. Although most teams were initially funded $1,000, we did not have a corporate sponsor. Initially, we contacted a company called Genesis which assists students with funds for startup projects. After several months, we were not given any funding which caused a large setback with our project. Without funding, we were reliant on our personal funding to build our hardware device and purchase software. Fortunately, we found free programs to use for our project, free printing resources for our hardware enclosure, and an inexpensive Raspberry Pi Zero W which helped minimize the costs, however it took extra time away from our project to search for them. By using our own funds and finding free software that would meet our needs, our budget obstacle was eliminated.

Towards the end of our project, time became an obstacle. As we approached the date for Open House, we were completing the basic functionality of our rhythmic learning tool. Once we completed the basic features of the hardware interface and software interface, all that remained was to integrate them together. Unfortunately, the integration step used more time than we had anticipated. During our first complete system tests, we noticed that the timing between the display and hardware controller were mismatched. The issue caused our rhythmic learning tool to read button presses at incorrect times compared to the displayed correct time. Our response to this problem was to preload all of the correct data to minimize latency, and to adjust the limits of correctness for the student hardware interface. This process took longer than anticipated because after each change we would need to test the entire system again. However, we were able to calculate the threshold correctness for button presses and releases, and were able to meet the deadline for Open House.

During the development of the rhythmic learning tool, our team face both budgetary and time constraints. We managed to work around the issues by finding free alternatives for our hardware components and software, and finding a less expensive device to process information. We stayed

highly organized throughout the semester with meeting agendas which helped to keep focus on the task at hand and the remaining time.

7.0 SAFETY AND ETHICAL ASPECTS OF DESIGN

While there are many safety and ethical issues one must consider when designing any electronic device, during the design the Rhythmic Learning Tool we also took into account the fact that this is a device intended for children and adjusted our safety standards accordingly. Some of the broader concerns for our device included the life cycle of our components as well as security regarding student and teacher records. Meanwhile, concerns specific to our project include things like safety related to devices for children such as the volume of the audio feedback as well as not having any electronic components exposed when the product is handled by children.

7.1 General Safety and Ethical Concerns

The general concerns for our device when it comes to ethics are the replacement of components as well keeping the records of students and teachers confidential. The hardware devices battery can be charged via micro USB through a port built into the 3D printed enclosure to prevent students or teachers having to open the device. However, if the battery fails to hold a charge or another component of the device malfunctions the enclosure can be opened very easily by loosening just a few screws and the malfunctioning components can be replaced. We would have liked to have implemented some security for our system so that each teacher can have secure access to their own database of lessons and students but were unable to include this in our project due to time constraints. If we were to move forward with this plan we would probably use something like the Google Identity Platform [1] for an easy login process for the teachers as well as functionality with the Google Firebase database.

7.2 Safety and Ethical Concerns Related to Children's Products

To make sure that our system is safe for children we made sure the onboard speaker wasn't loud enough to cause hearing damage as well as make sure that the device wouldn't expose children to electronic components. Since we developed the hardware portion of our system to be used in an elementary school classroom, we were mindful to create a device that will be durable enough to stand up to the use of young children and safe enough to be approved for school use.

Unfortunately, there are currently no concrete standards that a digital learning tool must meet to be used in a publicly funded school. The loose standard is based on the variation of standards between school districts and state legislature. However, we must ensure that our device will not cause any hearing damage. Based on our research, a volume for the onboard speakers that will be useful and safe for a device like ours falls in the range of 0 dB - 75 dB [2]. We used Sound Meter, an android phone application, to test the sound level of the speakers output and the loudest reading at our maximum volume was 58dB which falls into our allowable range. Lastly, we designed our 3D printed enclosure to not open without use of a screwdriver to ensure that children cannot access any electronic components.

8.0 RECOMMENDATIONS

Based on our design and implementation experiences through EE 464, our team has several recommendations for future work on the rhythmic learning tool system. Our recommendations focus on new features for rhythmic patterns and digital instrument design modifications.

8.1 Note Durations and Time Signatures

While the note durations and time signature currently implemented cover much of the basic rhythmic structure taught in elementary school music class, our team recommends implementing two more note durations and the ability to change time signatures to cover a wider range of lessons. The features our rhythmic patterns currently support include multiple note divisions to teach children the concepts of quarter notes, half notes, and whole notes in a 4/4 time signature. Eighth notes and sixteenth notes also commonly appear in teaching rhythm and would allow for more familiar rhythmic patterns based on known songs. We believe our system also lends itself towards teaching the more advanced topic of time signatures to students in a simple intuitive manner. Showing students 3/4 time for waltzes and 5/4 time for tangos by lighting up LEDs in corresponding patterns on the digital instrument has the ability to give children a visual representation often lost in verbal and even musical demonstrations. The additions of these two new note durations and time signatures would make the rhythmic learning tool more robust and flexible for a wider range of lessons.

8.2 Digital Instrument Modifications

Due to the time and cost constraints of our project, our team used a Raspberry Pi Zero, USB battery pack for power, and Trellis keypad with trade-offs in performance, form factor, and flexibility respectively. While our digital instrument successfully utilized a Raspberry Pi Zero for its on board Wi-Fi capabilities, we ran into issues implementing real-time features, such as audio and visual feedback, due to the extra layer of abstraction an operating system adds. Our team would recommend implementing future digital instruments with a microcontroller to overcome these issues. The digital instrument we created used a battery pack to power the device, and while it gave our instrument an impressive battery life of 45 hours of use, it resulted in a larger enclosure than originally planned. Our team would recommend using a small rechargeable lithium ion polymer battery or coin cell battery to achieve a smaller form factor in future designs. Finally, our device used a Trellis keypad for student input and LED feedback due to its small form factor and affordability. However, using the Trellis limited us to monochrome LEDs and buttons. To achieve greater functionality our team would recommend creating a custom keypad to allow for multicolor LEDs and force sensing resistors instead of buttons. Multicolor LEDs would allow for more detailed feedback to students based on the color of the LED, and force sensing resistors would help teach children dynamic range by basing volume off of the pressure of the presses. We believe the additions of these features would make the digital instrument more responsive, with a better form factor, and more engaging to students.

9.0 CONCLUSION

This document discussed the complete project development and implementation of our EE 464 Rhythmic Learning Tool Project. The design, implementation, evaluation, and further considerations required to create a system to improve elementary school music education are examined the preceding sections. We first examined how our project helps music educators teach elementary school students rhythmic structure and meter in the classroom more efficiently than standard methods by using a hardware interface and web application to record and evaluate student performance data. In the next sections we looked at the design concept and implementation of our rhythmic learning tool. We discussed how our web application allows teachers to manage lessons consisting of rhythmic patterns and concept summaries projected to a class of students, while the digital instrument offers a kinesthetic approach to learning as well as

provide visual and aural feedback to improve students' intuitive understanding of rhythmic structure. We used an online database to store lessons, student performance data, and allow wireless communication between the teacher's web application and the students' digital instruments. Following the examination of our design, the report details our successful testing and evaluation of the system and its modules. Our completed web application, digital instrument, and online database were tested by creating patterns via the web app, communicating the patterns to our instrument to be evaluated, and transmitting our evaluation back to the web app verifying that our specifications were met. Next, we looked at the time and cost considerations of our project as well as the safety and ethical concerns of an elementary education tool. Finally, we wrapped up our report by looking at recommendations for our project, such as new features for rhythmic patterns and modifications to our digital instrument.

In the end, our Rhythmic Learning Tool Project met the expectations and specifications we set for our design implementation plan and EE 364 documentation with the exceptions of a second digital instrument and the ability for multiple time signatures. We completed our project with a functioning web application, real-time database, digital instrument, and an overall third place finish at the ECE Senior Design Open House. We plan on further expanding our project by adding support for multiple digital instruments and deploying our website with the hopes of testing our rhythmic learning tool in an elementary school classroom.

REFERENCES

[1] *Google Sign-In.* [WebOnlinesite]. Available: https://developers.google.com/identity/

[2] “Interactive Sound Ruler: How Loud is Too Loud?”, *www.nidcd.nih.gov,* 2017. [Online]. Available: https://www.nidcd.nih.gov/health/interactive-sound-ruler-how-loud-too-loud

APPENDIX A – ENCLOSURE SCHEMATIC

APPENDIX A – ENCLOSURE SCHEMATIC