JnJ’s Clockwork

Software and Hardware Specification Sheet

Juan Sebastian Rodriguez, Jordan Pulido, Johnson Dinh

April 25, 2019

# Declaration of Joint Authorship

We, Juan Rodriguez, Jordan Pulido, and Johnson Dinh, confirm that this work submitted for the project is the joint work of ourselves, and does not infringe upon anyone’s copyright nor violate any proprietary rights. Any uses made with other authors’ ideas, equations, figures, techniques, or any other material from the work of other people included in the project are in accordance with the standard referencing practices. Johnson Dinh has handled the database connection along with the app integration for the hardware, Juan Sebastian Rodriguez created the foundation of the app, along with its features and functionality. Juan also designed the overall structure of the hardware’s appearance. Jordan Dave Pulido was responsible for the assembly and functionality of the Amplifier component of the hardware.

# 

# Proposal

January 17th 2019

***Proposal for the development of JnJ’s Clockwork***

Prepared by Johnson Dinh, Juan Rodriguez, Jordan Pulido  
*Computer Engineering Technology Student*https://github.com/JuanRodriguez19/JnJ-s-Clockwork

**Executive Summary**

As a student in the Computer Engineering Technology program, I will be integrating the knowledge and skills I have learned from our program into this Internet of Things themed capstone project. This proposal requests the approval to build the hardware portion that will connect to a database as well as to a mobile device application. The internet connected hardware will include a custom PCB with the following sensors and actuators: HTU21D-F Humidity/Temp Sensor (0x40), I2S 3W Class D Amplifier Breakout MAX98357, 0.56″ 4-Digit 7-Segment display. The database will store: Username, Password, Timestamp, Temperature Reading, Alarms Saved By Users. The mobile device functionality will include: Alarm Clock, Time Zones, Timers, Stopwatch, Temperature readings, User Information, Customization Features and will be further detailed in the mobile application proposal. I will be collaborating with the following company/department: Humber Prototype Lab. In the winter semester I plan to form a group with the following students, who are also building similar hardware this term and working on the mobile application with me. These are the following group members: Juan Rodriguez, Johnson Dinh, Jordan Pulido. The hardware will be completed in CENG 317 Hardware Production Techniques independently and the application will be completed in CENG 319 Software Project. These will be integrated together in the subsequent term in CENG 355 Computer Systems Project as a member of a 2 or 3 student group.

**Background**

The problem solved by this project is: As a youth, it becomes increasingly difficult to manage and maintain a proper sleeping schedule. The snooze button is used to give the user 5 more minutes to relax and properly wake up, however this is often abused and the user ends up repeatedly hitting the snooze button, which often leads to time wasting. A bit of background about this topic is: This project will consist of an alarm clock application which will link up to a physical hardware element via Wi-Fi. The hardware being developed would contain a display where the current time, alarm settings, and local temperature readings would appear. The app is where the user would be able to customize and select what they want to appear on the display. Each sensor in the hardware portion of the project serve their own purpose in conjunction with one another. The Humidity/Temp sensor would give the current readings of the temperature and store them in the database, the Haptic sensor will vibrate the device with an alarm goes off as a time of notification. The display screen will be responsible for displaying the core information requested by the user.

Existing products on the market include Google Home. I have searched for prior art via Humber’s IEEE subscription selecting Institute of Electrical and Electronics Engineers and have found and read A DIY approach to pervasive computing for the Internet of Things: a smart alarm clock which provides insight into similar efforts.

In the Computer Engineering Technology program, we have learned about the following topics from the respective relevant courses:

* Java Docs from CENG 212 Programming Techniques In Java,
* Construction of circuits from CENG 215 Digital And Interfacing Systems,
* Rapid application development and Gantt charts from CENG 216 Intro to Software Engineering,
* Micro computing from CENG 252 Embedded Systems,
* SQL from CENG 254 Database With Java,
* Web access of databases from CENG 256 Internet Scripting; and,
* Wireless protocols such as 802.11 from TECH152 Telecom Networks.

This knowledge and skill set will enable me to build the subsystems and integrate them together as my capstone project.

**Methodology**

This proposal is assigned in the first week of class and is due at the beginning of class in the second week of the fall semester. My coursework will focus on the first two of the 3 phases of this project:  
 Phase 1 Hardware build.  
 Phase 2 System integration.  
 Phase 3 Demonstration to future employers.

*Phase 1 Hardware build*

The hardware build will be completed in the fall term. It will fit within the CENG Project maximum dimensions of 12 13/16" x 6" x 2 7/8" (32.5cm x 15.25cm x 7.25cm) which represents the space below the tray in the parts kit. The highest AC voltage that will be used is 16Vrms from a wall adaptor from which +/- 15V or as high as 45 VDC can be obtained. Maximum power consumption will be 20 Watts.

*Phase 2 System integration*

The system integration will be completed in the fall term.

*Phase 3 Demonstration to future employers*

This project will showcase the knowledge and skills that I have learned to potential employers.

The brief description below provides rough effort and non-labour estimates respectively for each phase. A Gantt chart will be added by week 3 to provide more project schedule details and a more complete budget will be added by week 4. It is important to start tasks as soon as possible to be able to meet deadlines.

Display screen for hardware element required. Materials for creation of the device. Casing for sensors and Raspberry Pi. Additional connectors to link up sensors to one another.

**Concluding remarks**

This proposal presents a plan for providing an IoT solution. The hardware device is a convenient option for those that want to maintain a solid time schedule all while being able to view current temperature readings of the area around them without having to open up external applications. This is an opportunity to integrate the knowledge and skills developed in our program to create a collaborative IoT capstone project demonstrating my ability to learn how to support projects such as the initiative described by [3]. I request approval of this project.

**References**

[1]Google Home. (2016). Retrieved from https://store.google.com/ca/product/google\_home

[2]Institute of Electrical and Electronics Engineers. (2015, August 28). IEEE Xplore Digital Library [Online]. Available: https://ieeexplore.ieee.org/search/advsearch.jsp

[3]Scott, G. and Chin, J. (2013). A DIY approach to pervasive computing for the Internet of Things: A smart alarm clock - IEEE Conference Publication. [online] Ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org/document/6659445 [Accessed 15 Jan. 2019].

# 

# 

# 

# Abstract

The purpose of this project is to create a device which alarms the user more effectively for those wanting to be notified on time with ease. It will consist of an alarm clock application which will link up to a physical hardware element via Wi-Fi This technical report will give a thorough analysis of the development process regarding every aspect of the project, ranging from app structure to hardware assembly. Explanations revolving around certain ideals and decisions will be given to provide readers insight on why certain features are present along with their purpose.

Test cases will be examined during development to acknowledge mistakes that were made along the road along with the solutions for said mistakes. Any hardships and difficulties will also be documented for prevention purposes in the future of similar circumstances. Hardware explanation and breakdowns can be located in the Requirement Specification portion of the report.

# Table of Contents

[Declaration of Joint Authorship 1](#_Toc7100060)

[Proposal 2](#_Toc7100061)

[Abstract 7](#_Toc7100062)

[Table of Contents 9](#_Toc7100063)

[Illustrations 11](#_Toc7100064)

[Introduction 13](#_Toc7100065)

[Project Description 15](#_Toc7100066)

[Purpose of Project 15](#_Toc7100067)

[Problem 15](#_Toc7100068)

[Solution 15](#_Toc7100069)

[Requirement Specifications 16](#_Toc7100070)

[Hardware 16](#_Toc7100071)

[Software 16](#_Toc7100072)

[Database 17](#_Toc7100073)

[Build Instructions 19](#_Toc7100074)

[19](#_Toc7100075)

[Introduction 20](#_Toc7100076)

[Repository 20](#_Toc7100077)

[Budget for Materials Required 21](#_Toc7100078)

[UML Diagram 22](#_Toc7100079)

[Time Schedule 23](#_Toc7100080)

[Assembly of Pi 25](#_Toc7100081)

[Wiring 26](#_Toc7100082)

[PCB Design Files 28](#_Toc7100083)

[PCB Soldering 30](#_Toc7100084)

[Power Up 31](#_Toc7100085)

[Case Design 33](#_Toc7100086)

[Assembly for Hardware 34](#_Toc7100087)

[Installing CircuitPython 36](#_Toc7100088)

[Code for Sensors 38](#_Toc7100089)

[Create library object using our Bus I2C port 38](#_Toc7100090)

[Database Design 42](#_Toc7100091)

[Mobile Application 44](#_Toc7100092)

[Hardware Code 51](#_Toc7100093)

[**Alarm Function Code:** 51](#_Toc7100094)

[**Display Function Code:** 52](#_Toc7100095)

[**Temperature Function Code:** 54](#_Toc7100096)

[Status Reports 57](#_Toc7100097)

[Problems Encountered 65](#_Toc7100098)

[**Hardware Difficulties** 65](#_Toc7100099)

[**Database Issues** 65](#_Toc7100100)

[**Application Errors** 66](#_Toc7100101)

[**Case Design Problems** 66](#_Toc7100102)

[Solution 67](#_Toc7100103)

[**Hardware Fix** 67](#_Toc7100104)

[**Database Fix** 67](#_Toc7100105)

[**Application Fix** 67](#_Toc7100106)

[**Case Design Fix** 67](#_Toc7100107)

[Future Planning 69](#_Toc7100108)

[Reproduction of Product 70](#_Toc7100109)

[Conclusion 71](#_Toc7100110)

[Bibliography 73](#_Toc7100111)

# 

# Illustrations

[Figure 1 - HTU21D-F Temperature/Humidity Sensor(0x40) 19](file:///C:\Users\n01174532\Downloads\JnJ's%20Clockwork%20Report.docx#_Toc7100112)

[Figure 2 - 0.56″ 4-Digit 7-Segment display 19](file:///C:\Users\n01174532\Downloads\JnJ's%20Clockwork%20Report.docx#_Toc7100113)

[Figure 3 - I2S 3W Class D Amplifier Breakout MAX98357 19](file:///C:\Users\n01174532\Downloads\JnJ's%20Clockwork%20Report.docx#_Toc7100114)

[Figure 4 – Budget 22](#_Toc7100115)

[Figure 5 - UML Diagram 23](#_Toc7100116)

[Figure 6 - Schedule 24](#_Toc7100117)

[Figure 7 - Raspberry Pi 3 B+ Pinouts 28](#_Toc7100118)

[Figure 8 - Breadboard Layout 29](#_Toc7100119)

[Figure 9 - PCB Design 29](#_Toc7100120)

[Figure 10 - PCB Top View 30](#_Toc7100121)

[Figure 11 - PCB Bottom View 31](#_Toc7100122)

[Figure 12 - Interfacing Menu 32](#_Toc7100123)

[Figure 13 - I2C Option 32](#_Toc7100124)

[Figure 14 - I2C Addresses 33](#_Toc7100125)

[Figure 15- Case Design 33](#_Toc7100126)

[Figure 16 - Top View 34](#_Toc7100127)

[Figure 17 - Front View 35](#_Toc7100128)

[Figure 18 - Back View 35](#_Toc7100129)

[Figure 19 – Verification 36](#_Toc7100130)

[Figure 20 - Blinatest Output 38](#_Toc7100131)

[Figure 21 - Temperature Readings 39](#_Toc7100132)

[Figure 22 - Permission Screen 40](#_Toc7100133)

[Figure 23 - Alsamixer Volume Screen 41](#_Toc7100134)

[Figure 24 - Speaker Test Output Screen 42](#_Toc7100135)

[Figure 25 - Database Layout 43](#_Toc7100136)

# 

# Introduction

JnJ’s Clockwork is an android based alarm mobile application where users are able to set and customize alarms of their choice for daily use. The user will be able to create multiple alarm profiles along with being able to set a timer and utilize a stopwatch. When connected to its corresponding hardware component, the app furthers its capabilities by allowing users the ability to read local temperatures via the sensor included. The project is designed to give users ease of access to anything time related in a simple and clean form factor.

Students can benefit greatly from this product for those that struggle waking up at certain times of day and require an external source for assistance such as the app. It can also be used to set reminders for certain time periods such as deadlines for assignments etc. The temperature readings are an added bonus as it readily displays the current temperature in real time without having the need to open other respective applications for similar purposes.

In order for the app to function as intended, it is linked up with a database known as Firebase. This allows user information such as their own personal accounts, alarm templates and temperature values to be stored in a cloud which is ultimately located in Firebase. A Unique approach we are taking for this project would be the exclusion of a snooze button on the hardware itself which must be manually turned off in the app when alarms are triggered. This would make it harder for users to simply ignore the alarm they set up initially with a simple button press and motivate them to wake up.

## 

# Project Description

## Purpose of Project

### Problem

Our generation these days struggle to wake up from sleeping due to obvious reasons. More young adults and teens are oversleeping. With technology today, an easygoing and agreeable alarm is easily accessible within one's phone. Since alarms on your phone are helpful and very flexible to the user’s desires, younger audiences tend to rely heavily on their devices for the alarm. Most cases result in them having low battery on their phones which ultimately leads to it never going off. Either they forget to charge it or even set the alarm desired. Our goal is to combat this with a hybrid model which consists of an android application and physical alarm station.

### Solution

This alarm clock will not run out of battery, as it is connected to a power outlet. With the given settings this device provides, it has harsher features to force the user to manually open the app and disable it, rather than easily pressing a button on their phone to cancel the alarm sound. The user will have to manually enter the app, login in with their proper credentials, navigate to the hardware page and click cancel. The sound does not get stopped immediately, but rather in a 5 second countdown. The ideology behind this method is that by the time the sound stops, the user will be fully awake and not just simply turn it off and go back to sleep. JnJs Clockwork can help stabilize the user's sleeping schedule by stimulating a stronger alarm experience. Due to everything being handled under Wi-Fi connectivity, the user can set an alarm anytime on the app and it will be communicated instantly with the hardware.

## Requirement Specifications

### Hardware

The hardware portion of this project will be a joint effort between each member of the group as there are many responsibilities in order for everything to function as intended. In terms of the hardware design, enclosure, PCB, and Soldering, it will be handled by Juan. The functionality of each sensor will be tested and operational mainly by Jordan with help from Johnson when required. Connections between sensors and the Raspberry Pi will be accomplished by Johnson. The Integration of components may require additional help from every member due to problems that may occur during development.

The project utilizes many hardware components such as the Raspberry Pi 3B+, HTU21D-F Humidity/Temp Sensor (0x40), Audio amplifier with 3 Ohm Speaker, and a 4-digit-7 segment display screen. These sensors will be enclosed using 3D printed materials with a maximum dimension size of: 12 13/16” x 6” x 2 7/8”. The Humidity and display sensor are ready for integration with one another as they were already completed last semester. The new inclusion for the project is the Display Screen as it is crucial to display the time and temperature from the application. The android smartphone’s role will act as the device’s remote as it can communicate with the clockwork through Wi-Fi. An 8GB micro SD card will be used as storage as it can store the installation of the Raspbian OS and reading and writing values from the Clockwork. The PCB (printed circuit board) acts as the structure and support of the system for sensor connections.

### Software

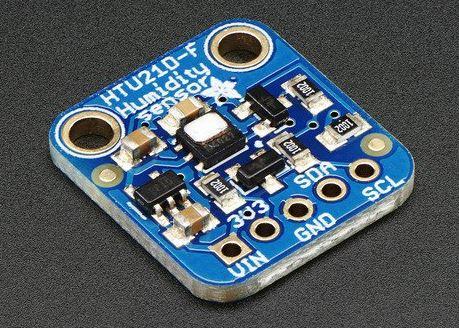
The android application will be developed and maintained by Juan and Johnson. Add ons and additional functionality will be incorporated with the help from Jordan. The app is mostly complete in its current state. The only things that are left to work on is Wi-Fi functionality and debugging. The app needs to respond to the hardware in order to display desired information from the application.

The project utilizes a smartphone capable of running Android API 21 or higher. An up to date version of Android Studio was used to build the mobile application. A Raspberry Pi 3 was implemented with connection between the hardware and application. Updating the Raspbian OS to its newest version was used throughout the project. The mobile application will be used to work alongside the hardware components. Firebase real-time readings is going to be used for communication such as storing user and temperature readings.

### Database

The database will be designed, created and upheld by Juan and Johnson. The database connection is established and connected to the mobile application. Reading and writing from the sensor to the database are also required. The database utilizes user-authentication to allow maximum security and protection for the user’s information. In order to read and write temperature, the user must be registered using a username and password through authentication processing. Offline mode allows access to the app, without the need to register and login. Offline mode skips user-authentication, and moves the user to the actual app. In offline mode, there will be no form of communications to the database. Therefore, the user is unable to read/write temperature to the database.

# Build Instructions



The HTU21D-F Temperature/Humidity sensor uses the ratio of air moisture to be able to measure and read both temperature and humidity. In this tutorial, the goal is to able to activate the senor and have it display readings of the temperature and humidity. It has a typical accuracy of ±2% with an operating range that's optimized from 5% to 95% RH.

Figure 1 - HTU21D-F Temperature/Humidity Sensor(0x40)

This sensor is a 4 digit - 7 segment numeric display

Figure 2 - 0.56″ 4-Digit 7-Segment display

which utilizes many LEDS built into the display. The screen is accompanied by a backpack which does all the heavy lifting. Includes a built-in clock to multiplex the display and current drivers for bright/vivid colors for the LEDS.

### C:\Users\n01164181\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Amplifier.JPG

The I2S 3W Class D Amplifier Breakout MAX98357 is a sensor which amplifies audio for digital sounds files. It takes standard I2S digital audio input and decodes it into analog, which also amplifies it directly into a speaker. The small mono amplifier is capable to deliver 3.2 Watts of power into a 4-ohm impedance speaker. Inside the chip is a class D controller, able to run from 2.7V-5.5VDC making it extremely efficient.

Figure 3 - I2S 3W Class D Amplifier Breakout MAX98357

### Introduction

JnJ’s Clockwork is an android based alarm mobile application where users are able to set and customize alarms of their choice for daily use. The user will be able to create alarm profiles along with being able to set timers and stopwatches. When connected to its corresponding hardware component, the app furthers its capabilities by allowing users the ability to read local temperatures via the sensor included. The project is designed to give users ease of access to anything time related in a simple and clean form factor.   
This project consists of the HTU21D-F Temperature/Humidity Sensor for reading local temperature values, a 1.2″ 4-Digit 7-Segment display for displaying current time and temperature readings, and a I2S 3W Class D Amplifier Breakout MAX98357 for audio when an alarm goes off.

### Repository

The repositories consist of all the files that were made throughout the production of JnJs Clockwork. One repository strictly consists of android code to replicate the JnJs Clockwork mobile application. Inside the documentation repository are folders that categorize how each of the work was handled. From the previous semester CENG319, a schedule was predetermined on how the work was going to be laid out. This work was accomplished by project and excel 2016. The schedule, report, proposal, and budget were being updated as we came upon obstacles that interfered within the plans. In the repository are the 5 status reports that were to be reported during this semester for the Capstone Project. The updates were rotated by students A, B, and C. These reports were to verify if the work is being done properly and if the project is on task as required. There are 3 folders in the repository that consists of 3 parts, the alarm clock case, the PCB, and pictures. The alarm clock case folder handles CorelDRAW files to create the alarm clock case. The creation of the case was handled through the CorelDRAW program. The PCB folder consists of fritzing files for the temperature sensor, segment display, and amplifier. With the Fritzing program, the files were merged to into one zip file containing Gerber files to create the PCB. The pictures folder consists of all the progress made throughout the stage of the JnJs Clockwork project. From the skeleton of the hardware, sensors, raspberry pi, and PCB are all contained within the folder. Additionally, there are screenshots of code that sets up the raspberry pi, amplifier, temperature sensor, and 7 segment display. For accurate measures, there are also pictures of code and outputs to guide audiences of the project. Different views of the hardware and PCB were taken to display accurate visuals on how the pin sockets were soldered. Inside the hardware code folder represents the final python code of the temperature sensor, 7 segment display, and amplifier to run as the project intended. All the code of this project is available to download directly as this project is completely open source for anyone to use.

### Budget for Materials Required

The required materials and budget for this project can be found in the documentation folder in the repository.

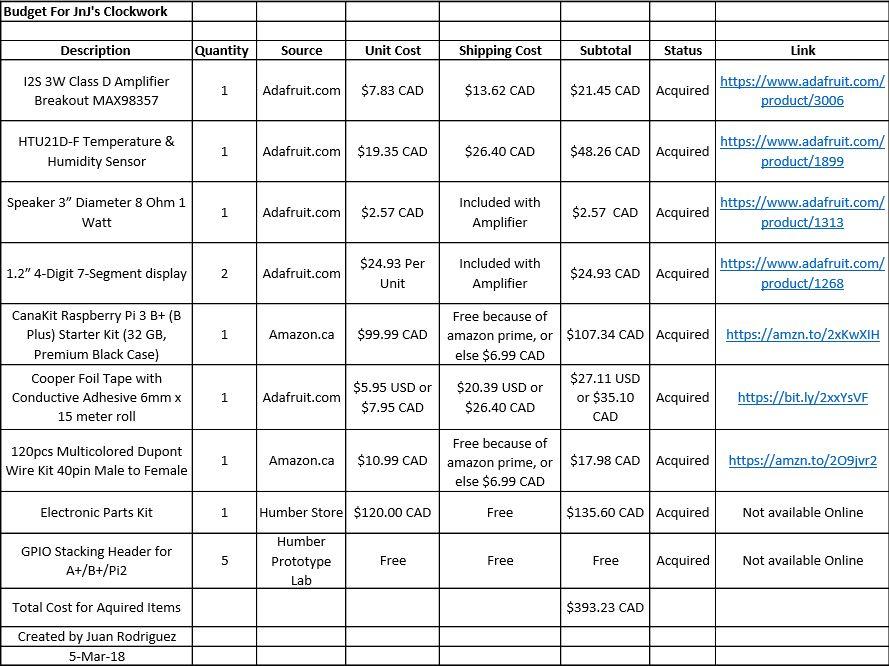


Figure 4 – Budget

### UML Diagram

With this UML diagram, it shows all established connections and how they interact with one other. It starts at registration which requires email and password. Once its set, it is this then sent to Firebase Authentication, where it will be stored. Firebase Authentication is its own database as it only stores emails, password, last signed in, account created, and it will set a user ID. With that the login is now available for use. When a username and password is inserted, the authentication is rechecked. If it rechecks and it doesn’t match with the authentication, the login will fail. When everything matches, the login will be successful. The user will now have been able to read and write to the database. There are 4 activities that all relate and correspond with the database. The temperature sensor provides the database with its readings, it will always update and write it to the database every 10 seconds. Within the android app, temperature display will then take those readings from the database and show it onto the android device. On the same activity the temperature display can also write back to the database with its saved temperature followed by a timestamp. On the android app, SetAlarm will send its values of hour, minute, cancel, and volume control to the database. The database will then communicate with AlarmFunction on its values and work appropriately.

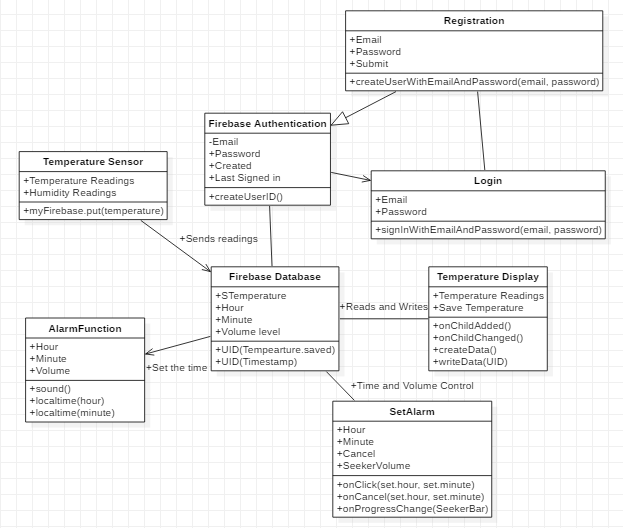
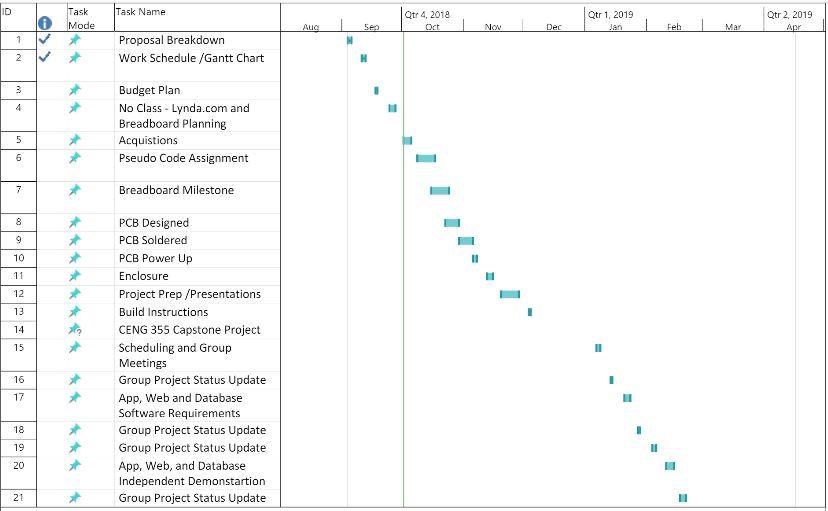


Figure - UML Diagram

### Time Schedule

Realistically, this project should take around 3-4 weeks to complete if all materials and facilities are available to you. The materials themselves might take a week to arrive due to shipping, but the actual process of assembling and programming should not take longer than a week if proper time is given. A couple of hours each day can be dedicated towards the different aspects of the project to make time usage more efficient and effective. For us, this project took around 2 whole semesters (8 months) to finish along with an average work time of around 2.5 hours a week. Here is the time schedule we followed:



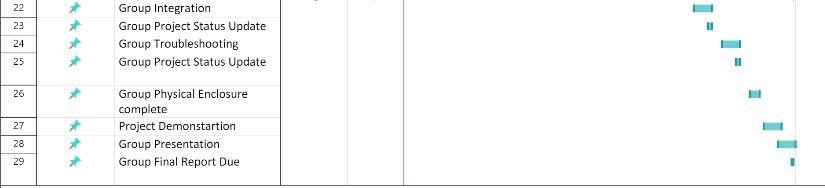


Figure 6 - Schedule

### Assembly of Pi

These steps will cover how to set up the Raspberry Pi 3 B+ properly so that you have the ability to log in and test your sensors capabilities.

1. Format an SD card with a minimum of 8GB to be used for the OS of the Pi. You can use the following link to download a SD card formatting software: https://www.sdcard.org/downloads/formatter\_4/index.html
2. Download and unzip the latest version of the OS for the Raspberry Pi to your SD card. Download NOOBS in the link as that will ensure that you will have almost everything required when starting: https://www.raspberrypi.org/downloads/noobs/
3. Once the image is on the SD card, remove it from the pc and insert it in the Pi. Now plug in a separate monitor, mouse, keyboard, HDMI, Ethernet cable, and power supply to the Pi in their corresponding ports. The Pi turns on automatically when the power is plugged in.
4. Upon the boot up session, select Raspbian as the operating system for the Pi and follow the instructions as they appear. You may also change the keyboard layout on the bottom during initial boot. The US layout is highly recommended.
5. Once installation is completed, you should be brought to the desktop. Connect yourself to either Wifi or wired connection in order to perform the next few steps.
6. Open the terminal in the top left corner of the screen and input the following lines (this takes quite a long time): Shell wget https://raw.githubusercontent.com/six0four/StudentSenseHat/master/firmware/hshcribv01.sh \ -O /home/pi/hshcribv01.sh chmod u+x /home/pi/hshcribv01.sh /home/pi/hshcribv01.sh
7. Now it is time to set up a VNC connection so that you can access your Pi on any computer screen. From the Start Menu, go -> Preferences->Raspberry Pi Configuration->Interfaces, then set VNC to Enabled. Now on the desktop in the top right corner, you should see a VNC logo. When you click it you should see an IP address for your Pi which will be used to connect it via the VNC software. Download the software on any computer you wish to communicate with the Pi: https://www.realvnc.com/en/connect/download/vnc/
8. Once the software is installed, connect the Ethernet cable from the Pi to your computer of choice to have a direct connection. Now you can simply input the same address you found in the Pi in the VNC software and it should connect.
9. To turn off the Pi, type sudo power down in the terminal.

If you are still unsure or struggling with a part in particular, this video provides a step by step explanation for everything required: <https://www.youtube.com/watch?v=xBlxuf_LSCM>

### Wiring

Before wiring each sensor to the breadboard, it is important to solder the pins that come included to the sensors corresponding pin layouts. Additionally, make sure to cut the excess pins that come included that will not be required for each sensor.   
   
When soldering, make sure you have safety glasses equipped along with having proper ventilation that contains an extractor arm for the fumes. A soldering toolkit is also required which is available in most labs. Here is a great soldering tutorial to help with those unsure.  
 https://www.youtube.com/watch?v=3230nCz3XQA

You can wire the sensors to the Raspberry Pi using the following charts:

**HTU21D-F Temperature/Humidity Sensor (0x40).**

|  |  |
| --- | --- |
| Device Pin | Pi |
| 1 (VIN) | [5.0v] |
| 2 (3.3v) | [3.3v] |
| 3 (GND) | [GND] |
| 4 (SDA) | [GPIO 2] |
| 5 (SCI) | [GPIO 3] |

**0.56″ 4-Digit 7-Segment display.**

The IO pin only needs to be connected if you are working with the 1.2" display which has the same pinouts as the 0.56" one.

|  |  |
| --- | --- |
| Device Pin | Pi |
| 1 (D) | [SDA] |
| 2 (C) | [SCL] |
| 3 (+) | [5.0v] |
| 4 (-) | [GND] |
| 5 (IO) | [3.3v] |

**I2S 3W Class D Amplifier Breakout MAX98357.**

|  |  |
| --- | --- |
| Device Pin | Pi |
| 1 (VIN) | [5.0v] |
| 2 (GND) | [GND] |
| 3 (DIN) | [GPIO21] |
| 4 (BCLK) | [GPIO18] |
| 5 (LRCLK) | [GPIO19] |

This is the layout for the pins of the Raspberry pi for guidance on where certain pins are located:

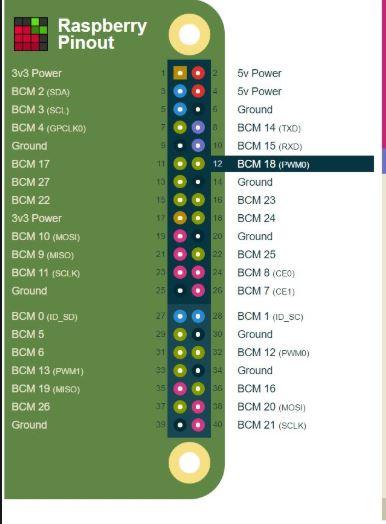


Figure 7 - Raspberry Pi 3 B+ Pinouts

### PCB Design Files

In order to develop the PCB design files, the application Fritzing is required along with the Corresponding sensors file which must be added to the application under MyParts. The files can be located here: https://github.com/adafruit/Fritzing-Library/tree/master/parts   
 Once the Sensor is added to parts, you can create a fritzing diagram for the wiring of the pi and sensor. It should look similar to this.

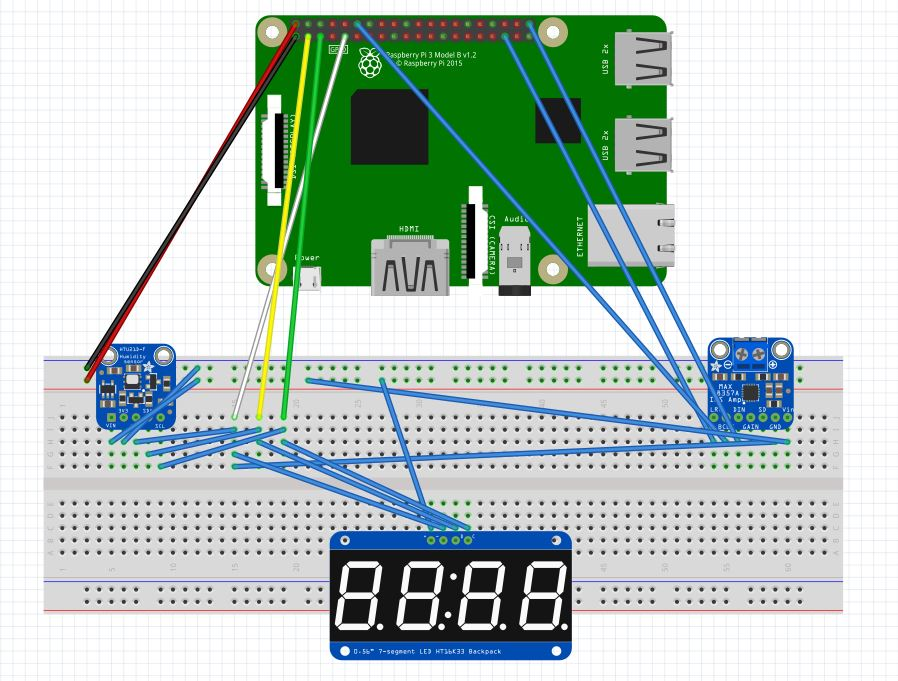


Figure 8 - Breadboard Layout

From here you can create the PCB design from the wiring you just designed. The PCB layout should look similar to this. Reminder, the IO pin is only connected to 3.3v for the 1.2” screen.

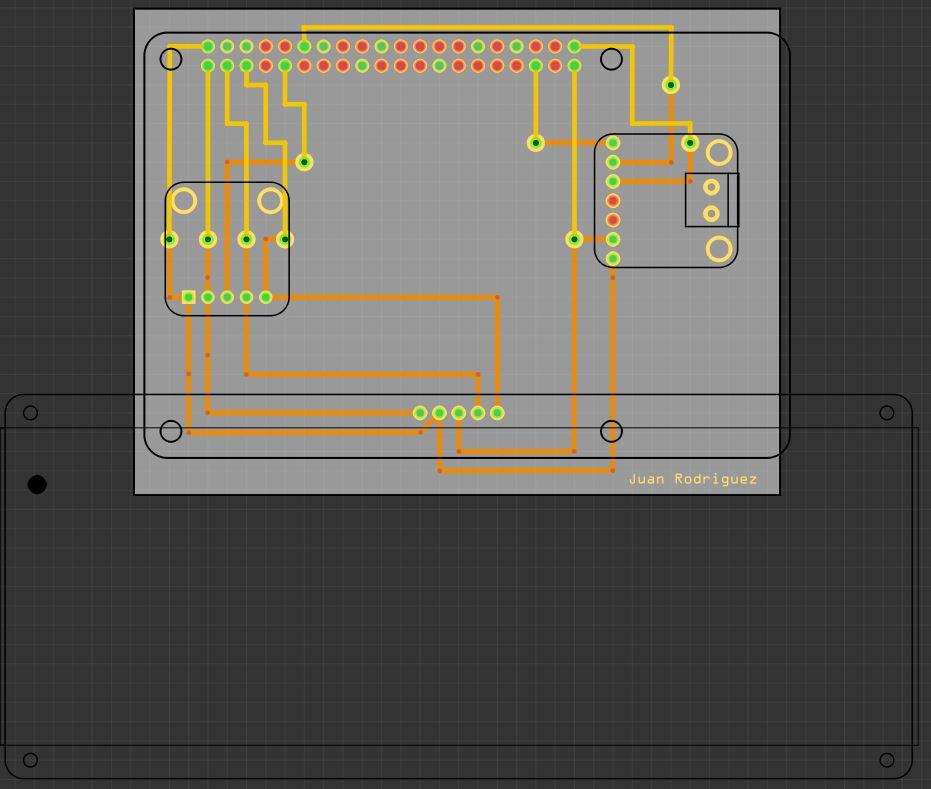


Figure 9 - PCB Design

With these now ready, you can put together your Gerber files and create your PCB using a laser cutter machine. The Gerber files are located in the repository: <https://github.com/JuanRodriguez19/JnJ-s-Clockwork>

### PCB Soldering

Using the same rules as when soldering the Sensors, solder pieces of wire in between the vias on the PCB board. Once that's done, solder the 20-pin socket to the PCB board to the corresponding holes for where the pi would connect. For the remaining pin sockets, you have, solder in the respective headers for each sensor in the appropriate locations. Your final board should look similar to this.

Top view:

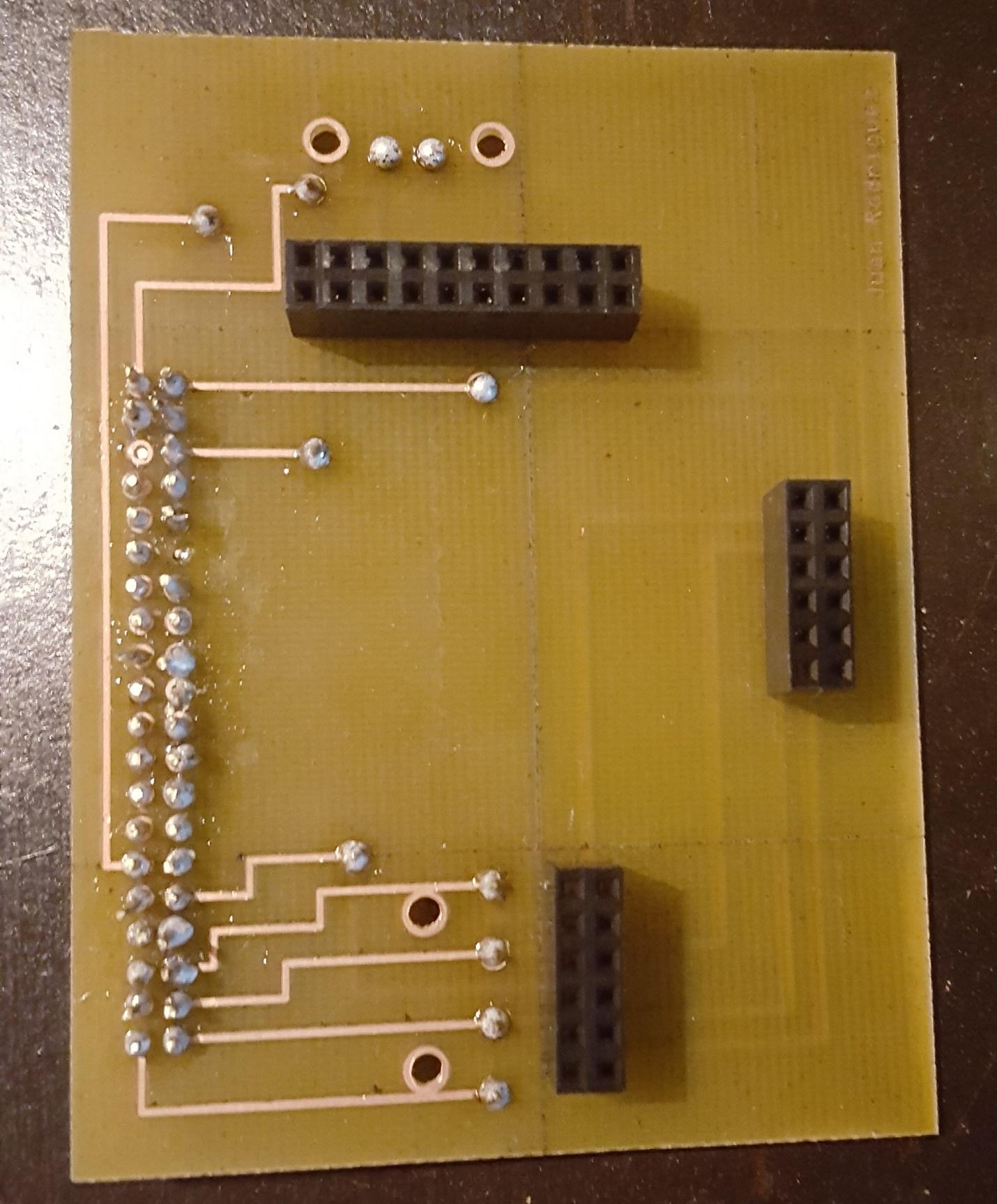


Figure 10 - PCB Top View

Bottom view:

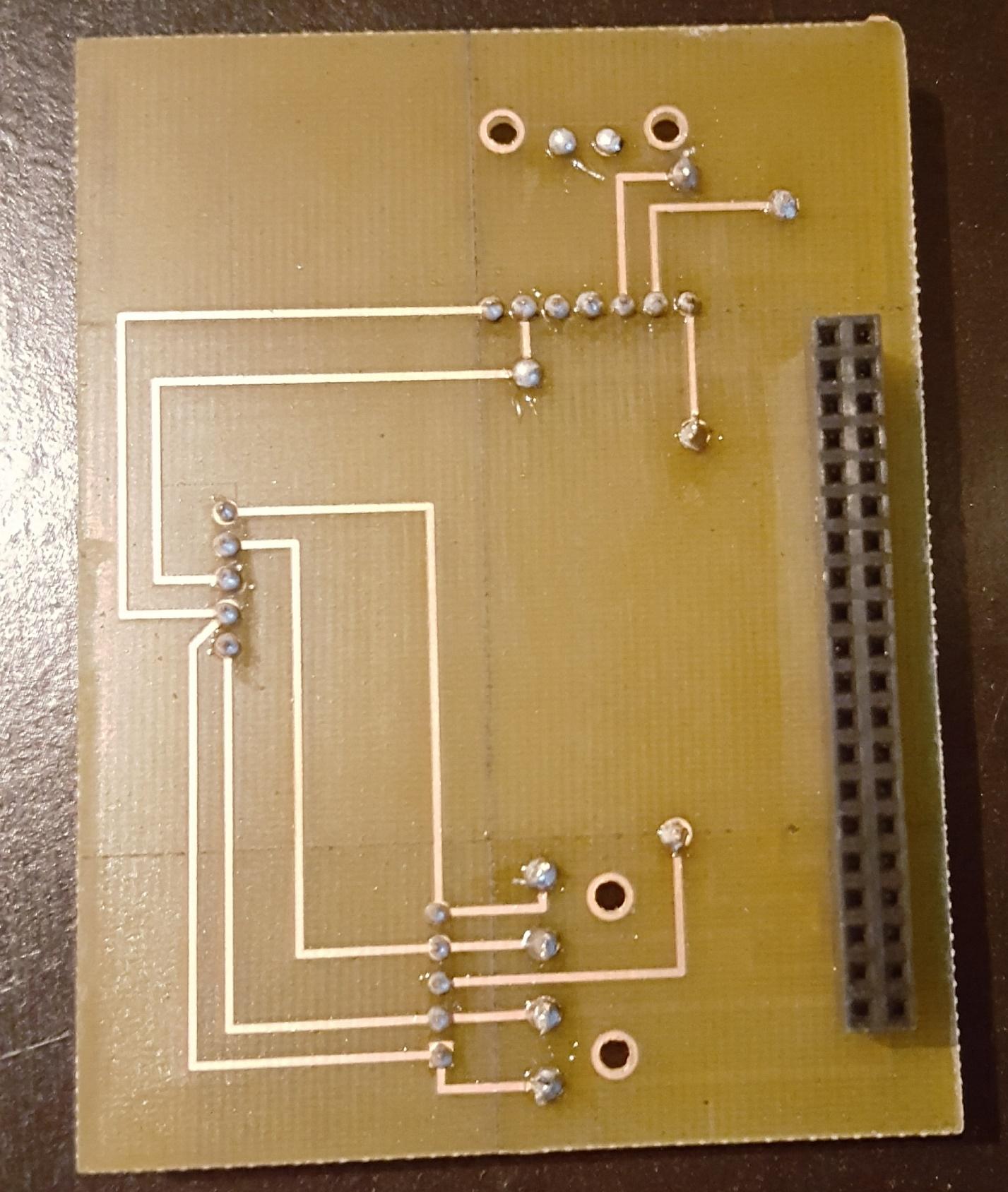


Figure 11 - PCB Bottom View

### Power Up

In this section, we will now see if everything works, this works on whether you have soldered your PCB or you normally wired it onto your circuit board. Once connected boot up the Raspberry Pi, open the terminal window and follow these steps:

1. This will bring you into the configuration tool

sudo rasp-config

1. Use your arrows keys to go down and select "Interfacing Options"

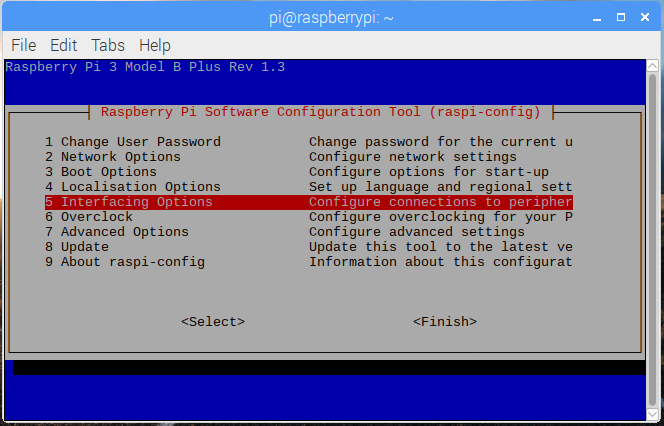
* 

Figure 12 - Interfacing Menu

1. Select I2C and submit yes. It should display ARM I2C is enabled.

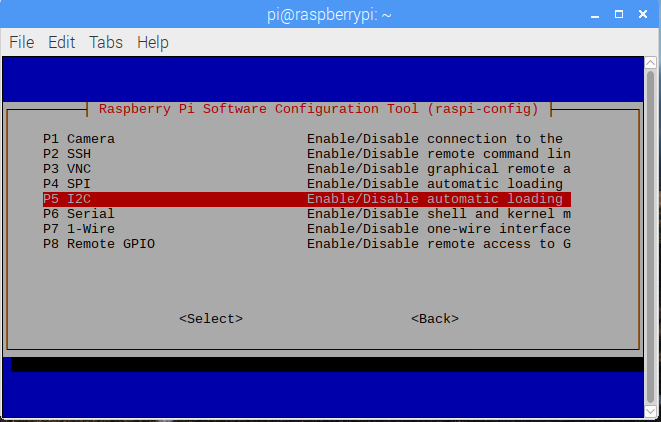


Figure 13 - I2C Option

1. Exit by selecting the finish option. By using the command below, it should your address for the sensors being (0x40) and (0x70).

sudo i2cdetect -y 1

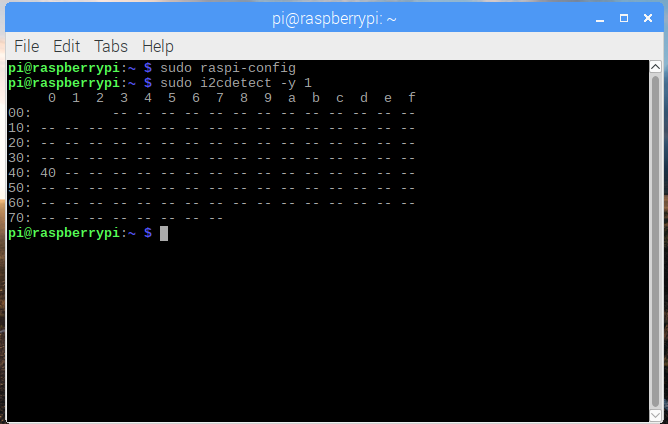


Figure 14 - I2C Addresses

### Case Design

The case was created using CorelDRAW software. The file can be located in the repository linked above. Each case side was carefully measured to ensure everything fit in their corresponding locations. It should look like this:

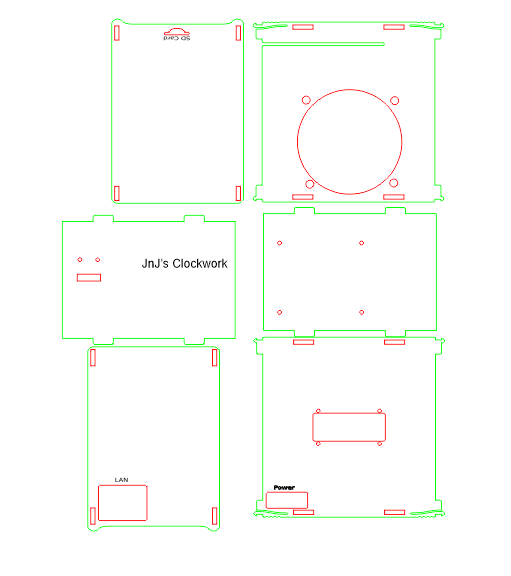


Figure 15- Case Design

For the case, we decided to mount the display screen, temperature sensor and speaker in order to give our project more appeal in terms of its design rather than simply being a box that contains 3 sensors within. The case overall is around 5.2 Inches tall giving it rather large appearance as it was necessary to fit the speaker on the back of the device. The temperature sensor was mounted on the top of the case so that it can receive more accurate readings and the display screen is in the front of the case. In terms of port accessibility, only the LAN and Power ports are open as the others are not necessary for the project to function. Their inclusion would only result in more open sockets that would allow dust and debris to enter which is not desirable.

### Assembly for Hardware

The finished product should look like this when assembled:

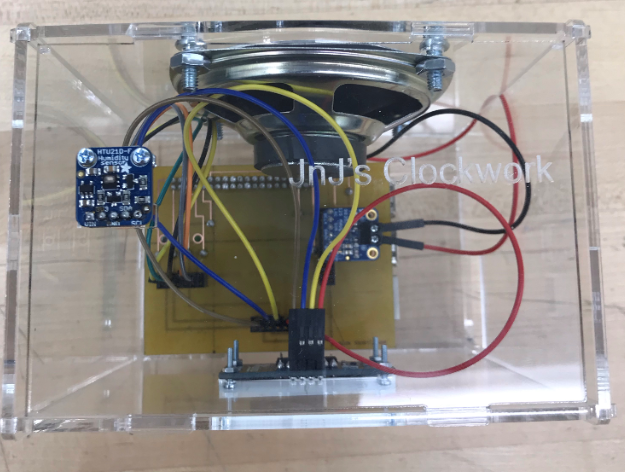


Figure 16 - Top View

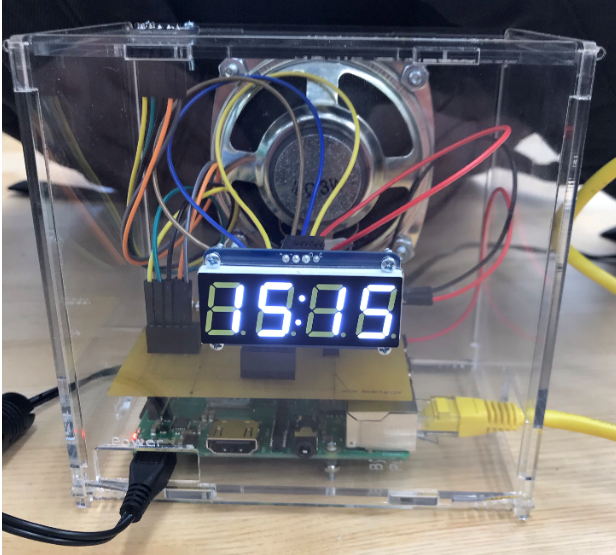


Figure 17 - Front View

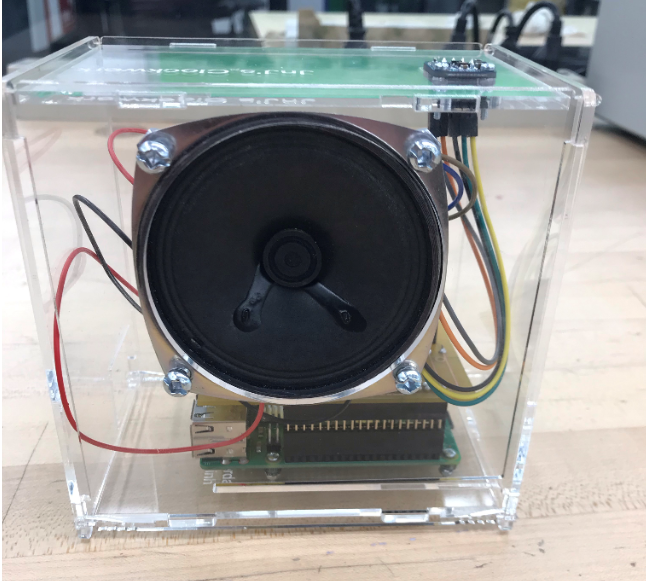


Figure 18 - Back View

### Installing CircuitPython

For this project alot of code will be used with Circuit Python so here is the installation process for it.

The following tests were experimented using Adafruit:

1. Run the update commands for the Raspberry Pi.  
 sudo apt-get upgrade

2. When done installing, run the command line for the python tools

sudo pip3 install --upgrade setuptools

3. Verify you have I2C Enabled

ls /dev/i2c\*



Figure 19 – Verification

4. Begin to install the Python Libraries

pip3 install RPI.GPIO

5. Use the following command to install adafruit-blinka:

pip3 install adafruit-blinka

To test if Python works, open python in the Raspberry Pi (it should be installed at this point), and write an example file to sample output.

import board  
import digitalio  
import busio  
   
print("Hello blinka!")  
   
# Try to great a Digital input  
pin = digitalio.DigitalInOut(board.D4)  
print("Digital IO ok!")  
   
# Try to create an I2C device  
i2c = busio.I2C(board.SCL, board.SDA)  
print("I2C ok!")  
   
# Try to create an SPI device  
spi = busio.SPI(board.SCLK, board.MOSI, board.MISO)  
print("SPI ok!")  
   
print("done!")

Save it, then run it on the command line by typing

python3 blinkatest.py

The following should be seen

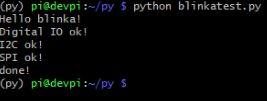


Figure 20 - Blinatest Output

### Code for Sensors

**HTU21D-F Temperature/Humidity Sensor:**

With the code provided in this repository, this test code should get your sensor to read and write temperature/humidity.

Firstly, though assuming you installed circuit python, all that is required is to run the command line to install HTU21D libraries.

sudo pip3 install adafruit-circuitpython-htu21d

You can copy this code and run it.

```import time import board import busio from adafruit\_htu21d import HTU21D

### Create library object using our Bus I2C port

i2c = busio.I2C(board.SCL, board.SDA) sensor = HTU21D(i2c)

while True: print(": %0.1f C" % sensor.temperature) print("Humidity: %0.1f %%" % sensor.relative\_humidity) time.sleep(2)

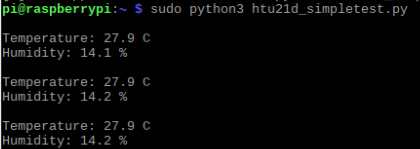
Your output should look like this.  


Figure 21 - Temperature Readings

**0.56″ 4-Digit 7-Segment display:**  
  
For this sensor we want to be able to display the current time. With that python and its libraries need to be installed:

sudo apt-get update sudo apt-get install build-essential python-dev

You would also need python smbus and python-imaging library:

sudo apt-get install python-smbus python-imaging

Clone the url onto your pi and move into it:

git clone https://github.com/adafruit/Adafruit\_Python\_LED\_Backpack cd Adafruit\_Python\_LED\_Backpack

This is the last library you need to install:

sudo python setup.py install

Now go into your file named examples:

cd examples

There are alot of test codes we can use here, but in case we just need our sensor to display the time:

sudo python ex\_7segment\_clock.py

**I2S 3W Class D Amplifier Breakout MAX98357:**  
  
For testing the sensor, the following line was used with the help of internet connectivity within the Raspberry Pi.

curl -sS https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/i2samp.sh | bash

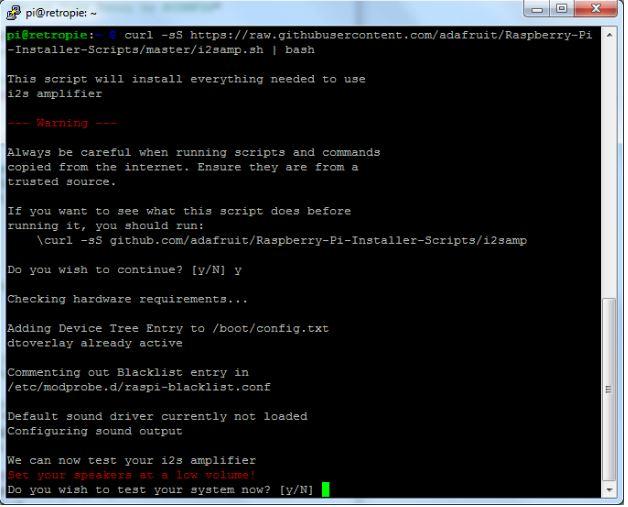
Select yes for the following questions the file asks by typing "y".  


Figure 22 - Permission Screen

The file will ask to reboot. Type "y" again to reboot the Raspberry Pi. When the device is rebooted, type in the same command

curl -sS https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/i2samp.sh | bash

The script should recognize the device at this point and noise should be coming out of the speaker. To adjust the volume, Alsamixer was used within the terminal. The recommended volume is 50 within Alsamixer.

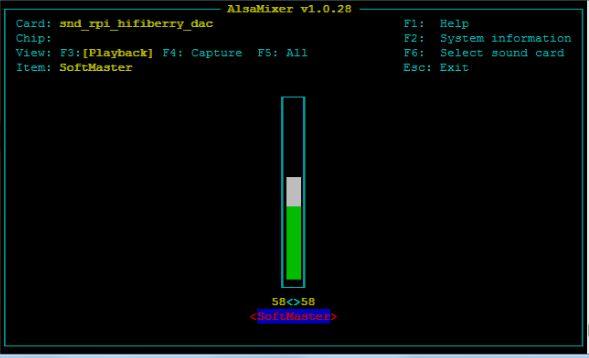


Figure 23 - Alsamixer Volume Screen

Press "ESC" to exit Alsamixer. To actually save the volume settings THE PI MUST BE REBOOTED TWICE, type in the following to reboot the system.

sudo reboot

Start up the Raspberry Pi, type in the following to generate a white noise coming out of the speaker

speaker-test -c2

If real sound wants to be heard, here is a demo

sudo apt-get install -y mpg123 mpg123 http://ice1.somafm.com/u80s-128-mp3 ```

Open another terminal window to access Alsamixer to adjust volume of the speaker.

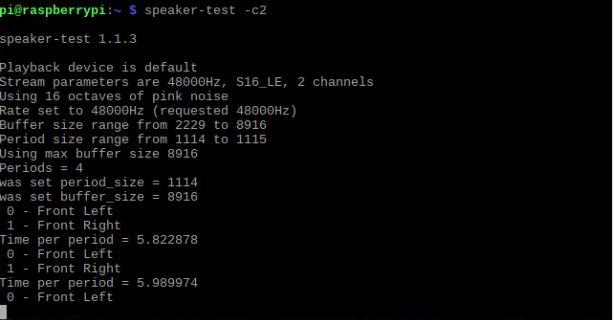


Figure 24 - Speaker Test Output Screen

If errors occur, refer to Adafruit's manual setup:

https://learn.adafruit.com/adafruit-max98357-i2s-class-d-mono-amp/raspberry-pi-usage

### Database Design

The database connection is established and connected to the mobile application. Reading and writing from the sensor to the database are also required. The database utilizes user-authentication to allow maximum security and protection for the user’s information. In order to read and write temperature, the user must be registered using a username and password through authentication processing. Offline mode allows access to the app, without the need to register and login. Offline mode skips user-authentication, and moves the user to the actual app. In offline mode, there will be no form of communications to the database. Therefore, the user is unable to read/write temperature to the database. The database connection works in conjunction with firebase.

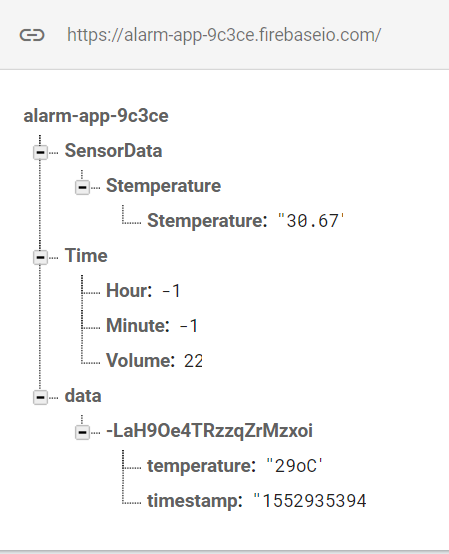


Figure - Database Layout

The data in database is broken into 3 data structures. The first structure (SensorData) gets the temperature readings from the temperature sensor. The second data structure (time) is the volume and alarm control where the user can input the Hour and minute for the Alarm setting. The third data structure (data) is the users Id that is currently logged in with their saved temperature reading timestamp.

### Mobile Application

The mobile application upon startup will provide the login screen. Through user authentication, the application will transfer to the main menu. We designed the user interface to be clean and simple so that it would feel inviting for the user rather than being complex and intimidating. Having the 3 main features of the app (Clock, Alarm, Timer) on the homepage via tab fragments would make it quick and easy for the user to traverse without having to flip through various menus and options. The data that the app uses such as different time zones, alarms, and stopwatch count would be provided by user input so that they could choose whatever they desire. The mobile app also includes a section where it connects to the firebase where it has the temperature readings. In terms of hardware connectivity, the user will be able to set an alarm from the app and that information will then be sent to the database. The alarm data will then be sent to the hardware component, which will ultimately trigger the hardware to go off.

**Test Cases**

Authentication:

This is important as it gives the user access to the database. The first thing they will see is if they can login, for new users they would have to register. If the user attempts to login without registering, a message will appear noting that the account is not registered. The register page consists of an email and password. When the user inputs an incorrect email address such as @email.com, it would not work due to the authentication recognizing that this is not a proper email address. Therefore, the user would have to use either Gmail, Hotmail, Yahoo, etc. Once everything is met, the account is now registered and saved onto the firebase authentication. They can now log in, if any mistakes are made a warning will appear prompting the user that there is a mistake. A successful login message will appear, if they meet all the necessary credentials and they will move on the to the clock screen.

Firebase Read/Write:

In order to read and write to the database, the user first has to be logged in. If the user is in offline mode, they will not be able to read or write to the database. The database is structured so that it will always use the UID as the primary key and then the temperature and timestamp under it. So, if a new user saves a temperature, a new structure of their UID, temperature readings, and timestamp will be displayed on the database. This in turn can always have accurate reads of timestamp and temperature for different selected users. When the user creates an alarm profile, it will be saved to the database under their user id along with their temperature values. This then allows the amplifier to receive the database data and act accordingly, in this case having audio play with the alarm condition is met.

Clock Screen Test:

This fragment is shown right after someone logs in, or go in offline mode. You can access this page anytime by clicking the Tab that has a label “Clock” on it. To test the app there are textViews displaying the current time, and the time of the selected time zone. Select a time zone by pressing the spinner, it will show a list of all the available time zones in standard time. Once you pick a time zone, the textView for the selected time should change to the appropriate time. Changing to landscape mode changes the layout design to fit appropriately.

Alarm Screen Test:

You can get to this Fragment by clicking the second tab labeled “Alarm”. At first it should show the text “No Alarm Set” clicking the set alarm button will open up a TimePickerDialog that, on default, should be on the exact time you click the button, you can set the time by picking the hour then selecting the minutes. Or you can change to a different mode to change the alarm manually. After selecting the time, the text view should show “alarm set for \_\_\_” and will send a notification once it reaches the time selected. To cancel the alarm, click the cancel button and the text should change to “alarm canceled”. Changing to landscape mode will retain an appropriate layout design. There is a button at the bottom of the page called hardware which sets an alarm that will communicate with the database and the hardware component, not solely the mobile phone. This will make so the audio that hits when the alarm goes off comes from the amplifier and speaker from the raspberry pi, not the phone.

Stopwatch Screen Test:

The stopwatch can be accessed by clicking the third tab Fragment labeled Stopwatch. In this screen it displays a chronometer for minutes and seconds. If you press the start button it will start to count up. If you press the pause button it will pause the stopwatch at the time, pressing start again will resume the stopwatch. Reset button will reset the time displayed back to 0. landscape has an appropriate design layout.

**Android Components**

Many android components and libraries are used during development of the app, each with their own specific purpose.

Major methods used for general overall use:

* **Intent (this, MainActivity.class)** - this method is often called when switching between screens (Often used with buttons or TextView links)
* **findViewById (int id)** - this method is used frequently to locate and interact with views found from layout resource files that are attached to the current activity
* **toastmakeTest(applicationContext, text, duration) -** this method is used often to send feedback to the user when they do a certain task like logging in.
* **setOnClickListener(new View.OnClickListener)** - When a button is placed this allows the button to be set. In doing so it becomes it creates a function called onClick(View v). As a result, the system allows all executeable code that is under onClick(View) once the button is pressed.
* **onCreate()** - When an activity is first created it comes with onCreate. Everything containing the activity is under the onCreate method. The onCreate method allows things such as create views, bind data to lists, strings, etc.
* **onOptionsItemSelected(MenuItem item)** - An item is hook each time a item in the option menu is selected it will run the item's selected output. MenuItem is the item that was selected and will execute, the item can never be NULL.

Main Methods for all Firebase Activities:

* **ChildEventListener ()** - Listens for any activities to the children.
* **ValueEventListener ()** - Looks for data changes in specific location of thedatabase.
* **onChildAdded ()** - This method allows the android device to pull data from the data structure and read from the database. In turn when a new child is added to the location to which it was added. For example, the temperature readings from Firebase.
* **onChildedChanged ()** - This method will update the data when a child location has changed. Whenever there is a new temperature reading.
* **onChildRemoved ()** - An event when the child is removed. Whenever content is being removing from the database.
* **onChildMoved ()** - This method is used when the location of the child changes. Used to sort data on the database.
* **onDataChange ()** - Similar to onChildedChanged, but will reads the static snapshot of the information at the given path at the timed event.
* **writeData()** - Will write the database with whatever the user is saving. This also includes the current user ID.

Main methods for Fragments:

* **onCreateView()** - similar to onCreate, but allows the view of other associated fragments.

Main methods for user authentication:

* **signInWithEmailAndPassword -** this method is called when the user gains access to their respective account once the proper email and password are entered
* **getCurrentUser() -** this method retrieves the data from the current user logged into the server, thereby accumulating further input they enter and save to the database

Main methods of Clock Fragment:

* **SimpleDateFormat** - allows you to set and use your own custom date format. Allows you to set the timezone format.
* **Calendar** - It's an abstract class with methods that allow you to set calendar fields such as Year. Month, Day of month, Hour.
* **TimeZone** - represents a timezone offset, you can use getavailableIds to set the spinner widget to display all the time zones.

Main methods of Alarm Fragment:

* **TimePicker** - A widget that allows you to pick a date and time 24h AM/PM, using TimerPickerDialog that displays a dialog to the user to set the time.
* **AlarmManager** - This allows you to schedule your application to be run at some point in the future, using intents to broadcast it. It retains the application when the device is asleep.
* **NotificationManager -** Class to notify the user of events that happen. This is how you tell the user that an alarm has been triggered in the background. You can set vibration, lights, sound, descriptions and the icon. You can also put default intensity settings for vibration or sounds.
* **onCreateDialog** - Allows custom dialog container. This mainly used to display a custom AlertDialog instead of the default dialog. No implementation required.
* **onRecieve** - Used when receiving an intent broadcast.

Main methods of Stopwatch Fragment:

* **Chronometer** - A class that implements a timer. You can set when the timer should start counting and also change it to a countdown timer.
* **SystemClock** - has access to all timekeeping facilities. The use of this method was to set the chronometer to 0 when you start it, so it started on the system clock time.

# Hardware Code

## **Alarm Function Code:**

|  |
| --- |
| from firebase import firebase |
|  | from pygame import mixer |
|  | import time |
|  | import alsaaudio |
|  |  |
|  |  |
|  | firebase = firebase.FirebaseApplication('https://alarm-app-9c3ce.firebaseio.com/') |
|  |  |
|  | def sound(): |
|  | mixer.init() |
|  | mixer.music.load('/home/pi/Downloads/old-school-bell.mp3') |
|  | mixer.music.play() |
|  |  |
|  |  |
|  |  |
|  | while True: |
|  | result = int (firebase.get('/Time/Hour',None)) |
|  | result2 = int (firebase.get('/Time/Minute',None)) |
|  | result3 = int (firebase.get('/Time/Volume',None)) |
|  |  |
|  | m = alsaaudio.Mixer('PCM') |
|  | vol = m.getvolume() |
|  | m.setvolume(result3) |
|  |  |
|  | if result == time.localtime().tm\_hour and result2 == time.localtime().tm\_min: |
|  |  |
|  | sound() |
|  | print('playing alarm') |
|  |  |
|  | #break |
|  |  |

A python file was used to create the alarm function with the amplifier and hardware. In the code, it imports firebase, python libraries, the sensor libraries, and the raspberry pi volume control in order to enable alarm functionalities. Then 3 variables were declared to always equal the hour, minute, and volume represented in the firebase. Whenever the raspberry pi's time did not equal to the time outputted by the user, nothing would happen. However, when the integer values correspond with the pi and firebase, the alarm would emit a sound, and the file would notify that the alarm is playing. As for the volume, the firebase represents numbers within integers of 1 to 100. Whenever the volume is changed in the database, the python file will declare the raspberry pi's volume equal to that of the outputted integer in the firebase.

## **Display Function Code:**

|  |
| --- |
| #!/usr/bin/python |
|  |  |
|  | import time |
|  | import datetime |
|  |  |
|  | from Adafruit\_LED\_Backpack import SevenSegment |
|  |  |
|  | # =========================================================================== |
|  | # Clock Example |
|  | # =========================================================================== |
|  | segment = SevenSegment.SevenSegment(address=0x70) |
|  |  |
|  | # Initialize the display. Must be called once before using the display. |
|  | segment.begin() |
|  |  |
|  | print "Press CTRL+Z to exit" |
|  |  |
|  | # Continually update the time on a 4 char, 7-segment display |
|  | while(True): |
|  | now = datetime.datetime.now() |
|  | hour = now.hour |
|  | minute = now.minute |
|  | second = now.second |
|  |  |
|  | segment.clear() |
|  | # Set hours |
|  | segment.set\_digit(0, int(hour / 10)) # Tens |
|  | segment.set\_digit(1, hour % 10) # Ones |
|  |  |
|  | # Set minutes |
|  | segment.set\_digit(2, int(minute / 10)) # Tens |
|  | segment.set\_digit(3, minute % 10) # Ones |
|  | # Toggle colon |
|  | segment.set\_colon(second % 2) # Toggle colon at 1Hz |
|  |  |
|  | # Write the display buffer to the hardware. This must be called to |
|  | # update the actual display LEDs. |
|  | segment.write\_display() |
|  |  |
|  | # Wait a quarter second (less than 1 second to prevent colon blinking getting$ |
|  | time.sleep(0.25) |

The display screen utilizes python and is on a constant while loop. The while loop will always be updating and displaying the local time. The local time is based on the current time on the raspberry pi and uses a function called datetime.datetime.now(). The 7-segment screen will place the correct sections of each numeric value.

## **Temperature Function Code:**

|  |
| --- |
| import time |
|  | import board |
|  | import busio |
|  | from adafruit\_htu21d import HTU21D |
|  | from firebase import firebase |
|  |  |
|  | myFirebase = firebase.FirebaseApplication('https://alarm-app-9c3ce.firebaseio.com/') |
|  |  |
|  | # Create library object using our Bus I2C port |
|  | i2c = busio.I2C(board.SCL, board.SDA) |
|  | sensor = HTU21D(i2c) |
|  |  |
|  | while True: |
|  | temp = sensor.temperature |
|  | temp = '%.2f' % (temp) |
|  |  |
|  | print("\nTemperature: %s C" % temp) |
|  | print("Humidity: %0.1f %%" % sensor.relative\_humidity) |
|  | upload ={'Stemperature':temp} |
|  | myFirebase.put('/SensorData',"Stemperature",upload) |
|  | time.sleep(2) |

The temperature sensor runs on Python 3. The code requires a while loop in order for constant updates of temperature readings. Those temperature readings are then sent to firebase and then displayed on to the app. The temperature sensor will be sending its readings to the firebase in an interval of 10 seconds.

# Status Reports

**Status Report 1:**

Dear Kristian,

The following is the status report for JnJ’s Clockwork:

**Physical Format**

The product will take on a box form factor in the end as it is supposed to replicate a alarm clock that would be on a typical nightstand. The current dimensions of the project are unknown however the max would be 12 13/16" x 6" x 2 7/8" = 32.5cm x 15.25cm x 7.25cm.

**Recent and Current Progress**

Johnson

Fixed few bugs in the application that were still present from last semester. Currently researching how to integrate Bluetooth connectivity with the Hardware portion.

Jordan

Recently purchased 2 display screens which we plan on using for the Hardware, 20×4 RGB Backlit LCD display and 1.2″ 4-Digit 7-Segment display. When the products arrive, Jordan will start to work on how to integrate them with the other sensors.

Juan

Created rough schematic and layout for how the hardware device should appear and function. Assisted Johnson in fixing database errors from last semester in the application. Next step is to design the internal layout of the project to determine locations for sensors and wiring.

**Problems and Hyperlinks**

Some problems we have encountered has to deal with the application itself in terms of how to actually display what we want to the screens purchased. Johnson and I are not quite sure what code to implement that would allow such a process to occur. Research is still ongoing so that once the displays arrive, we may attempt a few theories we currently have.

**Financial**

The price for the 2 display screens we recently purchased are as follows:

20×4 RGB Backlit LCD - $32.83 CAD

1.2″ 4-Digit 7-Segment display - $24.93 CAD

More components may be purchased in the future depending on current circumstances.

**Status Report 2:**

Dear Kristian,

The following is the status report for JnJ’s Clockwork:

**Recent and Current Progress**

Johnson

Attempted to connect the temperature sensor to the database. I have proper C code working however in order to simply things, I am trying to get python code to upload to the database. While also displaying a real-time clock on the 7-Segment sensor.

Jordan

Worked with the amplifier and speaker, managed to produce white noise, and is now working on emitting an alarm sound.

Juan

Soldered all the sensors, and has completed all designs for the fritzing diagram. His next step is to print out the PCB, solder, and test it with all connected sensors. Once that is complete, he can move onto creating the casing for the combined hardware.

**Problems and Hyperlinks**

One issue occurred when testing the 7-Segment display, we learned that the display can only take in 3.3v rather than 5v. Because of this the original display screen was short-circuited and a new one had to be purchased. An issue that was discovered with the raspberry pi was it required an amplifier and a speaker in order to emit sound.

**Financial**

The price for the 2 display screens we recently purchased are as follows:

I2S 3W Class D Amplifier Breakout MAX98357- $7.83 CAD

Speaker 3” Diameter 8 Ohm 1 Watt - $2.57 CAD

1.2″ 4-Digit 7-Segment display - $24.93 CAD

More components may be purchased in the future depending on current circumstances.

**Status Report 3**

Dear Kristian,

The following is the status report for JnJ’s Clockwork:

**Recent and Current Progress**

Juan

Printed out the PCB, soldered, and tested with all connected sensors. The next step is to create a case given that the PCB and sensors are all connected to the Pi, and finding dimensions for the alarm case. The dimensions of the alarm box are not yet determined, but will fall under the maximum dimensions of 12 13/16" x 6" x 2 7/8" = 32.5cm x 15.25cm x 7.25cm.

Johnson

Got the temperature sensor to connect with the database. He is now managing to add more python code into the data. Johnson is now working on displaying the temperature readings that are stored in the database to the application itself and also sending timer instructions to the raspberry pi.

Jordan

I got actual sound coming out of the speaker through the amplifier. An alarm sound has been emitted. Currently working with Johnson on triggering the alarm sound from data taken through the database by using python.

**Problems and Hyperlinks**

A problem occurred while soldering the PCB. We were measuring each pin and via with an ohm meter to identify its connection, and we came upon some vias that were not soldered accordingly. Another PCB had to be printed and soldered, while acquiring more pin sockets from the prototype lab. When the 7-Segment digital display was connected to the PCB, a problem occurred where the i2c would detect every single address from the sensor. A 4.7k ohm resistor was implemented into the IO slot to acquire the specified address. However, the same situation applies where the sensor displays every single address from the sensor; this only happens for the display screen and not the other sensors.

**Financial**

Nothing has been purchased lately as we retrieved pin sockets from the prototype lab.

More components may be purchased in the future depending on current circumstances.

**Status Report 4:**

Dear Kristian,

The following is the status report for JnJ’s Clockwork:

**Recent and Current Progress**

Johnson

Has full database connections with the temperature sensor and is able to get its readings from firebase to the android app. Also assisted Jordan with alarm functionality. Currently is researching on how to get the pi to start up and have its applications run automatically.

Jordan

Created an activity in the android app to get the user to send its alarm settings to firebase, the hardware is able to detect and read the set alarm. He will also attempt to implement multiple alarm ringtones and volume control from the android app.

Juan

Updated the GitHub to provide its current version of android code. Is still working on the enclosure. He continues to work on the report in terms of the app build details and has included the build instruction, status reports, and illustration to the report.

**Problems and Hyperlinks**

One of the issues we encountered was with the display screen. When we originally used the PCB to connect the display screen, the screen would output all i2c numbers and would no longer run. We had this issue with 2 screens that we purchased prior. As a solution we decided to get a 0.56” 7-segment display screen instead of a 1.2”. Thankfully the 0.56” had the exact same pins as the 1.2” except for the IO pin and also was on Amazon so it came in fast. In terms of the amplifier, Jordan encountered an issue where the amplifier was unable to output sound when the alarm is supposed to go off. Jordan is still working on this issue and will hopefully have this issue resolved by the weekend.

**Financial**

The price for the display screen we recently purchased are as follows:

Adafruit 0.56" 4-Digit 7-Segment Display w/I2C Backpack White [ADA1002] - $30 CAD

More components may be purchased in the future depending on current circumstances.

**Status Report 5:**

Dear Kristian,

The following is the status report for JnJ’s Clockwork:

**Recent and Current Progress**

Johnson

Managed to get all the python files in the raspberry pi to start up upon booting the system. Organised the python files needed for the hardware within the raspberry pi. Assisted Juan with production of enclosure, and also assisted Jordan with debugging the alarm functionality. He is now planning on picking up the display board.

Jordan

Fixed an issue where the amplifier won’t trigger the alarm when it is supposed to. Worked on the alarm layout within the android app. Added a volume feature that can control the hardware’s alarm. Finished the alarm python file where it would receive the hour, minute, and volume from the application. This was done by sending values to the firebase using an array list and volume bar. Has full database connections with the amplifier, alarm, and application. Assisted Juan with production of enclosure.

Juan

Updated the project’s GitHub to represent its current version of android code. Juan worked on CorelDRAW to create the case outside of class and printed it in the prototype lab.  He progresses to work on the report in terms of the app build details.

**Problems and Hyperlinks**

An issue that we encountered was with the enclosure. While assembling the case numerous times, slot sizes were not the same, and slot positionings were not corresponding lengths. After correcting the dimensions, another problem we encountered was while assembling the case, the mono amp was interfering with the upper corner of the case. The case had then been made taller. Another problem we encountered was the alarm’s functionality. While attempting to make a file for the alarm, the alarm would only play once without a continuous loop. The solution to the problem was to create a while loop that was always true. When the alarm would reset (hour = 0 minute = 0), the alarm would go off during midnights due to the 24 hour clock format. This was solved by setting the hour and minute value to -1.

**Financial**

No more components are needed to be purchased as the project is completed.

# Problems Encountered

During the development of this project, we ran into many problems and roadblocks which halted progression and created extra hurdles for us to overcome. These ranged from database communication errors to hardware malfunctions.

## **Hardware Difficulties**

In terms of hardware failure, we initially wanted to work with a 1.2” 4 -digit 7-segment display from adafruit. The display used the same pins as the 0.56” model besides an IO pin which is required to be plugged into either 3.3v or 5v depending on the processor being used, in this case being a raspberry pi 3 B+. We purchased the display and got it to output the current time as desired, however upon second boot up, the display stopped being recognized and no longer functioned. Upon checking for the failure point, we came to the conclusion that the display short circuited due to the IO pin being plugged into 5v rather than 3.3v. 5v only works for processors such as Arduinos, and not really stable with raspberry pi’s. With this knowledge, we purchased the same display and used the 3.3v pin instead, however it resulted in the same conclusion. The display appeared to short circuit again. We could not figure out the reasoning behind it and could not afford to waste time ordering a new one. We also encountered issues with the PCB board that was created in terms of soldering, where the pins connecting the Pi to the board (20xpins) were improperly soldered, resulting in the sensors not receiving any current.

## **Database Issues**

When it comes to the database, one issue we encountered was that we weren’t able to get the temperature readings to send to the firebase. This is due to the rules of firebase that are automatically implemented when starting a project. The rules of read/write are set to where authentication is required. The problem lies where we weren’t aware of this ruling and therefore was unable to send any sort of data to the firebase.

## **Application Errors**

When working with the application itself, there were many instances where a (R.) error code occurred in android studio. This had to do with strings that were created upon making new xml and java pages that had similar or identical ids.

## **Case Design Problems**

Another area during the projects development where several issues occurred was with the cases design and construction. The case was designed using a software tool called Corel Draw and was created by Juan. It was very difficult to use as we had little to no experience using such software which resulted in many attempts when creating said case. The problem stemmed from measurement errors and incorrect placements for the connectors. Every sensor and speaker were to be mounted so cutouts for each device was needed to be accurately measured and positioned on the sides. As all the measuring was done by hand, it was prone to human error and resulted in many attempts to get just right. Another issue was that the when the speaker was mounted, it interfered with the PCB which was not desired.

# Solution

## **Hardware Fix**

For hardware, we decided to purchase a similar display to the one that kept malfunctioning, the 0.56” model which lacks an IO pin as that seemed to be the issue with the bigger variant. All other pins are the same which made readjustments simple. This completely fixed the issue and we were able to have the sensor display the time without it short circuiting. In terms of PCB, we needed to remake and solder the board as the old one was well beyond repair.

## **Database Fix**

The database issue was resolved by setting the rules to true. This allowed all access to read and write to firebase. This was not clearly evident at the start, hence why it took longer than anticipated.

## **Application Fix**

In order to fix the (R.) issue, we need to navigate to the strings.xml file and check change string ids that were the same. This did not always fix the error initially however; a successful reboot would clear the errors that were still present.

## **Case Design Fix**

The case design was eventually fixed after a lot of trial and error however the dimension tool in CorelDraw assisted greatly with the final product. The main issue was that after altering certain sides, others became distorted or their respective cutouts became enlarged. This was difficult to notice as the change was minuscule but still important. The dimension tool helped with keeping track of the distance for each respective part so that when one thing was altered, the other sides can be fixed respectfully. It took much longer as each and every side was required to keep track off however, this was still the most efficient method. In order to fix the speaker issue, the case needed to be raised by 1 inch to leave space between the speaker (when mounted) and the PCB. While it may seem like a simple fix, all other sides needed to be taken into factor as the respective holes needed to be adjusted as well.

# Future Planning

Plans that are to be determined in the future can go a long way. Be it the sound, hardware, or the mobile application. For starters, the alteration of volume can be implemented. Using these variations of volume can further alter the alarm’s sound to the user’s desires. Another plan that was made throughout the project is the effectiveness of the alarm sound. Having more than 1 alarm clock being set to 1 mobile application will increase the alarms sound overall. An example of this can be at one’s house. Multiple alarm clocks can be set in various rooms of the house (i.e. Living room, bedroom, basement, etc.) that can all be programmed to one mobile device. In this case, wherever the person is in, be it the bedroom or living room, the alarm will effectively go off all throughout the house, all at the press of 1 button in the app. With this feature, teens can be sleeping in the living room without their mobile devices, and the alarm can still go off effectively even though they are not in their bedroom and while their phone is nowhere in sight. As we want the alarm to be at its full functionality, we plan to implement Bluetooth to the alarm clock as Wi-Fi is not always accessible. If someone is on vacation and/or is far away from technology, the Bluetooth feature can allow communication with the mobile application and hardware without the use of internet. This way, the phone and alarm clock can be used almost anywhere without the worry of having to use Wi-Fi via internet. Another plan that be set forth to the future is the effectiveness of the alarm clock. Since our goal is to wake up the user with this device, adding another hardware to force the user to physically be active would make the alarm clock more efficient than ever. As a result of this feature, the consumers will experience a tougher time to turn off the alarm which will keep them in a more “awake” state. With this plan, the alarm’s sound will keep on ringing until the goal is achieved. Another plan that can be set into motion is to make it more difficult to access the cancel button. As the user wants to turn off the alarm to stop the disruptive noise, we can implement a tougher way for the user to get into the application to ironically stall their goal.

# Reproduction of Product

A feature that was planned during the production of the project is to produce multiple ringtones/sounds to emit from the speaker as the alarm. To further enhance this option, having the user to implement their own sound or ringtone can be added to the mobile application. On the other hand, the mobile application will additionally add various volumes such as bass, tone, tempo, etc. To make the mobile and hardware device communicate with each other without internet via Wi-Fi, Bluetooth can be installed and accessed within the mobile application and hardware device. Other features that can be implemented to the project in the future is a hardware that can detach and reattach itself for the alarm clock.For example, when the alarm goes off, simultaneously a hatch will open in the device and shoot out a flying object (in this case let’s use a bird). Without the bird put back in its place, the alarm will keep emitting a sound until the bird is back at its initial place. Another feature to add a puzzle towards the login screen. Simply adding various difficulties of riddles, and puzzles can alter the user’s path and time to turn off the alarm.

# Conclusion

In Conclusion, JnJ’s Clockwork is an android based project designed to give users ease of access to anything time related in a simple and clean form factor. The project consists of mainly layers in order to function as designed. It utilizes 3 distinct sensors being the 4-digit, 7 segment display, Humidity/Temperature sensor, and audio amplifier. All sensors communicate with our database in firebase which allow interactivity with the android app. Users have the ability to create alarm profiles, set different time zones, create stopwatches, and view/save local temperature readings all in one condensed form factor.

# 

# 

# 

# 

# 

# 

# 

# Bibliography

Adafruit. (2018, December 21). Adafruit/Adafruit\_CircuitPython\_HTU21D. Retrieved April 22, 2019, from https://github.com/adafruit/Adafruit\_CircuitPython\_HTU21D

Adafruit. (2018, September 11). Adafruit/Adafruit\_Python\_LED\_Backpack. Retrieved April 22, 2019, from https://github.com/adafruit/Adafruit\_Python\_LED\_Backpack

Adafruit HTU21D-F Temperature & Humidity Sensor. (n.d.). Retrieved April 22, 2019, from https://learn.adafruit.com/adafruit-htu21d-f-temperature-humidity-sensor/python-circuitpython

Adafruit Industries. (n.d.). Adafruit HTU21D-F Temperature & Humidity Sensor Breakout Board. Retrieved April 22, 2019, from https://www.adafruit.com/product/1899

Adafruit Industries. (n.d.). Adafruit 0.56" 4-Digit 7-Segment Display w/I2C Backpack - Yellow. Retrieved April 22, 2019, from https://www.adafruit.com/product/879

Adafruit Industries. (n.d.). Adafruit I2S 3W Class D Amplifier Breakout - MAX98357A. Retrieved April 22, 2019, from https://www.adafruit.com/product/3006

Adafruit Industries. (n.d.). Speaker - 3" Diameter - 8 Ohm 1 Watt. Retrieved April 22, 2019, from https://www.adafruit.com/product/1313

Adafruit MAX98357 I2S Class-D Mono Amp. (n.d.). Retrieved April 22, 2019, from https://learn.adafruit.com/adafruit-max98357-i2s-class-d-mono-amp/raspberry-pi-test

Adafruit's Raspberry Pi Lesson 4. GPIO Setup. (n.d.). Retrieved April 22, 2019, from https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c

Butter, B. (2018, July 05). How to make "ALARM" in python (Simple method). Retrieved April 22, 2019, from https://www.youtube.com/watch?v=xgcFQzChknk

CanaKit Raspberry Pi 3 B (B Plus) with 2.5A Power Supply (UL Listed). (n.d.). Retrieved April 22, 2019, from https://www.amazon.ca/CanaKit-Raspberry-Power-Supply-Listed/dp/B07BC6WH7V

Copper Foil Tape with Conductive Adhesive (1inch X 12yards) - Slug Repellent, EMI Shielding, Stained Glass, Paper Circuits, Electrical Repairs - Extra Long Value Pack At A Great Price. (n.d.). Retrieved April 22, 2019, from https://www.amazon.ca/Copper-Conductive-Adhesive-1inch-12yards/dp/B018RDZ3HG

Dyson, M. (n.d.). AlarmPi. Retrieved April 22, 2019, from http://mattdyson.org/projects/alarmpi/

ELEGOO 120pcs Multicolored Dupont Wire 40pin Male to Female, 40pin Male to Male, 40pin Female to Female Breadboard Jumper Wires Ribbon Cables Kit for arduino. (n.d.). Retrieved April 22, 2019, from https://www.amazon.ca/Elegoo-120pcs-Multicolored-Breadboard-arduino/dp/B01EV70C78

Google. (n.d.). Read and Write Data on the Web  |  Firebase Realtime Database  |  Firebase. Retrieved April 22, 2019, from https://firebase.google.com/docs/database/web/read-and-write