Security for the Smart Home and Office

Prepared by

Denald Demirxhiu
Jacob Ladan
Marko Javorac

Humber College Institute of Technology and Advanced Learning

Discipline: Computer

April 25, 2019

Declaration of Group Authorship

Roomi hereby certifies that this technical report we are submitting is entirely our own original work except where otherwise indicated. We are aware of the College's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. A detailed breakdown of the individual work to be completed for this project can be found in *Figure 1 (pg. 6)*, as per the list of illustrations.

Student #1 Name: _____ Signature: _____

Student #2 Name: _____ Signature: _____

Student #3 Name: _____ Signature: _____

Date of Submission: _____

Abstract

This project combines the sum of our skills and knowledge gained through the first five semesters of the Computer Engineering Technology program at Humber College. As a result, the final product will encompass a door latching mechanism with NFC/RFID reader, graphical display, cloud storage database, and an accompanying Android application. The goal is to give users a modern approach to security systems for residential and commercial dwellings. This project demonstrates our extensive knowledge of both hardware and software. This technical report will outline the complete process of research, construction, testing, and implementation of each aspect of the project.

Introduction

Today, many rooms being electronically secured lack the flexibility of information display and mobile user management. The problem herein lies with the decrease in usability, and the sole focus of Roomi is to remedy this. Through the tying together of an LCD screen and existing RFID technology, users will be able to view additional important information about secured rooms, as well as allow administrators to update this information easily and without the need for support from IT.

A 2.4" Digole LCD Display will be used to display relevant information unique to every room. A PN532 NFC/RFID reader will allow individuals access to configured rooms, based on a predefined access level setup through the Android application. A mini push-pull solenoid will be used to demonstrate how the locking mechanism will function.

The mobile application will allow the administration to modify user and room access and display settings in a user-friendly environment. The hardware and application will work in unison with a cloud service hosted on Google's Firebase platform to store relevant information for rooms, user authentication, and personnel registered through the application.

Table of Contents

List of Illustrations

Requirements Specification

Software

In this section, we will describe the requirements for the application and the database that will work in parallel with the hardware for the capstone project of the Computer Engineering Technology program at Humber College. These requirements will outline the specifications of the Broadcom development platform as well as the three sensors/effectors that will be interfaced.

Mobile Application

Purpose

This application will give administrators the utilities needed to manage room and lab access as well as display relevant information regarding each individual room and lab.

Operating Environment

This application will operate on a mobile device running Android OS updated to at least version 5.0 (Lollipop). User settings will be delivered to a cloud service database (Firebase) through which a Raspberry Pi 3B+ will be also communicating. The Raspberry Pi will perform the necessary actions to control room, user, and personnel access/traits.

Assumptions and Dependencies

The most significant dependencies of this mobile application include the server-side database (where the data will be stored), the actual hardware it is going to communicate with, and time as the time constraint we have is restricted to 10 weeks. These constraints do affect

the development time and the list of features that are going to be implemented, therefore in the final version some of the features might not be fully integrated and compatible, but the functionality will be implemented. There might be feature drop as a result of the time constraint, which can be implemented at a later version of the application.

Database

Purpose

The database will be used to store relevant room information for rooms that have been registered with the application, as well as information about employees/students who have been registered with RFID cards. Additionally, administrators using the application to configure rooms will be required to log in, as a result, account information will be stored.

Operating Environment

The database will be hosted on Google's Firebase cloud service. The information will be stored in JSON format, utilizing RESTful architecture as a NoSQL structure. User authentication information will be stored using Firebase's authentication database, which allows us to ensure obfuscation of sensitive information such as email and password.

Assumptions and Dependencies

Due to the nature of our application and its hardware, the most significant dependency going forward is the requirement of having 100% connectivity to the internet. In order for the hardware, and the application to function accordingly, they will both need to be connected to our cloud service at all times. Additionally, we are utilizing Firebase's free plan to store data. As such, in order to not incur additional charges or reduce functionality for the user, we will need to limit data rates to the specified plan.

Database Structure

At the top level, only one branch is presented, each child holding a unique string value for registered users. Continuing, each user node holds the pertinent information used for authentication as well as two branched nodes; rooms and personnel. The rooms branch holds all the information associated with the user's rooms that have been set up through the application (access level and name). The personnel branch, similar to the rooms branch, holds all the information associated with the personnel set up by the user through our application (name, access level, and avatar colour).

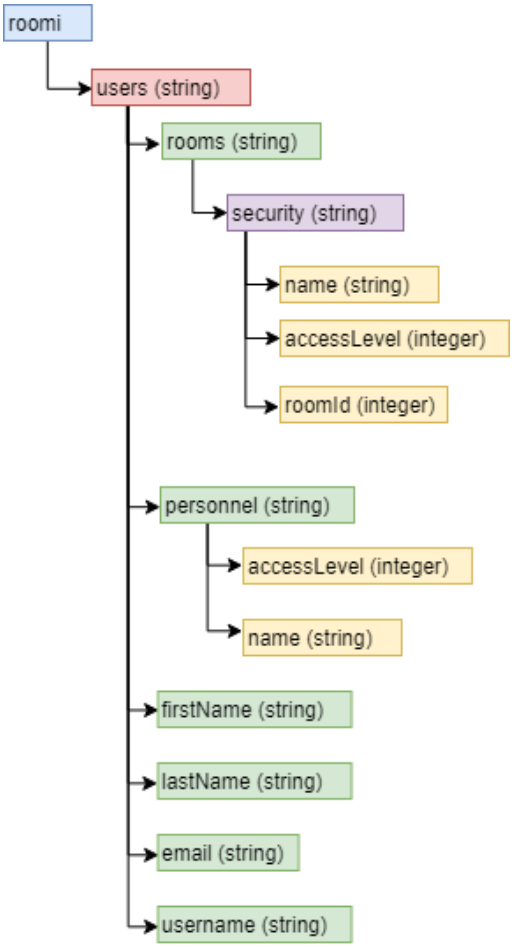

Figure 1 - Database Structure

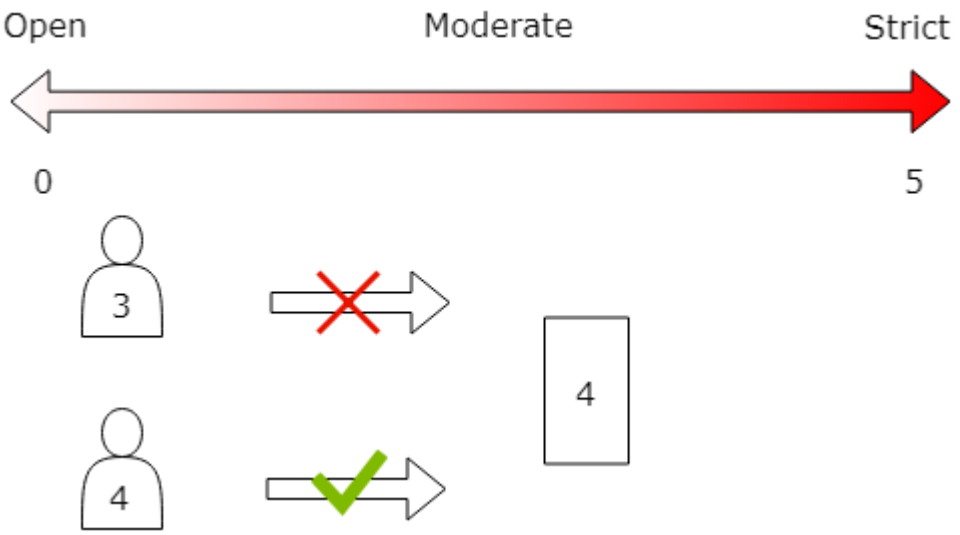

Figure 2 - Access Level Representation

Firmware

Purpose

The purpose of the firmware will be to provide the software platform of communication between the hardware, database, and mobile application. The firmware will also provide the user interface at the point of contact.

Operating Environment

The operating environment is called Raspbian. Raspbian is a free operating system based on Debian GNU/Linux and optimized for the Raspberry Pi hardware.

Assumptions and Dependencies

It is critical for Raspbian to receive continual support throughout its planned lifetime. This is important for security and I2C driver compatibility.

Hardware

In this section, we will describe the hardware specifications for the development of the capstone project. These requirements will outline the specifications of the 3 unique sensors, the Broadcom Development Platform (Raspberry PI 3B+), and system enclosure. The functionalities of all these components will be tightly coupled together to provide seamless communication between the hardware operations and the software aspect of the project.

Development Platform

Purpose

The development platform that is going to be the main focus of the entire project is the Raspberry PI 3B+. It offers a wide variety of features including the ability to communicate with sensors and actuators via I2C, a fast CPU for handling various tasks, 40 general input/output pins that can be used as the main way of communication with external components, and more. The purpose of this device is to process the information retrieved from the RFID sensor mounted on to it and based on the data stored on the database make decisions by triggering the push-pull solenoid to act as a door lock. Additionally, it will display relevant information to the Digole display as a way to communicate with users of the device.

Operating Environment

As it was mentioned above, the Raspberry Pi will be used as the central board to connect to and communicate with the respective sensors/actuators. The board will be mounted on the wall where it will provide a way to interact with RFID cards. The cards will be used to grant access to specific rooms whose access level information has been set by the appropriate administrators. With the exception of the mini push-pull solenoid, the RFID sensor and the

Digole display will be mounted on the Raspberry PI, which will allow a level of interactivity with the whole system.

Assumptions and Dependencies

The major dependency of the hardware will be power, which will be assumed to be supplied at all times. The power to each of the components will be provided by the Raspberry PI itself, however, the power for the Raspberry PI will be supplied by the micro USB charger. The operation of the entire system also depends on a reliable network that ensures communication with the database to make the appropriate decisions at all times.

Interface boards and sensors

The main sensors/actuators that are going to be used in the making of the entire project include the Digole LCD Display, the PN532 NFC reader/writer, and the mini push-pull solenoid. The purpose of each sensor is described briefly below.

Digole LCD touch screen

The purpose of the LCD screen will be to display pertinent information related to the room it is associated with. The information will include the room's name and access level. Additionally, the LCD screen will display general information such as the date. The LCD will be implemented using a Raspberry Pi 3B+. The Pi will connect to the internet using wifi and communicate directly with the database retrieving and updating any information necessary for display.

PN532 NFC Reader/Writer

The purpose of the NFC reader/writer will be to control access to the room it is associated with. Along with the LCD screen, the NFC will be implemented using a Raspberry Pi

3B+. The Pi will connect to the database using wifi to retrieve the access level associated with the room. Knowing this information the NFC will be able to grant access if the personnel's access level is equal to or greater than the room's access level or deny them if it is lower.

Mini Push-Pull Solenoid

The purpose of the mini push-pull solenoid will be to demonstrate the mechanism that will deal with locking and unlocking doors.

Other Accessories and Enclosure

The development platform will require a power source, display, and a network connection. The enclosure should be designed in CorelDraw and laser cut from acrylic. It may be further improved by 3D printing (responsibility: Mechanical Engineering Technology collaborators).

Build Instructions

Repository: https://github.com/roomi-develop/roomi

Budget

A full list of materials along with a detailed view of costs can be downloaded from within the project's repository. The total cost of producing this prototype is heavily inflated due to the cost of the soldering kit that was supplied in the lab during development. Any generic soldering iron can be used for this project.

The total cost after removing the soldering kit is: $474.66 after taxes. This includes all the tools used in completing the prototype (e.g. wire cutters, needle nose pliers, breadboard, etc.).

Notable purchases include: Raspberry Pi 3B+ Kit ($94.95 CAD), Digole 2.4" LCD Display with Touch Capabilities ($17.49 USD), RFID/NFC Reader ($39.95), and the Mini Push-Pull Solenoid ($21.23).

Time Commitment

A detailed view of the schedule followed for this project can be downloaded from within the project's repository. The schedule uses a weekly breakdown that follows the CENG 317 and CENG 355 class schedules for fall 2018 and winter 2019. This prototype could be completed in a more compact time frame if the builder so chooses. The schedule is helpful in outlining the overall flow and the order in which each milestone for the project is completed. Originally, the

project was completed over the 30 weeks needed for the school year. However, it could more reasonably completed from within 2-4 weeks dependent on how many of the materials the builder already possesses. Access to the facilities necessary for producing the PCG/case, and shipping times.

Mechanical Assembly and Bread Boarding

Digole 2.4” LCD

Firstly, in order for the Digole LCD to use I2C communication with the Pi, one modification must be made to the screen's logic board. The builder must solder a short between the middle and the I2C adapters as outlined in the image below.

Figure 3 - Digole Jumper for I2C

Next, the following connection scheme will be used in order to connect the Digole LCD to the Raspberry Pi's GPIO pins.

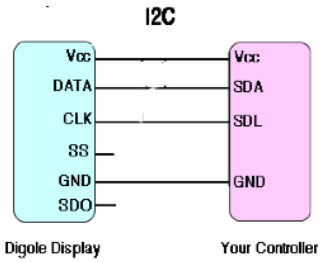

Figure 4 - Digole to GPIO Connections

The LCD will be connected to a breadboard, and the following connections will be made from the board to the GPIO pins on the Pi. With GPIO on the left and the corresponding LCD connections on the right. The SS and SDO pins on the LCD will be left open:

- Pin01(3.3V) --> VCC
- Pin03(BCM 2 Data) --> DATA
- Pin05(BCM 3 Clock) --> CLK
- Pin06(Ground) --> GND

As a result the builder will now have produced something similar to the construction depicted below.

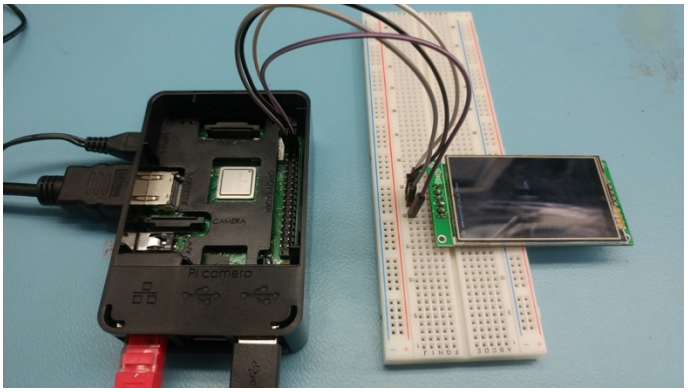

Figure 5 - Digole LCD to Raspberry Pi Breadboard Connection

Note: A 6 pin header has been soldered to the LCD connectors in order to allow it to attach properly to the breadboard and later the PCB.

With the Pi powered on, and I2C communications enabled, the LCD will power on and display its factory default message indicating that proper power has been supplied, ground has been connected, and the LCD is in working order.

Next, to ensure the LCD has been connected properly for I2C communications, the following command should be entered in to the Pi's terminal: `sudo i2cdetect -y 1`. This will display a simple graphic listing each device connected to the I2C bus and its corresponding address. The address the LCD uses is 0x27.

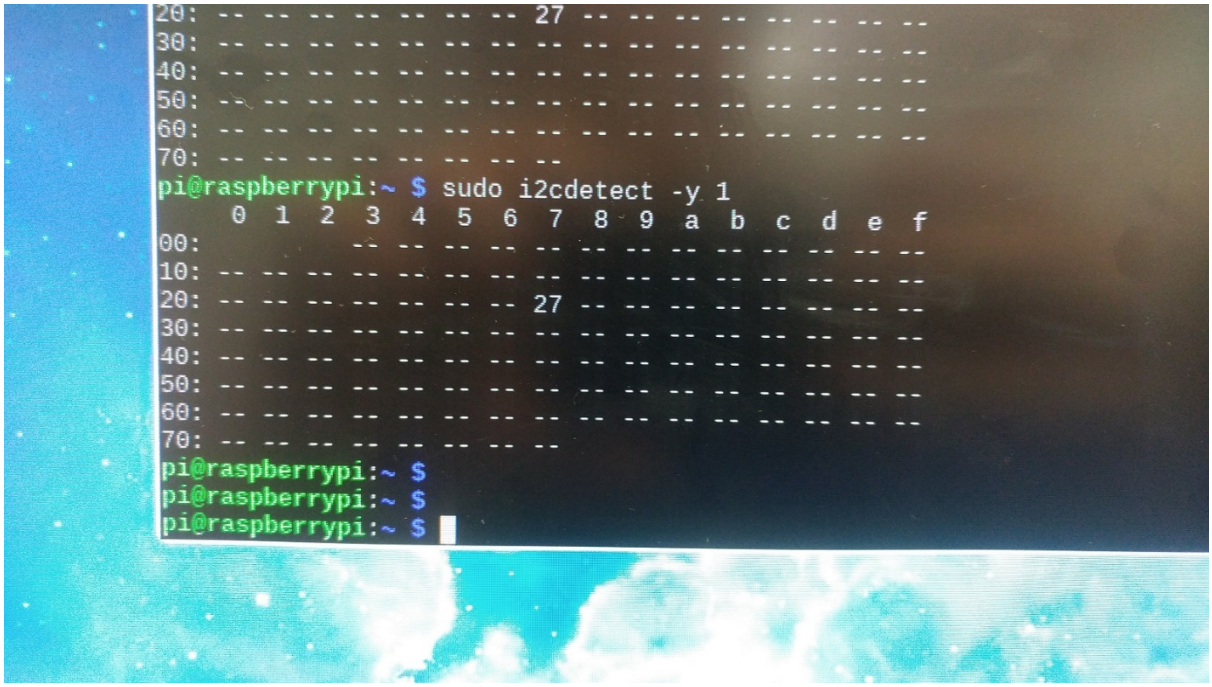

Figure 6 - Digole LCD I2C Connectivity Verification

NFC Reader

To enable communication between the Pi and the PN532 Board, you will have to choose a communications protocol and solder jumpers on the designated protocol. For this tutorial, we will be using I2C. The jumpers are included in the PN532 kit.

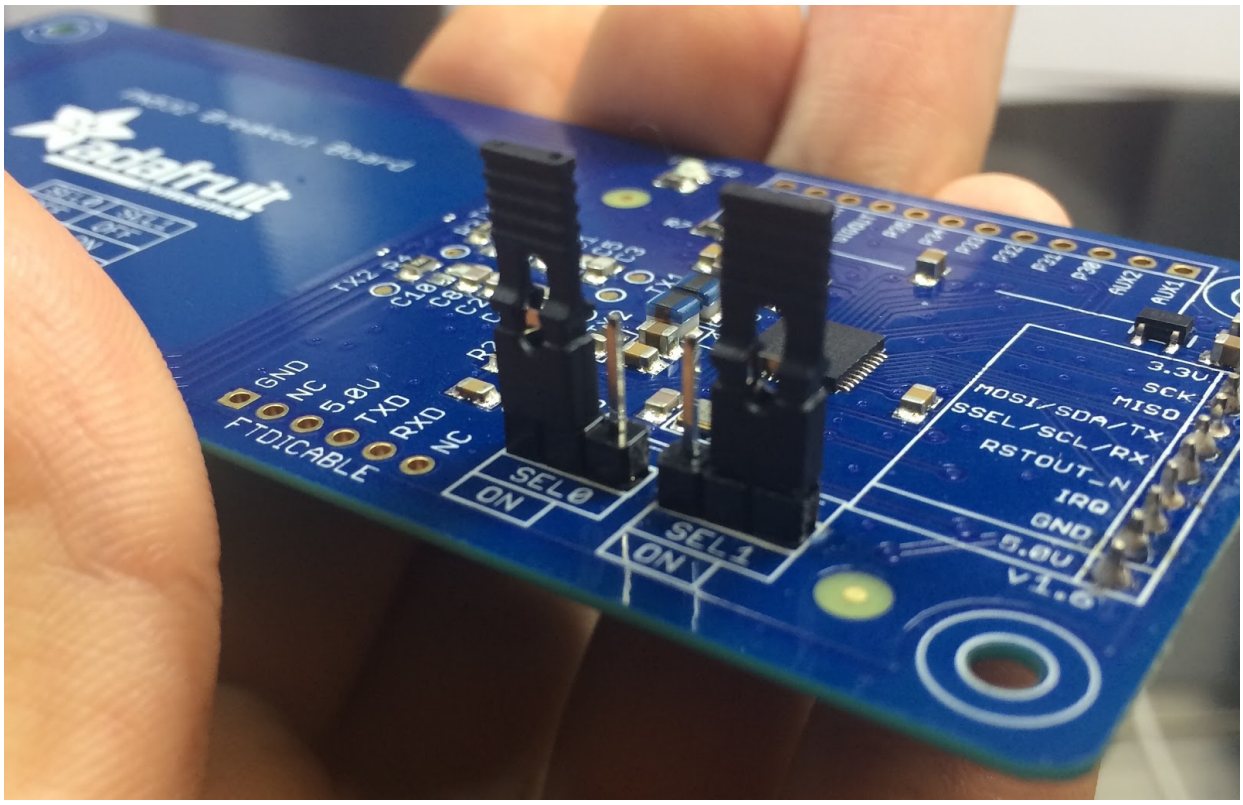

Figure 7 - PN532 I2C Jumper Configuration

To enable I2C or any other communication protocol, the jumpers must be configured correctly. Setup table is below.

Working Interface	Channel 1	Channel 2
HSU	OFF	OFF
I2C	ON	OFF
SPI	OFF	ON

Figure 8 - PN532 Jumper Connection Configurations

We will give it a quick test to make sure all our components are working. A prototyping breadboard is a great way to do this. The pins must be configured the right way for our sensor. Be extra careful as you do this to not damage you board.

- Pin01 --> 3.3V
- Pin03 --> SDA

- Pin05 --> SLC
- Pin06 --> GND

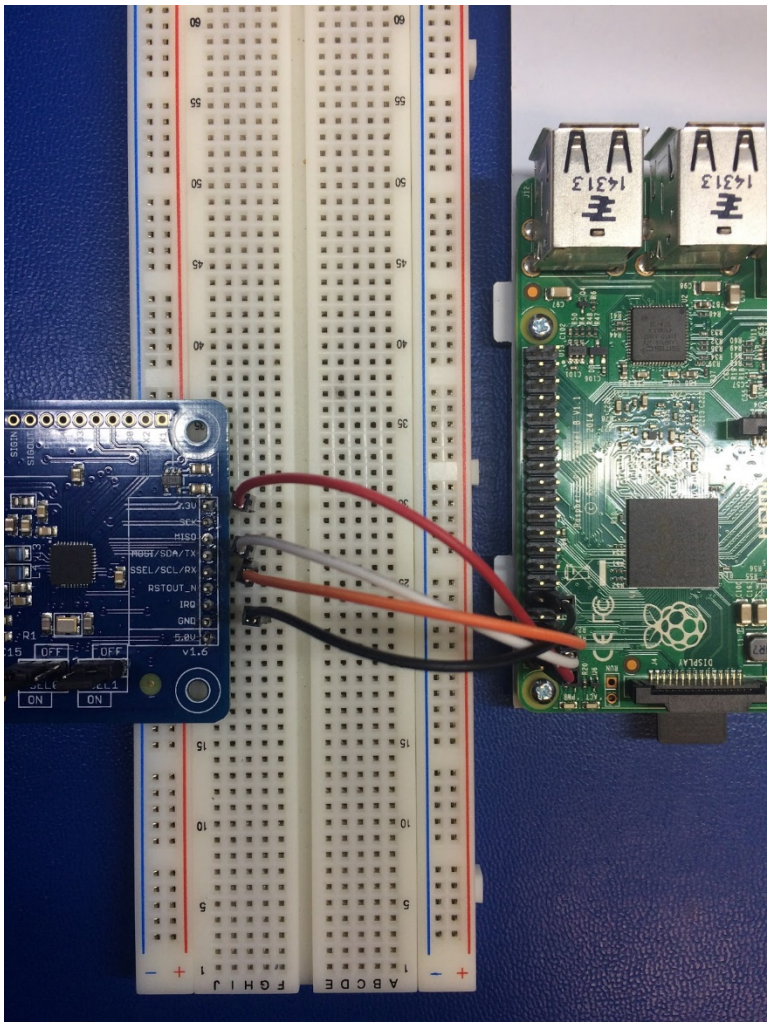

Figure 9 - PN532 to Raspberry Pi Breadboard Connections

Running the i2cdetect program will let you know if your PI can actually see the i2c device. Your address might vary from mine and that's fine. The goal is just get an address.

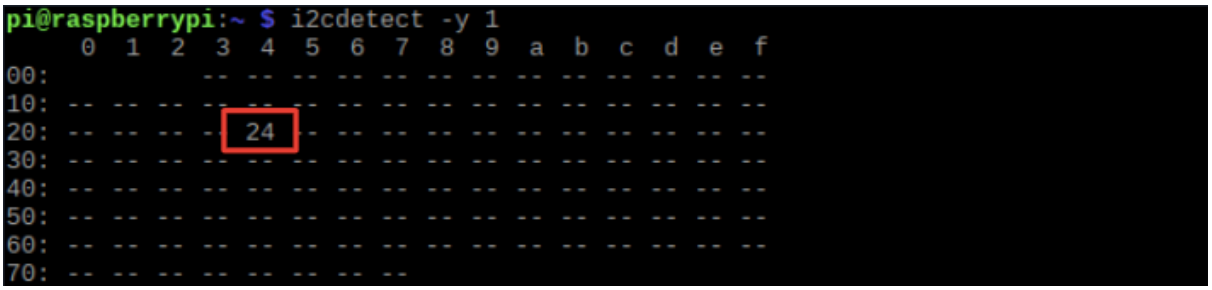

Figure 10 - PN532 I2C Connection Verification

Using libnfc, we can run a simple poll program that when a card is detected, it will state simple information and confirm we can read NFC cards.

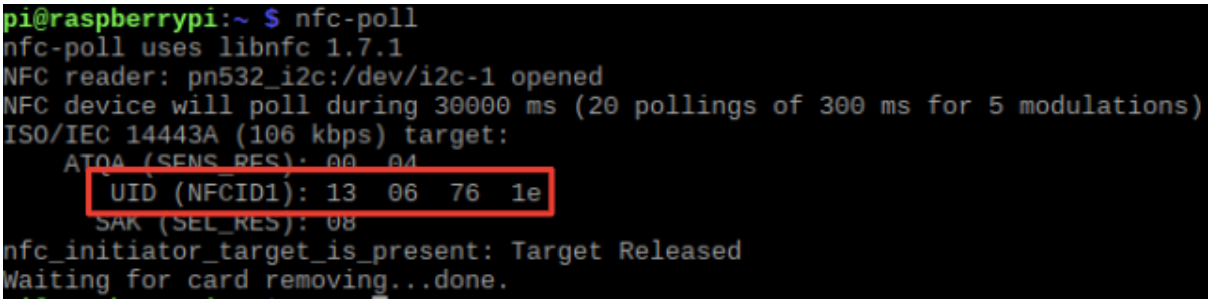

Figure 11 - PN532 Hardware Test with nfc-poll

PCB and Soldering

Soldering Tutorial and Tips

Soldering is a very important step in building the printed circuit board and assembling all the components to work properly with the sensors and actuators. This section is helpful for anyone that has not done any soldering prior to this project. The main two components that are needed to solder are the soldering iron and solder. A soldering iron can be purchased at any hardware store and prices may vary. The solder refers to the alloy that typically comes as a long thin wire in spools or tubes. It is used to join together two pieces of metal in what is called a solder joint. When selecting a solder, however, there are two choices. Leaded solder is harmful to humans and can lead to lead poisoning when exposed to large amounts, so soldering with one should be done carefully. There is another alternative to this type of solder. Lead-Free solder is very similar to its leaded counterpart, apart from the fact that it contains no lead. Since there is no exposure to harmful fumes, this type of solder is relatively safer than the leaded one.

After having the components ready it is time to start soldering. Some tips to have in mind when soldering is to be cautious when handling the hot iron. Furthermore, always set the iron at a good medium heat (325-375 degrees Celsius). It is also important to tin the tip with solder before each connection to help prep the joint. When soldering, it is a good idea to heat both the pad and the part you want to solder evenly and at the same time. A good solder joint should look like a volcano, not a ball or clump. Start by turning your soldering iron on and setting it at the recommended temperature. At this point, touch the tip of the iron to the copper pad and the resistor lead at the same time. You need to hold the soldering iron in place for 3-4 seconds in order to heat the pad and the lead. Continue holding the soldering iron on the copper pad and the lead and touch your solder to the joint. It is recommended to touch the solder directly to the

tip of the iron. The joint should be hot enough to melt the solder when it is touched. If the joint is too cold, it will likely form a bad connection. After completing the soldering of that joint, remove the soldering iron and let the solder cool down naturally. Blowing on the solder is not recommended, as this will cause a bad joint. When the process is completed, the solder joint should be smooth, shiny and it should look like a volcano or cone shape. That is all the necessary information needed to accomplish soldering the circuit board.

Printed Circuit Board

Design Files and Fritzing

The PCB (Printed Circuit Board) for this application is a custom design planned to work seamlessly with the Raspberry Pi 3B+ and the custom enclosure design we have designed and built. The software that was utilized to design the circuit board is Fritzing, which is an open-source hardware design software that makes electronic mapping accessible for everyone. Fritzing can be downloaded from their website (http://fritzing.org/download/) and it offers compatible versions for various operating systems including Windows, Linux, macOS, and even the source code is available on Github. Installation notes are presented in the webpage provided above. To design the circuit board, we first required to lay out all the electronic components that are necessary for the correct functionality. The main parts that are crucial to the circuit board are two 2x20-pin stackable headers, one 1x9-pin stackable header, one 1x2 stackable header, one 1 kilo-ohm resistor, two 1.5 kilo-ohm resistors, one IN4004 diode, one MJE3055 transistor, and of course the components of the project that are needed to test the board when finished constructing all of it. We recommend getting two of each part since they are very inexpensive to buy at an electronics store, and can be easily swapped with their homologous counterparts if a problem occurs during soldering.

The Fritzing file supplied in the Github repository of this project contains the circuit board layout that is needed to be printed. It also contains information on the location of the components in the board itself, which can be very useful when trying to solder the components properly and in the right place. The components like the 9-pin stackable header, 2-pin stackable header, diode, and resistors should be soldered with the leads facing down the circuit board. The Fritzing design file contains a label showing which side is the top of the circuit board. It also contains information on how the components should be connected in a breadboard if there is a necessity to test the components prior to assembling everything. However, we recommend that every component is tested using a digital multimeter before making any breadboarding attempt, or even soldering them to the board. The 20-pin stackable header should be soldered with the leads facing the top of the board. A picture of the actual board design is provided below, however, if there is a need to make any changes, or add any labeling that helps in the assembly the design files can be downloaded from the Github project repository.

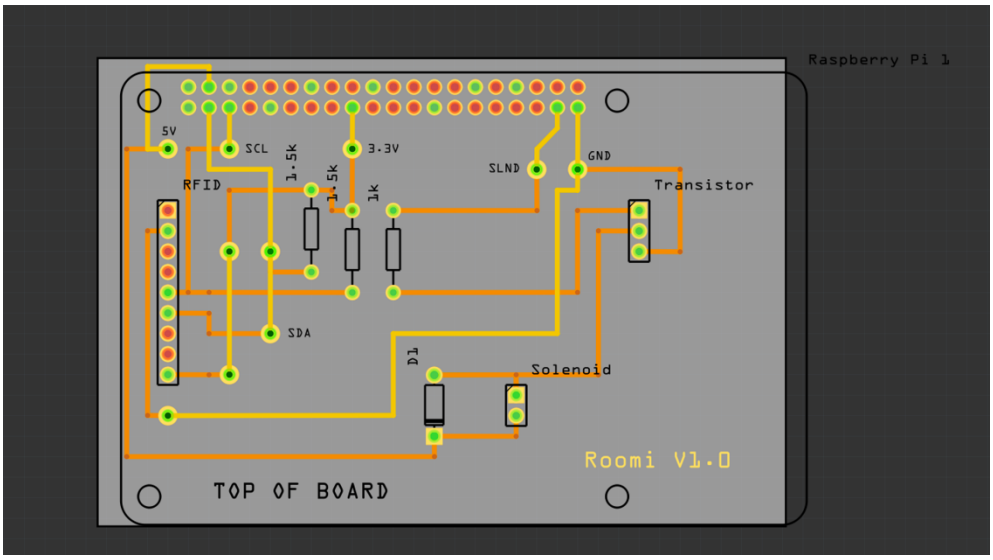

Figure 12 - PCB Design Schematic

Testing the Components

The instructions supplied below are very useful to test the individual components. The only tool that is necessary for the testing procedures is a digital multimeter. For the resistors, the testing procedure requires setting the multimeter in resistor reading mode or placing the reading setting in kilo-ohms. It does not matter where the multimeter leads are placed, as long as they are on the opposite resistor leads. The value read by the multimeter should be close to the value of the appropriate resistor, but not exactly the same, since resistors have tolerance. Tolerance is the percentage of error in the actual value of the resistor during manufacturing. This value can usually be between 5-10% of the theoretical value. If the multimeter displays O.L (Open Leads), it means that the resistor is not functional. In that case, you have to replace the resistor with another equivalent. Some multimeters include a diode testing setting, however, a resistor reading setting will work just as well.

Testing the diode requires first to determine which side is the cathode and which one is the anode. The main idea behind this is that current in a diode is allowed to flow from the anode to the cathode, but is blocked in the other direction. Usually, diodes have a thin line or strip that designates the cathode. We first take the ohmmeter and place the positive probe on the anode of the diode (the part of the diode without the strip indicator) and the negative probe on the cathode of the diode (the indicator line). In this setup, the diode should read a moderately low resistance. When reversing the multimeter leads, the reading should display O.L (Open Leads), because the diode is reverse biased and therefore it has a very high resistance. If you get an open circuit in one direction indicating the current is blocked, and a low resistance reading in the other direction, the diode is good. If there is an open circuit in both directions, the diode has failed with an open circuit. If there is low resistance in both directions, the diode has failed with a short. In both cases, the diode should be replaced.

Lastly, to test the transistor we have to first determine if it is an NPN or a PNP transistor. NPN transistors have the collector and the emitter as negative and the base as positive, while PNP transistors are the opposite of NPN ones. As with the diode, we need to set the multimeter in diode reading mode or in resistor reading mode. To test the transistor, we measure one diode junction with the multimeter leads situated one way and then we flip the leads of the multimeter to switch polarity. One side of the diode junction should read a very high resistance (the anode-to-cathode side) and the other side should read a much lower resistance (the cathode-to-anode side). We are going to test these junctions: emitter to base, base to emitter, emitter to collector, collector to emitter, collector to base, and finally base to collector. Each pair should have one side with very high resistance, and the other side with a much lower resistance of a few hundred thousand ohms. If this is the case for all the transistor leads, the transistor is good. If not, the transistor is defective and needs to be replaced.

Soldering the Components

There are only 6 Raspberry Pi general purpose pins that are utilized for connecting every component on the circuit board. Since the communication sensors and actuators require I2C communication, pins 3 and 5 (SDA and SCL respectively) are utilized for the connection. I2C supports multiple slave devices, therefore, we do not need to make any extra connections to the GPIO header pins. To easily determine the correct pins to solder and connect the components to, a useful website and service is PINOUT (https://pinout.xyz/), that is designed to be a quick and interactive reference to the Raspberry Pi GPIO pins. Pin number 4 is used to provide 5V power, pin number 17 is required to provide 3.3 Volt power, pin number 37 is utilized to control the mini push-pull solenoid, and finally, pin number 39 is ground. These pins are required to power, communicate, and control the respective components sitting on the board.

The recommended method of soldering the components is to start with the vias. The vias are holes in the board that allow the bottom and top lines of the board to connect. There are multiple vias in the board and they can easily be distinguished since these holes are bigger than the others. The process of soldering the vias is simple; it only involves putting a conductor wire in the via, bending it on one side so it is stable on the other side, and solder each side independently. The long ends of the wire can be cut after the soldering process is complete.

On the other hand, the first item that should be soldered is the 2x20-pin stackable header. Soldering this component is tricky and it requires a bit of getting used to. Above this section, some soldering tips were given to maximize soldering efficiency. Ideally, a fine tip soldering iron would be perfect for this type of application, however, a normal standard tip should also be just fine. The stackable header pins should be inserted in the circuit board facing upwards in the direction of the top of the board. It is recommended to solder the first four pins (2 at each side) to hold the stackable header in place so that it is easier to solder the rest of the pins. Soldering just the required pins would be just fine, however, soldering all the pins in the stackable header would provide for better-suited support of the circuit board when mounted to the Raspberry Pi.

The next step is to solder the 9-pin and the 2-pin stackable headers. The female ends of the stackable headers should be facing the top of the board, with the pins facing the bottom of the board. Again, just like with the 2x20 stackable header, it is recommended to solder the two pins at the end of the sides of the header, and then solder the rest of the pins. After finishing soldering these pins, it is necessary to cut the long ends of the headers because they are not going to be used. For the 1x2 pin header, it is necessary to bend the pins in close to a 70-degree angle, so that it lays flat in the board. The soldering process is exactly the same for this component as well.

Next, the resistors should be soldered in their respective position. All of them are parallel and close to each other on the board. In this step, it is recommended to use the design file to help denote where each resistor should be placed on the board. It does not matter which direction they are placed, as long as the resistors are facing the top of the board and their leads are facing the bottom. The resistor pins should be bent close to 90-degrees before being placed on the board. After placing the resistors in their correct spot, their leads should be bent in order to prevent them from falling off the board when attempting to solder the leads. Another component that can be soldered just as easily is the diode, but it needs to be done carefully because the diode can only work in one direction. First, the cathode side needs to be determined before moving forward. This is very simple since diodes have a thin line that indicates the cathode side. When sitting upright, the cathode of the diode should be facing the direction of the label of the board opposite the 2x20 pin stackable header. The soldering process, however, is identical to the resistors.

After the vias, stackable headers, the resistors, and the diode have been soldered properly, the last step is to solder the transistor. The pins of the transistor should also be bent in an angle close to 70-degrees, before being placed in the board. The flat side of the transistor is the one that is going to be bent towards the right side of the board, opposite the 9-pin header on the left side. The soldering process of the transistor is exactly identical to the other components of the board. After all the components have been soldered to the board, it should look like the images provided below:

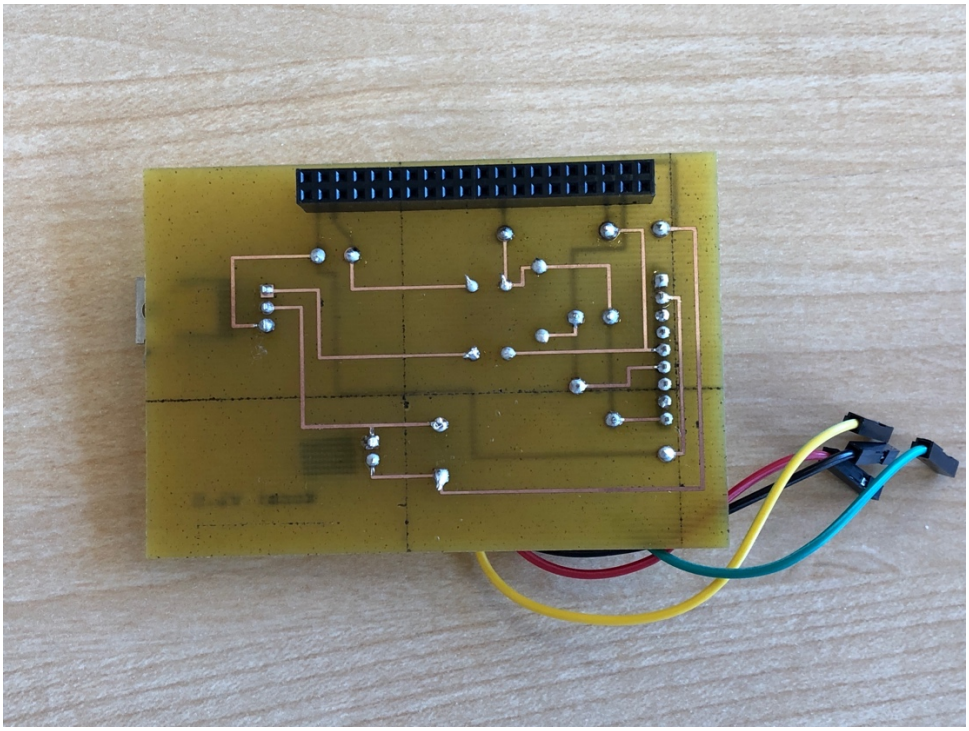

Figure 13 - PCB Bottom

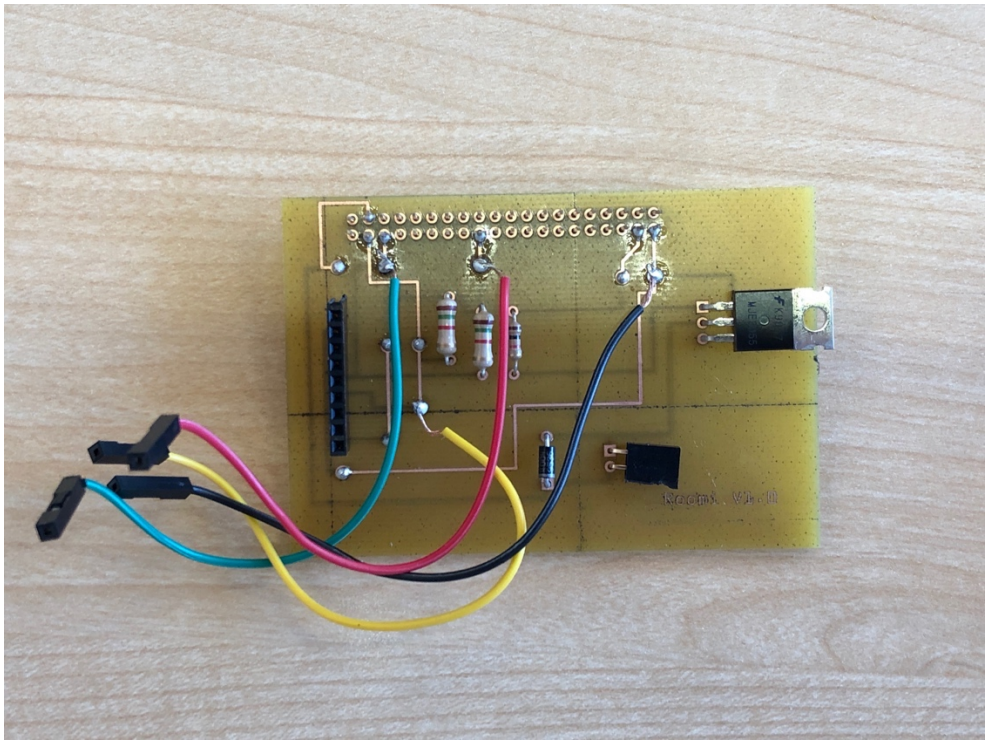

Figure 14 - PCB Top

Installing Raspbian OS

To run the raspberry PI, you will need to install Raspbian OS. This is the default OS designed for the Raspberry Pi and is the backbone of the project. All software will be running on this OS including the web server and real-time python script. It is critical to install the Raspbian correctly to avoid issues later on in development.

To create the OS, you will need to purchase an microSD card that is minimum 8GB but 16GB recommended. A quick search on Amazon pulls up over 10,000 results at varying price points. You can also purchase the card at any electronics store.

Once you have the card you will then need to image it with a copy of Raspbian. The official website raspberrypi.org/downloads/raspbian/ maintains and provides the latest versions. There are multiple variations offered but it is recommended to go with the Raspbian with desktop and recommended software. The package contains all the critical components with some extra that the development team deems valuable to any project. You can download it in either ZIP of torrent format.

Once it is done downloading, you will have to unzip the file using 7Zip (Windows) or The Unarchiver (Macintosh). The next step is to image the card with flashing software. The recommended software is Etcher. You will have to insert your card to the SD reader and then begin Etcher. After you have successfully flashed the card, you should be able to insert it into the Pi and power up. If all goes well, you will see a rainbow coloured screen the raspberry icons.

Software

Android Application

The Android application can be downloaded in its entirety from the roomi repository here - github.com/roomi-develop/roomi/tree/master/roomi

Choice of Languages

Java

Java is an object-oriented, class-based, and portable programming language. It is the standard language for mobile applications built to run on Google's Android operating system.

XML

Extensible Markup Language (XML) is used to design the layouts for the different pages of the application. Paired with Java, XML is the standard for designing Android applications and was chosen for this reason.

Developing Environment

To develop the application the Android Software Development Kit was used (SDK). The SDK gives us an environment in which to produce both the Java and XML code base, as well as gives us the tools necessary to test the application on a virtual device, and debug any issues that arise during development.

The Android SDK can be downloaded from the android developer website with the following link - developer.android.com/studio

Database

Linking the Application and Firebase

The database chosen for roomi, as discussed in other sections of this report is Google's Firebase platform. The two reasons for choosing Firebase as the database for roomi is firstly due to it being a real-time database allowing the different facets of roomi to communicate with each other seamlessly, and secondly because it uses a NOSQL JSON structure which enables us to easily access and update data from both the Java code driving the application and the Python code driving the hardware.

Google's Firebase can be accessed from the following link - console.firebase.google.com

In order to begin using Firebase for roomi, a Google account must be created in order to access Firebase's features. Once accessible, the application must be tied to the database with certain unique attributes from the application.

From the homepage of Firebase's console an application can be synchronized by first clicking on the cogwheel at the top of the page, and second, by clicking "Add firebase to a project". From here, users are directed to a form in which they are required to fill out certain details pertaining to the application such as, the package name, and encryption key in order to utilize authentication services. After the information has been entered, users are required to download and add to their project the file named "google-services.json", which holds all the necessary information for your application to connect to the database.

This file must be placed within the "app" directory, within the application's root directory. After this step, the app can now be tested to confirm that it can communicate with Firebase by

using Firebase's online tool to do so. If all of the previous steps are followed correctly, the application will now be able to access the database.

Connecting to Database from the Application

In order to connect the application to the database, and to gain access to reading and writing values with it the following steps are required:

1. **private FirebaseDatabase database** - initializing database variable
2. **private DatabaseReference dbRef** - initializing database reference variable
3. **database = FirebaseDatabase.getInstance()** - returns an instance of the database by utilizing the contents of the "google-services.json" file
4. **dbRef = database.getReference(String: dbPartentNode)** - returns a reference to the parent node within the database containing the child nodes for which values will be read or updated

Reading Database Values

In order to read values from the database, the child nodes' information must be stored within a predefined data structure using the following methods:

1. **public void onDataChange(@NonNull DataSnapshot dataSnapshot) {}** - this handler function will be called whenever a value has been updated from within the database. Passing through a snapshot of the reference created in the previous section
2. **List<data structure> structureList = new ArrayList<>()** - A list is created containing a data structure filled with the information for each of the child nodes returned from the

database. The database data structures can be found within the application's Java source code directory

3. After these two steps, the data from the database can now be accessed through the list and each subsequent data structure created

Writing Database Values

After determining which values will be updated and from which child node from within the database the application can now alter the values of an existing child node or create a new one if they so choose. Using the same instance and reference from the database connection, a new data structure containing the updated values will be created and uploaded to the database. The following method is called in order to write data to the database:

- **dbRef.child(String: childNode).setValue(new dataStructure(any: valueOne, any: valueTwo, ...))** - Uses the database reference variable to access the child node to be updated, or a new one to be created and through the use of the data structure will write the values accordingly

Deleting Database Values

The following method is called in order to delete a child node from the database. Again utilizing the same references and instances of the database as mentioned in the previous section:

- **dbRef.child(String: childNode).removeValue()** - Uses the database reference variable to access the child node to be deleted and removes the entry of it from the database

Flow

The moment the user opens the application they are presented with a login form in order to gain secure access to the various configurations of both the card holders (personnel) and the configured rooms that the Raspberry Pies have been attached to. Users are also able to access a “sign up” form in order to create an account to access the application’s features.

Once authenticated, the user is able to configure either the name or access levels for the aforementioned properties or delete them permanently from the application and database. A hamburger menu is also present from the main pages that allows users to navigate the various portions of the application, as well as view information about the developers or log out from the application entirely.

The image below depicts a graphical representation of the application’s navigation and flow as discussed within this section.

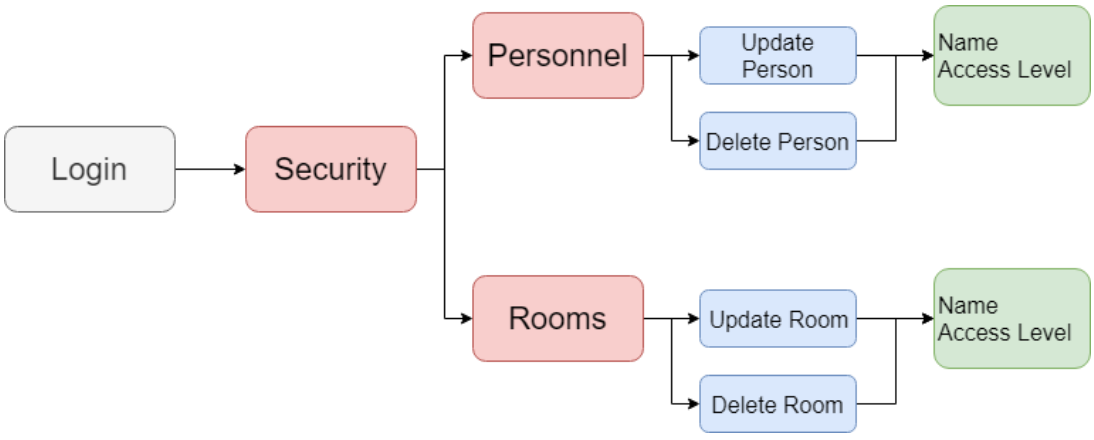

Figure 15 - Application Flow

Web Interface and Hardware Drivers

As both the hardware drivers and web interface work interconnectivity they will be discussed from within the same section. The reasoning for including a web interface with roomi is due to the fact that we required a software solution that would enable users to interact directly with the Raspberry Pi and the hardware modules directly. Something the Android application would now allow. The source files for both the hardware drivers and web interface can be downloaded from the roomi repository here - github.com/roomi-develop/roomi/tree/master/desktop_app

Choice of Languages

Python 3.5

Python is a high-level, general purpose, and object-oriented programming language. The reason Python (version 3.5) was chosen for roomi is due to two reasons.

Firstly, third party open source Python modules that have been previously developed were available for both the PN532 NFC reader/writer and the digole display, which significantly reduces the amount of time commitment required.

Secondly, a graphical user interface was required for certain tasks related to roomi's configuration as an end user. Python, utilizing a framework named "Flask" which allows a Python script to run as a local server on the Raspberry Pi and respond to HTTP requests from a web browser to serve web pages.

Due to this reasoning and the fact that only one language would be required for both programming the hardware and serving the web interface Python was chosen.

Python can be downloaded from the Python website at the following link - www.python.org/downloads/

HTML5

Hypertext Markup Language, similar to XML, is a language used to define the structure and design of user interfaces. More specifically, graphical user interfaces that are viewed and interacted with from a web browser. As HTML is the standard language for the layout of web pages it was chosen for roomi.

CSS3

Cascading Style Sheets is a styling language used to describe the presentation of the HTML on a webpage. It uses classes and selectors to give individual styling options to the various elements marked up with HTML. As with HTML, CSS is the standard styling language used for styling web pages and was chosen for this reason.

Javascript

Javascript, also referred to as simple JS, is a prototype-based object-oriented programming language that is used to program the front-end functionality of web pages. JS gives us the tools necessary to communicate with the back-end of the software (Python), as well as enables us to capture user input, such as button clicks and form inputs. As with HTML and CSS, JS is the third standard layer from which web pages are built upon and was chosen for this reason.

Python Dependencies

In order for the Python script to interact with both the hardware and serve the web interface, certain Python modules are needed in order do so. To download python modules on to the Raspberry Pi so that the Python script can access them and utilize the various functions they supply, a module manager must be downloaded first. The module manager used for this project was “Pip”. Pip comes pre-installed with python, and as such is the reason it was chosen.

In order to install a Python module with pip, the following command is to be used.

- *pip3 install <module name>*

Note that as we are using Python version 3.5, pip3 was used instead of pip in order for the script to be able to access the modules. The following modules are required for running the roomi software and as such must be also imported into the Python script.

- digole
- adafruit-circuitpython-pn532
- adafruit-circuitpython-lis3dh
- rpi.gpio
- board
- pyrebase
- flask

Python Web Server (Flask)

As mentioned previously, the Python framework Flask was used to serve the web pages for the graphical user interface. Flask allows the Python script to listen to HTTP request from a web browser and respond by either navigating to a certain web page, or process data that might be passed to it through JS.

In order to define which browser URL will be used to access which web page, routes are defined from within the script to serve the corresponding web pages. For example, when the user accesses the server at its route with the url “/” the corresponding index.html is sent to be processed and displayed by the browser.

These HTML files are stored within a separate directory that Flask knows to look for when processing requests. This directory is named “templates”. The corresponding CSS, JS, and image files that are used to give the HTML its presentation and functionality are stored in a separate directory named “static”, which Flask also knows to look for when processing requests.

Passing Data Between the Web Browser to the Server

In order to pass the user input data from the web interface forms (ex. Name and access level of a newly added personnel) the data is processed using JS and then passed to the Python server using HTTP GET requests. The JS function used has three parameters; first, the URL on the server to pass the data to. Second, the data to be passed, in the form of JSON. Third, the function to be called when the request has been successful.

If data is to be passed back to the web browser, the Python function simple returns a JSON object which is then parsed by the JS. An example of data that would be passed from the

Python server to the web page would be either a confirm or deny that the data has been process properly and as expected.

Digole LCD Driver

The purpose of the Digole LCD is to communicate with the user and give context to the processes running. The necessary steps to configure the Digole LCD for use from within the Python script are found under the comment labeled “Digole Setup and Clear Screen”. These functions define the I2C address with which the LCD will be communicating, it clears the screen of any text left over from previous uses, and displays the roomi name.

From here, the three main functions that the script uses are:

- **digoleClearScreen()** - Sends to the LCD “CL” which is interpreted to clear the screen of any text and images.
- **digoleWriteText(String: text)** - Accepts a string that is to be sent to the LCD and displayed. The string is appended to “TT” which tells the screen to display any of the following characters .
- **digoleWriteCommand(String: command)** - Similar to the write text function, this function accepts a string as a command that is to be sent to the LCD. The difference being, the string is not appended to “TT” so that just the command is sent and is interpreted as such. The two commands that were used for roomi were “TRT” which sends a carriage return to the LCD in order to add line breaks and “ETP99” which adjusts the position of the cursor where text will be written.

PN532 NFC Driver

The PN532 NFC reader/writer is used by roomi to process the NFC cards as they are brought within range of the sensor. The necessary steps to configure the Digole LCD for use from within the Python script are found under the comment labeled “NFC Setup”. These functions define the I2C address with which the LCD will be communicating, and configure the device for use with roomi.

From here, the main function that the script uses is:

- **read_passive_target(timeout)** - Tells the device to look for NFC cards that come within range of the device. Returns the unique ID that is encoded on the card

Solenoid Driver

The software that drives the solenoid valve, that is used to simulate a latching mechanism, is the simplest of the three. In order to control the position of the solenoid either 5V or 0V will sent through its corresponding GPIO pin. When the solenoid’s coil receives electrical potential, it induces an electromagnetic field for which the metal pin is attracted to and is pulled in. Once the coil has been de energized, the magnetic field dissipates and the coil is returned to its open position. The two GPIO functions that are used to access the solenoid’s pin are as follows:

- **GPIO.output(int: pin, GPIO.HIGH)** - Sets the GPIO pin specified by the integer to output a high signal (5V)
- **GPIO.output(int: pin, GPIO.LOW)** - Sets the GPIO pin specified by the integer to output a low signal (0V)

Database Communication

As mentioned in the application's database section of this report, the database used for roomi is Google's Firebase platform.

Connecting to Database from the Script

In order to connect the Python script to the database the Python module pyrebase is used. This gives accesses to a variety of methods necessary for communication with Firebase. The following steps are taken to link the script with the database:

1. A JSON object is created holding the necessary data to connect to the database with the script. Similar to how the "google-services.json" file held the data for the application's connection. This information can be found from within the aforementioned JSON file.

```
config = {
        "apiKey": String,
       "authDomain": String,
       "databaseURL": String,
        "storageBucket": String
}
```

2. **firebase = pyrebase.initialize_app(JSONObj: config)** - returns an instance of the database
3. **db = firebase.database()** - returns a reference to the database using the reference created in the previous step

Reading Database Values

In order to read values from a specific child node from within the database, the database reference is used from the connection section outlined previously. Using this reference, the following function is called in order to read a certain value from the child node:

- **db.child(String: childNode).get().val()** - Returns the value associated with the given child node. In order to navigate through further child nodes, more calls to **child()** can be made passing the associated child node name before calling the last two **get()** and **val()** functions

Writing Database Values

As in reading values from the database, the same database reference will be used to write values. The following steps are taken in order to write new values to the database:

1. A JSON object is created to hold the values needing to be added to the database with the structure as follows:

```
JSONobj = {
        "valueOne": value,
        "valueTwo": value,
}
```

2. **db.child(String: childNode).set(JSONobj)** - Adds a new child node with the navigated child node as its parent node. Similar to reading, more **child()** calls can be made before the **set()** call in order to navigate further in to the nodes of the database

Web Interface Functionality

Main Page

The associated web page for this piece of functionality is the document labeled “index.html”. Here users are presented with three options, in the form of buttons, to access the other three portions of the web interface. Simply clicking a button will navigate to the associated functionality and request the webpage from the server.

Assign Room

The associated web page for this piece of functionality is the document labeled “assign_room.html”. Here users are presented with a form with two inputs for each the name and access level of the room they wish to assign the Raspberry Pi to and two buttons, one for submitting the form, and one for canceling and returning to the main page of the web interface.

When the submit button is clicked by the user the first thing to run will be the JS. The associated JS document is the one labeled “assign_room.js” and as mentioned in previous sections can be found from within the directory labeled “static”. Here the inputs are validated, making sure they are not empty, and contain only letters and numbers through the use of regular expressions. After the data has been validated, the two values (name and access level) are passed to the server using an HTTP GET request. The route requested for this step is the one labeled “/add_room_to_db”.

This python function extracts the values passed to it from the JS method using the function **request.args.get(String: JSONKey, String: defaultValue)**. Once these values are extracted the Raspberry Pi’s MAC address is obtained and used as a string value to give the database entry a unique parent key. The values are then sent to the database following the

steps outlined in the web interface database section and a confirmation message is displayed on the LCD with the information entered and the Raspberry Pi's MAC address. The web interface is then redirected to the home page.

Add Personnel

The associated web page for this piece of functionality is the document labeled "add_personnel.html". Similar to the "assign room" functionality, users are presented with a form with which to enter both the name and access level of the person they wish to add to the database and assign an NFC card to and both a cancel and submit button. Once the submit button has been clicked, the form data is validated and the JS handler will start a 10 second timer associated with a progress bar on the webpage. The route requested "/poll_for_card" is also called through an HTTP GET request.

Within this 10 second timer, users are tasked with tapping the card in which they are assigning to a personnel in order to extract its unique identifier. If a card is seen by the NFC reader and an ID can be found then it is returned to the JS function.

From here the JS will the pass all three pieces of information to the server to be added to the database (name, access level, and card id). The route requested is "/add_personnel_to_db" and from within this function the values are extracted using **request.args.get(String: JSONKey, String: defaultValue)**. Once these values are extracted the unique card identifier is used as a string value to give the database entry a unique parent key. The values are then sent to the database following the steps outlined in the web interface database section and a confirmation message is displayed on the LCD with the information entered. The web interface then redirects to the main page.

Normal Running Mode

The associated web page for this piece of functionality is the document labeled “running.html”. This page simply displays a message that roomi has secured the assigned room and users are presented with a stop button to cancel the security process and return to the main page. Most of the functionality for this page is handled by the server. Except for, a handler to start the security process when the page is loaded, and a handler to stop it when the cancel button is clicked.

When the page is loaded an HTTP GET request is sent to the server with the following route “/start_normal_mode” after which a global boolean variable is toggled to true. When the page’s cancel button is clicked, another HTTP GET request is sent to the server with the route “/stop_normal_mode” where the server will toggle the global boolean variable false.

The variable is used from within a threaded process that has been started with the server and continues to run until the server is stopped. Inside this thread is a while loop that checks the status of the global boolean variable and either loops if it’s true or doesn’t if it’s false.

Within this while loop is the basic security functionality designed for the security process of roomi. The loop waits for a card to be read from the NFC reader and then triggers additional processes when a card ID is found. Using the ID extracted from the NFC card, the personnel’s access level is retrieved from the database. Using the MAC address of the Raspberry Pi, the access level of the room is retrieved from the database. These values are then compared and based on this comparison, the solenoid will be opened simulating “access granted” or it does not, simulating “access denied”.

While roomi is in “normal mode” and the room is secured, a message is displayed indicating such on the LCD and the name and access level of the room. When a user has been permitted access to the room, the LCD displays the name of personnel the card is assigned to, their access level, and a date/time stamp of their card tap. The solenoid is then charged and simulates an opening latch for 3 seconds and then closes. If a user is denied access a message on the LCD is displayed indicating such. This process will run until the cancel button is clicked on the web interface and the global variable is toggled false.

Enclosure

To develop the enclosure, there are are some critical skills and tools you will need. Firstly, you will need to be experienced with SolidWorks and its unique workflow. Next, you will need to have access to a laser acrylic cutter or something equivalent. The enclosure is made from 3mm clear plastic which provides the structural integrity and aesthetic housing for the unit. To assemble the case you will also need a variety of screws and glues to keep the enclosure together.

Design

The enclosure is designed in SolidWorks 3D which is a proprietary software and pricing ranges for a variety of use cases. More information is available at my.solidworks.com/ and should be your starting point for any troubleshooting as well.

Once you acquire/install SolidWorks, you will be required to have a basic understanding of its tools and layout. To learn more about SolidWorks, you should always start with the official documentation provided at my.solidworks.com/ but there are always a variety of sites dedicated to learning new software.

Located in the Roomi Github repository, there is a folder labeled Enclosure which contains all the necessary files for SolidWorks. Our design divided up all our components into unique individual parts allowing us to prototype and test in the virtual world before consuming resources in the real world. This also allows easy modification for future upgrades without requiring a complete rebuild.

The procedure for designing the case was to initially measure and design each of the different hardware components (PCB, Digole LCD, Raspberry Pi, PN532 NFC). From here we added each of the components to a single work space to arrange them as they would be inside the case. Finally, each of the 6 sides of the case were modeled individually and added around the assembled components to form the case. Additionally, two tabs were designed to be glued to the bottom panel of the case with which will be connected with screws to the 5 other panels of the case.

Laser Cutting

Once you generate the full design, you have to laser cut it out on acrylic. There are a variety of ways to do this. You can digitally submit it on a online service that delivers the print to your home. Sites like sculpteo.com/en/lasercutting/ offer competitive pricing and a variety of shipping methods. There are plenty of companies that do this so you will have to research your options. There are also physical vendors in all major cities that can help you print the design. Again, you will have to do research and decide what's best for you.

Assembly

Once your components are cut out, you will have to assemble the enclosure. The case provided is labelled on the top with a Roomi tag and an ID card so the user knows where to place their NFC card for detection. There is also a large opening where the display can be viewed by the user. Start assembling by mounting the digital display with 4 screws and 4 nuts to secure it to the acrylic panel.

The next step is to mount the Pi to the bottom panel. There are 4 distinct holes designed to fit a Raspberry Pi Model 3 B+. You will need 4 micro nuts and bolts to secure it.

There are two side panels that act as the middle point of the case. The laser cutting files contain 4 small panels that need to be glued to the bottom and side panels which will then fastened with nuts and bolts with the tabs attached to the bottom of the case. The images in this section will help you as a point of reference to assembling the enclosure.

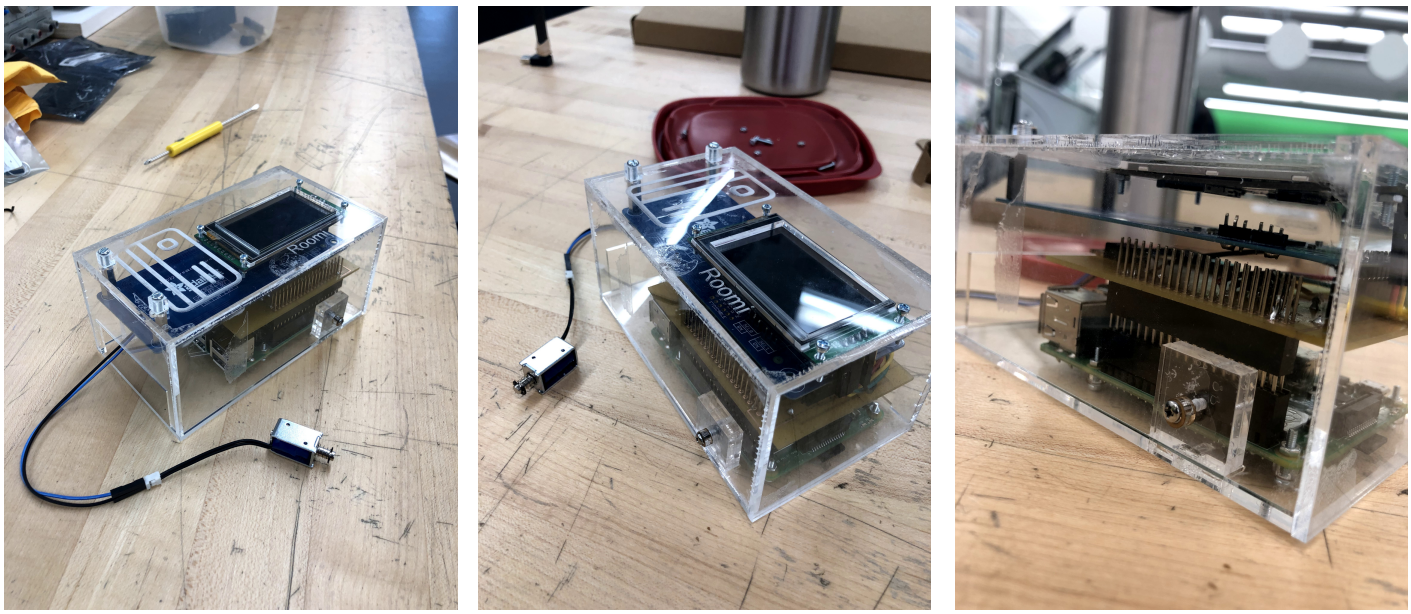

Figure 16 - Assembled Case

All other panels will have to be glued together along the edges using concrete glue and claps to secure it for overnight drying. Once the panels are dried, the PCB and NFC Shield can be combined to bring it all together. You will then secure the end of the shield to the top panel by using spacers and screws. The last step is to connect all the wires and power supplies.

Once you complete this process, you should be able to move the unit around without fear of damaging the internal components. Keep in mind that you should follow the recommended waiting time for the specific glue you used.

Conclusion

This project combined the sum of our skills and knowledge gained through the first five semesters of the Computer Engineering Technology program at Humber College. As a result, the final product encompasses a door latching mechanism with NFC/RFID reader, graphical display, cloud storage database, and an accompanying Android application. The goal was to give users a modern approach to security systems for residential and commercial dwellings. This project demonstrated our extensive knowledge of both hardware and software. This technical report outlined the complete process of research, construction, testing, and implementation of each aspect of the project.

Bibliography

Resources

- Roomi Github Repository - https://github.com/roomi-develop/roomi
- CENG 355 Weekly Breakdown Repository – https://six0four.github.io/ceng355/
- Raspbian Operating System - https://www.raspberrypi.org/downloads/raspbian/
- The Ontario Association of Certified Engineering Technicians and Technologies - https://www.oacett.org/
- Android Developers Documentation - https://developer.android.com/
- Firebase Developer Console & Documentation - https://firebase.google.com/
- Fritzing - http://fritzing.org/home/
- Solidworks - https://www.solidworks.com/

Appendix

Glossary of Terms

- HTTP - HyperText Transfer Protocol is a standard form of communication over web based technologies and is used to pass information from the web browser and server
- XML - Extensible Markup Language is used to design graphical layouts and structures of elements
- HTML - Hypertext Markup Language is used to define the structure and design of user interfaces for web applications
- CSS - Cascading Style Sheets is a styling language used to describe the presentation of the HTML on a webpage
- JS - Javascript is a prototype-based object-oriented programming language that is used to program the front-end functionality of web pages on the client
- URL - Uniform Resource Locator specifies a location on a computer network or web server
- Flask - A Python framework used to run as a web server in order to server web pages to browsers
- GPIO - General Purpose Input/Output is the main line of communicating with hardware connected to the Raspberry Pi. A total of 26 pins with seperate functionalities
- LCD - Liquid Crystal Display is used to visually represent text and images on a physical screen
- NFC - Near Field Communications, short-range wireless connectivety standard
- RFID - Radio Frequency Identification, uses electromagnetic fields to automatically identify and track tags attached to objects

- PCB - Printed Circuit Board, platform for which components will be soldered to and to connect
- I2C - I squared C, serial 2 wire communication protocol
- UART - Universal Asynchronous Receiver/Transmitter communication interface
- SPI - Serial Peripheral Interface communication interface
- JSON - JavaScript Object Notation, lightweight data-interchange format
- SQL - Structured Query Language (database language)
- NoSQL - Not Only SQL, standard language for storing manipulating and retriving data in databases
- REST - Representational State Transfer
- OS - Operating System, software that supports the basic functions of a computer
- GNU - GNU not Unix, open source UNIX based OS
- CPU - Central Processing Unit, computer’s electronic circuitry that carries out instructions

