|  |
| --- |
| Robert Dinh |

NFC Inventory Management System

Status

/1 Hardware present?

/1 Title Page

/1 Declaration of Joint Authorship

/1 Proposal (500 words)

/1 Executive Summary

# Declaration of Joint Authorship

We, Jonathan Luong, Robert Dinh, and Collin LeDonne, confirm that this work submitted is the joint work of our group and is expressed our own words. Any uses made within it of the works of any other author, in any form (ideas, equations, figures, texts, tables, programs), are properly acknowledged at the point of use. A list of the references used is included. The work breakdown is as follows: Each of us provided functioning, documented hardware for a sensor or effector. Student A provided the Adafruit HD44780 LCD. Student B provided the Adafruit PN532 NFC/RFID Controller. Student C provided the Adafruit VCNL 4010 Proximity Sensor. In the integration effort Collin LeDonne is the lead for further development of our mobile application, Robert Dinh is the lead for the Hardware, and Jonathan Luong is the lead for connecting the two via the Database.

# Proposal

We have created a mobile application, worked with databases, completed a software engineering course, and prototyped a small embedded system with a custom PCB as well as an enclosure (3D printed/laser cut). Our Internet of Things (IoT) capstone project uses a distributed computing model of a smart phone application, a database accessible via the internet, an enterprise wireless (capable of storing certificates) connected embedded system prototype with a custom PCB as well as an enclosure (3D printed/laser cut), and are documented via this technical report targeting OACETT certification guidelines.

Intended project key component descriptions and part numbers:  
Development platform: Raspberry Pi 3 B+  
Sensor/Effector 1: Adafruit HD44780 LCD  
Sensor/Effector 2: Adafruit PN532 NFC/RFID Controller  
Sensor/Effector 3: Adafruit VCNL 4010 Proximity Sensor

We will continue to develop skills to configure operating systems, networks, and embedded systems using these key components to create a functional device that will trigger a proximity sensor to enable a RFID controller in read mode and turn on an LCD display. This device will be used to read an NFC NDEF message from a phone application to identify the student and their requested parts. The Android application that we will develop will allow students to select the parts they need and the application will also generate the NDEF message to be scanned over NFC using dedicated protocols. Also, a web browser application will also be developed mainly to monitor requests and inventory. It will also act as an alternative method for requesting parts. These three components will create an inventory management system that will benefit both the students and the staff. It will seamlessly allow students to request items and pick their parts up without the hassle of purchasing bags and RFID tags for individual students.

Our project description/specifications will be reviewed by, Professor Kristian Medri and Vlad Porcila (Both within the Faculty of Applied Sciences and Technology at Humber College), ideally an employer in a position to potentially hire once we graduate. They will also ideally attend the ICT Capstone Expo to see the outcome and be eligible to apply for NSERC funded extension projects. This typically means that they are from a Canadian company that has been revenue generating for a minimum of two years and have a minimum of two full time employees.

The small physical prototypes that we build are to be small and safe enough to be brought to class every week as well as be worked on at home. In alignment with the space below the tray in the Humber North Campus Electronics Parts kit the overall project maximum dimensions are 12 13/16" x 6" x 2 7/8" = 32.5cm x 15.25cm x 7.25cm.

Keeping safety and Z462 in mind, the highest AC voltage that will be used is 16Vrms from a wall adapter from which +/- 15V or as high as 45 VDC can be obtained. Maximum power consumption will not exceed 20 Watts. We are working with prototypes and that prototypes are not to be left powered unattended despite the connectivity that we develop.

# Executive Summary

The purpose of this document is to lay out the requirements and specifications for the hardware and the client-side application for the NFC Inventory Management System. This system will work in conjunction with an online management interface to maintain inventory in an efficient and organized manner. This product will mainly be developed as a Parts Crib solution at Humber College North Campus but it can also be utilized by other companies for their own purposes. This product will allow the client’s customers(students) to request items through a mobile or online interface and sign the items out with their phones as a form of identification through NFC. This removes the hassle of providing a card as a form of ID and eliminates the use of RFID tags that have to be individually programmed for each person. All the data that needs to be organized and managed to make the system work flawlessly, will be handled by the database with an online interface, which allow clients to audit their inventory and manage their customer’s information.

Contents

[Declaration of Joint Authorship 3](#_Toc34124120)

[Proposal 5](#_Toc34124121)

[Executive Summary 7](#_Toc34124122)

[List of Figures 13](#_Toc34124123)

[1.0 Introduction 15](#_Toc34124124)

[1.1 Product Scope 17](#_Toc34124125)

[1.2 Requirements and Specifications 18](#_Toc34124126)

[2.0 Background 19](#_Toc34124127)

[3.0 Methodology 21](#_Toc34124128)

[3.1 Required Resources 21](#_Toc34124129)

[3.1.1 Parts, Components, Materials 21](#_Toc34124130)

[3.1.2 PCB and Enclosure Design 23](#_Toc34124131)

[3.1.3 Tools and Facilities 24](#_Toc34124132)

[3.1.4 Shipping, duty, taxes 26](#_Toc34124133)

[3.1.5 Time expenditure 27](#_Toc34124134)

[3.2 Development Platform 28](#_Toc34124135)

[3.2.1 Mobile Application 28](#_Toc34124136)

[3.2.2 Image/firmware 38](#_Toc34124137)

[3.2.3 Connectivity 39](#_Toc34124138)

[3.2.4 Breadboard/Independent PCBs 41](#_Toc34124139)

[3.2.4 Printed Circuit Board 48](#_Toc34124140)

[3.2.5 Enclosure 48](#_Toc34124141)

[3.3 Integration 48](#_Toc34124142)

[3.3.1 Enterprise Wireless Connectivity 49](#_Toc34124143)

[3.3.2 Database Configuration 49](#_Toc34124144)

[3.3.3 Security 49](#_Toc34124145)

[3.3.4 Testing 49](#_Toc34124146)

[4.0 Results and Discussions 51](#_Toc34124147)

[5.0 Conclusions 53](#_Toc34124148)

[6.0 References 55](#_Toc34124149)

[7.0 Appendix 57](#_Toc34124150)

[7.1 Firmware code 57](#_Toc34124151)

[7.1.1 main.c File Reference 57](#_Toc34124152)

[Functions 57](#_Toc34124153)

[Detailed Description 58](#_Toc34124154)

[Function Documentation 58](#_Toc34124155)

[main.c 58](#_Toc34124156)

[7.1.2 NDEFData.c File Reference 60](#_Toc34124157)

[Functions 60](#_Toc34124158)

[Variables 60](#_Toc34124159)

[Detailed Description 60](#_Toc34124160)

[Function Documentation 61](#_Toc34124161)

[NDEFData.c 61](#_Toc34124162)

[NDEFData.h File Reference 65](#_Toc34124163)

[Detailed Description 65](#_Toc34124164)

[NDEFData.h 65](#_Toc34124165)

[7.1.3 Presence.c File Reference 66](#_Toc34124166)

[Functions 66](#_Toc34124167)

[Detailed Description 66](#_Toc34124168)

[Function Documentation 66](#_Toc34124169)

[Presence.c 67](#_Toc34124170)

[Presence.h File Reference 69](#_Toc34124171)

[Data Structures 69](#_Toc34124172)

[Typedefs 69](#_Toc34124173)

[Detailed Description 69](#_Toc34124174)

[Presence.h 69](#_Toc34124175)

[7.1.4 Lcd.py File Reference 70](#_Toc34124176)

[7.2 Application code 71](#_Toc34124177)

# List of Figures

[Figure 1. Main Activity 29](#_Toc34736209)

[Figure 2. Create Account Screen 29](#_Toc34736210)

[Figure 3. Login Screen 30](#_Toc34736211)

[Figure 4. Home Screen 31](#_Toc34736212)

[Figure 5. Navigation Drawer 31](#_Toc34736213)

[Figure 6. Items Category Screen 32](#_Toc34736214)

[Figure 7. Items Name Screen 33](#_Toc34736215)

[Figure 8. Item Description Screen 33](#_Toc34736216)

[Figure 9. Cart Screen 34](#_Toc34736217)

[Figure 10. Orders Screen 35](#_Toc34736218)

[Figure 11. Data Visualization 36](#_Toc34736219)

[Figure 12. LCD Schematic 42](file:///D:\Humber\2019_20\CENG355Project)\Git\CENG355NFCSystemsProject\Documentation\Technical_Report.docx#_Toc34736220)

[Figure 13. Proximity/Luminosity Sensor 43](#_Toc34736221)

[Figure 14. NFC Controller Schematic 44](#_Toc34736222)

[Figure 15. Schematic for all devices wired together 45](#_Toc34736223)

[Figure 16. Breadboard Layout. VCNL4010(Top), LCD(Bottom), PN532(Right) 46](#_Toc34736224)

[Figure 17. PCB Design 49](#_Toc34736225)

[Figure 18 Top of PCB 49](#_Toc34736226)

[Figure 19 Bottom of PCB 50](#_Toc34736227)

[Figure 20 Top of PCB after soldering 50](#_Toc34736228)

[Figure 21. Bottom of PCB after soldering 51](#_Toc34736229)

[Figure 22. Completed PCB with connected components 52](#_Toc34736230)

# Introduction

Report

/1 Hardware present?

/1 Introduction (500 words)

/1 Scope and Requirements

/1 Background (500 words)

/1 References

The idea for this project is to create a solution to the current sign in and out feature at the parts crib at Humber College. The current system has the user download buggy software that is not very intuitive and rather difficult to use without instruction. One must download the software from a dropbox and then the software may need to be loaded several times to work properly.

Our goal is to create a solution to this problem by creating a smart phone application that uses HTTP protocol to send and receive data from a server along with hardware to make the process of borrowing parts easier. The server currently being used is the Humber Apollo Server (apollo.humber.ca) under a student’s profile for testing, but can be moved elsewhere for deployment elsewhere. The mySQL database on the server has 3 dedicated tables for accounts, inventory, and transactions. Each table has one field or more that will be able to relate to another table.

The Android application is designed to interact with the database. It has a graphical user interface to allow students to create a user, sign in, request items and to view previous requests. The process has the user queue the parts they require from the mobile application, which operates like an e-commerce store, which will then notify the employee at the parts crib via web application. When the user queues up parts, this will create a unique NFC code for the user to scan at the parts crib. The employee will then prepare the order for the user to pick-up. When the user arrives at the parts crib, they will scan their phone on the NFC reader which will then update the status of the unique transaction on the database that the employee can see. This will then update the order status on the database until the user scans their phone again on the NFC reader when items have been returned.

The web application will give the employee at the parts crib the ability to release orders, see the contents of the orders, and any relevant information requiring the order. The hardware includes a VCNL 4010 proximity sensor, Adafruit PN532 NFC Sensor, and a 16x2 HD44780 LCD Display. The NFC reader is responsible for receiving the information from the users phone, the LCD display is responsible for displaying the information from the NFC reader to the user, and the proximity sensor will turn up the brightness of the LCD screen when the user puts their phone near the NFC reader. We will be designing a PCB with the intention that the sensors will be placed next to each other to reduce the size of the case. All the components will fit into a case that will be laser cut and the layout of the components will be important so it will be easier to use and understand.

## 1.1 Product Scope

The purpose of our project is to create an inventory management system that permits equipment provisioning for the Parts Crib that provides a friendly user experience for students and employees. The process will interact with a database through requests and finalize transactions through an interaction between a user’s mobile device and our NFC reader located at the crib. The system will implement functionality from the current system as well as provide features for both parties to stay informed regarding items pertaining to them to ensure users are updated accordingly per use.

The database will contain user account information and inventory that range from various tools and equipment. The goal of our project is to create an easy experience for the users of our product and intuitive enough for new users so it is simple and convenient for students to check out equipment.

Our IoT capstone project will implement a distributed computing model comprised of a smart phone application, and a database accessible via PHP scripts using a web browser or our NFC reader embedded system prototype with a custom PCB along with enclosure. The process will be documented via an OACETT certification acceptable technical report.

The project will be limited to the database we have created and will only be tested through simulation of the process. Requiring the Parts Crib for our test is not necessary if we are able to simulate high volume periods. Also, because the application will have personal information of the user there are security clearances that we do not have but we will be implementing secure practices in the handling of our account database involving unique IDs and hashed passwords (Hasing Security, 2019).

## 1.2 Requirements and Specifications

**Application**

* developed for android mobile devices starting at API 21
* requires internet connection over WiFi
* mobile device must be NFC-equipped

**Hardware**

* Requires wall outlet for power
* Wi-Fi to communicate with database
* Raspberry Pi 3 B+ with WiFi and I2C enabled

**Database**

* MySQL Database to read/write/store data
* Requires use of PHP scripts and/or web browser to communicate to the database stored on student account in Humber College servers

# Background

Radio frequency identification, also known as RFID, is the utilization of tags with integrated chips and coiled wire that can be activated using radio signals to retrieve their unique identification number or other information. Tags that are often passive, do not require a power source as its main source is from a reader that sends out an electromagnetic signal (Butterfield & Szymanski, 2018). These RFID tags are commonly found in everyday activities such as tapping a membership card to get into a car wash, to make a payment for food or to get a bus/train(transit).

Near Field Communication, also known as NFC, is a subset of RFID. NFC operates at a frequency of 13.56MHz and transfers data at a rate of 106k-424k bits per second (Butterfield & Szymanski, 2018). NFC capable devices can be switched to emulate passive or active states, such as smartphones to read and write to each other. To standardize the flow of information from device to device, NXP Semiconductors, Sony, Nokia and Philips have joined together to create protocols that are found under the NFC Forum name (Vanderkay, 2004)Currently, NFC Forum has 5 different specifications for tags, each unique for their own purposes. For this project, the Type 4 Forum Tag (ISO/IEC 14443) specification will be used to communicate between the hardware and an Android mobile device which. This involves 4 parts: physical characteristics, radio frequency power and signal interface, and initialization and anti-collision (Sabella, 2016). The Type 4 tag was selected due to the ability to hold large amounts of data in NFC Data Exchange Format (NDEF). This will be furthered explained in conjunction with Application Protocol Data Unit(APDU) commands specified in ISO/IEC 7816-4 (Organization, security and commands for interchange) to show how one device can access data on another device .

Modern day smartphones, capable of doing many tasks, have been very convenient for everyday users. With the ability to perform host card emulation on these Android devices, contactless payment and other data transactions are made possible through NFC from one device to another. This type of technology can be applied to modern day inventory management problems such as the Parts Crib. The Parts Crib currently uses RFID tags that are individually scanned in and out for each incoming student, costing precious time and money. Having a contactless transaction of information with a mobile device will allow the Parts Crib to operate efficiently and effectively without requiring any extra manpower. Since students carry around their phones every day, it is simpler to just tap their phones than to rummage through purses and wallets. This system will allow students to securely transfer their information to the Parts Crib for easy access.

# 3.0 Methodology

## 3.1 Required Resources

Report

/1 Parts/components/materials (500 words)

/1 PCB, case (500 words)

/1 Tools, facilities (500 words)

/1 Shipping, duty, taxes (250 words)

/1 Working time versus lead time (250 words)

### 3.1.1 Parts, Components, Materials

The NFC System Management system revolves around a hardware device that will sense an incoming device, scan NDEF messages containing data via NFC and display the status of the device. The internal hardware component has 4 main devices with supporting electronics parts to deliver the functionality.

The first main component is a Raspberry Pi 3 B that revolves around a system on a chip and has GPIO pins for power and data transfer. The device’s main purpose is to gather data from the sensors and display data to the effectors. To manage and handle the data, it will run scripts to interpret the data and send it off to a database server for further processing.

The second device is the Adafruit PN532 NFC Controller that is capable of many functions but will mainly act as an NFC reader to get access to the NDEF messages sent by an Android mobile device. For the device to power up, it requires a connection to the 3.3V and ground lines from the Raspberry Pi. To communicate with the Raspberry Pi, it needs to be wired in the I2C configuration which requires connections to the Serial Data Line (SDA) and a Serial Clock Line (SCL) line.

The third device is the Adafruit HD44780 16x2 LCD Display that will be used to output messages to the user based on their interaction with our system. After completing a transaction through the device above the output will either display a success or instructions/error code based on the action. The device requires a 5V and ground connection to the Raspberry Pi. To communicate to the Pi using I2C a PCF8574P IC I/O expander chip is used to communicate to the device, this requires 3.3V and ground from the Pi as well as connections to the SDA and SCL lines. To control the contrast and backlight of the LCD there are two 10k ohm potentiometers wired to the LCD as well as a 10k and 4k7 ohm resistor for better control over the lighting options. This setup also provides protection to the LCD so less voltage is being input to the device.

The fourth device is the Adafruit VCNL 4010 proximity sensor which will be used to for user input to the device. The proximity sensor will be set up in such a way so that when the user puts their phone near the device, it will brighten the LCD screen. The VCNL 4010 has an SCL and SDA pin what are used to transfer data. The values the 4010 returns are unit less but need to be converted. The closer you get to the sensor, the higher the proximity value and the lower the ambiance. The sensor’s IR led operates from 3.3 to 5 volts and is not very large as it is smaller than a quarter.

### 3.1.2 PCB and Enclosure Design

When the hardware components have been wired and tested on a breadboard, a printed circuit board (PCB) will be designed in Fritzing with the Adafruit library. It will be designed so that it can be a connecting point for all the main devices and its supporting components. It will also be designed for modularity with troubleshooting in mind so that nonfunctional components can be easily accessed and replaced.

The PCB design will contain holes for female header pins to be soldered on. One 2x20 40-pin female header will be used to connect the PCB to the raspberry pi’s GPIO pins. Also, a 6-pin female header, 9-pin female header, and a 16-pin header will be used for a proximity sensor, an NFC reader and an LCD display respectively. Each header that is dedicated to each device will have lines providing power and I2C communication running from the 40-pin header. To route these lines, vias will be created so that the lines will not overlap, allowing them to be placed both on top and bottom of the PCB. Any other supporting electrical components such as resistors and transistors will have holes dedicated to them as well. For the PCB to be completely secure and held in place, 4 mounting holes around 2.5-3mm in diameter will be put in so that screws can threaded through to hold the PCB to the nylon standoffs that will be put under it. When the PCB is finished in its designed phase, the files will be sent off to a local prototype lab at Humber College to be printed.

When the PCB is being sent off to be printed, the design files will then be imported into AutoCAD and Inventor to design the case to surround the internal hardware. The external design will be boxy in shape similar to that of a rectangular prism.

The base of the enclosure will be based on the dimensions of the Raspberry Pi. Holes with standoffs will be created so that the Raspberry Pi does not rest on the soldered joints but rather on elevated pads. This will prevent the device from overheating and possibly melting the plastic underneath it. The holes will be countersunk so that the screws to mount the Raspberry Pi to the case will be not exposed.

The sidewalls of the device will have holes created to give access to the Raspberry Pi’s micro USB for power, HDMI, USB ports, Ethernet port, and audio port. Having access to these ports allows easy access for future development and mainly for troubleshooting the device. Extra holes will be put in place to allow the Raspberry Pi’s LED and the NFC reader’s LED to check on the status of other devices.

The removable top-lid of the device will be designed to have holes for the LCD Screen and a very small hole for the proximity sensor. 2.5-3mm mounting holes will be put in place for each device so that they do not shift around during transportation or during a transaction.

### 3.1.3 Tools and Facilities

The initial setup of our final product will require the integration of all our devices that we will build on a breadboard and interact with through a computer to set up necessary files to run our program. This will be mostly done in the J232 Lab room at Humber College. Use of the digital multimeter in the room is important for ensuring the right voltage levels are present in specific parts of the build. Schematics, breadboard designs, and PCB design will all be created on Fritzing before each phase is actually made.

With past experience it is noted important in PCB design to ensure leads at each connector is facing a certain way so the solder matches the same side. This makes soldering for each connector onto the board a lot easier to do.

After completing the tests required for our devices to all work on the same platform together, the custom PCB (Printed Circuit Board) and enclosure will be designed and created through the facilities at Humber College.

All of our past custom PCBs were created after submitting our designs to the Prototype Lab at Humber College. Our past enclosures were split between submitting a design for laser-cutting and 3D printing at the same location. Through comparing the pros and cons of both methods we plan on using the same methods to have our final product’s PCB and enclosure created at the Prototype Lab with the enclosure method leaning towards a heavier laser-cut design. Preference is over a full laser-cut enclosure however through 3D printing the portions of the enclosure requiring mounted screws can be created with a better design.

After the PCB is created the assembly will take place in the J232 Lab room. This facility provides soldering stations with fume extractors to provide user safety. This room will be an important factor towards the assembly step of the PCB as it is an easy access point to being able to solder our PCB and in safe work conditions. A long with the facility the use of pliers and wire cutters are required to prepare fine wire, shape pieces such as connectors, and trim the smaller pieces that will be soldered onto the PCB (example: resistors). Safety goggles will be needed for any worker near the solder work to prevent any exposure of chemicals to the eye, and the fume extractor will lower the risk of inhaling of any chemicals.

Through past solder work, the digital multimeter was found to be important in this step as well as it helps ensure connections are made on the PCB as well as be able to test voltage levels for the devices when it is assembled.

After this phase of testing is completed we are able to attach our PCB onto the Raspberry Pi to ensure the devices work and can interact with our microprocessor. The testing here will be similar to the breadboard testing however we will also confirm the size dimensions with the product as a whole and determine the finalization for sizes and designs towards the enclosure.

Lastly enclosure assembly will require some adhesive for some portions and the use of a screwdriver to add the necessary screws and nuts to bring the pieces of the enclosure together with the final product fastened to it.

### 3.1.4 Shipping, duty, taxes

The main components, PN532 NFC, Raspberry Pi, HDF8574P LCD screen, and the VCNL 4010 proximity sensor, were all obtained mainly from amazon or directly from the manufacturer. The majority of these places do apply taxes but do not charge for shipping or duty. The individual costs of the parts are the following: Raspberry pi was $45.75, the HD44780 LCD screen was $9.95, the VCNL 4010 proximity sensor was $7.50 and the PN532 NFC reader was $71.42. All these combined was $134.62 CAD. With taxes, this brings to total to $152.12 CAD which is 17.50 in taxes. The other smaller components include: the nylon stand-off kit, which was $13.99, the PCF8574P Remote 8-bit I/O expander IC which was $9.95, a 10K resistor which was $0.15, and a 4k resistor which was also $0.15. With these components added it then brings the grand total to $161.86 and $182.90 after tax. This means that the taxes we will be paying will be $21.04 for all our materials. The shipping states that the parts should arrive in 5-7 business days however, in the past, orders were delayed and adjustments had to be made to the project plan in order to ensure that the project would be done on time. For example, when Colin ordered his blue pill, it got delayed by a week and he had to make adjustments to his schedule to ensure he finished it on time. We are prepared to make changes to our schedule in case unforeseen events like delayed shipments take place.

### 3.1.5 Time expenditure

Working time for our project will mostly be done outside of the allocated lab time. The exception to this will be for soldering the PCB and printing and assembling the case. We plan to assemble most of our hardware outside of the lab as we have allocated out workload and set dates to follow. This makes it easier and more efficient as we can get more work done in a shorter period of time since all of the group members will be contributing to different parts of the project at the same time. We plan to use most of the lab time as a meet up to troubleshoot any issues and to make plans for future steps. We will be spending out working time polishing the mobile application, developing the web application, and creating the hardware for the application. The lead time will consist of brainstorming, troubleshooting, and discussing future plans and any possible changes to schedule or overall direction. Working time will also be used to test and debug any issues that might occur during the development of this project. Since we do most of the development outside of lab time, we communicate any issues we have via a group chat and assist each other when possible. Following this work ethic is what we believe to be the best possible way to approach this project in terms of working time and lead time.

## 3.2 Development Platform

The mobile application will be developed using java with Android Studio version 3.5 with API 21. The app will also coexist and communicate with the web server that handles requests through PHP to an SQL database.

### 3.2.1 Mobile Application

The app is currently working as intended and communicates with the server with no issues. The only thing that must be completed on the application is the communication with the web application and polishing the GUI. The home screen on the app (Figure 4) is going to include a recycler view were the parts crib employees can send notifications to users via the web application. The GUI also needs some work as it was made quick and with low priority since we needed the major functions of the app to work. Going further into the semester Colin will be working to polish the GUI to make it more appealing and easier to follow since that was one of the goals. Since the major functions of the app are now operational, polishing the GUI should not be too time consuming since it just requires fixing the XML files. The mobile application is due March the 14th 2020 and there should be no issues getting it done by that date.

The mobile application was made in android studio using API level 21. The current stable version on the application works as intended and just needs to be polished to be more aesthetically pleasing. The app was developed with the intention to make it easy and straight forward to use. The project was divided into three parts; the GUI, the database, and the back end. These parts were allocated to the three group members and work was completed on time. The app has a total of 11 screens and was designed to be easy to navigate. The main activity that will load after the splash screen is the login screen.

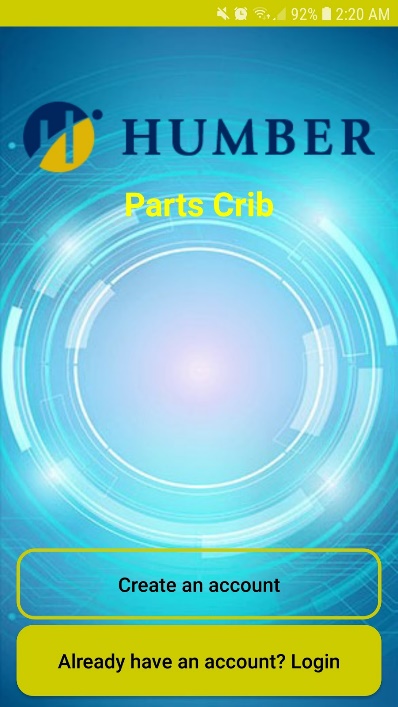


Figure 1. Main Activity

The main activity (Figure 1) was made using two custom buttons and a simple image placed at the top of the screen. The two buttons lead to the two corresponding activities.

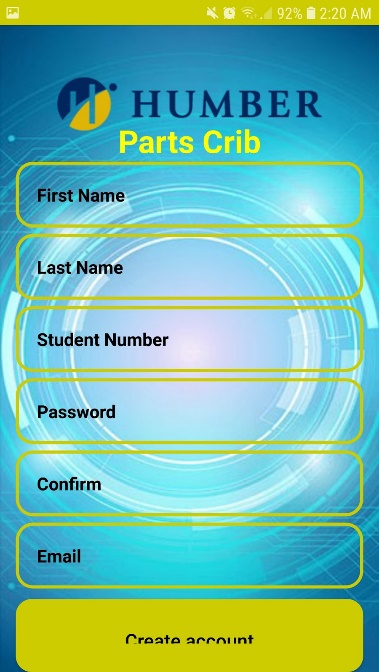
****

Figure 2. Create Account Screen

Figure 2. Create Account Screen

Figure 2. Create Account Screen

The create account (Figure 2) activity was made with custom input boxes and a custom button. This screen was made to mirror other create account pages or screen. In order for an account to be created, all of the fields must have valid information in them. If the user tries to make an account while having no information entered, it will alert the user that they need to fill in the information. This was crucial when creating this screen since the database would be full of incomplete information.

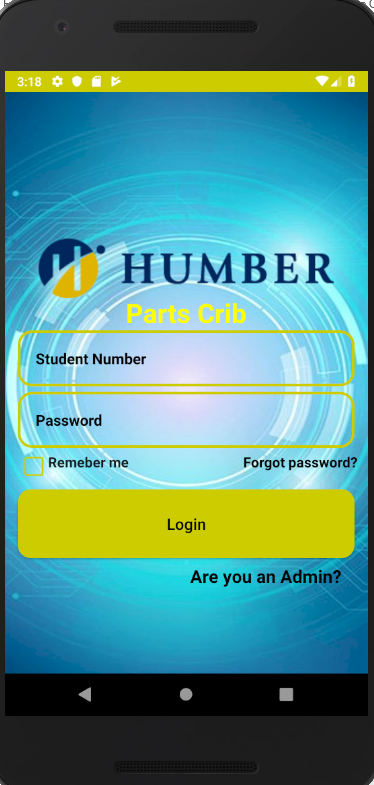


Figure 3. Login Screen

The login screen (Figure 3) was made to also be simple and straight forward. This screen is also where the admin can login once the web application is completed. This screen was made with two custom input boxes, a custom button, a remember me check box, that saves the users login information on the phone’s memory, and an admin login.

****

Figure 4. Home Screen

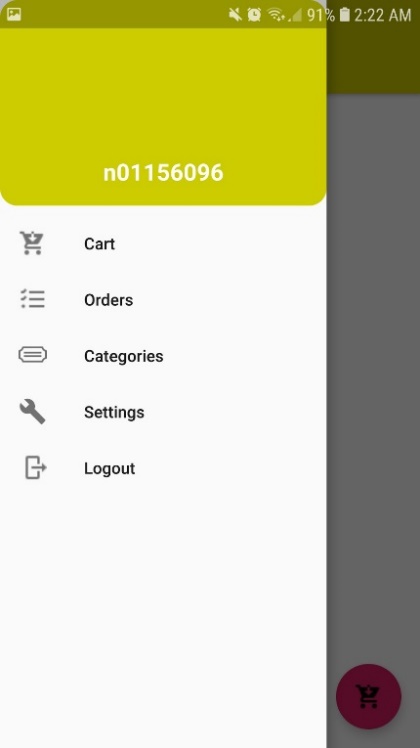
****

Figure 5. Navigation Drawer

The home activity (Figure 4) was made by using a navigation drawer (Figure 5) that lead to the other main activities. This screen will also have notifications from the parts crib. The notifications feature will be implemented when the web application is completed.

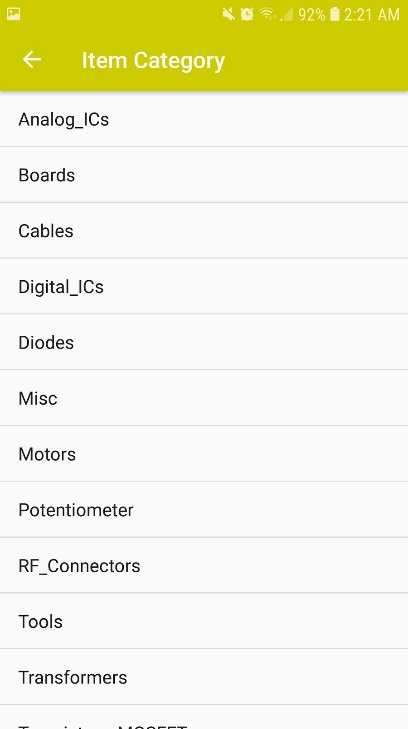


Figure 6. Items Category Screen

The items category (Figure 6) uses a list view that pulls data from the SQL database to display the current items available at the parts crib that the user can borrow from the parts crib. All of screens from the navigation drawer also have a back button on the top left to make navigation easier. This was done by making separate processes in the manifest file.

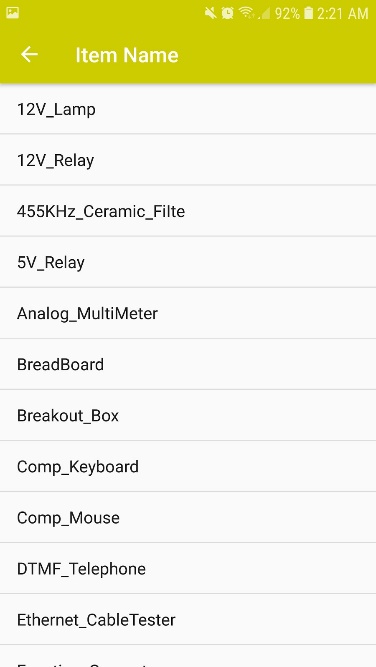


Figure 7. Items Name Screen

The items name screen (Figure 7) is similar to the item’s category screen (Figure 6) as they both use a list view and pull data from the SQL data base to display to the user.

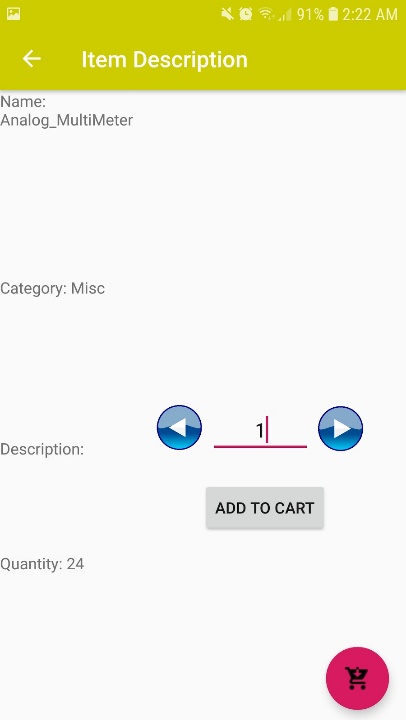


Figure 8. Item Description Screen

The item description screen (Figure 8) can is accessed from pressing on the desired item and it pulls the corresponding information from the SQL server to display to the user.

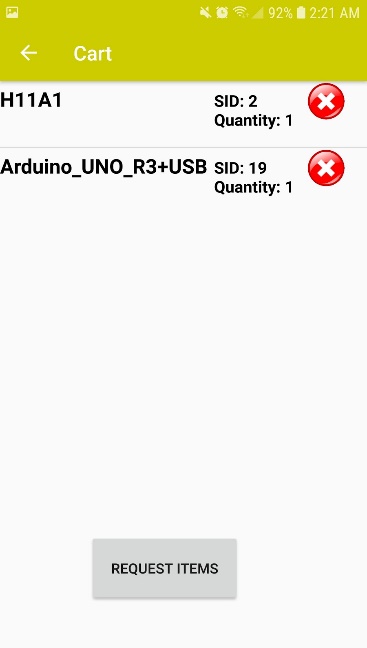
****

Figure 9. Cart Screen

The cart screen (Figure 9) is another list view that gives the user an overview of what they are currently planning to request. The cart screen can be accessed from the navigation drawer (Figure 5) or the floating action button on the home screen (Figure 4). Once the user presses the request button, the parts crib will receive a notification the web application to prepare the order.

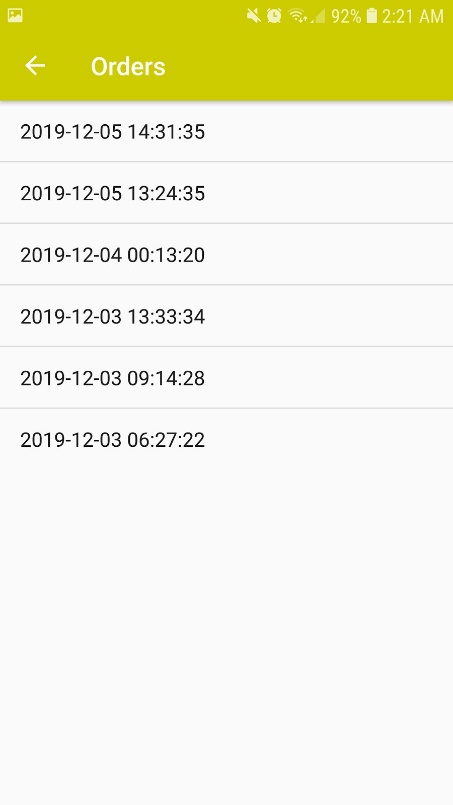
****

Figure 10. Orders Screen

The orders screen (Figure 10) displays all the previous orders that the user has made. This is just a list view that pulls order details from the SQL server. Pressing on the order will display what was in the order and if it has been returned or not.

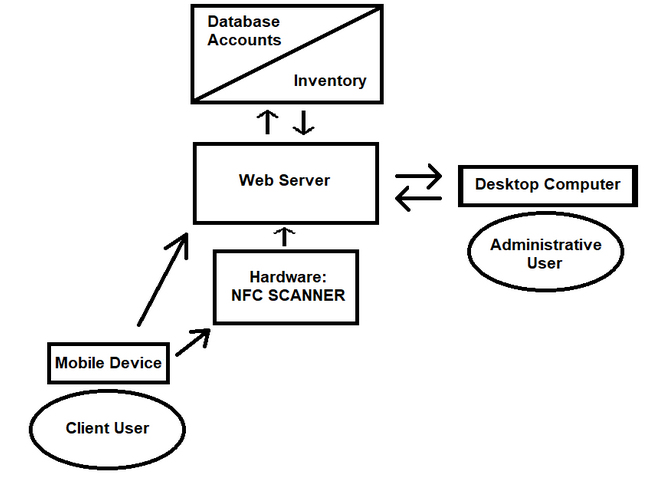


Figure 11. Data Visualization

The data in our app mainly flows from the client’s mobile device to the web server. The web server also communicated with the database in order to pull information to display to the user. For example, when the user wants to view their older orders, the data is sent from the user and then the server pulls the information from the database to display to the user’s mobile device. The user’s mobile device is also going to communicate with the NFC reader at the parts crib. Once the user is ready to pick up or return their order, they tap a button in the app to get into an NFC state to transfer transaction ID to the NFC reader. This transaction ID will be used to look up the order in the database and update the status of the order as picked up/returned.

/1 Hardware present?

/1 Memo by student A + How did you make your Mobile Application? (500 words)

/1 Login activity

/1 Data visualization activity

/1 Action control activity

Include screenshots such as Figure 1. Testing. Progress.

### 3.2.2 Image/firmware

The Raspberry Pi uses the current up-to-date image of Raspbian (Buster) operating system found at their main website. The image was flashed onto the 16GB SanDisk SD card that is used with the Pi. The process was done using an application called ‘balenaEtcher’, this application is used for flashing operating systems onto SD cards and USB drives.

The firmware used for our product was compiled through three different methods used by each member to initially implement their sensor/effector from our hardware projects last semester. Each device was setup to communicate with the Pi via I2C.

The VCNL4010 Proximity Sensor is written in C++ with the source code provided by Adafruit repository for the sensor. The main code used reads the proximity value given by the sensor every 3ms, when the value surpasses 2300 units the program will initiate the set of code for the NFC portion of the task.

The NFC reader PN532 utilizes example code provided by nfc-tools.org as well as their wiringPi libraries and is written in C. Upon activation the NFC sensor begins a process by first detecting the nearby device then selecting the application ID to receive an NDEF (NFC Data Exchange Format) message that contains the transaction ID of the client from their mobile device. If the device does not have the corresponding application ID the process will terminate. When a NDEF message is successfully the program will initiate the script for the LCD with the information given through a system call.

Using a guide provided by Rototron.info as well as the libraries provided by Adafruit the HD44780 LCD screen is able to display strings written to its respective address from the NFC function. The script used to display strings on the LCD was written in Python.

Our goal is to achieve communication between our device and the PHP script that handles our database. We plan to do this over HTML using GET and POST to update the server and complete the transaction process at the Parts Crib when the client has tapped their mobile device. Documentation of the resources used to achieve this will be provided at the repository for reference.

### 3.2.3 Connectivity

The user has the two different methods should the microprocessor need to be accessed for any adjustment or maintenance.

The Raspberry Pi when setup and in Wi-Fi range can be accessed remotely from another terminal using RealVNC Server/Client. Provided in the wpa\_supplicant.conf file on the Pi are the credentials for access to the Humber College Wi-Fi under a student account. With this enabled any user logged into the RealVNC team on the client application can access the device remotely. Internet connection can also be established by connecting an Ethernet cable between the Pi and any networking device that has connection to the internet.

If internet connection cannot be established the Pi can also be communicated to by connecting the Pi to another device via Ethernet and using the network application Bonjour or any similar functioning tool. Bonjour is the application selected by our team and it is used to help setup a network between connected devices. This allows the connected device to assign an IP to the Pi so that it can be remotely connected to.

If neither of these options are a viable option for the scenario then peripherals can be connected directly to the Pi and interacted with.

3.2.2 Image/firmware

Status

/1 Hardware present?

/1 Memo by student B + How did you make your Image/firmware? (500 words)

/1 Code can be run via serial or remote desktop

/1 Wireless connectivity

/1 Sensor/effector code on repository

### 3.2.4 Breadboard/Independent PCBs

The hardware seen in this system were all designed in the previous semester. The LCD, the proximity/luminosity sensor and the NFC Controller were all carefully tested using the I2C interface before the integration on two separate breadboards. Due to the large amounts of components and required space, the LCD needed a dedicated breadboard while the rest shared a single breadboard.

The LCD, in general, requires a lot of wire when it used in a parallel interface. Though it is almost impossible to run all those connections to a Raspberry Pi GPIO, the LCD uses a PCF8574 IC to use the I2C interface to convert the parallel interface into a serial one. To run in an I2C interface, the PCF8574 IC needs 3V, SDA, SCL and ground to be connected from the Raspberry Pi while the remaining connections are meant to be used to control the LCD. Using the potentiometers with the 4k7 and 10k resistors tied to them allows control over the backlight of the display and the light of the text (the output) during the testing phase of the device. The potentiometer will later be replaced with fixed value resistors to keep the device more compact.

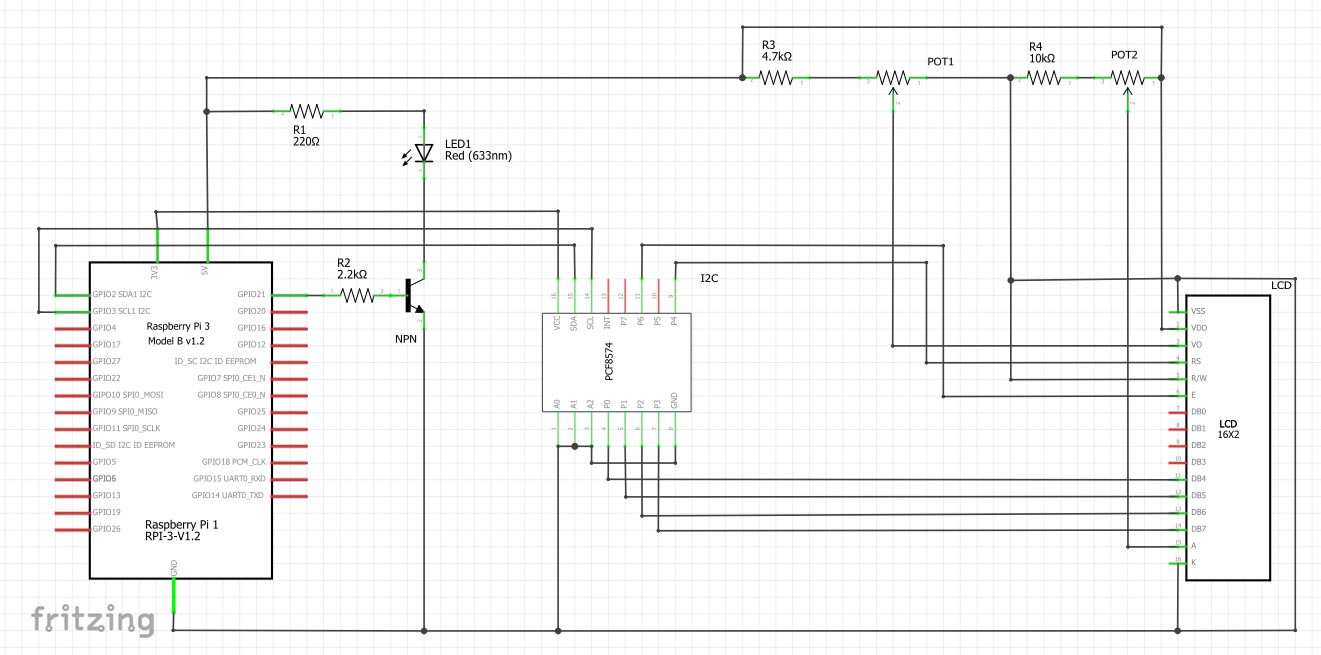


Figure 12. LCD Schematic

The VCNL 4010 proximity/luminosity sensor, previously tested on an STM32 V2 link Blue Pill (an Arduino alternative that uses its IDE), also uses the I2C interface. It just needs to be daisy chained off of the LCD’s connections which are the: 3V, SDA, SCL, and ground. To connect to the device through I2C its address is 0x13.

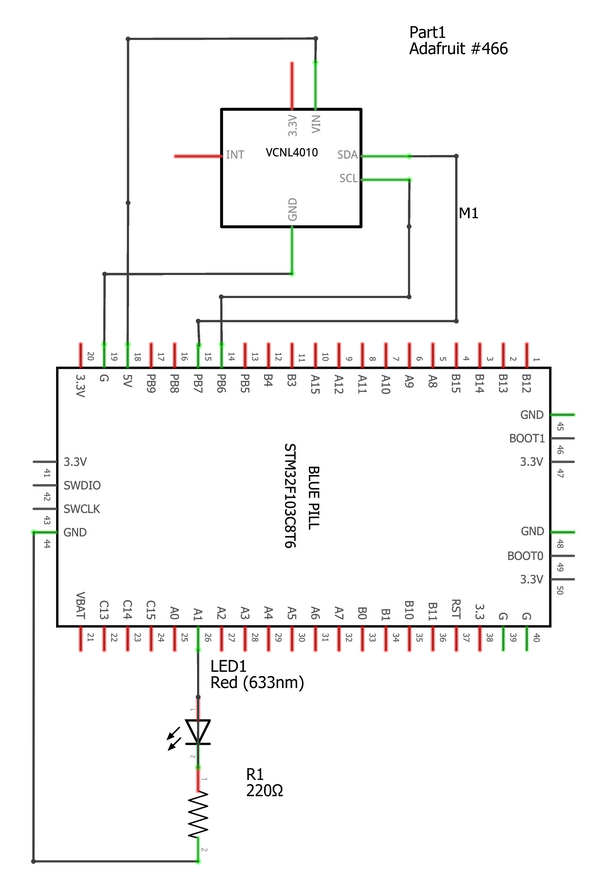


Figure 13. Proximity/Luminosity Sensor

The PN532 controller also uses the I2C interface which is daisy-chained off of the VCNL 4010 using the 3V, SDA, SCL and ground connections. The hardware itself requires a jumper on two sets of pins. Located on the board is labeled SE0 and SE1 where they have 3 pins each. There are several interfaces that can be selected on this NFC controller which include: I2C, UART, and SPI. In this case, I2C is being used so SE0 has to have the jumpers on the ON position while SE1 is switched off. To communicate with the device, it uses a I2C address of 0x24.

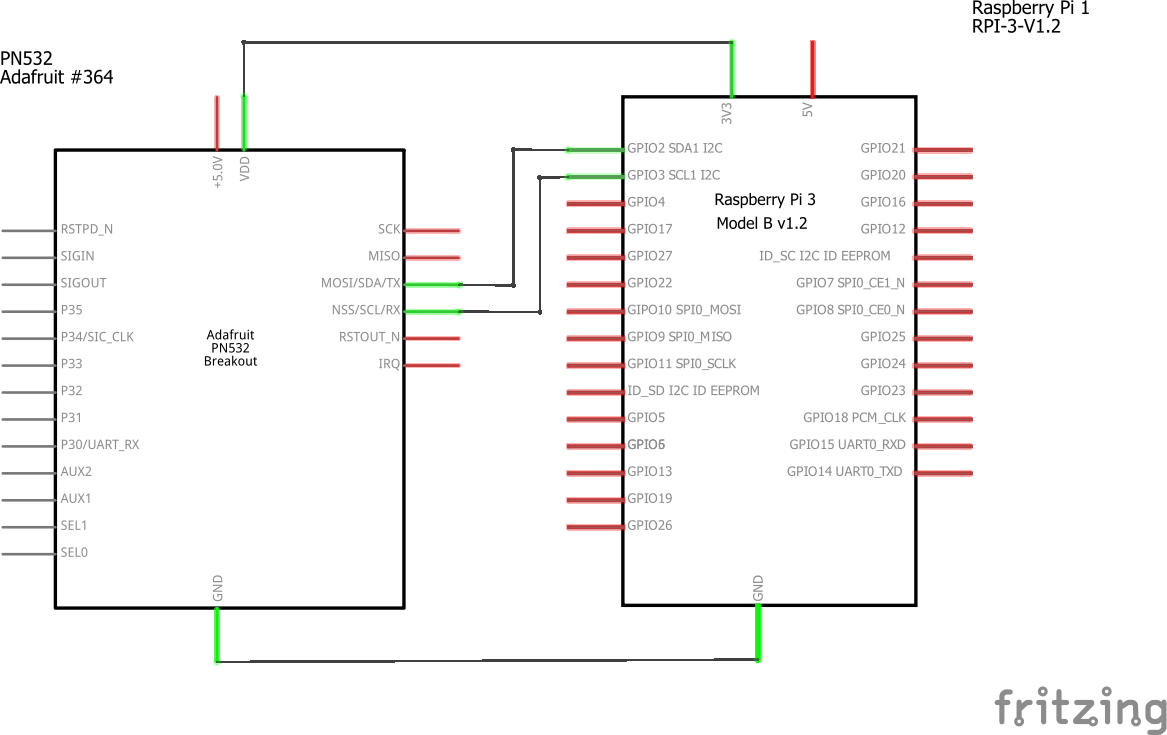


Figure 14. NFC Controller Schematic

Even though the device as a whole currently is operational, there are still some technical difficulties that need to be addressed. The PN532 is the core of the project and is currently experiencing some issues. It is currently reading corrupted data when there is an interruption with an ongoing data transaction. When it is unable to complete a data transaction, it will begin to corrupt the next incoming data transaction as well. Upon further inspection, there is a reset pin on the device. The reset pin can be attached to a GPIO pin on the Raspberry PI to control when to reset the device, but more reading has to be done on the device before any changes are made. Also, the group had also noticed that the backlight did not need to be on all the time especially when no one is using it because it was a waste of energy. The idea of a transistor was brought up to control the backlight and will be later implemented into the design of the project.

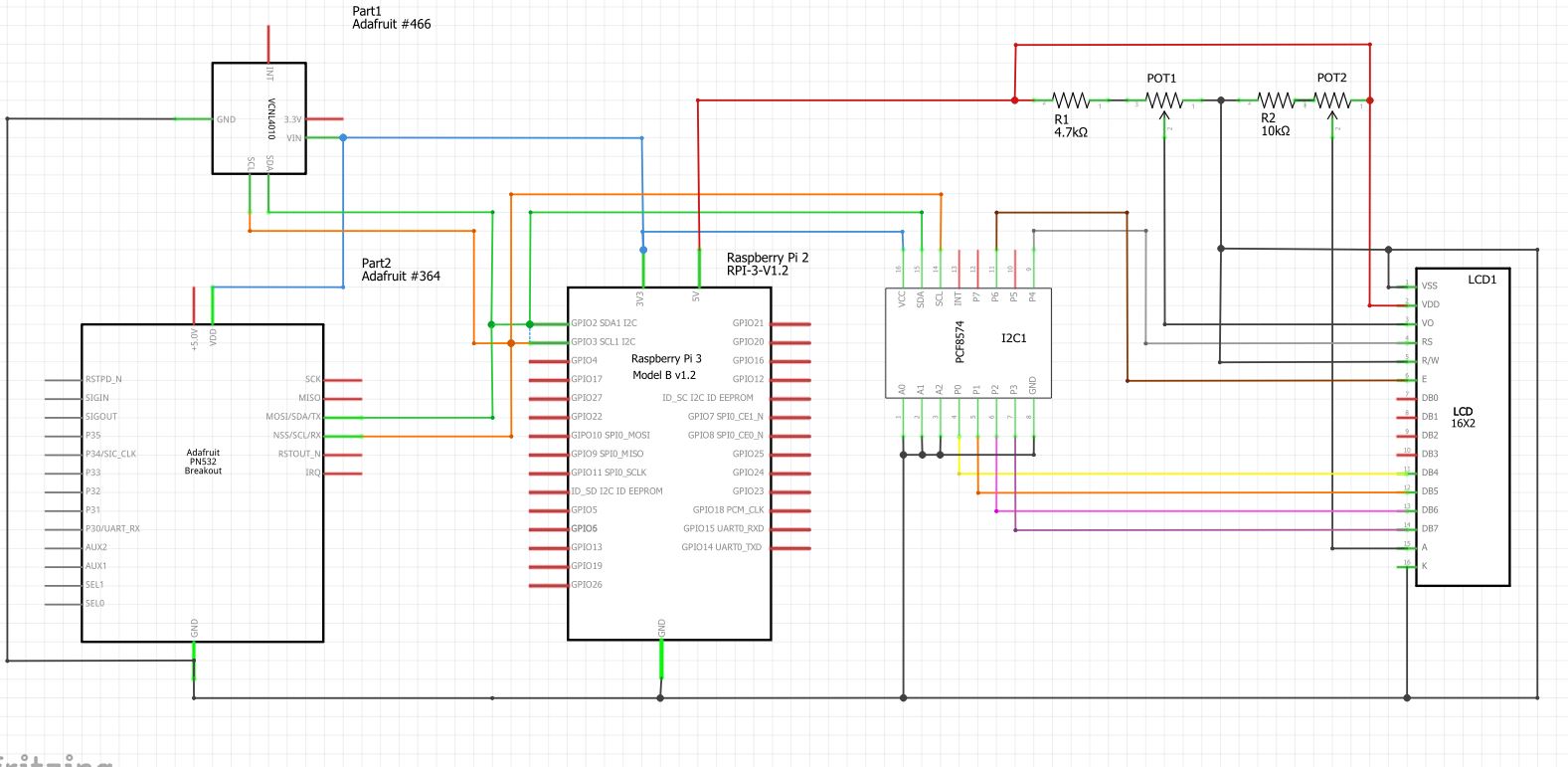


Figure 15. Schematic for all devices wired together

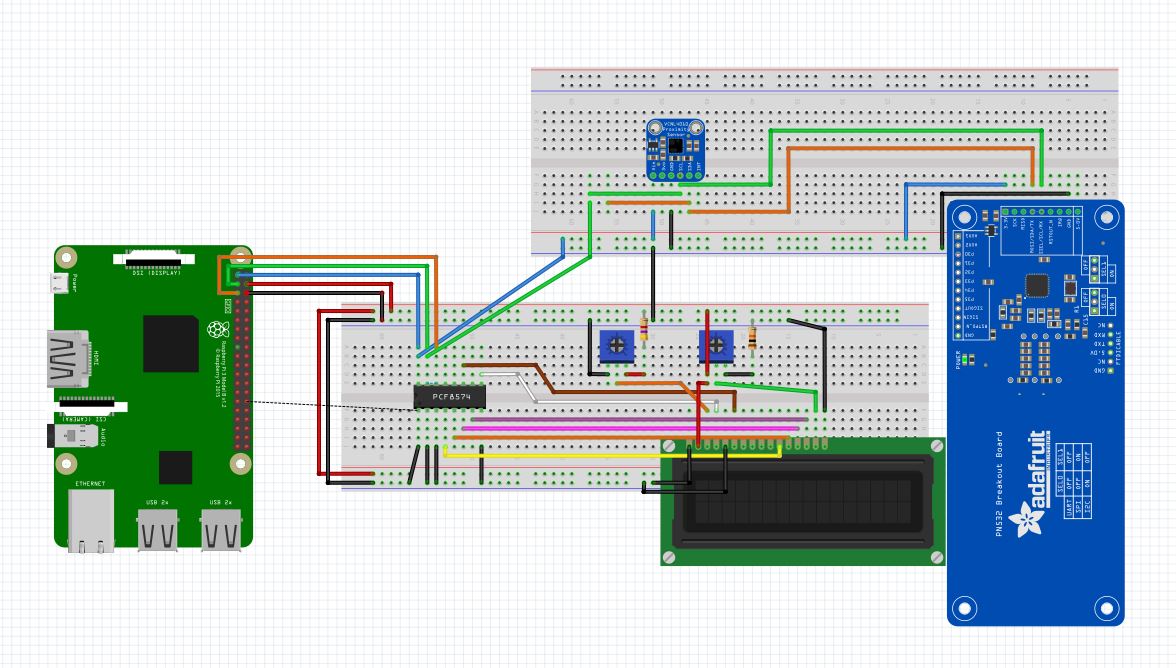


Figure 16. Breadboard Layout. VCNL4010(Top), LCD(Bottom), PN532(Right)

Status

/1 Hardware present?

/1 Memo by student C + How did you make your hardware? (500 words)

/1 Sensor/effector 1 functional

/1 Sensor/effector 2 functional

/1 Sensor/effector 3 functional

The initial schematic design, Figure 2, based on datasheets (Bosch Sensortec, 2019) led to a breadboard layout Figure 3 that was realized Figure 4.

How did you build your Prototype: Breadboard?

Then a PCB was designed, Figure 5, and populated (Figure 6). Bill of Materials, Case, Time commitment. Testing. Progress.

### 3.2.5 Printed Circuit Board

This section documents the PCB created for our product. Outlined will be the process as well as figures to explain how the prototype PCB was created.

The first step in our process was taking a look over each of the independent PCBs from our previous semester and removing the components that would not be required for the combined prototype (LEDs, potentiometers). After close inspection, we were left with our two sensors, the LCD display, the IC chip, and a few resistors to handle the lighting for the LCD. With the initial schematic for the LCD, the back and foreground lights were adjustable with potentiometers but were later replaced with resistors and a 2N4124 NPN transistor that is controlled by a GPIO pin, so that the backlight can be controlled by software. The last step of the planning process was deciding the size of the PCB that would affect the enclosure and the prototype as a whole. The size was determined by the largest component the NFC sensor with the LCD display and proximity sensor sitting above the NFC sensor and the Raspberry Pi underneath to seat the PCB itself.

After the initial preparation, the PCB design was later designed in Fritzing. The wiring necessary for each component was designed to lead into connectors that would have pin connections for only the pins required for each component. Due to the size of the NFC sensor the decision was made to have a wired connection from the connectors to the LCD display and proximity sensor, then the two components are to be seated within the enclosure. This decision was made to prevent the dimensions of the enclosure from being too large.

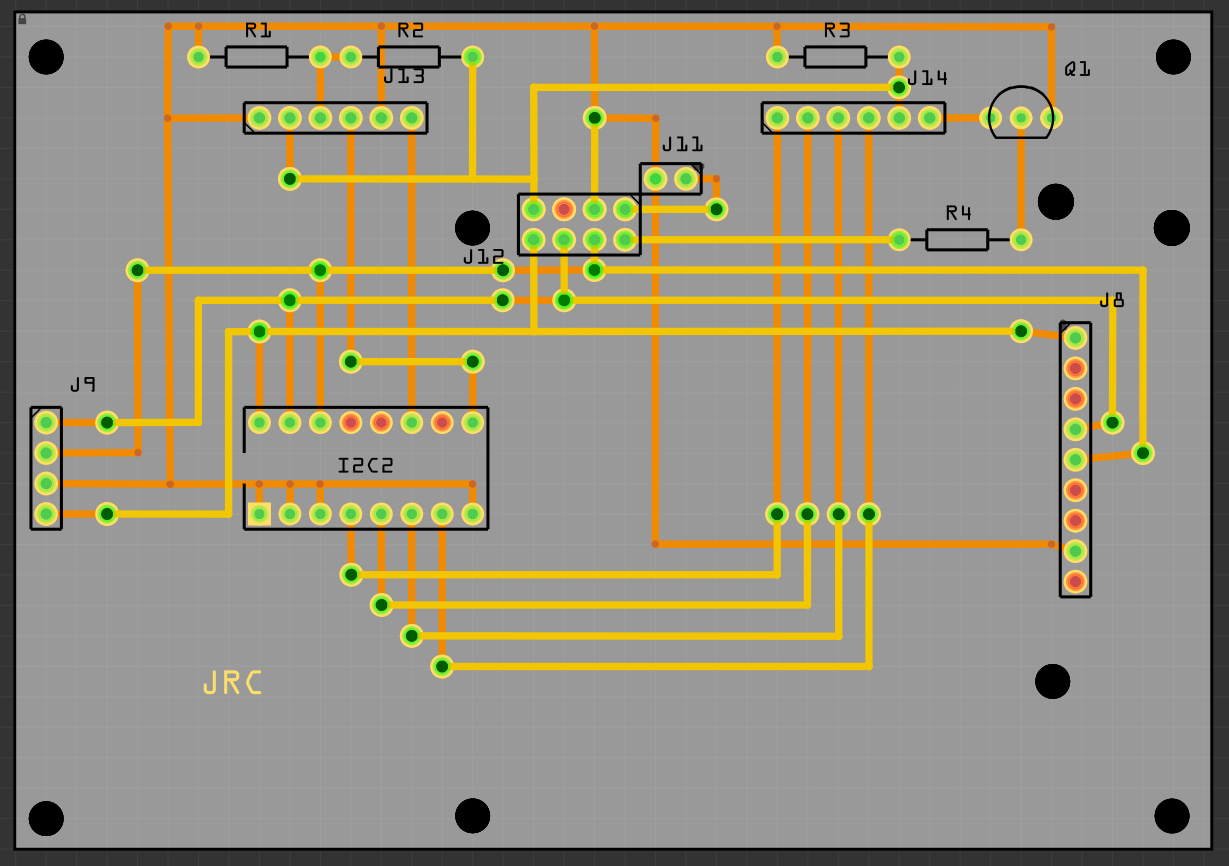


Figure . PCB Design

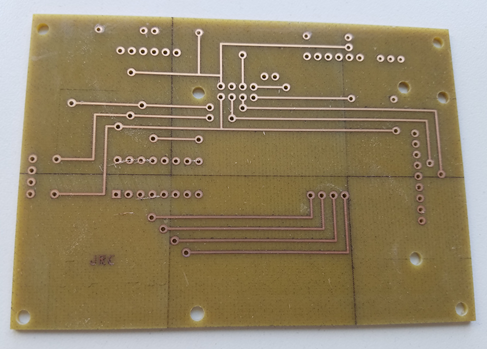


Figure Top of PCB

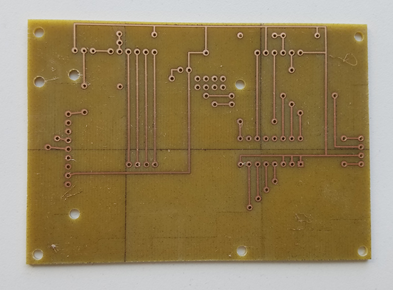


Figure Bottom of PCB

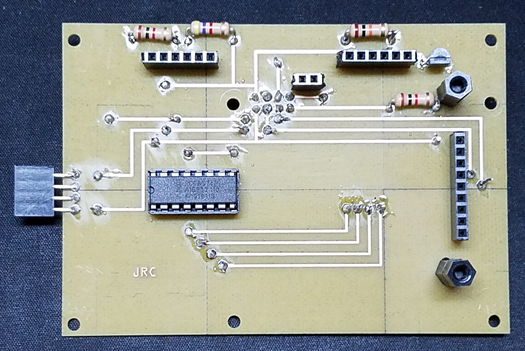
The file that was designed in Fritzing was then submitted to the Humber Prototype lab for the PCB to be developed. The soldering of the connectors, resistors as well as completing the wires through the VIAs onto the PCB took less than two hours to put together.

Figure Top of PCB after soldering

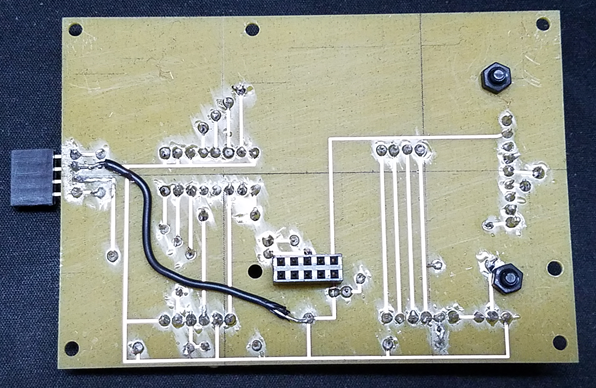


Figure . Bottom of PCB after soldering

The testing of the PCB was conducted with a multi-meter to ensure all the copper lines were connected, correct voltages were being delivered and to ensure there were no short circuits. With the success of this testing, the next step was to test the PCB with the main components plugged into the connectors. At this stage, there was one issue that required troubleshooting. The IC chip used for our setup was running at a high temperature. To resolve this issue, a wire was ran from the grounding point near the IC directly to the ground of the female header that is connected to the Raspberry Pi. Further testing showed that all devices had valid I2C addresses shown to the Pi and all components could communicate through our software.

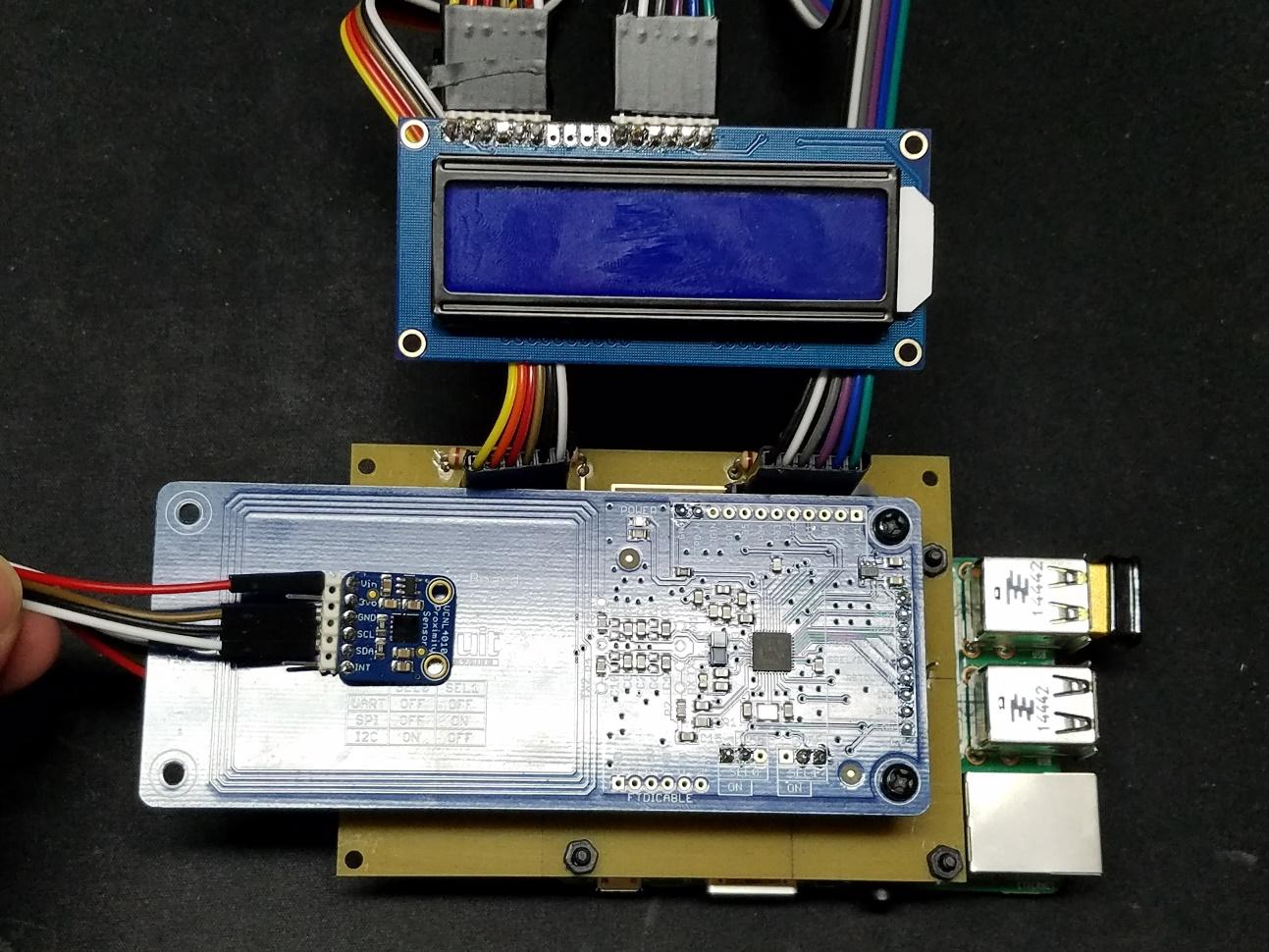


Figure . Completed PCB with connected components

Demo

/1 Hardware present?

/1 PCB Complete and correct

/1 PCB Soldered wire visible but trim, no holes or vacancies

/1 PCB Tested with multimeter

/1 PCB Powered up

How did you build your Prototype: PCB?

### 3.2.5 Enclosure

Demo

/1 Hardware present?

/1 Case encloses development platform and custom PCB.

/1 Appropriate parts securely attached.

/1 Appropriate parts accessible.

/1 Design file in repository, photo in report.

How did you build your Prototype: Case?

## 3.3 Integration

Demo

/1 Hardware present?

/1 Data sent by hardware

/1 Data retrieved by mobile application

/1 Action initiated by mobile application

/1 Action recieved by hardware

Report

/1 Enterprise wireless connectivity (250)

/1 Database configuration (250 words)

/1 Security considerations (500 words)

/1 Unit testing (900 words)

/1 Production testing (100 words)

### 3.3.1 Enterprise Wireless Connectivity

How did you make a Database accessible by both your Prototype and Mobile Application?

### 3.3.2 Database Configuration

### 3.3.3 Security

### 3.3.4 Testing

Unit testing and Production testing.

# 4.0 Results and Discussions

Is your prototype perfect? What did you learn?

# 5.0 Conclusions

If you were making 1000 of these.

Report

/1 Hardware present?

/1 Checklist truthful

/1 Valid Comments

/1 Results and Discussion (500 words)

/1 Conclusion

# 6.0 References

Bosch Sensortec. (2019, July). *BME680 - Datasheet.* Retrieved from Robert Bosch GmbH: https://ae-bst.resource.bosch.com/media/\_tech/media/datasheets/BST-BME680-DS001.pdf

Butterfield, A., & Szymanski, J. (2018). *In A Dictionary of Electronics and Electrical Engineering.* Oxford University Press.

*Hasing Security*. (2019, June 5). Retrieved from Crack Station: https://crackstation.net/hashing-security.htm

OACETT. (2017, March). *I need to Complete a Technology Report*. Retrieved from The Ontario Association of Certified Engineering Technicians and Technologists: https://www.oacett.org/Membership/Technology-Report-and-Seminar

Sabella, R. (2016, April 11). *NFC for dummies*. Retrieved from dummies: https://www.dummies.com/consumer-electronics/nfc-standard-isoiec-14443/

Vanderkay, J. (2004, March 18). *Nokia, Philips And Sony Establish The Near Field Communication (NFC) Forum*. Retrieved from NFC Forum: https://nfc-forum.org/nokia-philips-and-sony-establish-the-near-field-communication-nfc-forum/

# 7.0 Appendix

## 7.1 Firmware code

Demo

/1 Hardware present?

/3 Code runs concurrently for all sensors/effectors

/1 Project repository contains integrated code

Status

/1 Memo including updates

/1 Financial update

/1 Progress update

/1 Modified Code Files in Appendix

/1 Link to Complete Code in Repository

## 7.1.1 main.c File Reference

Main Program that will run the VNCL 4010, PN532 and LCD.

#include <stdio.h>

#include <stdlib.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <nfc/nfc.h>

#include "Presence.h"

#include "NDEFData.h"

### Functions

* int [main](#AAAAAAAAAC) (void)

*Function used to initialize Presence and setup the NFC reader.*

### Detailed Description

Main Program that will run the VNCL 4010, PN532 and LCD.

##### Author

Robert Dinh

##### Date

01MAR2020

Definition in file [main.c](#AAAAAAAAAP).

### Function Documentation

#### int main (void )

Function used to initialize Presence and setup the NFC reader.

The fuction sets up the VNCL4010 and PN532 using the Wiring Pi library through I2C. It then sets up the registers of the devices so that data can be retrieved. When a presence is detected by the VNCL4010 it triggers the NFC reader into a read state to to grab an NDEF encoded message from a mobile device.

Definition at line [27](#AAAAAAAAAQ) of file [main.c](#AAAAAAAAAP).

## main.c

00001

00008 #include <stdio.h>

00009 #include <stdlib.h>

00010 #include <signal.h>

00011 #include <string.h>

00012 #include <unistd.h>

00013 #include <nfc/nfc.h>

00014 #include "[Presence.h](#AAAAAAAAAA)"

00015 #include "[NDEFData.h](#AAAAAAAAAB)"

00016

00017

00027 int [main](#AAAAAAAAAC) (void)

00028 {

00029 [initNFC](#AAAAAAAAAD)();

00030 int fd = [initPresence](#AAAAAAAAAE)();

00031

00032 int present = 0;

00033

00034 fprintf(stdout,"%d \n",fd);

00035 present = [detectPresence](#AAAAAAAAAF)(fd);

00036 if (present)

00037 [getNdef](#AAAAAAAAAG)();

00038

00039 [stopFunction](#AAAAAAAAAH)(2);

00040 return 0;

00041 }

## 7.1.2 NDEFData.c File Reference

Various Functions to read an NDEF message from a mobile device.

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <nfc/nfc.h>

### Functions

* void [stopFunction](#AAAAAAAAAH) (int sig)

*Function used to stop the NFC reader in the case of a System Interrupt.*

* void [initNFC](#AAAAAAAAAD) (void)

*Function used to initialize the PN532 NFC Reader.*

* int [CardTransmit](#AAAAAAAAAI) (nfc\_device \*pnd, uint8\_t \*capdu, size\_t capdulen, uint8\_t \*rapdu, size\_t \*rapdulen)

*Function used to send APDU commands.*

* void [getNdef](#AAAAAAAAAG) (void)

*Function used to initialize Presence.*

### Variables

* nfc\_device \* **pnd**
* nfc\_context \* **context**

### Detailed Description

Various Functions to read an NDEF message from a mobile device.

##### Author

Robert Dinh

##### Date

01MAR2020

The following does contain non-original code but was modified to suit certain needs. Original code was developed by NFC - TOOLS. <http://nfc-tools.org/index.php/Libnfc:APDU_example>

Definition in file [NDEFData.c](#AAAAAAAAAU).

### Function Documentation

#### int CardTransmit (nfc\_device \* pnd, uint8\_t \* capdu, size\_t capdulen, uint8\_t \* rapdu, size\_t \* rapdulen)

Function used to send APDU commands.

Definition at line [70](#AAAAAAAAAV) of file [NDEFData.c](#AAAAAAAAAU).

#### void getNdef (void )

Function used to initialize Presence.

Definition at line [106](#AAAAAAAAAW) of file [NDEFData.c](#AAAAAAAAAU).

#### void initNFC (void )

Function used to initialize the PN532 NFC Reader.

Definition at line [42](#AAAAAAAAAX) of file [NDEFData.c](#AAAAAAAAAU).

#### void stopFunction (int sig)

Function used to stop the NFC reader in the case of a System Interrupt.

Definition at line [28](#AAAAAAAAAY) of file [NDEFData.c](#AAAAAAAAAU).

## NDEFData.c

00001

00013 #include <stdlib.h>

00014 #include <string.h>

00015 #include <unistd.h>

00016 #include <signal.h>

00017 #include <nfc/nfc.h>

00018

00019 //NFC Variables

00020 nfc\_device \*pnd;

00021 nfc\_context \*context;

00022

00023

00028 void [stopFunction](#AAAAAAAAAH)(int sig)

00029 {

00030

00031 printf("Stopping now.\n");

00032 nfc\_close(pnd);

00033 nfc\_exit(context);

00034 exit(EXIT\_SUCCESS);

00035

00036 }

00037

00042 void [initNFC](#AAAAAAAAAD)(void)

00043 {

00044 signal(SIGINT,[stopFunction](#AAAAAAAAAH));

00045 nfc\_init(&context);

00046

00047 if (context == NULL) {

00048 printf("Unable to init libnfc (malloc)\n");

00049 exit(EXIT\_FAILURE);

00050 }

00051

00052 pnd = nfc\_open(context, NULL);

00053

00054 if (pnd == NULL) {

00055 printf("ERROR: %s", "Unable to open NFC device.");

00056 exit(EXIT\_FAILURE);

00057 }

00058

00059 if (nfc\_initiator\_init(pnd) < 0) {

00060 nfc\_perror(pnd, "nfc\_initiator\_init");

00061 exit(EXIT\_FAILURE);

00062 }

00063

00064 }

00065

00070 int [CardTransmit](#AAAAAAAAAI)(nfc\_device \*pnd, uint8\_t \* capdu, size\_t capdulen, uint8\_t \* rapdu, size\_t \* rapdulen)

00071 {

00072 int res;

00073 size\_t szPos;

00074

00075 printf("=> ");

00076 for (szPos = 0; szPos < capdulen; szPos++)

00077 {

00078 printf("%02x ", capdu[szPos]);

00079 }

00080 printf("\n");

00081

00082 if ((res = nfc\_initiator\_transceive\_bytes(pnd, capdu, capdulen, rapdu, \*rapdulen, 500)) < 0)

00083 {

00084 return -1;

00085 }

00086 else

00087 {

00088 \*rapdulen = (size\_t) res;

00089 printf("<= ");

00090

00091 for (szPos = 0; szPos < \*rapdulen; szPos++)

00092 {

00093 printf("%02x ", rapdu[szPos]);

00094 }

00095

00096 printf("\n");

00097 return 0;

00098 }

00099 }

00100

00101

00106 void [getNdef](#AAAAAAAAAG)(void)

00107 {

00108 nfc\_target nt;

00109

00110

00111 const nfc\_modulation nmMifare = {

00112 .nmt = NMT\_ISO14443A,

00113 .nbr = NBR\_106,

00114 };

00115

00116 printf("Polling for target...\n");

00117

00118 while (nfc\_initiator\_select\_passive\_target(pnd, nmMifare, NULL, 0, &nt) <= 0);

00119 printf("Target detected!\n");

00120

00121 uint8\_t capdu[264];

00122 size\_t capdulen;

00123 uint8\_t rapdu[264];

00124 size\_t rapdulen;

00125

00126

00127 // Select application

00128 memcpy(capdu, "\x00\xA4\x04\x00\x07\xF0\x39\x41\x48\x14\x81\x00\x00", 13);

00129 capdulen=13;

00130 rapdulen=sizeof(rapdu);

00131 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00132 exit(EXIT\_FAILURE);

00133 if (rapdulen < 2 || rapdu[rapdulen-2] != 0x90 || rapdu[rapdulen-1] != 0x00)

00134 exit(EXIT\_FAILURE);

00135

00136 printf("Application selected!\n");

00137

00138 // Select Capability Container

00139 memcpy(capdu, "\x00\xa4\x00\x0c\x02\xe1\x03", 7);

00140 capdulen=7;

00141 rapdulen=sizeof(rapdu);

00142 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00143 exit(EXIT\_FAILURE);

00144 if (rapdulen < 2 || rapdu[rapdulen-2] != 0x90 || rapdu[rapdulen-1] != 0x00) {

00145 capdu[3]='\x00'; // Maybe an older Tag4 ?

00146 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00147 exit(EXIT\_FAILURE);

00148 }

00149

00150 printf("Capability Container selected!\n");

00151

00152 // Read Capability Container

00153 memcpy(capdu, "\x00\xb0\x00\x00\x0f", 5);

00154 capdulen=5;

00155 rapdulen=sizeof(rapdu);

00156 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00157 exit(EXIT\_FAILURE);

00158 if (rapdulen < 2 || rapdu[rapdulen-2] != 0x90 || rapdu[rapdulen-1] != 0x00)

00159 exit(EXIT\_FAILURE);

00160

00161 printf("Capability Container header:\n");

00162

00163 size\_t szPos;

00164 for (szPos = 0; szPos < rapdulen-2; szPos++)

00165 {

00166 printf("%02x ", rapdu[szPos]);

00167 }

00168 printf("\n");

00169

00170 // NDEF SELECT

00171 memcpy(capdu, "\x00\xA4\x00\x0C\x02\xE1\x04", 7);

00172 capdulen=7;

00173 rapdulen=sizeof(rapdu);

00174 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00175 exit(EXIT\_FAILURE);

00176 if (rapdulen < 2 || rapdu[rapdulen-2] != 0x90 || rapdu[rapdulen-1] != 0x00)

00177 exit(EXIT\_FAILURE);

00178

00179 printf("NDEF SELECTED!\n");

00180

00181 // NDEF Read Binary

00182 memcpy(capdu, "\x00\xb0\x00\x00\x02", 5);

00183 capdulen=5;

00184 rapdulen=sizeof(rapdu);

00185 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00186 exit(EXIT\_FAILURE);

00187 if (rapdulen < 2 || rapdu[rapdulen-2] != 0x90 || rapdu[rapdulen-1] != 0x00)

00188 exit(EXIT\_FAILURE);

00189 printf("NDEF Read Binary NLEN!\n");

00190

00191 //READING NDEF DATA

00192 memcpy(capdu, "\x00\xb0\x00\x00\x0f", 5);

00193 capdulen=5;

00194 rapdulen=sizeof(rapdu);

00195 if ([CardTransmit](#AAAAAAAAAI)(pnd, capdu, capdulen, rapdu, &rapdulen) < 0)

00196 exit(EXIT\_FAILURE);

00197 if (rapdulen < 2 || rapdu[rapdulen-2] != 0x90 || rapdu[rapdulen-1] != 0x00)

00198 exit(EXIT\_FAILURE);

00199

00200 printf("NDEF DATA ! \n\n");

00201 size\_t szPos2;

00202 char ndefMsg[100] = "";

00203 char chr[1] = "\0";

00204 int counter = 0;

00205

00206 for (szPos2 = 8; szPos2 < rapdulen-2; szPos2++) {

00207 ndefMsg[counter]=(char)rapdu[szPos2];

00208 counter++;

00209 }

00210

00211 printf("%s",ndefMsg);

00212

00213 char cmdS[100] = "python lcd.py Phone Scanned\n";

00214

00215

00216 system(cmdS);

00217 sleep(1);

00218 printf("\nDone...\n");

00219 }

## NDEFData.h File Reference

Function prototypes for NDEFData.

### Detailed Description

Function prototypes for NDEFData.

##### Author

Robert Dinh

##### Date

01MAR2020

Definition in file [NDEFData.h](#AAAAAAAAAZ).

## NDEFData.h

00001

00008 #ifndef NDEFData\_H

00009 #define NDEFData\_H

00010

00011

00013 // Function Prototypes

00014 void [stopFunction](#AAAAAAAAAH)(int sig);

00015 void [initNFC](#AAAAAAAAAD)(void);

00016 int [CardTransmit](#AAAAAAAAAI)(nfc\_device \*pnd, uint8\_t \* capdu, size\_t capdulen, uint8\_t \* rapdu, size\_t \* rapdulen);

00017 void [getNdef](#AAAAAAAAAG)(void);

00019

00020

00021

00022 #endif

## 7.1.3 Presence.c File Reference

Various functions to aid in Presence Detection with the VNCL4010.

#include <stdio.h>

#include <wiringPiI2C.h>

#include <unistd.h>

#include "Presence.h"

### Functions

* int [initPresence](#AAAAAAAAAE) (void)

*Function used to initialize Presence.*

* int [detectPresence](#AAAAAAAAAF) (int fd)

*Function used to detect the presence of a person.*

* int [getProximity](#AAAAAAAAAL) (int fd)

*Function used get the proximity of the sensor.*

* int [getLuminosity](#AAAAAAAAAN) (int fd)

*Function used get the luminosity of the sensor.*

### Detailed Description

Various functions to aid in Presence Detection with the VNCL4010.

##### Author

Robert Dinh

##### Date

01MAR2020

Definition in file [Presence.c](#AAAAAAAABB).

### Function Documentation

#### int detectPresence (int fd)

Function used to detect the presence of a person.

Definition at line [44](#AAAAAAAABC) of file [Presence.c](#AAAAAAAABB).

#### int getLuminosity (int fd)

Function used get the luminosity of the sensor.

Definition at line [80](#AAAAAAAABD) of file [Presence.c](#AAAAAAAABB).

#### int getProximity (int fd)

Function used get the proximity of the sensor.

Definition at line [65](#AAAAAAAABE) of file [Presence.c](#AAAAAAAABB).

#### int initPresence (void )

Function used to initialize Presence.

The fuction sets up the VNCL4010 using the Wiring Pi library through I2C. It then sets up the registers of the device so that data can be retrieved.

Definition at line [20](#AAAAAAAABF) of file [Presence.c](#AAAAAAAABB).

## Presence.c

00001

00008 #include <stdio.h>

00009 #include <wiringPiI2C.h>

00010 #include <unistd.h>

00011 #include "[Presence.h](#AAAAAAAAAA)"

00012

00013

00020 int [initPresence](#AAAAAAAAAE)(void)

00021 {

00022 //Setup Wiring Pi thorugh I2C

00023 int fd = wiringPiI2CSetup(0x13);

00024

00025 //Select Command Register

00026 wiringPiI2CWriteReg8(fd,0x80,0xFF);

00027

00028 //Select Prox Register

00029 wiringPiI2CWriteReg8(fd,0x82,0x03);

00030

00031 //Select Current for IR LEDto 200ma

00032 wiringPiI2CWriteReg8(fd,0x83,0x14);

00033

00034 //Select ALS Register

00035 wiringPiI2CWriteReg8(fd,0x84,0x9D);

00036

00037 return fd;

00038 }

00039

00044 int [detectPresence](#AAAAAAAAAF)(int fd)

00045 {

00046 [presenceD](#AAAAAAAAAJ) pData = {0};

00047 while (1)

00048 {

00049 pData.[proximity](#AAAAAAAAAK) = [getProximity](#AAAAAAAAAL)(fd);

00050 pData.[luminosity](#AAAAAAAAAM) = [getLuminosity](#AAAAAAAAAN)(fd);

00051

00052 fprintf(stdout,"Proximity:\t%d\t Luminosity:\t%d\n",pData.[proximity](#AAAAAAAAAK),pData.[luminosity](#AAAAAAAAAM));

00053 if (pData.[proximity](#AAAAAAAAAK) > 2300)

00054 return pData.[proximity](#AAAAAAAAAK);

00055

00056 usleep(300000);

00057 }

00058

00059 }

00060

00065 int [getProximity](#AAAAAAAAAL)(int fd)

00066 {

00067 int prox1= 0;

00068 int prox2= 0;

00069

00070 prox1 = wiringPiI2CReadReg8(fd,0x87) \*256; //Upper Byte

00071 prox2 = wiringPiI2CReadReg8(fd,0x88); //Lower Byte

00072

00073 return prox1+prox2;

00074 }

00075

00080 int [getLuminosity](#AAAAAAAAAN)(int fd)

00081 {

00082 int lux1= 0;

00083 int lux2= 0;

00084

00085 lux1 = wiringPiI2CReadReg8(fd,0x85) \*256; //Upper Byte

00086 lux2 = wiringPiI2CReadReg8(fd,0x86); //Lower Byte

00087

00088 return lux1 + lux2;

00089 }

## Presence.h File Reference

Constants, structures, function prototypes for Presence.

#include <wiringPiI2C.h>

### Data Structures

* struct [PresenceData](#AAAAAAAAAJ)

### Typedefs

* typedef struct [PresenceData](#AAAAAAAAAJ) **presenceD**

### Detailed Description

Constants, structures, function prototypes for Presence.

##### Author

Robert Dinh

##### Date

01MAR2020

Definition in file [Presence.h](#AAAAAAAABH).

## Presence.h

00001

00008 #ifndef PRESENCE\_H

00009 #define PRESENCE\_H

00010

00011 #include <wiringPiI2C.h>

00012

00013 // Structures

00014 typedef struct [PresenceData](#AAAAAAAAAJ)

00015 {

00016 int [proximity](#AAAAAAAAAK);

00017 int [luminosity](#AAAAAAAAAM);

00018 }[presenceD](#AAAAAAAAAJ);

00019

00021 // Function Prototypes

00022 int [initPresence](#AAAAAAAAAE)(void);

00023 int [detectPresence](#AAAAAAAAAF)(int fd);

00024 int [getProximity](#AAAAAAAAAL)(int fd);

00025 int [getLuminosity](#AAAAAAAAAN)(int fd);

00027

00028

00029

00030 #endif

## 7.1.4 Lcd.py File Reference

from Adafruit\_CharLCD import Adafruit\_CharLCD

import Adafruit\_GPIO.PCF8574 as PCF

import wiringpi as wiringpi

import time

from time import sleep

import sys

#Initialization of the Wiring Pi Library

wiringpi.wiringPiSetup()

wiringpi.wiringPiSetupGpio()

wiringpi.wiringPiSetupPhys()

wiringpi.pinMode(7, 1)

#Getting the I2C address of the LCD

GPIO = PCF.PCF8574(address=0x38)

#Define PCF pins connected to the LCD

lcd\_rs = 4

lcd\_en = 6

d4,d5,d6,d7 = 0,1,2,3

cols,lines = 16,2

#instantiate LCD display

lcd = Adafruit\_CharLCD(lcd\_rs, lcd\_en, d4, d5, d6, d7,

cols, lines, gpio=GPIO)

lcd.clear()

test = ' '.join(sys.argv[1:])

if len(test)>16:

test = test[0:15] + "\n" + test[15:]

#Display the Contents to the LCD. Turn on Backlight and turn off.

print str(test)

print "LCD ON"

wiringpi.digitalWrite(7,1)

lcd.message(str(test))

sleep(5)

wiringpi.digitalWrite(7,0)

print "LCD OFF"

## 7.2 Application code

Demo

/1 Hardware present?

/1 Memo by student A

/1 Login activity

/1 Data visualization activity

/1 Action control activity

Report

/1 Login activity

/1 Data visualization activity

/1 Action control activity

/1 Modified Code Files in Appendix

/1 Link to Complete Code in Repository