**SuperStore Interactive Helpdesk**

**Technical Report**

**Authors:**

**Guru Sharma**

**Saqib Jaweed Syed**

**Abhay Singla**

Date: 24-04-2018

# **Table of contents**

[**Table of contents**](#_2wuq8r4jzxx4) **1**

[2.4 Software Requirement Specifications](#_mj93oi7a61hr) 6

[2.4.1 Database (Guru)](#_3fck34z7cko4) 6

[2.4.2 Mobile Application (Guru)](#_bygoqtcwhhgs) 7

[2.5.2 Time Commitment](#_3yqkwk9bxvqh) 9

[2.8 Walkthrough of the system](#_ysg1l98lgz3r) 10

[2.8.1 Microcontroller](#_h53zi3smd7ck) 10

[2.8.2 Microprocessor](#_pkmqrke5dw99) 10

[2.8.3 Database (Guru)](#_vu4p6kxiykwu) 10

[Why Firebase?](#_o20uy2nh22hi) 10

[Structure and Features](#_1had55dsg8e4) 11

[Illustration 1. Overall view of Firebase Database](#_veha1fkrlg2y) 11

[Illustration 2. Items Table View Expanded](#_l9mgx755h36r) 12

[Illustration 3. Single Item View Expanded](#_f6kn7impkqjt) 12

[Illustration 4 (a). Users Table View Expanded, (b). Expanded view of Users with and without items in a cart](#_b1ksx7gyx09l) 13

[2.8.4 Android App (Guru)](#_3h6k4z1fnfkz) 13

[Illustration 5. App’s Home screen with options to “SIGN IN” or SIGN UP”](#_3kz62896maa3) 14

[Illustration 6. App Home Screen with “SIGN UP” page for new users and an option to “SIGN IN” for existing users](#_8jml7g6w8u1) 15

[Illustration 7. App’s Welcome page with Catalog (Item Count reflecting items in online catalog displayed)](#_90gn9a9vdrqb) 16

[Illustration 8. “Product Description” page for Samsung Galaxy S8 (Please note that the Description view is scrollable)](#_cwighzh1x85c) 18

[Illustration 9. “Shopping Cart” page for a user with 5 items in Cart (Total items and Total Price displayed)](#_ftbfnhqc80oz) 19

[Illustration 10. “Log Out” option in Action Bar](#_j69onctzb5i9) 20

Appendix 45

**Declaration of Joint Authorship**

Superstore Interactive Helpdesk  
From: Saqib Jaweed Syed, Abhay Singla & Guru Sharma  
Program: Computer Engineering Technology  
Course: CENG-355 0NA  
Instructor: Austin Tian & Kristian Medri  
Date: 2/25/2018  
   
We, Saqib Jaweed Syed, Abhay Singla and Guru Sharma, confirm that this work submitted for assessment is the joint work of ourselves, and is expressed in our own words. Any uses made within of other works of any other author, in any form (ideas, equations, figures, previous technologies, tables, programs, texts) are properly acknowledged at the point of use. A list of the references used is included. Saqib Jaweed Syed has handled the Hardware Build and Integration, Abhay Singla has handled the Rpi software and GUI, and Guru Sharma has handled Android Application and database aspects of this project.

Abstract  
There is a large need for small-scale agricultural operations to constantly monitor the environment around newly growing plants. To remedy this, a system should be created to actively monitor the environment during the germination process of a plant in a way that allows a user to remain up to date on the environment without having to be present. This system monitor ambient temperature, relative humidity, light levels towards the plant, and the moisture of the soil, and allow the plant to be watered should the soil moisture drop too low. These sensors will store the data temporarily on the Development Platform before it is sent to an offsite database. This database will hold all of the sensor data, and make it available to both an Android application, and a website. The application will display the most recent data entries, as well as a limited history of data entries from the database, including graphs for ease of viewing. The website will display the most recent batch of entries, and provide the full history for each individual sensor’s data, as well as a graph for ease of viewing trends. This system has the potential to make small-scale growing operations easier to monitor, allowing users to check their plants’ conditions from anywhere at any time, provided they have internet connectivity.  
  
Approved Proposal

|  |  |
| --- | --- |
| Submission Date | 2/5/2018 |
| Project Name | Store Helpline |
| Student Names | Guru Sharma, Abhay Singla, Saqib Jaweed Syed |
| Project repository | <https://github.com/gurusharma/Store_Helpline> |
| SensorsEffectors choices | RFID Reader, PIR Sensor, Touch Screen |
| The database will store | User Authentication and Store ItemList |
| The mobile device functionality will include | Generation Of Recipt, Display the Cart and Feedback |
| I will be collaborating with the following company/department | Prototype Lab and Humber Tech. Club |
| My group in the winter semester will include | The people I was working with in the last semester. |
| 50 word problem statement | Every supermarket has to hire employees to help their customers with various problems regarding products, for example, finding the right product in the right place. But the associates usually fall behind in assisting the customers and therefore some customers leave the store with bad impression. Using our design, the associate and the user will be aware of the current line and time estimates. |
| 100 words of background | Whether in-charge of a small, individually-owned grocery store or the one that is part of a larger chain, managing a grocery store successfully involves considerable responsibility. Grocery store managers must ensure that the store runs smoothly, that items are priced competitively and that customers are satisfied. Having a thorough understanding of key concepts, involved in effective grocery store management, is imperative for any manager dedicated to the success of his/her store. What customers need in the current era is as follows: 1. Sell what they need and have it in stock when they want it.  2. An easier way for them to shop and find what they are looking for.  3. Get all the information they need in order to quickly decide what to buy.  4. Have friendly helpful people available to make the shopping experience a pleasant one. Currently, some of the stores have placed 'Help' buttons at a few locations within that the customers can press and wait until an associate gets free to help them. |
|  |  |
| Existing research IEEE paper APA citation | Advertisement and shopping guide system for large supermarkets based on wireless sensor network. (2012, May 25). Retrieved September 18, 2017, from http://ieeexplore.ieee.org/document/6272826/ |
| Brief description of planned purchases | Raspberry Pi 3, Touch Screen input Module, PIR Sensor, RFID Reader, connecting cables. |
| Solution description | We propose a solution to the above problems with interactive touch screens installed at various locations throughout the superstore, which allows users to look through the products and ask for help. Once they have registered for help, the right associate will help them. Once the process is complete, the software will send the user a short survey which will help in store's future decisions. |

Executive Summary

As students in the Computer Engineering Technology program, We will be integrating the knowledge and skills we have learned from our program into this Internet of Things themed capstone project. This proposal requests the approval to build the hardware portion that will connect to a database as well as to a mobile device application. The internet connected hardware will include a custom PCB with the following sensors and actuators RFID Reader, PIR Sensor, Touch Screen. The database will store User Authentication and store item list. The mobile device functionality will include Product Lookup, Display the Cart. We will be collaborating with the Prototype Lab and Humber Tech. Club . In the Fall 2018 semester we planed to form a group with each other, who are also building similar hardware this term and working on the mobile application with me The people I was working with in the last semester.. The hardware will be completed in CENG 317 Hardware Production Techniques independently and the application will be completed in CENG 319 Software Project. These will be integrated together in the subsequent term in CENG 355 Computer Systems Project as a 3 student group.

Background

The problem solved by this project is that every supermarket has to hire employees to help their customers with various problems regarding products, for example, finding the right product in the right place. But the associates usually fall behind in assisting the customers and therefore some customers leave the store with bad impression. Using our design, the associate and the user will be aware of the current line and time estimates.. A bit of background about this topic is Whether in-charge of a small, individually-owned grocery store or the one that is part of a larger chain, managing a grocery store successfully involves considerable responsibility. Grocery store managers must ensure that the store runs smoothly, that items are priced competitively and that customers are satisfied. Having a thorough understanding of key concepts, involved in effective grocery store management, is imperative for any manager dedicated to the success of his/her store. What customers need in the current era is as follows: 1. Sell what they need and have it in stock when they want it.

2. An easier way for them to shop and find what they are looking for.

3. Get all the information they need in order to quickly decide what to buy.

4. Have friendly helpful people available to make the shopping experience a pleasant one. Currently, some of the stores have placed 'Help' buttons at a few locations within that the customers can press and wait until an associate gets free to help them..

Existing products on the market includeI have searched for prior art via Humber’s IEEE subscription selecting “My Subscribed Content”[2] and have found and read [3] which provides insight into similar efforts.

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

* Java Docs from CENG 212 Programming Techniques In Java,
* Construction of circuits from CENG 215 Digital And Interfacing Systems,
* Rapid application development and Gantt charts from CENG 216 Intro to Software Engineering,
* Micro computing from CENG 252 Embedded Systems,
* SQL from CENG 254 Database With Java,
* Web access of databases from CENG 256 Internet Scripting; and,
* Wireless protocols such as 802.11 from TECH152 Telecom Networks.
* How to couple and control hardware sensors and interface them through RPi3 in Hardware Project CENG 317
* Developing Android Applications in Software Project CENG 319

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

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

[3] Advertisement and shopping guide system for large supermarkets based on wireless sensor network. (2012, May 25). Retrieved September 18, 2017, from http://ieeexplore.ieee.org/document/6272826/

Methodology

This proposal is assigned in the first week of class and is due at the beginning of class in the last week of the winter semester. Our coursework will focus on all 3 phases of this project mentioned below:

Phase 1 Hardware build.

Phase 2 System integration.

Phase 3 Demonstration to future employers.

Phase 1 Hardware build

The hardware build will be completed in the Winter 2018 term. It will fit within the CENG Project maximum dimensions of 12 13/16" x 6" x 2 7/8" (32.5cm x 15.25cm x 7.25cm) which represents the space below the tray in the parts kit. The highest AC voltage that will be used is 16Vrms from a wall adaptor from which +/- 15V or as high as 45 VDC can be obtained. Maximum power consumption will be 20 Watts.

Phase 2 System integration

The hardware system will be linked to the GUI Software and the Android App in week 9 or 10 of the semester. This will include the app working on the phone while the GUI software gives the user a bit more convenience by adding items to cart and requesting help to call an associate and ask product related questions.

Phase 3 Demonstration to future employers

This project will showcase the knowledge and skills that we have put together to build this product that solves a real life problem potential employers.

Concluding remarks

This proposal presents a plan for providing an IoT solution for We propose a solution to the above problems with interactive touch screens installed at various locations throughout the superstore, which allows users to look through the products and ask for help. Once they have registered for help, the right associate will help them. We plan to add a functionality where when the process is complete, the software will send the user a short survey which will help in store's future decisions.. This is an opportunity to integrate the knowledge and skills developed in our program to create a collaborative IoT capstone project demonstrating our ability to learn how to support projects such as the initiative described by [3]. We request approval of this project.

Abstract  
Illustration List  
1. Introduction

This project was made to meet the requirements of our final semester capstone project, the hardware consists of a motion sensor a touch screen, an RFID reader and an LED coupled to Raspberry Pi 3B. The project works in the following manner: by default, there is a screensaver running on the screen that is just playing advertisements and as soon as a customer walks within 1 meter of the touchscreen, it launches the helpdesk application on the screen t0 enable user to use the app. The user then has the option to request help from an associate, when the user presses the help option, the LED lights up until an associate walks up to the kiosk and taps their provided RFID tags against the RFID readers. We have also used firebase-database in our project to store data and have also written an Android Application for users who might want to use the features remotely.

2. Project Description

The main idea behind this project is to provide convenience to the user by letting them browse and select items through a touchscreen or a mobile application that is available on the Android play store. As you might have noticed on every visit a super store that all the items that you need on a daily basis for example milk bread and butter are shelved at the rear of the Superstore because the superstores want you to walk through the aisles and pick up items that you might not even need. This idea will not only save money for the consumer but it will also be a money saver for the store owners are franchises as this will eliminate the need for extra Associates to help customers in the future we plan to add a checkout option at the kiosk terminal and through the mobile application.

2.1 Problem

You might have seen that when you are looking for an item at a superstore, you might have to spend five to eight minutes looking around the store finding the item but then again you have to look at every shelf and every aisle and even ask Associates for directions. The project that we have made enables the user to use a touch screen located at the entrance of the superstore through which users can browse items available in the store, look at the item’s price and even request help from an associate.  
2.4 Software Requirement Specifications

### 2.4.1 Database (Guru)

It was identified that for solving the problem stated in the section 2.1 the implementation would need a secure database. The specifications for the database design were finalized after a few iterations over the course of three months. Older version of the specifications can be found in the appendix section 7.8. To best address the problem, following requirements were identified and finalized after a few iterations.

Following are the requirements that were specified for the database component of the project:

REQ-1: The database shall store the information about the products.

REQ-2: The product information in the database shall be accessible to an authenticated user only.

REQ-3: The database shall be secure and shall have good storage capacity.

REQ-4: The database shall be easy to edit and update.

REQ-5: The database shall be accessible via internet (on cloud) to avoid storing information on devices.

REQ-6: The database shall be compatible to be used with android studio for effective implementation by the app developers.

REQ-7: The database shall be compatible to be used with Raspberry Pi.

REQ-8: The database shall store information about users who sign up, using the app, in a secure way.

REQ-9: The database shall support following functionalities for the app developer:

* User Login Authentication (to ensure reliability, durability, concurrency, security and correctness)
* Allow users to see inventory and cart only after a successful login
* User Specific view of cart
* Updating user cart data when user adds an item using touchscreen or app
* Updating user cart data when user deletes items from cart using the app

REQ-10: Database shall be scalable and easy to maintain.

### 2.4.2 Mobile Application (Guru)

The specifications for the mobile application were finalized after various iterations over the course of three months. Older version of the specifications can be found in the appendix 7.7.1. To serve the potential users with an easily accessible interface, the need of a mobile app was identified. The app would be able to provide users with unique experience. The customers of the superstores with use of a mobile app can get optimal user experience. Following are the specified requirements for the Mobile application:

REQ-11: The app shall ensure reliability, durability, concurrency, security and correctness.

REQ-12: The app shall ensure that the displayed data is not pixelated but easily readable.

REQ-13:The app shall not freeze or lag.

REQ-14:The app shall be compatible with android devices.

REQ-15:The app shall allow users to Sign up using a valid email id and password.

REQ-16:The app shall allow users to sign in using a pre registered email id and password.

REQ-17:The app shall allow users to view the items added to their cart using the touch screen.

REQ-18:The app shall allow users to view description of the products in their cart.

REQ-19:The product description shall include an image of the product, product name, product brand, price of a single unit, SKU (Stock Keeping Unit), aisle number indicating where the product is located in the store and a brief description about the product highlighting its features.

REQ-20:The app shall show the count of number of items in a user’s cart.

REQ-21:The app shall compute and display the total cost of the purchase (i.e. items in the cart).

REQ-22:The app shall allow users to delete items in the cart.

REQ-23:The app shall display a list of items they can purchase using the app.

REQ-24:The app shall display some user information to greet the user on the shopping page.

REQ-25:The app shall allow users to view description of the products without having to add those in user’s cart.

REQ-26:The app shall allow users to add items to cart from the list of items available to purchase.

REQ-27:The app shall allow users to update their cart i.e. add or remove an item in a single click.

REQ-28:The app shall allow users to sign out without the risk of losing their information.

REQ-29:The app shall allow users to sign back in using same account or a different account after signing out.

REQ-30:The app shall work optimally when connected to internet.

REQ-31:The app shall be able to synchronize with the database and hence the touch screen in real time.

REQ-32:The app shall display most up to date information.

REQ-33:The app shall reflect changes made to the items database (for example price change) in real time.

REQ-34:The app shall be user friendly.

REQ-35:The app shall be easy to use.

REQ-36:The app shall allow user to sign up for an account in minimal number of steps.

REQ-37:The app shall have an easy to read display.

REQ-38:The app shall have display items in user’s cart as list.

2.4.3 Touch Screen Interface  
 2.4.4 Networked Platform Communication Software  
 2.4.5 Microcontroller Firmware  
2.5 Project Overview  
 2.5.1 Bill of Material

* Raspberry Pi 3 Model B $49.99
* PIR Sensor $4.00
* RFID Reader with Tags $35.00
* Touch Screen 7’’ $50.00
* 3D printed Case $00.00
* Jumper cables $4.00
* Resistor 2.2K Ohm & 3.3K Ohm $1.00
* Project PCB Board Slotted Bus type $3.99
* 40 pin Female to Male connector Long $0.60
* Lead free Solder with flux core (Electric Grade) $8.50
* 5mm 3.3VDC LED $0.25

Note: All the above equipment was ordered from Amazon through vendor Adafruit and Lamptron

Database (Guru): the cost of building and maintaining the database is negligible. We utilized Google’s Firebase for the project which gives developers an option to signup for free for its “Spark Plan” using an already existing google account or making a new one. Internet access charges do apply when using on a paid network, however in case of free wifi on public places or accessible school network no/negligible internet charges may apply.

App (Guru): The cost of making the app was $0.00 as the developer already owned the tools necessary for developing the app, such as a Laptop with Android Studio installed and an Android phone to test the app. Also, no paid libraries were used for the development so the cost for building the app was zero dollars.

### 2.5.2 Time Commitment

The whole project from scratch and no build instructions might take around 10 to 12 days to make depending on skill level. But since the project has now been made once and one can follow the build instructions, it will only take 3 to 5 hours to make provided that all parts and code is available on hand before starting.

Database (Guru): The total time committed for creation, update and maintenance of database was more than 6-8 hours. This included a couple of hours of exploring different option, about 5 hours of learning about Firebase, its features, usage guidelines and following tutorial for initial setup. Approximately 1 to 2 hours were dedicated to come up with informal database structure compatible to client needs and atleast 2-3 hours were dedicated to implementing the database on Firebase (for all iterations including bug fixes in the structure to make it easily accessible from within the app). For anyone to follow our instructions, the setup would take less than 30 minutes, given the size of inventory is small (for larger inventory, the time may exceed).

Android App (Guru): The app creation task requires dedication and patience. For a beginner it requires some extra time devoted just to read documentation and examples on android studio’s website as well as following tutorials for setting up certain components of the app. There are times when certain components do not work as expected; careful debugging is required at those times. Therefore when developing the app one must dedicate some extra time in case some debugging is required. For the purpose of this project the developer devoted on average 8 -10 hours each week. If using just the code provided in the appendix, the development may take 1 to 2 hours of work.

2.5.3 Mechanical Assembly

**For RFID**

Build Instructions for RFID Reader Project

**Table of Contents**

1. [Introduction](https://github.com/gurusharma/RFID/blob/master/README.md#introduction)
2. [Bill of Materials](https://github.com/gurusharma/RFID/blob/master/README.md#bill-of-materials)
3. [Time Commitment](https://github.com/gurusharma/RFID/blob/master/README.md#time-commitment)
4. [Mechanical Assembly](https://github.com/gurusharma/RFID/blob/master/README.md#mechanical-assembly)
5. [PCB or Soldering](https://github.com/gurusharma/RFID/blob/master/README.md#pcb-or-soldering)
6. [Power Up](https://github.com/gurusharma/RFID/blob/master/README.md#power-up)
7. [Unit Testing](https://github.com/gurusharma/RFID/blob/master/README.md#unit-testing)
8. [Production Testing](https://github.com/gurusharma/RFID/blob/master/README.md#production-testing)
9. [Reproducible](https://github.com/gurusharma/RFID/blob/master/README.md#reproducible)

### **Introduction**

On this page, you will find the instructions for building your own project on Raspberry Pi 2 utilizing RFID Serial tag reader. This project can be used to identify different RFID tags. RFID tag readers are already used in many applications, such as automatic user identification, robotics, inventory tracking and payment systems. This project is part of a bigger project where we plan to utilize the functionality of RFID reader to authenticate users of a shopping mall to conveniently login and browse helpline service.

##### **Note:**

Please note that I will be referring to Raspberry Pi 2 as Pi occasionally.

### **Bill of Materials**

You would need a budget of approximately $200.00 if you do not own or have access to any of the parts listed below. It is possible to obtain better deals, so feel free to do a little research before making a purchase.

|  |  |  |  |
| --- | --- | --- | --- |
| **Item** | **Quantity** | **Cost** | **Supplier** |
| Raspberry Pi 2 (Includes USB cable connectors) | 1 | Only Pi: $ 45.95; Ultimate Starter Kit: $119.95; Complete Starter Kit: $99.95; Basic Starter Kit With Power Adaptor: $59.95 | [Cana Kit](https://www.canakit.com/raspberry-pi/raspberry-pi-kits) |
| RFID Card Reader and Tags | 1(reader) + 3(tags) | $ 59.99 | [Parallax Inc](https://www.parallax.com/product/32390) |
| Female-Female Jumper Wires | 6 | $ 3.95 \* | [adafruit](https://www.adafruit.com/product/266) |
| Resistors | 2 | $ 6.86 \*\* | [Amazon](https://www.amazon.com/dp/B01ERPXFZK/ref=sspa_dk_detail_3?psc=1&pd_rd_i=B01ERPXFZK&pd_rd_wg=MK4GM&pd_rd_r=60TKJ3G1SDZ08G6YW486&pd_rd_w=REy7O) |
| Soldering Kit (with Soldering Iron) | 1 | $ 31.00 \*\*\* | [Amazon](https://www.amazon.ca/Primacc-Adjustable-Temperature-Controlled-Interchangeable/dp/B06XCZC4PF/ref=sr_1_3?ie=UTF8&qid=1516578338&sr=8-3&keywords=soldering+kit) Humber Prototype Lab |
| 3-D Case for RFID reader | 1 | $ 1.80 | [Toronto Public Library](https://www.torontopubliclibrary.ca/using-the-library/computer-services/innovation-spaces/3D-design-print.jsp) |

\* Premium Female/Female Jumper Wires - 40 x 6" (you can strip wires to make individual ones)

\*\* Elegoo Electronics component pack with resistors, LEDs, Switch, Potentiometer for Arduino UNO, MEGA2560, Raspberry Pi

\*\*\* Primacc Soldering Iron Kit

##### **Note:**

Prices may change over time and some of the prices are in USD, so, please verify before making a purchase.

### **Time Commitment**

This was part of a project course, divided over a period of 12 weeks, but the actual work hours spent on it were 3 to 5 hours every week. If the project is the only thing you have to do, then it should take less than a week, depending on how you divide tasks and deal with delays associated with finding and purchasing items. The table below gives an approximate time breakdown per task:

|  |  |
| --- | --- |
| **Tasks** | **Time Required to Finish Successfully** |
| Look for parts and make a purchase \* | 2 hours |
| Wait for delivery | 1 day to 1 week |
| Soldering and Testing Voltage Divider | < 1 hour |
| Create .stl file for custom case 3D printing | < 1 hour |
| Print the 3D case | ~ 2 hours |
| Configure the Pi and download libraries | ~ 2 hours |
| Project setup | < 10 minutes |
| Demo and Testing | < 30 minutes |

\* Less than 30 Minutes if you follow the links provided in the previous section

From the breakdown above, it is clear that the project does not take very long to complete. If you follow the instructions correctly and are dedicated, you can build your own prototype in less than a week.

### **Mechanical Assembly**

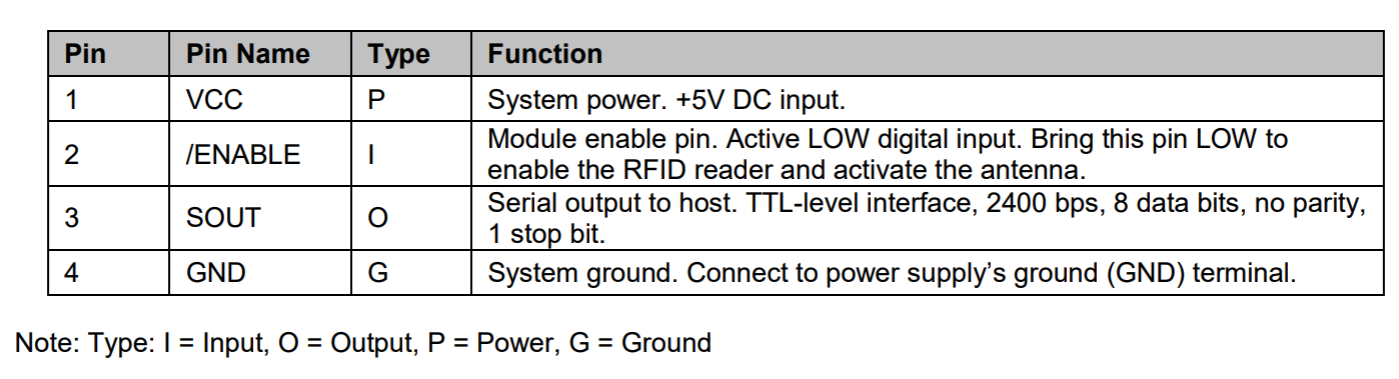
The Parallax Serial RFID (Radio Frequency Identification) Card Reader (#28140) can be connected to any host microcontroller easily using only four connections. For the purpose of this project, I am using Raspberry Pi 2. There are 4 pins on the RFID reader and following table (from official documentation) lists their type and functionality: [](https://github.com/gurusharma/RFID/blob/master/RFID%20Pins.PNG)

Figure 1: RFID Serial Card Reader's Pin Type and Functionality

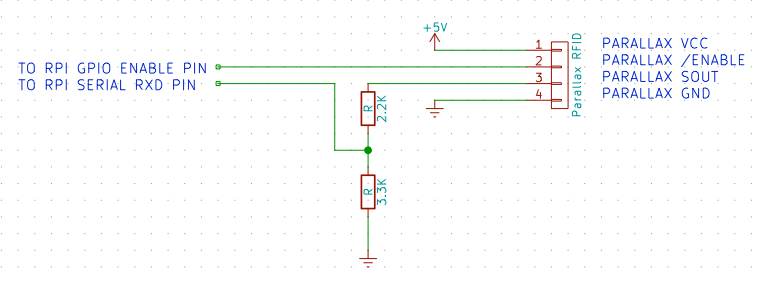


Figure 2: Schematic for connecting RFID Serial Card Reader to Raspberry Pi 2

Use the schematic in Figure 2 above, to establish connections. The SOUT pin of the RFID reader is connected to a voltage divider. Use a 2200 Ω (R1) resistor and a 3300 Ω (R2) resistor to reduce 5 volts coming out of the RFID reader's SOUT to about 3 volts. This is done to avoid the risk of damaging Pi with higher voltages. You can solder these resistors along with a jumper wire to create the voltage divider as shown in the picture below (also put a heat shrink on the soldered part). Make sure you check the output of voltage divider using a multimeter before incorporating it in the assembly.



Figure 3: Voltage Divider

Now create the circuit in Figure 2; this should not take more than 10 minutes if done carefully.

#### **Linux Configuration:**

Use raspi-config to ensure that the shell is configured to run on serial port. From the main menu, select “Advanced Options” followed by “Serial”. Select “No” for the prompt to login shell over serial port. “Finish” and save configuration. Before you reboot also check that the “enable\_uart” filed in “/boot/config.txt” is set equal to “1” and not “0”. Now reboot.

#### **Install Python Packages:**

You may choose to program in any other language compatible with Pi. For the purpose of these build instructions, I will explain steps of programming Pi using Python. You will need to have “Python GPIO” and “Serial” packages installed. Raspberry Pi Linux distribution usually have some python packages installed. In case they are missing you can use the following commands to install it:

sudo apt-get install python  
sudo apt-get install python-dev  
sudo apt-get install libjpeg-dev  
sudo apt-get install libfreetype6-dev  
sudo apt-get install python-setuptools  
sudo apt-get install python-pip

After installing Python, you may now install above mentioned packages using:

sudo pip install RPi.GPIO  
sudo pip install pySerial

If you run into any issues with installation, follow this [link](https://jeffskinnerbox.wordpress.com/linux-python-packages-for-my-raspberry-pi/).

#### **Write a Python Script:**

You can download the script [here](https://github.com/gurusharma/RFID/blob/master/rfid.py) and modify as needed. This script simply sets up Pi’s serial port and GPIO header. It identifies the tags being read by the reader inside a while loop. The reader reads the input tag when it is enabled low. When you run the code, it makes reader enabled low and goes into the while loop and waits for the tag to be read. It is worth knowing that the RFID tag has 12 bytes of data, the validate\_rfid function ensures that the tag being read has 12 characters.

#### **Print a Case for the Sensor:**

I utilized [TINKERCAD](https://www.tinkercad.com/) to create .stl file needed for 3D printing. You may utilize the [file](https://github.com/gurusharma/RFID/blob/master/RFID%203D%20case/RFID%203D%20case.stl) I created and edit it as needed or use any other freely available online tool of your choice to do it from scratch. Once you are satisfied with .stl file, you may get it printed from any facility of your choice. If you choose to do it with Toronto Public Library, make sure to read the detailed instructions on their [website](https://www.torontopubliclibrary.ca/using-the-library/computer-services/innovation-spaces/3D-design-print.jsp).



Figure 4: 3D Printed Sensor Case

##### **TIP:**

The software they use for reading .stl file and printing is “CURA”, which by default tends to select the thickness of printing thread to be 0.4 mm, which usually takes more than 2 hours to finish. Make sure to change this thickness to 0.6 mm (recommended) or 0.8 mm in order to get the print done in 2 hours. In case you still have any issues, ask the staff; they are very helpful.

### **PCB or Soldering**

Only soldering needed for this project is the one needed to create a voltage divider circuit. Here are the steps:

1. Get the following parts:
   1. two resistors R1 (2200 Ω) and R2 (3300 Ω)
   2. one jumper wire and heat shrink
   3. soldering iron
2. Strip both ends of the wire
3. When soldering iron is ready to use:
   1. solder two resistors at one end and one end of the jumper wire between them
   2. now put the heat shrink on the soldered part

Following these steps, you will get your voltage divider ready. Make sure to check the output voltage using a multimeter.

### **Assembly**

Now assemble all the parts as shown in the image below and put the RFID tag inside its case. Also, you may utilize any 'ground pin' for assembly.

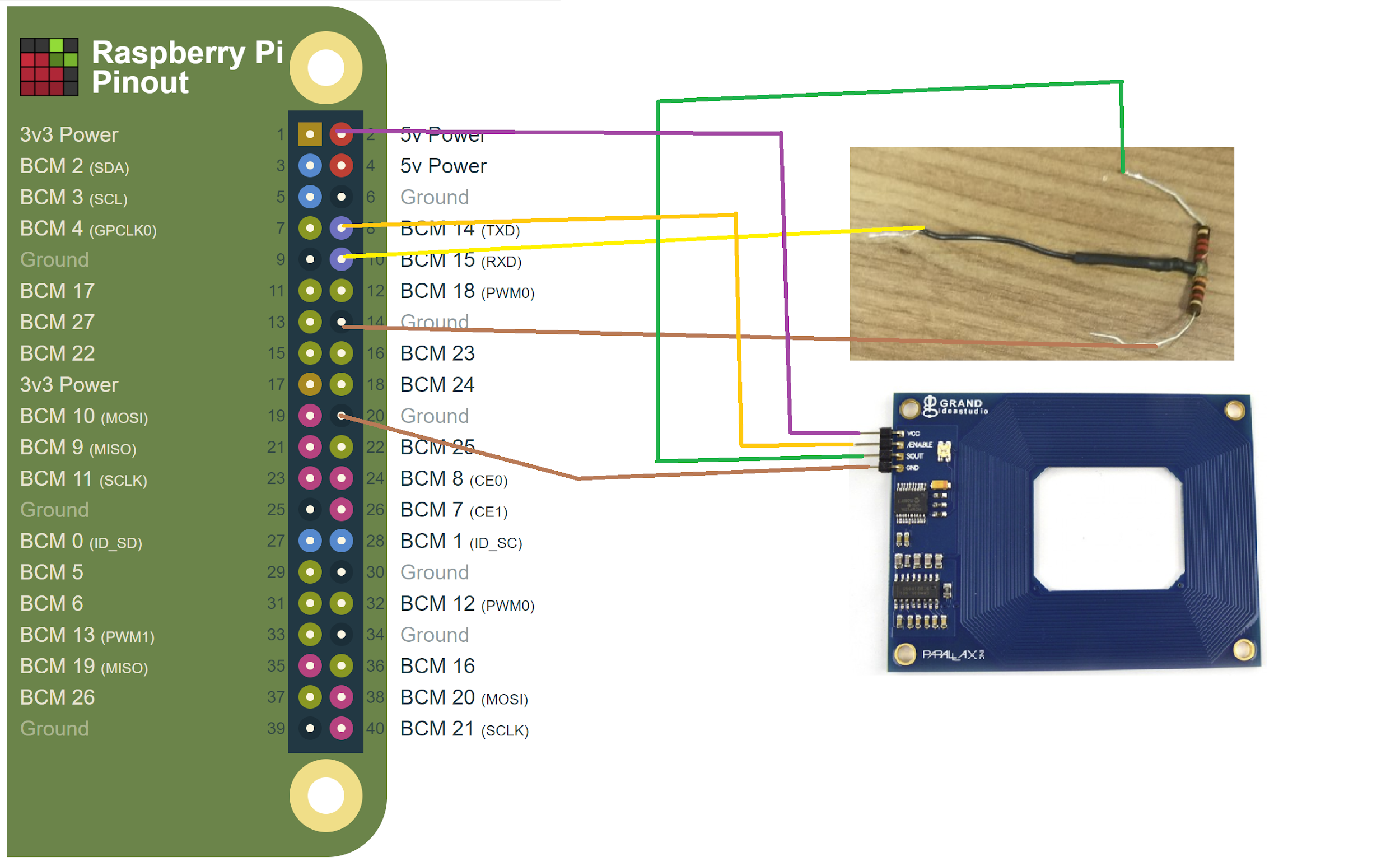


Figure 5: Circuit Assembly with Ports

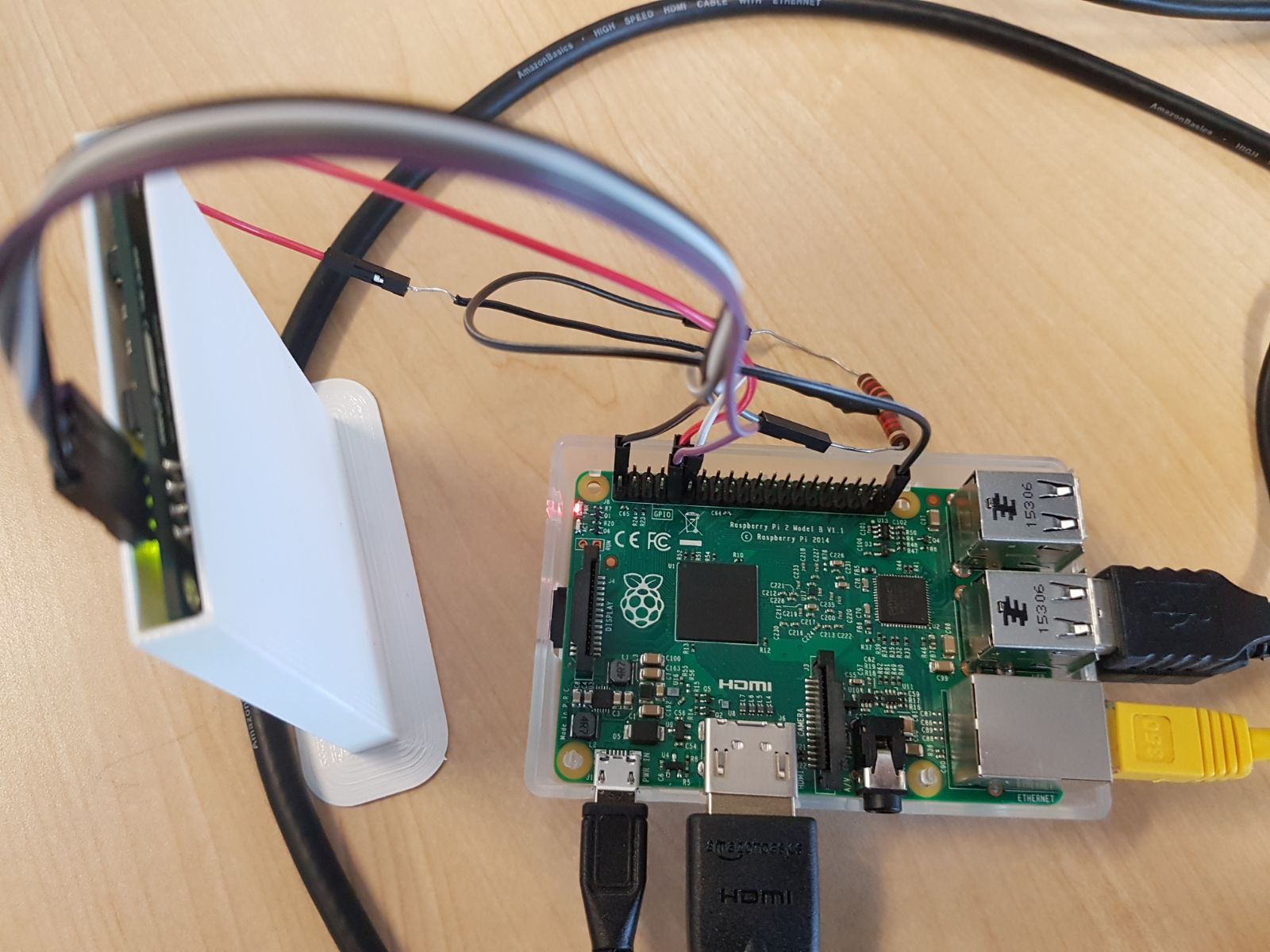
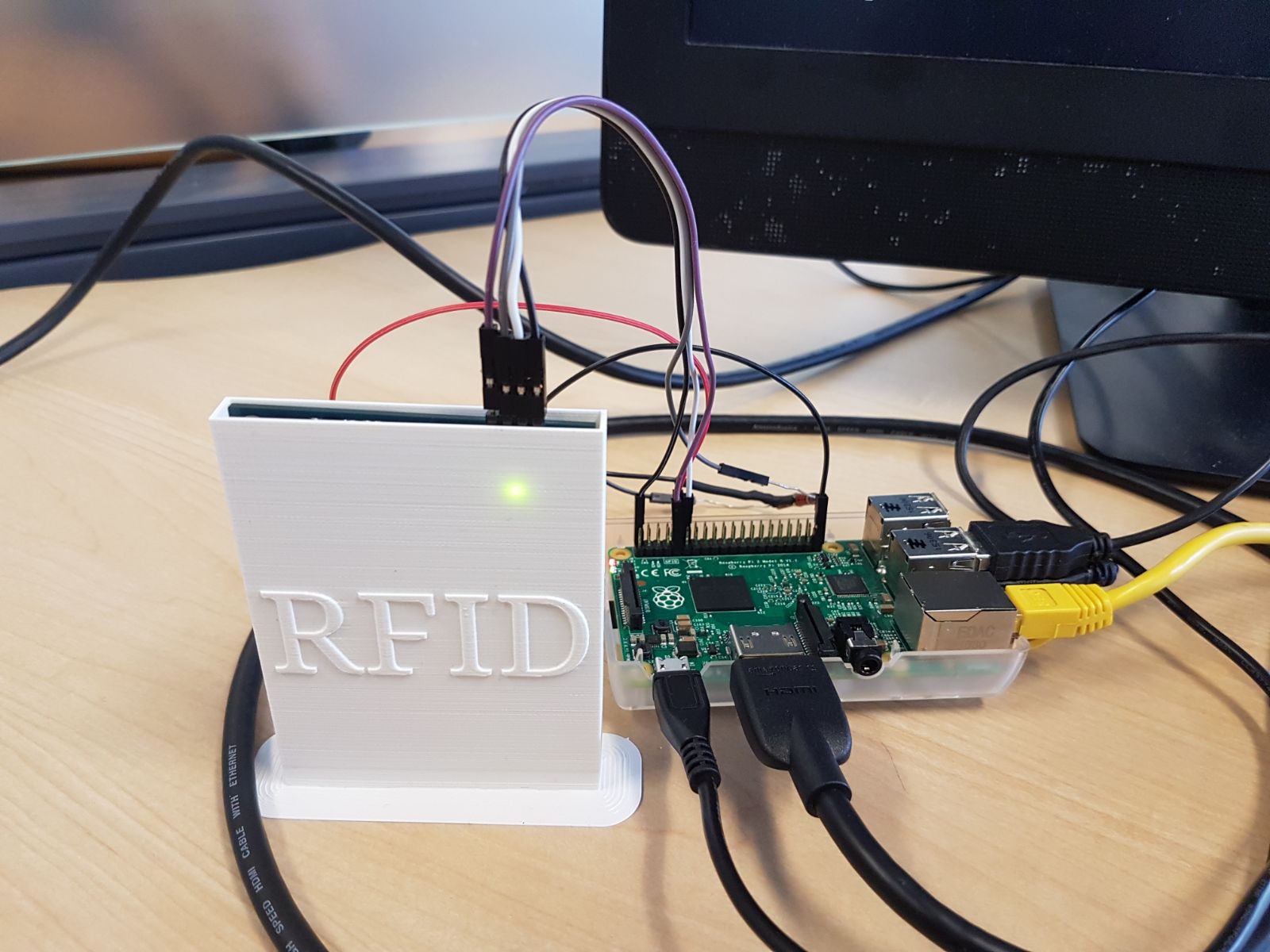
 [](https://github.com/gurusharma/RFID/blob/master/Assembly2.jpg)

Figure 6: Complete Assembly

**For Motion Sensor**

Ignore the Camera part in the instructions below

## Stuff you need (All the prices are from amazon.ca and may change by time you see them):

* Raspberry Pi 3 with a case and a power-supply(I only worked on a pi3, but its only a simple python code so it should run on other models, but I am not sure of it

-- $79.99 + Tax

* PIR Sensor

-- $8.99 + Tax for 1 sensor

-- $12.99+ Tax for 3 sensors

* Female-Female jumper wire

-- There are options from $6 for a 20 pack, you need only 3 wires for this project.

### Extras(This is just to load the libraries and code to the pi)

* HDMI Display
* KeyBoard
* Mouse

## Sensor Housings

I used the case in the kit for the pi, and I 3D printed housings for the sensor. The .stl files for them are in the [zip forler](https://github.com/AbhaySingla/college/blob/master/3d%20printing.zip).

This process might take you upto a day.

## Mechanical Assembly

## Take the PI out of the package.

1. Put the PI in the case, and then add the sd card to it.
2. Add the PIR sensor to the PI

## Setting up RaspberryPI

1. Connect the display in the HDMI port, connect the keyboard and he mouse and also give the pi an ethernet connection.
2. On the first boot, with NOOBS installed on your sd card, you will be asked to install the OS.
3. Select the raspbian OS, and wait for it to install. This might take upto 15 mins. [](https://github.com/AbhaySingla/college/blob/master/20171216_122916.jpg)

## Start (This whole process will take you upto 4 hours.)

#### Testing the PIR sensor:

1. Connect the sensor as shown in the schema (VCC- pin2, GND- pin6, DigitalOutput- pin7)
2. Open the console
3. Open a new python script
4. sudo nano pirtest.py
5. Write the following code in the the file:

import RPi.GPIO as GPIO   
 import time  
   
 GPIO.setmode(GPIO.BOARD)  
 pir = 7  
 GPIO.setup(pir, GPIO.IN)  
 print ("Waiting for sensor to settle")  
 time.sleep(2)  
 print ("Detecting motion")  
 while True:  
 if GPIO.input(pir):  
 print ("Motion Detected!")  
 time.sleep(2)  
 time.sleep(0.1)

1. Make the scipt excecutable
2. sudo chmod u+x pirtest.py
3. Execute the script, to exit the script use Ctrl. + c
4. python pirtest.py
5. If test the sensor by moving infront of it, remember there is a 2 seconds delay in between two readings in the script provided above.

2.5.4 PCB and Soldering

There is not a major soldering involved in the making of this project and the PCB board used is fairly simple

2.5.5 Power Up

After successfully completing the tasks above, you are ready to power up. Make sure your circuit connections are correctly established; fix errors in the setup if you encounter any. Now plug the USB into Pi and connect to a computer device. In order to get the output displayed on computer, plug in the HDMI cable. Turn on the Pi. Make sure you have all the packages listed above installed in order to run the python script.

2.5.6 Production Testing

Our project at present state takes 5 hours to build and is not feasible on a production line to make. To replicate this project on a production line have to be made some major cosmetic change and designing of the case whole project itself the software won't be affected as much got the look and feel of the project has to be changed completely.

2.6 Problems Encountered  
 2.6.2 Leaks in Casing

Since our project is designed to work indoors, leaks were not tested.

2.6.3 Limited Space for Electronics

Space behind the pi was enough to store all extra components and wirings.

2.6.4 Data Communication  
2.7 Approaches  
 2.7.2 Leaks in Casing

Since our project is designed to work indoors, leaks were not tested.

2.7.3 Limited Space for Electronics

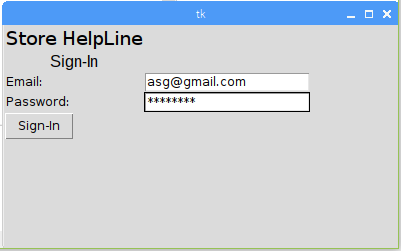
Space behind the pi was enough to store all extra components and wirings.

2.7.4 Local Data Communication

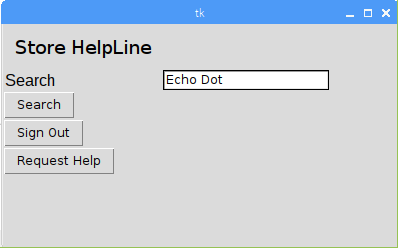
## 2.8 Walkthrough of the system



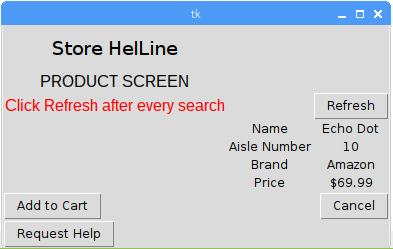
As, one walks in front of the screen the PIR sensor detects motion and the code switches from generic advertisements the firs GUI screen that is the sign-in screen



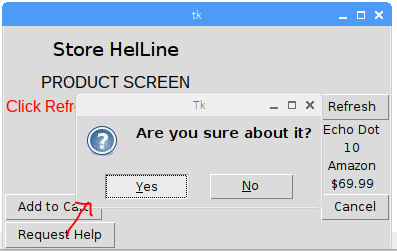
Once logged in correctly, the GUI askes for the Item that you have to search.



Hitting the search button, the page is redirected to the product detail page.



At any point of the in the process if the user needs any kind of assistance, he/she can click the Request Help button to call a store associate.



## 2.8.1 Database (Guru)

For the purpose of this project we utilized Google’s Firebase real-time database.

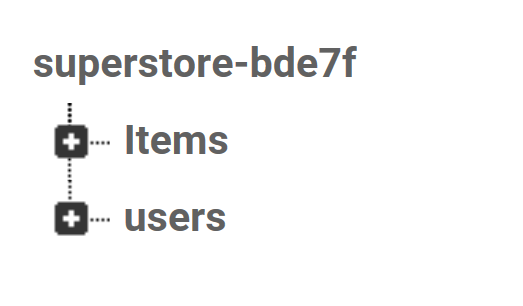
#### Why Firebase?

Here is a brief description of features offered by Firebase that influenced our decision to utilize it for our project. Firebase allows to store and sync data between users and devices in realtime using a cloud-hosted, NoSQL database. It also allows to sync updated data across all connected devices in milliseconds, and the data remains available while the app goes offline, providing a better user experience. It allows app’s clients to share one Realtime Database instance and automatically receive updates with the newest data. Another reason to choose Firebase was that it provides developers opportunity to signup for various features free of cost with “Spark Plan (Generous limits for hobbyists, Free)” and then pay for any exceeded usage making it easier to scale. Firebase is easy to use, maintain, and scale. Minimal requirements to create a firebase database is internet connection and a google account. Using Firebase we could now connect both the app and the touch screen with the same database instead of creating one database for each.

#### Structure and Features

This section describes the structure and features of the database created using Firebase.

Illustration 1 below is the overall structure of our Firebase database collapsed, showing names of two separate tables : “Items” and “users”. Further sections discuss the details in each of these table and their purpose.



### Illustration 1. Overall view of Firebase Database

For our project we use Firebase to store the information about each “Item” in the store catalog. This information includes: SKU (Stock Keeping Unit), name, price (of single unit), brand, an image url, brief description and an aisle number to locate the item physically in the store. The items are stored in “Items” table in the database with the unique SKU number being the primary key for each item; refer to Illustration 2 and Illustration 3 below for details.

### Illustration 2. Items Table View Expanded

### 

### Illustration 3. Single Item View Expanded

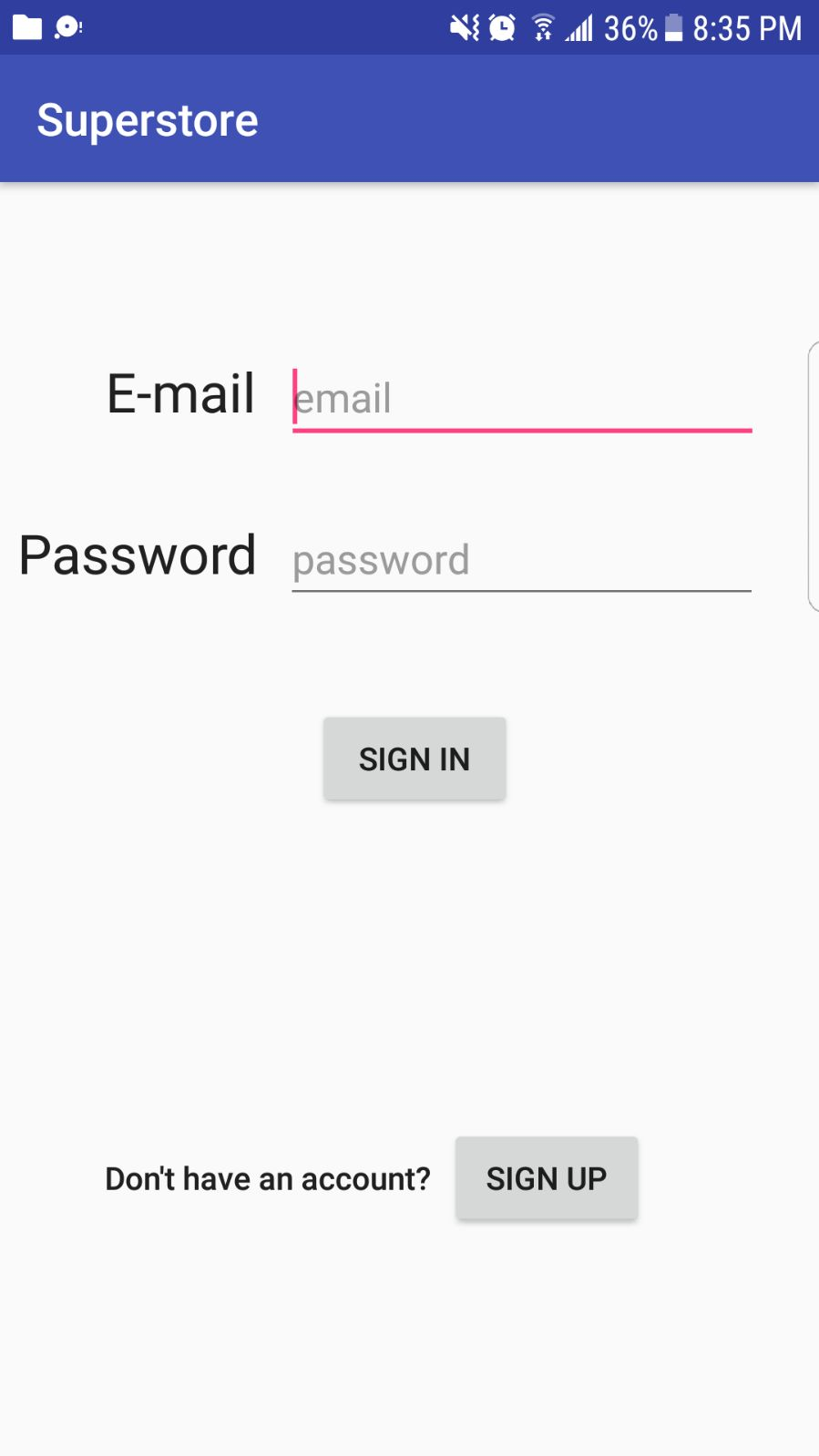
Our database has another table dedicated to our users. Each user after successfully signing up, with our app, is assigned a unique user id (automatically generated by Firebase). This unique id is the primary key for “users” table as illustrated by Illustration 4 (a). Each user’s email (used for sign up) is kept as a name attribute to greet the user on signing in. For future iteration of the app we would allow users to set up their display name at signup, but is currently beyond the scope of this project. Each user “node” has a child node called “cart” (see Illustration 4 (b)) which comes into existence when user adds an item to her/is cart using either the mobile interface or the touch screen interface (Illustration 4 (b)). The cart contains SKUs of all the items that user adds utilizing either of the interfaces. SKUs in user cart are assigned a value of ‘true’, which is sufficient for current iteration as for the purpose of current project users can add an item to the cart only once. More details such as an integer value representing quantity of each item can be added for further iteration of the app.

### Illustration 4 (a). Users Table View Expanded, (b). Expanded view of Users with and without items in a cart

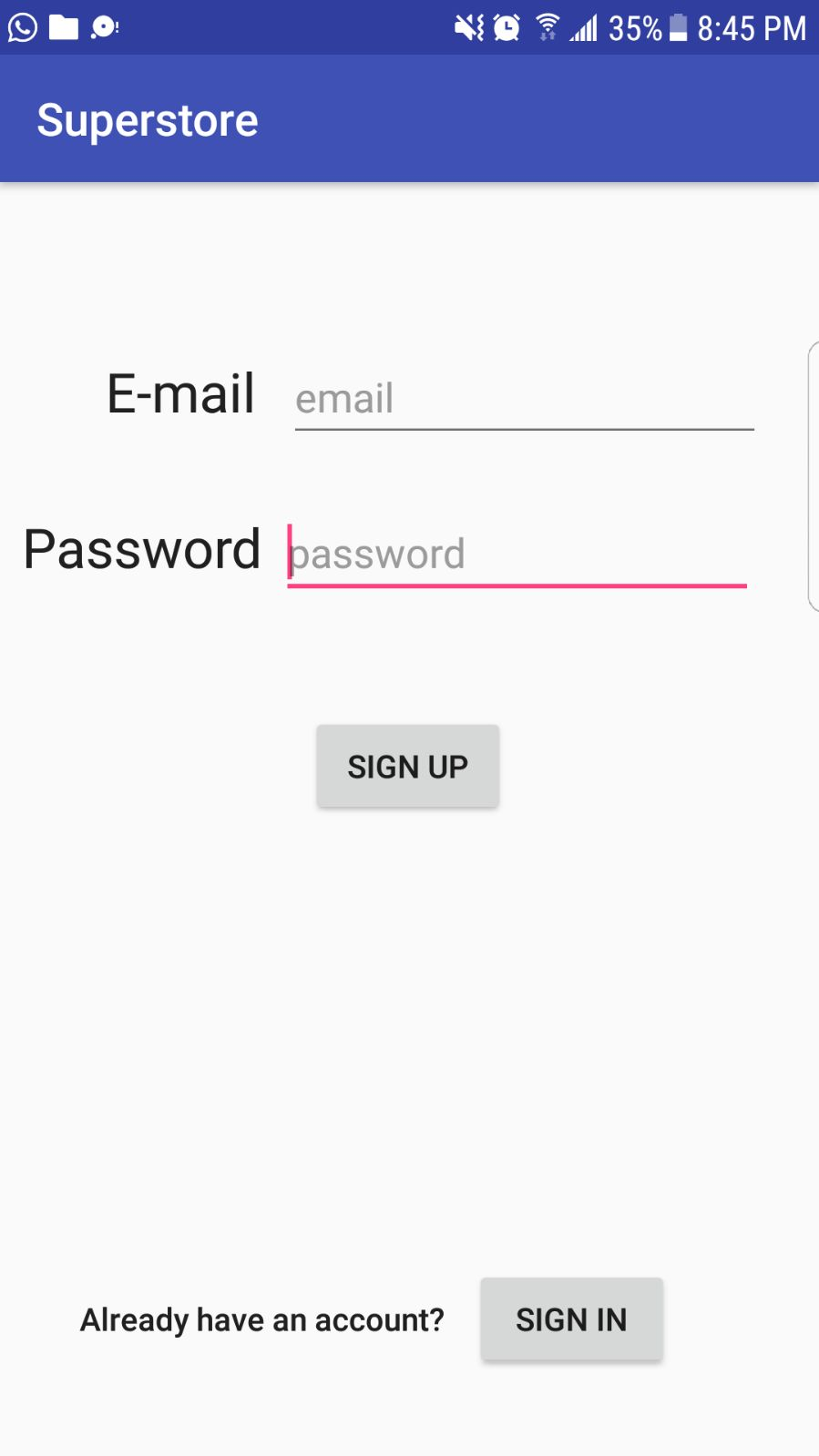
Unit testing suggests that this Firebase database works as expected and synchronizes information in realtime with both the Android app and the touch screen. The database is easy to edit and update. The current design of the database is satisfies all of the specification specified in section 2.4.1 above.

### 2.8.2 Android App (Guru)

For implementation of the mobile application we chose to build an android app that could be easily run on android devices (phones and tablets) as android users are a large proportion of the population (in North America). However for the future iteration of the product, we plan to extend it for ios users as well. Current iteration of the app is compatible with systems are running on API (Application programming interface) 21 and above. The app development has been done utilizing Android Studio IDE (Integrated development environment). The programming languages used are java for app functionality implementation and XML (Extensible Markup Language) for layout design. This section describes the functionality and the features offered by the app.



### Illustration 5. App’s Home screen with options to “SIGN IN” or SIGN UP”



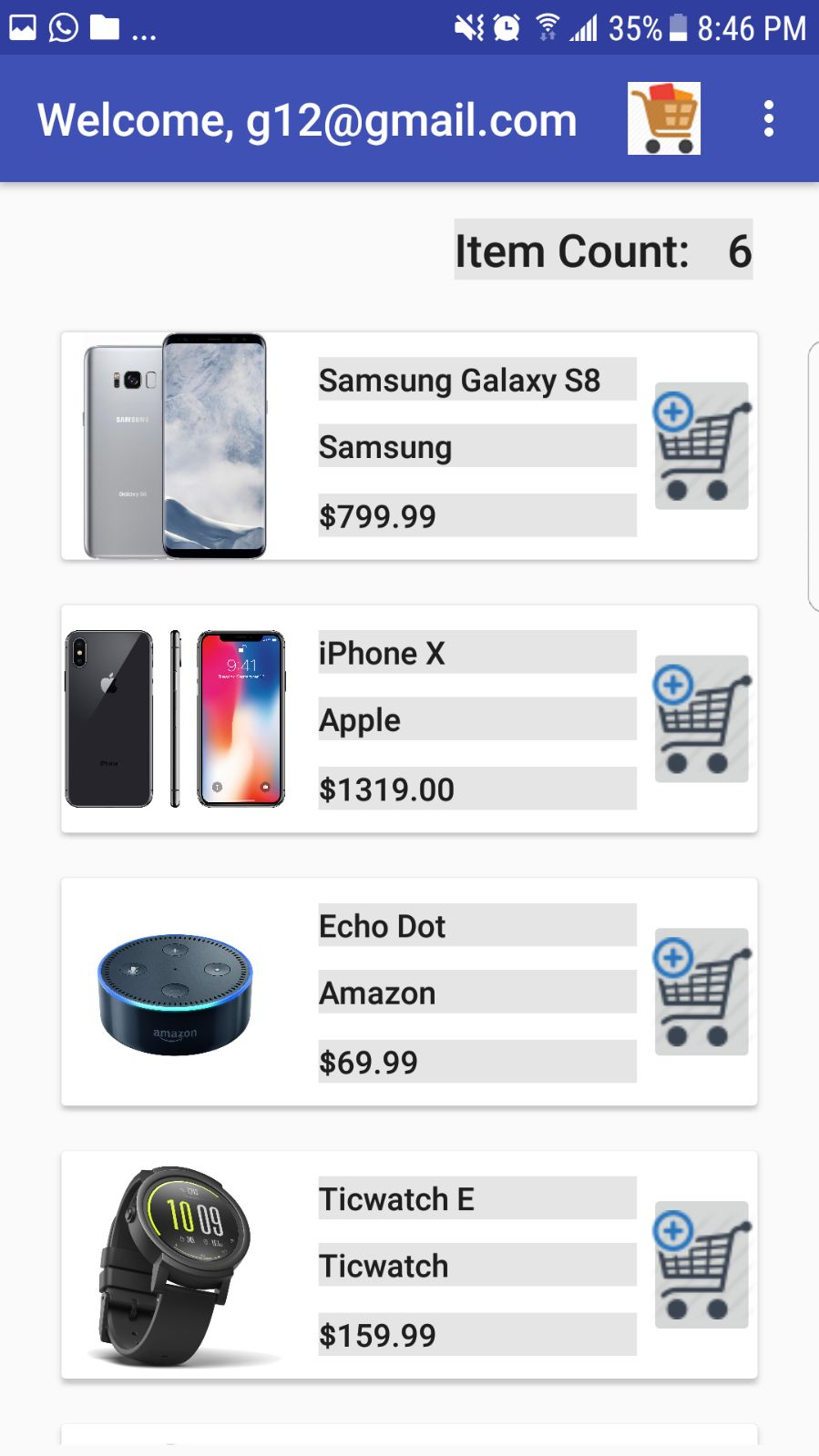
### Illustration 6. App Home Screen with “SIGN UP” page for new users and an option to “SIGN IN” for existing users

### 

### 

### 

The android app is called “Superstore”; once installed successfully on the system, the app presents the user with a home screen (as shown in Illustration 5 and Illustration 6) asking to either “SIGN UP” or SIGN IN”. Input fields on both of these pages have validation on them in order to make sure the user enters the data in the right format. For a new user, the app allows the user to signup with a valid email and password. The valid email format is “local-part@domain”, for example “[username@example.com](mailto:username@example.com)”. The valid password should be atleast 8 characters long. Once the user signs up, the information is updated to the Firebase database in realtime, successfully registering the user. The user can now sign in to the app with the same credentials used while registering. Again the format of the email and password should be valid.



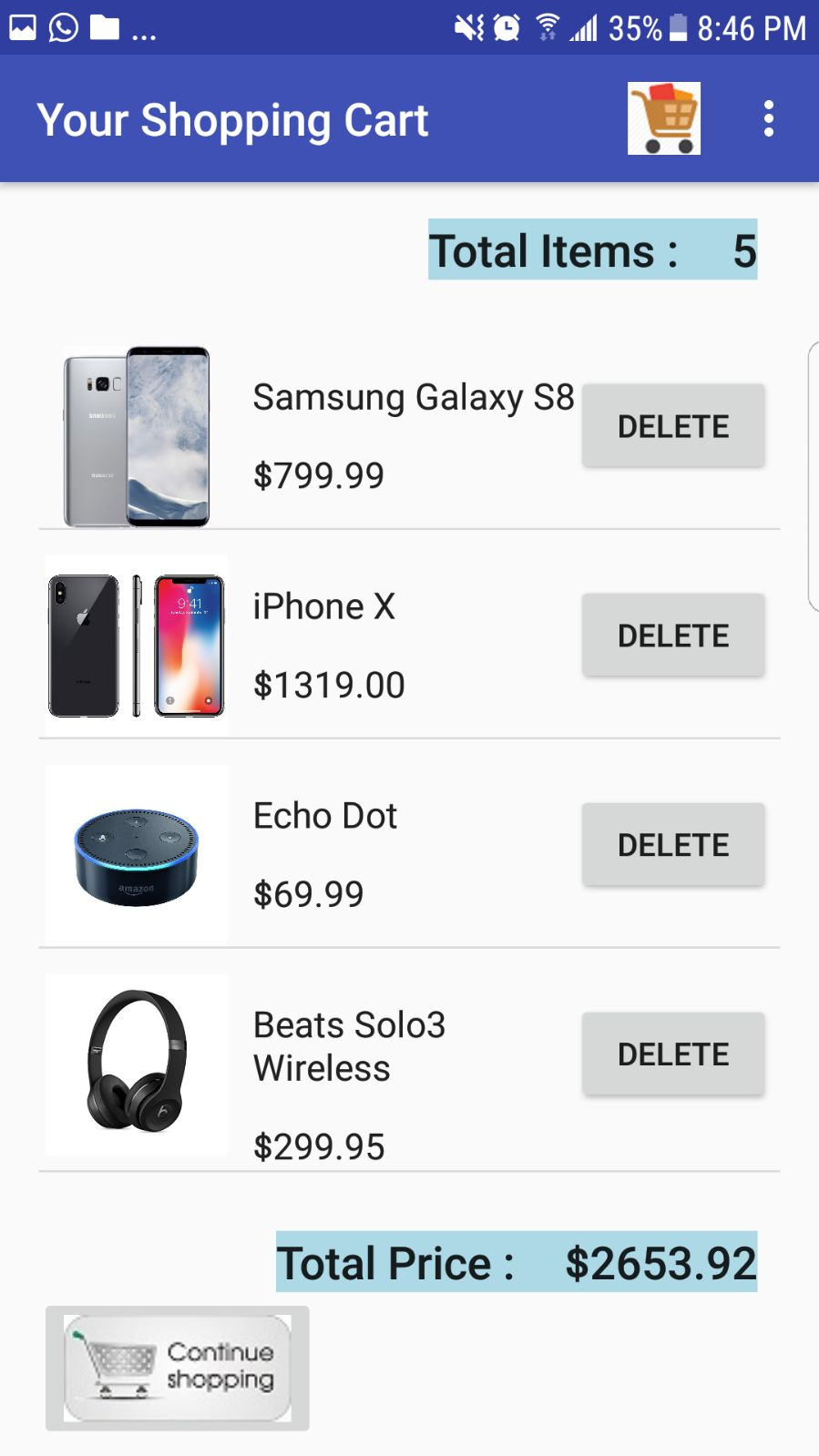
### Illustration 7. App’s Welcome page with Catalog (Item Count reflecting items in online catalog displayed)

After a successful sign in the user sees the item catalog with a list of items as shown in Illustration 7 above. The user email used to sign in is also displayed at the left of top “Action Bar” as “Welcome, [username@example.com](mailto:username@example.com)” to let the user know what account are they signed in from. The list of items is displayed using RecyclerView(an advanced and flexible version of the ListView, where different components work together to display data) with cards inside RelativeLayout. This is done in order to show each item in the catalog with partial details including Item name, item brand, item cost and an image.

The catalog page also displays total number of items available in online catalog as “Item Count”. There is also a clickable ‘add to cart’ image button with relevant visually appealing icon. Users may click on an item to see more detailed information about the item. This detail includes a larger image of the product, its name, aisle number (where the item can be located in the store), unique SKU (Stock Keeping Unit), product brand, price and small description illustrating product features. The detail is displayed on the “Product Description” page; refer to Illustration 8 below for illustration. The user can navigate to the “Cart” by either clicking on the cart icon located at the top right (left to “action overflow” icon with three dots) of the “Action bar” or by clicking “BACK TO CART” button from within the “Product Description” page. The app allow users to view description of the products without having to add those in user’s cart.



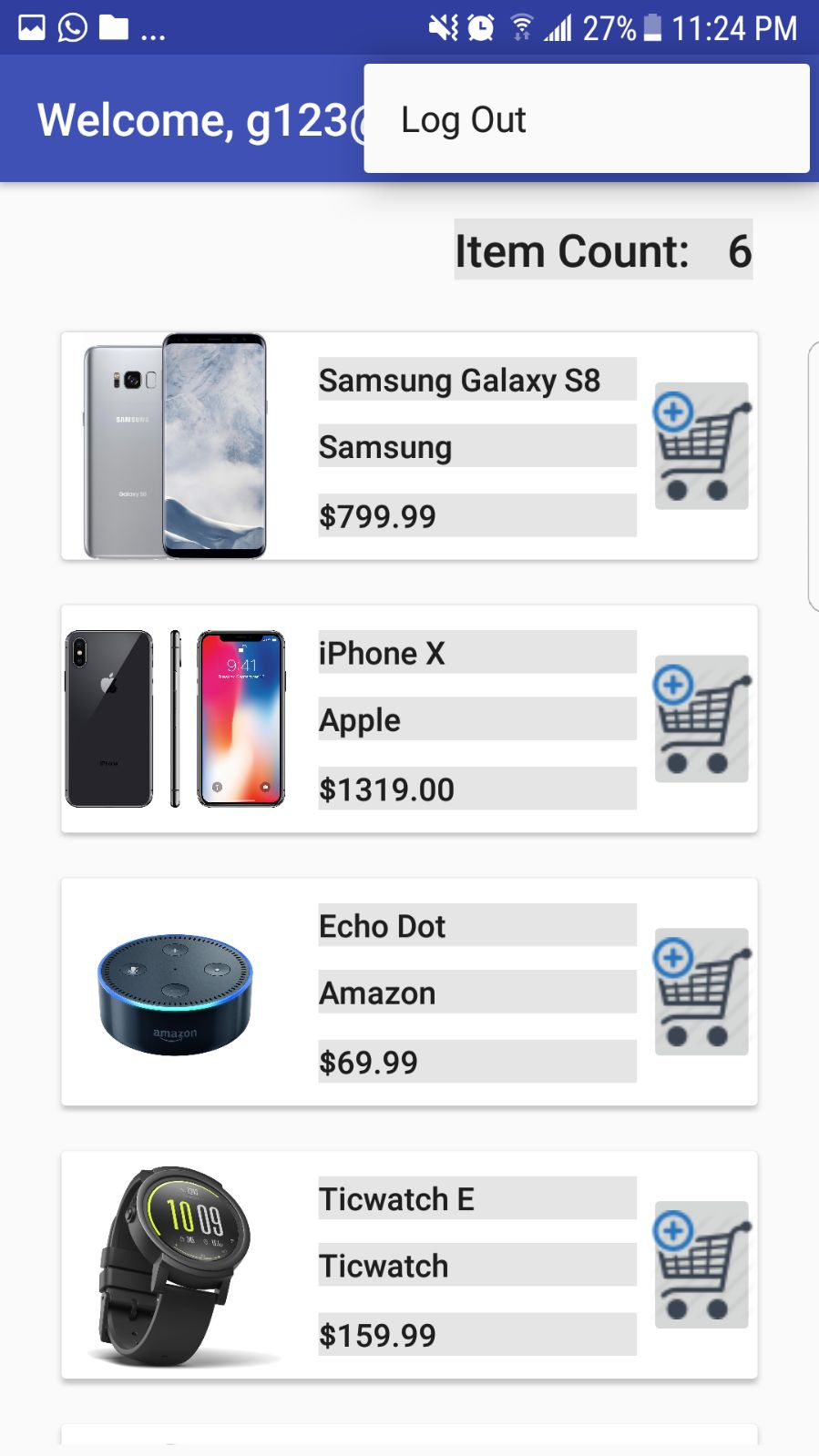
### Illustration 8. “Product Description” page for Samsung Galaxy S8 (Please note that the Description view is scrollable)



### Illustration 9. “Shopping Cart” page for a user with 5 items in Cart (Total items and Total Price displayed)

The “Cart” page as shown in Illustration 9 above indicates, total number of items added to user’s cart either via touch screen or using the app itself. This information is displayed above the list of items as “Total Items: 5” (for example). The “Action Bar” text on the left says “Your Shopping Cart”. The Cart page uses a customized list view created by the developer. This view for created to provide users with more detailed and visual information about the products in their cart. Each item in the cart now displays product image on the left, product name and price of the single unit. The also has an integrated “DELETE” button which allows users to delete that particular item from the cart (if they change their mind to make the purchase). Similar to the catalog, if the user clicks on an item in the cart, she/he would be able to view the detailed product description: a larger image of the product, its name, aisle number (where the item can be located in the store), unique SKU (Stock Keeping Unit), product brand, price and small description illustrating product features, as app navigates to “Product Description” page.

The user may log out of the app any time by clicking “action overflow” icon with three dots on the “Action Bar” and selecting the “Log Out” option (as shown in Illustration 10 below) without losing any of the information. The cart page also computes and displays the total cost for the items in a user’s cart. This information is displayed at the bottom of the list of items as “Total Price”. The unit of currency used is dollar and the total cost does to include GST. The value in “Total Price” and “Total Items” is updated right away when the user deletes an item. The same applies when user adds an item to the cart. The user may click on “Continue shopping” button at the bottom left of the cart page. This functionality allows users to add more items to the cart if desired.

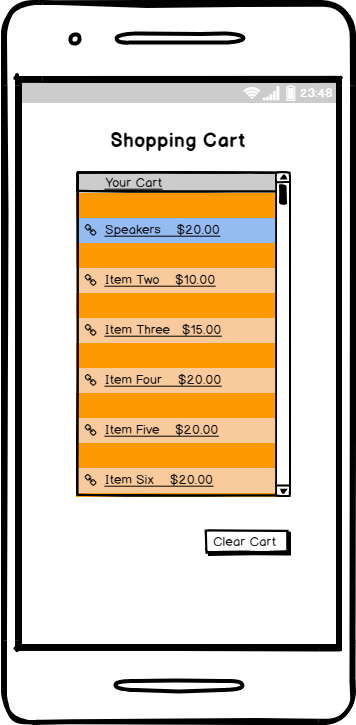
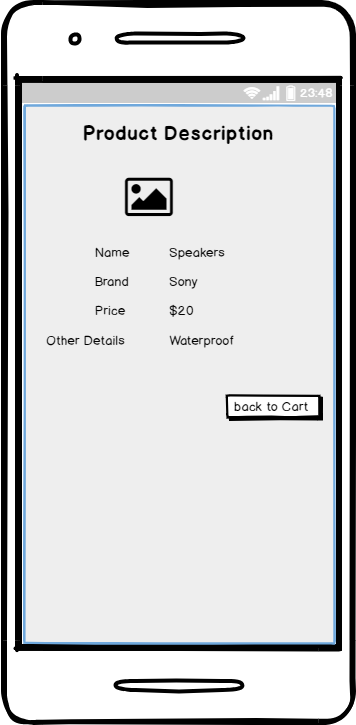


### Illustration 10. “Log Out” option in Action Bar

For current iteration user may add only one unit of desired product to the cart, if user tried to click the add to cart button multiple times, a toast is displayed indicated the item “Already exist!” in the user cart. For each action a relevant toast is displayed for the user so they get a confirmation that the action they tried to perform was successful or not. The app is connected to the Firebase database, therefore any changes made to product description by the superstore, for example change in item price or change in location within the store would be reflected on the app for the user in real time. Also any addition to cart made using the touch screen would be reflected in the real time as well. The app allows users to add an item back after deleting it intentionally or accidentally. The user information is not lost even if the user chooses to exit the app without logging out. The app does not logout a user after the user exits the app. The user may therefore resume from where she/he left off before exiting the app. The app requires user to have connectivity to internet while using the app in order to display the relevant information as it communicates to firebase realtime database which requires connection to internet. Despite that no information is lost if the connection is lost; the information would be correctly displayed after gaining the connection as it was before losing the connection. Also user does not need to worry about getting logged out if the connection is lost.

To conclude the app features and functionality, it is easy to use and has visually appealing and user friendly interface. The app does not use loud colors that may be visually disturbing for some users. The app provides easy and fast one click use interface. The sign up form for the app is really straightforward and easy to follow. The unit testing results indicate that the app does not crash or lag (a lag may be possible when there is poor network connectivity). The app clearly meets all the user needs specified in the section 2.4.2 above. The layout of the app and its different components is relatively easy to follow and meets the system specifications.

3. Progress Reports  
3.1 Report 1

Work Breakdown  
Guru is responsible for creating the android application and database. Saqib is responsible for integrating all hardware components together and Abhay is responsible for creating GUI for our interactive touch screen and linking it with database.  
Application and Work Breakdown:  
Below is the basic layout of how our app is going to look and the functionality it is going to have:  
1.   
This is the first page that appears when a user opens the app and it will allow user to Sign In into the app. New users will have to Sign Up first in order to log in to the app. Input fields on this page will have validation on them in order to make sure the user enters the data in the right format. After successful sign in, the user will be directed to the shopping cart page.   
   
  
2.   
Sign Up page will allow first time users to Sign Up. After successfully signing up, the users will be redirected to the Sign In page, where they have to sign in to be able to see their shopping cart.  
  
3.   
This is the page where users can see their shopping cart. This page is linked with real time database on the Firebase. Users can add items to the shopping cart from the interactive touch screen (part of hardware component), in order to see them on the app. Since our interactive touch screen will also be linked with the Firebase, any changes made to the shopping cart there will be reflected immediately on the Android app. From within the app, the users will have the option to clear their cart and they can also see product details by clicking on the item.   
4.   
The users will be directed to this page if they want to see the details of an item in their shopping cart. The page will have some details and a picture of the item. The user can return to cart by clicking on the "back to cart" button.  
  
  
  
  
  
  
  
  
  
  
  
  
7.8 Proposed Specifications for Database:

Database and Work Breakdown:  
Firebase database will store the items and their information. Interactive touch screen and the Android app will communicate with this real time database to add items to the cart and to display item description.   
  
Below is the rough draft of how our current database on the Firebase. We might change or modify it later according to our needs:  
  
{  
 "Items" : {  
 "Samsung Galaxy S8" : {  
 "Brand" : "Samsung",  
 "Name" : "Samsung Galaxy S8",  
 "Price" : "$1000",  
 "Waterproof" : "Yes"  
 },  
 "iPhone X" : {  
 "Brand" : "Apple",  
 "Name" : "iPhoneX",  
 "Price" : "$1000",  
 "Waterproof" : "Yes"  
 }  
 }  
}

**Appendix:**

**Sqube\_Solutions.py**

|  |  |
| --- | --- |
| # The code for changing pages was derived from: http://stackoverflow.com/questions/7546050/switch-between-two-frames-in-tkinter |  |
|  | # License: http://creativecommons.org/licenses/by-sa/3.0/ |
|  |  |
|  |  |
|  | #https://pythonprogramming.net/tkinter-depth-tutorial-making-actual-program/ |
|  | #https://gist.github.com/SirRobo/c503014bbb03088bec37a61231036461 |
|  | #https://blog.devcolor.org/heating-up-with-firebase-tutorial-on-how-to-integrate-firebase-into-your-app-6ce97440175d |
|  |  |
|  |  |
|  | import Tkinter as tk |
|  | import tkMessageBox |
|  | import pyrebase |
|  | import editedrfid |
|  | import time |
|  |  |
|  | config = { |
|  | "apiKey": "AIzaSyD3w34u\_ldmgK61PaH5TAlQV64jj4ro1l8", |
|  | "authDomain": "superstore-bde7f.firebaseapp.com", |
|  | "databaseURL": "https://superstore-bde7f.firebaseio.com", |
|  | "projectId": "superstore-bde7f", |
|  | "storageBucket": "gs://superstore-bde7f.appspot.com", |
|  | "messageSenderId": "superstore-bde7f.appspot.com", |
|  | "serviceAccount": "/home/pi/Desktop/Sqube\_Solutions/SuperStore-5486286531d1.json" |
|  | } |
|  |  |
|  | # Initializing firebase for the GUI |
|  | firebase = pyrebase.initialize\_app(config) |
|  |  |
|  | # Standard fonts used in the app |
|  | H1\_FONT= ("Verdana", 14) |
|  | H2\_FONT= ("Helvetica", 12) |
|  |  |
|  |  |
|  | # Container class that has all the frames of the GUI |
|  | class StoreHelpLine(tk.Tk): |
|  |  |
|  | def \_\_init\_\_(self, \*args, \*\*kwargs): |
|  |  |
|  | tk.Tk.\_\_init\_\_(self, \*args, \*\*kwargs) |
|  | container = tk.Frame(self) |
|  |  |
|  | container.pack(side="top", fill="both", expand = True) |
|  |  |
|  | container.grid\_rowconfigure(0, weight=1) |
|  | container.grid\_columnconfigure(0, weight=1) |
|  |  |
|  | self.frames = {} |
|  |  |
|  | for F in (StartPage, Search\_Page, Call\_Help, Product\_Details, Wrong\_Login): |
|  |  |
|  | frame = F(container, self) |
|  |  |
|  | self.frames[F] = frame |
|  |  |
|  | frame.grid(row=0, column=0, sticky="nsew") |
|  |  |
|  | self.show\_frame(StartPage) |
|  |  |
|  | def show\_frame(self, cont): |
|  |  |
|  | frame = self.frames[cont] |
|  | frame.tkraise() |
|  |  |
|  |  |
|  | # The default start page of the app (Sign in page) |
|  | class StartPage(tk.Frame): |
|  |  |
|  | def Sign\_In(self): |
|  |  |
|  | emailSt = emailEL.get() |
|  | pwordSt = pwordEL.get() |
|  |  |
|  | ## Saving the email id of the user for the "Add to cart" Function |
|  | file = open("emailid.txt","w") |
|  | file.write(emailSt) |
|  | file.close() |
|  |  |
|  | ## Initializing the Authentication |
|  | auth = firebase.auth() |
|  | try: |
|  | user = auth.sign\_in\_with\_email\_and\_password(emailSt, pwordSt) |
|  | self.controller.show\_frame(Search\_Page) |
|  | except: |
|  | self.controller.show\_frame(Wrong\_Login) |
|  |  |
|  |  |
|  | def \_\_init\_\_(self, parent, controller): |
|  | tk.Frame.\_\_init\_\_(self,parent) |
|  |  |
|  | global emailEL |
|  | global pwordEL |
|  |  |
|  | self.controller = controller |
|  |  |
|  | label = tk.Label(self, text="Store HelpLine", font=H1\_FONT) |
|  | label.grid(row=0,column=0,sticky="ew") |
|  |  |
|  | lable2 = tk.Label(self, text="Sign-In", font=H2\_FONT) |
|  | lable2.grid(sticky="ew") |
|  |  |
|  | emailL = tk.Label(self, text='Email: ') |
|  | pwordL = tk.Label(self, text='Password: ') |
|  | emailL.grid(row=2, sticky="w") |
|  | pwordL.grid(row=3, sticky="w") |
|  |  |
|  | emailEL = tk.Entry(self) # The entry input |
|  | pwordEL = tk.Entry(self, show='\*') |
|  | emailEL.grid(row=2, column=1) |
|  | pwordEL.grid(row=3, column=1) |
|  |  |
|  | button = tk.Button(self, text="Sign-In", command=self.Sign\_In) |
|  | button.grid(columnspan=2, sticky="w") |
|  |  |
|  |  |
|  | # The search page to search the items in the database |
|  | class Search\_Page(tk.Frame): |
|  |  |
|  | def search\_function(self): |
|  |  |
|  | searchSt = searchEL.get() |
|  |  |
|  | # Saving the searched item to a file, so that it can be used later |
|  | file = open("temp.txt","w") |
|  | file.write(searchSt) |
|  | file.close() |
|  |  |
|  | self.controller.show\_frame(Product\_Details) |
|  |  |
|  | def call\_for\_help(self): |
|  |  |
|  | result = tkMessageBox.askquestion("Tk", "Are you sure about it?") |
|  | if result == "yes": |
|  | editedrfid.main() |
|  |  |
|  | def \_\_init\_\_(self, parent, controller): |
|  | tk.Frame.\_\_init\_\_(self, parent) |
|  | label = tk.Label(self, text="Store HelpLine", font=H1\_FONT) |
|  | label.grid(pady=10,padx=10) |
|  |  |
|  | global searchEL |
|  |  |
|  | self.controller = controller |
|  |  |
|  | searchl = tk.Label(self, text="Search", font=H2\_FONT) |
|  | searchl.grid(row=2, sticky="w") |
|  |  |
|  | searchEL = tk.Entry(self) |
|  | searchEL.grid(row=2, column=1) |
|  |  |
|  | button2 = tk.Button(self, text="Search", command=self.search\_function) |
|  | button2.grid(columnspan=2, sticky="w") |
|  |  |
|  | button1 = tk.Button(self, text="Sign Out", |
|  | command=lambda: controller.show\_frame(StartPage)) |
|  | button1.grid(columnspan=2, sticky="w") |
|  |  |
|  | button3 = tk.Button(self, text="Request Help", command=self.call\_for\_help) |
|  | button3.grid(columnspan=2, sticky="w") |
|  |  |
|  | global waitl |
|  | waitl = tk.Label(self, text=" ") |
|  | waitl.grid(columnspan=2, sticky="w") |
|  |  |
|  |  |
|  | # Displays the Product details and give the option to "Add to cart" |
|  | class Product\_Details(tk.Frame): |
|  |  |
|  | ## This is a function that creats a dialog box when ever a person calls for help |
|  | def call\_for\_help(self): |
|  |  |
|  | result = tkMessageBox.askquestion("Tk", "Are you sure about it?") |
|  | if result == "yes": |
|  | editedrfid.main() |
|  |  |
|  |  |
|  | def add\_to\_cart(self): |
|  |  |
|  | file = open("emailid.txt","r") |
|  | email = file.readline() |
|  | db = firebase.database() |
|  |  |
|  | userdata = db.child("users").get() |
|  |  |
|  | user\_checker = False |
|  |  |
|  | # looking for the user in the database and then adding cart for it |
|  | for user in userdata.each(): |
|  | eachuser = db.child("users").child(user.key()).get() |
|  |  |
|  | for myuser in eachuser.each(): |
|  |  |
|  | if myuser.key() == "Name" and myuser.val() == email: |
|  | user\_checker = True |
|  | else: |
|  | user\_checker = False |
|  | break |
|  |  |
|  | if user\_checker: |
|  | db.child("users").child(user.key()).child("cart").child(sku).set(True) |
|  |  |
|  | def jugad(self): |
|  |  |
|  | global sku |
|  | aisle\_num = -1 |
|  | brand = None |
|  | desc = None |
|  | name = None |
|  | price = None |
|  |  |
|  | file = open("temp.txt","r+") |
|  | searchItem = file.readline() |
|  | db = firebase.database() |
|  | searchval = db.child("Items").get() |
|  |  |
|  | # Looking for the searched item in the database |
|  | for item in searchval.each(): |
|  | about = db.child("Items").child(item.key()).get() |
|  |  |
|  | for itemdes in about.each(): |
|  | if itemdes.key() == "Name": |
|  | if itemdes.val() == searchItem: |
|  | name = itemdes.val() |
|  | item\_checker = True |
|  | else: |
|  | item\_checker = False |
|  | break |
|  | elif itemdes.key() == "Aisle Number": aisle\_num = itemdes.val() |
|  | elif itemdes.key() == "Brand": brand = itemdes.val() |
|  | elif itemdes.key() == "Description": desc = itemdes.val() |
|  | elif itemdes.key() == "Price": price = itemdes.val() |
|  | elif itemdes.key() == "SKU": sku = itemdes.val() |
|  |  |
|  | if item\_checker: |
|  | break |
|  | else: |
|  | aisle\_num = -1 |
|  | brand = None |
|  | desc = None |
|  | name = None |
|  | price = None |
|  |  |
|  | file.close() |
|  |  |
|  | labelna.config(text = name) |
|  | labelas.config(text = aisle\_num) |
|  | labelbr.config(text = brand) |
|  | labelpr.config(text = price) |
|  |  |
|  | ## This is the constructor to build the GUI on the screen |
|  | def \_\_init\_\_(self, parent, controller): |
|  | tk.Frame.\_\_init\_\_(self, parent) |
|  |  |
|  | global labelna |
|  | global labelas |
|  | global labelbr |
|  | global labelpr |
|  |  |
|  | label = tk.Label(self, text="Store HelLine", font=H1\_FONT) |
|  | label.grid(pady=10,padx=10) |
|  |  |
|  | label2 = tk.Label(self, text="PRODUCT SCREEN",font=H2\_FONT) |
|  | label2.grid(row=2) |
|  |  |
|  | self.controller = controller |
|  |  |
|  | labelref = tk.Label(self, text="Click Refresh after every search", font=H2\_FONT, fg="Red") |
|  | labelref.grid(row=3) |
|  |  |
|  | button4 = tk.Button(self, text="Refresh",command=self.jugad) |
|  | button4.grid(columnspan=1, column=2, sticky="w", row=3) |
|  |  |
|  | labelname = tk.Label(self, text="Name") |
|  | labelname.grid(row=4, column=1) |
|  | labelna = tk.Label(self, text="Click Refresh") |
|  | labelna.grid(row=4, column=2) |
|  |  |
|  | labelaisle = tk.Label(self, text="Aisle Number") |
|  | labelaisle.grid(row=5, column=1) |
|  | labelas = tk.Label(self, text="Click Refresh") |
|  | labelas.grid(row=5, column=2) |
|  |  |
|  | labelbrand = tk.Label(self, text="Brand") |
|  | labelbrand.grid(column=1, row=6) |
|  | labelbr = tk.Label(self, text="Click Refresh") |
|  | labelbr.grid(column=2, row=6) |
|  |  |
|  | labelprice = tk.Label(self, text="Price") |
|  | labelprice.grid(column=1, row=7) |
|  | labelpr = tk.Label(self, text="Click Refresh") |
|  | labelpr.grid(column=2, row=7) |
|  |  |
|  | button1 = tk.Button(self, text="Add to Cart", command=self.add\_to\_cart) |
|  | button1.grid(columnspan=1, sticky="w", row=8) |
|  |  |
|  | button2 = tk.Button(self, text="Cancel", |
|  | command=lambda: controller.show\_frame(Search\_Page)) |
|  | button2.grid(columnspan=1, column=2, sticky="e", row=8) |
|  |  |
|  | button3 = tk.Button(self, text="Request Help", command=self.call\_for\_help) |
|  | button3.grid(columnspan=2, sticky="w") |
|  |  |
|  |  |
|  | class Wrong\_Login(tk.Frame): |
|  |  |
|  | def \_\_init\_\_(self, parent, controller): |
|  | tk.Frame.\_\_init\_\_(self, parent) |
|  | label = tk.Label(self, text="StoreHelpLine", font=H1\_FONT) |
|  | label.pack(pady=10,padx=10) |
|  |  |
|  | label2 = tk.Label(self, text="Worng Credentials", font=H2\_FONT) |
|  | label2.pack(pady=10,padx=10) |
|  |  |
|  | button1 = tk.Button(self, text="OK", |
|  | command=lambda: controller.show\_frame(StartPage)) |
|  | button1.pack(pady=10,padx=10) |
|  | app = StoreHelpLine() |
|  | app.mainloop() |

**Editedrfid.py**

|  |  |
| --- | --- |
| import RPi.GPIO as GPIO |  |
|  | import serial |
|  | import time |
|  |  |
|  | ENABLE\_PIN = 14 # The BCM pin number corresponding to GPIO1 |
|  |  |
|  | SERIAL\_PORT = '/dev/ttyS0' # RPi 3 has apparently used 'ttyAMA0' for |
|  | # Bluetooth and assigned 'ttyS0' to the GPIO |
|  | # serial port, so uncomment the appropriate |
|  | # SERIAL\_PORT definition for your setup. |
|  | # Failing that, check the output of: |
|  | # $ dmesg | grep serial |
|  | # to get an idea as to where serial has been |
|  | # assigned to. |
|  |  |
|  |  |
|  | def validate\_rfid(code): |
|  | # A valid code will be 12 characters long with the first char being |
|  | # a line feed and the last char being a carriage return. |
|  | s = code.decode('ascii') |
|  |  |
|  | if (len(s) == 12): |
|  | return s[1:11] |
|  |  |
|  | def main(): |
|  | # Initialize the Raspberry Pi by quashing any warnings and telling it |
|  | # we're going to use the BCM pin numbering scheme. |
|  | GPIO.setwarnings(False) |
|  | GPIO.setmode(GPIO.BCM) |
|  |  |
|  | # This pin corresponds to GPIO1, which we'll use to turn the RFID |
|  | # reader on and off with. |
|  | GPIO.setup(ENABLE\_PIN, GPIO.OUT) |
|  |  |
|  | # Setting the pin to LOW will turn the reader on. You should notice |
|  | # the green LED light on the reader turn red if successfully enabled. |
|  |  |
|  | print("Enabling RFID reader and reading from serial port: " + SERIAL\_PORT) |
|  | GPIO.output(ENABLE\_PIN, GPIO.LOW) |
|  |  |
|  | # Set up the serial port as per the Parallax reader's datasheet. |
|  | ser = serial.Serial(baudrate = 2400, |
|  | bytesize = serial.EIGHTBITS, |
|  | parity = serial.PARITY\_NONE, |
|  | port = SERIAL\_PORT, |
|  | stopbits = serial.STOPBITS\_ONE, |
|  | timeout = 3) |
|  | #ser = serial.Serial("/dev/ttyAMA0",baudrate=2400, timeout=5) |
|  | # Wrap everything in a try block to catch any exceptions. |
|  | try: |
|  | var = 1 |
|  | # Loop forever, or until CTRL-C is pressed. |
|  | while var: |
|  | GPIO.setup(6,GPIO.OUT) |
|  | GPIO.output(6,GPIO.HIGH) |
|  | #Read in 12 bytes from the serial port. |
|  | print("Reading tag...") |
|  | data = ser.read(12) |
|  | #Attempt to validate the data we just read. |
|  | code = validate\_rfid(data) |
|  |  |
|  | #If validate\_rfid() returned a code, display it. |
|  | if code: |
|  | print("Read RFID code: " + code); |
|  | if ("415D7D42C" in code): |
|  | print("Welcome User1") |
|  | var = 0 |
|  | GPIO.output(ENABLE\_PIN, GPIO.HIGH) |
|  | GPIO.output(6,GPIO.LOW); |
|  | elif ("415EA0D36" in code): |
|  | print("Welcome User2") |
|  | var = 0 |
|  | GPIO.output(ENABLE\_PIN, GPIO.HIGH) |
|  | GPIO.output(6, GPIO.LOW); |
|  | else: |
|  | print("Please contact admin to register first") |
|  | var = 0 |
|  | GPIO.output(ENABLE\_PIN, GPIO.HIGH) |
|  | GPIO.output(6, GPIO.LOW); |
|  |  |
|  | except Exception as e: |
|  | print (e) |
|  | # If we caught an exception, then disable the reader by setting |
|  | # the pin to HIGH, then exit. |
|  | print("Disabling RFID reader...") |
|  | GPIO.output(ENABLE\_PIN, GPIO.HIGH) |
|  | GPIO.output(6, GPIO.LOW); |
|  |  |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | main() |

**Pirtestwithsaver.py**

|  |  |
| --- | --- |
| import RPi.GPIO as GPIO |  |
|  | import os |
|  | import sys |
|  | import time |
|  | GPIO.setmode(GPIO.BOARD) |
|  | pir = 32 |
|  | GPIO.setup(pir, GPIO.IN) |
|  | print ("Waiting for sensor to settle") |
|  | time.sleep(5) |
|  | print ("Detecting motion") |
|  | while True: |
|  | if GPIO.input(pir): |
|  | print ("Motion Detected!") |
|  | os.system ("xscreensaver-command -deactivate") |
|  | os.system('python Sqube\_Solutions.py') |
|  | else: |
|  | os.system ("xscreensaver-command -activate") |
|  | time.sleep(2) |
|  | time.sleep(0.1) |