

Point Of Sale Grocery Cart

Michigan State University

Senior Design – ECE 480 – Team 6
Fall 2014

Sponsors:

Michigan State College Of Engineering
Dr. Satish Udpa

Faculty Facilitator:

Dr. Jian Ren



Team Members:

Ben Lauzon
Matt Rasmussen
Steven Hartz
Husain Aleid
Shuangfei Liu
Taoping Zhu

Acknowledgements

Dr. Satish Udpa: As the team's sponsor from Michigan State University College of Engineering, Dr. Udpa gave vital assistance and inspiration for the project. He provided the team with guidance and was very helpful in addressing any questions the team had throughout the semester. His level of passion for the project helped fuel the passion of all the team members on the project.

Dr. Jian Ren: As the team's facilitator, Dr. Ren helped the team stay on track by checking in regularly to ensure the group's success. We appreciate all of the time that he set-aside during semester to listen to our progress and critique our presentations and technical reports.

Executive Summary

The modern shopping experience has remained virtually unchanged throughout history. Customers walk around the store collecting the items that they want to purchase. Once complete, the customer must wait in a potentially long line just to have the items taken back out of the cart scanned and returned to the cart. The inefficiency of the shopping experience is an issue that needed to be addressed.

The *point of sale shopping cart* allows the users to skip the long lines and checkout as they shop. As each item is grabbed from the shelf it is scanned by the user's smartphone and added to their virtual shopping cart. After all items have been gathered the shopper simply checks out directly from the phone app. Upon leaving the store, the attendant verifies the contents of the cart and the shopper is all set to leave.

The *point of sale shopping cart* provides time savings for the customer as well as cost savings for the store. The store can drastically cut down on the number of checkout lanes that need to be open at a single time, which cuts operation costs significantly. The increased popularity of self-checkout has paved the way for the success of the *point of sale shopping cart* and demonstrates the shoppers desire for a change in the checkout process.

Table of Contents

Chapter 1	1
1.1 Design Project Overview	1
1.2 Design Objectives	1
Chapter 2	3
2.1 Function Analysis	3
2.1.1 FAST Diagram	3
2.1.2 Gantt Chart	3
2.1.3 House of Quality	4
2.1.4 Budget	5
2.2 Conceptual Designs	5
2.2.1 Communication	5
2.2.2 Microcontroller Choice	7
2.2.3 Load Cell and Analog Circuitry	8
2.3 Final Design	8
Chapter 3	9
3.1 Hardware Design	9
3.1.1 Amplification Circuit	9
3.1.2 Voltage Regulation	10
3.1.3 Wireless Charging	10
3.1.4 Arduino Yun	12
3.2 Software Design	12
3.2.1 Central Server	12
3.2.1.1 Web Server Setup	13
3.2.1.2 Communication With Microcontroller	14
3.2.1.3 Communication With The Phone	15
3.2.1.4 Server-Side Weight Validation	17
3.2.2 Android Application	18
3.2.2.1 Creating A Project and App Overview	18
3.2.2.2 Layout	20
3.2.2.3 Communication With The Server	22

3.2.2.4 Building A List.....	24
3.2.2.5 Checkout System.....	25
3.2.2.6 Communicating Between Activities.....	26
3.2.3 Scale Code.....	29
Chapter 4	33
4.1 Final Product Evaluation.....	33
Chapter 5	37
5.1 Final Cost.....	37
5.2 Schedule.....	37
5.3 Future Enhancements.....	38
5.4 Conclusion.....	39
Appendix 1: Design Team Composition and Responsibilities	40
Ben Lauzon: Team Manage.....	40
Matt Rasmussen: Webmaster.....	41
Shaungfei Liu: Document Prep.....	42
Taoping Zhu: Document Prep.....	43
Husain Aleid: Presentation Prep.....	44
Steven Hartz: Lab Manager.....	45
Appendix 2: References	46
Appendix 3: Code	46

Chapter 1

1.1 Design Project Overview

The system used in grocery stores around the world is one everyone is familiar with. It has been around for a very long time and is very out of date. The major issue with today's system is the unnecessary stop at the checkout line when a shopper is finished gathering items. It involves a lot of unnecessary waiting. At the front of this line is a cashier who is being paid to remove all the items you just placed into your cart, scan them, and then place them right back where they were. This results in a lot of the shopper's time being wasted, adds to congestion near the store's exit, and adds cost for the store that has to pay cashiers. This system is in dire need of an update.

The goal of the *point of sale grocery cart* is to create an easier system for shoppers to use, while cutting down on the costs for the store. This new system involves placing the scanner normally wielded by the cashier into the hands of every shopper. As a patron of the store travels between aisles and gathers items, they can use their smartphone to scan the items as they put them into their cart. Once an item has been scanned, the cart, which is equipped with a load cell scale, reads the weight of the item added to the cart and verifies it with the store's database. This helps confirm the correct item was placed in the cart. In case of having the wrong item, the server will send an error message and not add the item to the virtual shopping cart. When finished shopping, the shopper will be able to use a credit card or PayPal to checkout directly from their phone. This means they can skip the long checkout lines for good. The current checkout system is in desperate need of an update to save time and money for both sides of the grocery store.

1.2 Design Objectives

The aim of this project is to prototype a new, streamlined checkout experience for the everyday shopper. The final design is not expected to be store-deployable, but provide a proof-of-concept model for future work. By combining a weight-sensing cart with a smartphone app, customers will be able to skip checkout lines by checking themselves out straight from their cart. The

new system must use traditional grocery carts with minor additions; this system should allow storeowners to refit their current carts rather than buying entirely new ones.

The cart additions must be rugged, as they will have to withstand daily abuse from customers as well as weather and temperature-related stresses like cold, rain, and snow. The carts' weight-sensing and communication systems must be powered by a wirelessly rechargeable battery and last a full day on a single charge. The weight sensor placed in the cart must accurately measure weights ranging from an ounce to a full cart of groceries, at least 130 pounds, and must do so reliably no matter where on the cart the item is placed. This imposes both a sensitivity and dynamic range constraint on the weight-sensing system.

The phone application will be used by customers of all technical backgrounds, and therefore must be simple, intuitive, and user friendly. Additionally, the application must be secure to resist hacking. The application must also be able to handle items without UPC codes.

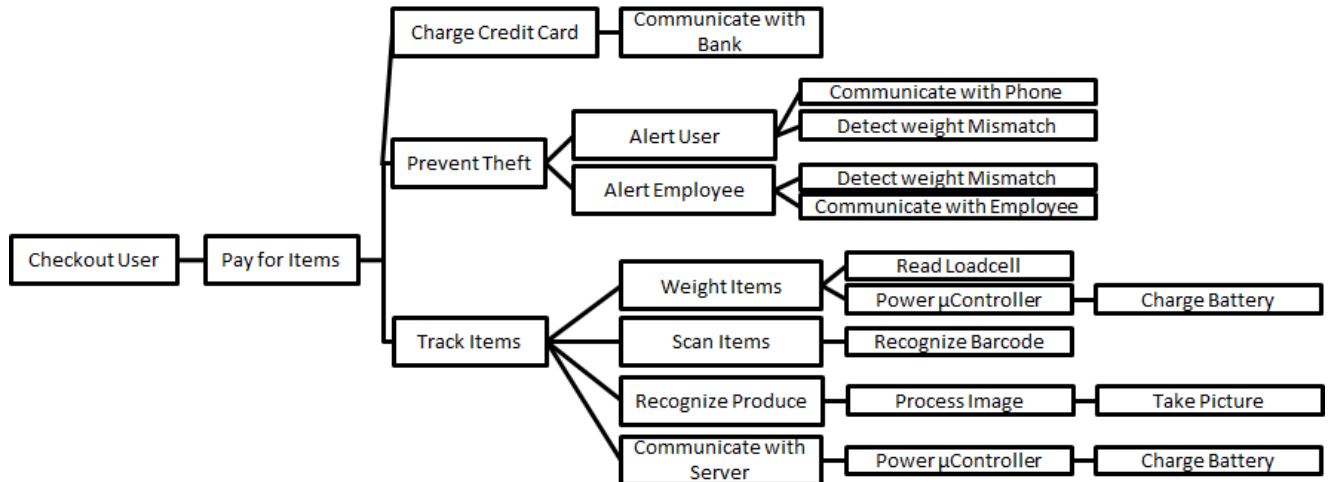
Two additions to the current grocery store infrastructure have been allowed. First, WiFi Internet access may be expected from grocery stores. This is done to allow phones to initiate payment transactions; stores in places with poor cell phone data coverage still want users to have a consistent checkout experience so payment over a user's 3G or 4G connection is not sufficient. Second, modification to the cart corral at the store's entrance is allowed so that wireless charging can be achieved.

Many different designs of this project have been discussed and various paths were suggested to follow in order to build a prototype for this project. However, to maximize the proficiency of the project and the ease of troubleshooting, the project was divided into three separate categories that will be built individually. These categories include cart communication method, battery charging design, and cart scale design.

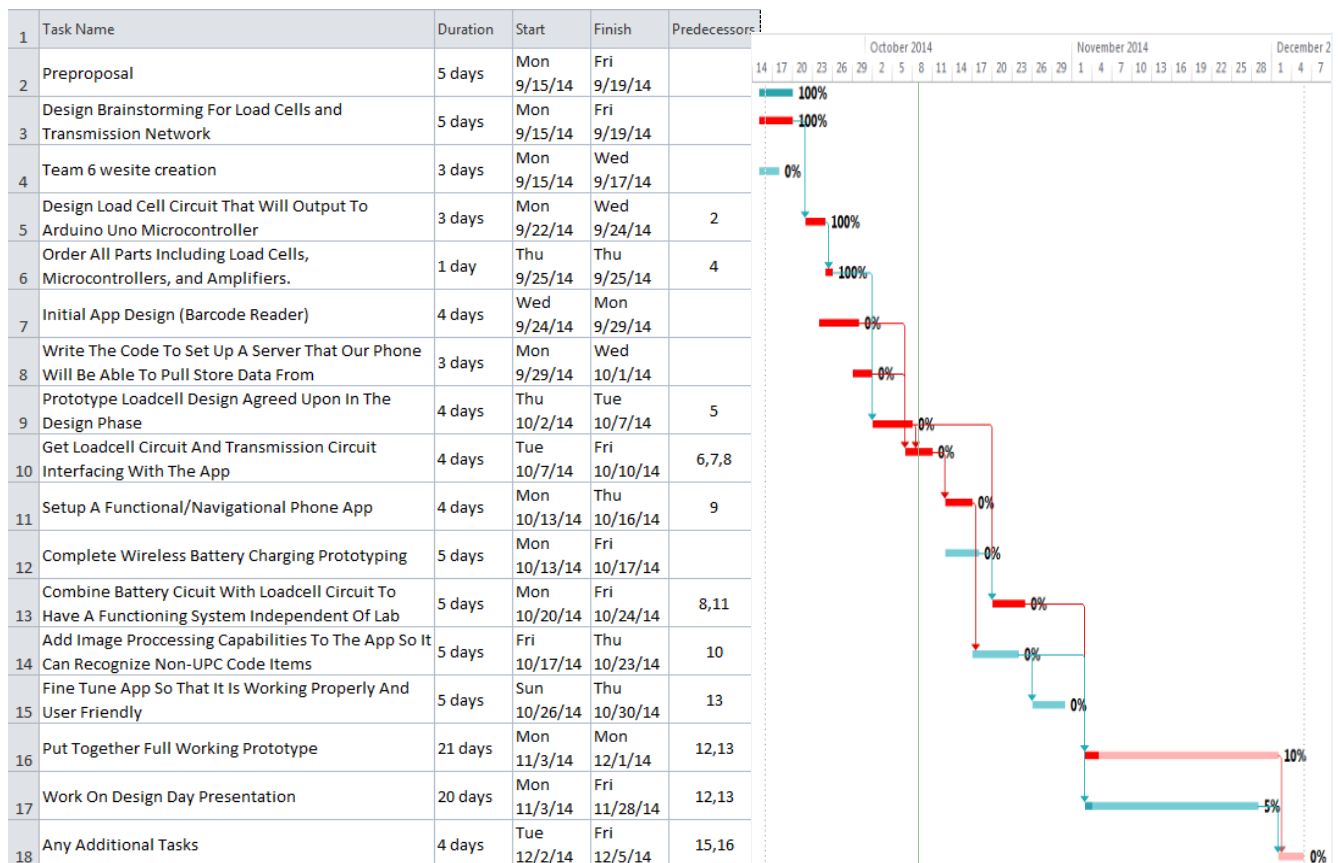
Chapter 2

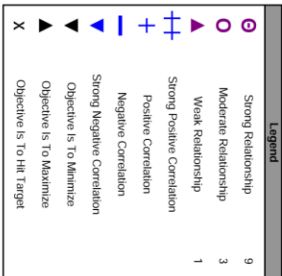
2.1 Design Project Overview

2.1.1 Fast Diagram



2.1.2 Gantt Chart





2.1.4 Budget

Item	Cost
Arduino YÚN	\$ 71.00
USA Raspberry Pi Micro USB Power Supply Charger	\$ 6.81
Edimax EW-7811Un 150M 11n Wi-Fi USB Adapter	\$ 8.70
Raspberry Pi Model B+ with 8GB SD Card	\$ 40.00
Accuteck Heavy Duty Postal Shipping Scale	\$ 21.99
RAVPower® 2nd Gen Mini 3000mAh Ultra-Compact Portable Charger	\$ 14.99
DIY Wireless Charging Transmitter + Receiver Solution Module	2 x \$ 8.45
Motorola Universal Charger & 3000mAh Portable External Battery Charger	\$ 20.98
Arduino Shield	\$ 27.95
Accuteck Heavy Duty Postal Shipping Scale	2 x \$ 21.990
SimpleLink Wi-Fi CC3200 LaunchPad	\$ 30.08
Arduino Uno	\$ 28.28
Linear Tech 5v to 3.3V Switching Regulator	3 x \$ 4.900
Total amount	\$ 346.36

2.2 Conceptual Designs

The system components were broken down into 3 major subsystems and conceptual designs for each subsystem were proposed. The first of the major subsystems pertains to how the cart and Smartphone App communicate and ensuring they do so securely. The cart's onboard microcontroller was seen as the second major subsystem. Lastly, the load cells and their associated analog circuitry comprised the final subsystem.

2.2.1 Communication

The first proposed method of communication between the cart and Smartphone App was a direct Wi-Fi connection. Over this connection, weights could be sent from the cart to the phone, verified, and items added to the current shopping list. This fully meets the customer's stated desire to add very little to current carts and does not use anything external to the cart and phone. Wi-Fi

was chosen as the communication method between the devices because nearly every Smartphone supports Wi-Fi whereas Bluetooth is slightly less supported.

The alternate method that was proposed was to add a Wi-Fi network throughout the store, managed and run by the store itself. While contrary to the sponsor's stated desire to add only to the cart, the sponsor found the addition of Wi-Fi to the store acceptable. Via this network, the cart would periodically report its current weight to a central server. The phone would then scan barcodes and send them to the same central server. The server would then respond either confirming that the scanned UPC matched the weight received from the scale or that the weight was incorrect.

While the first method is the simplest and the least expensive, it has a number of drawbacks. Possibly the most important is that there is no guarantee the user is running the store's application. By running a visually similar application that displays a copy of the real App's completed checkout page, a malicious customer could leave the store without ever actually paying for their groceries even with an employee checking the phone's screen for confirmation that they paid.

Another, less critical, disadvantage would be seen when the store wanted to have a sale. If the list of items with associated weights and prices is stored on the phone, and within the App, the App must be updated with each new sale. A store would then need employees to ensure customers are running the current version of that app and not an older one where items were not previously on sale.

The alternative design proposed involves the cost of installing a Wi-Fi network in stores that don't already have one. However, this cost is fixed relative to the number of carts. Additionally, Internet access is required to perform a credit card transaction. In areas where cellular network data coverage is poor or non-existent, a Wi-Fi network with Internet access would be required anyway to process payments.

The addition of a central server solves the two major issues of the first design. Since the cart communicates directly with the server, a response code

could be sent to indicate that the cart is bearing more or less weight than the UPC codes sent from the phone would indicate. An LED on the cart could be illuminated to indicate to store employees at a glance that the customer is trying to shoplift or is experiencing an issue with the application and needs to be checked out manually. The database of items, weights, and prices would also be stored on the server. Sales could then modify the prices of items without the App having to be updated.

2.2.2 Microcontroller Choice

Two microcontrollers and a Raspberry Pi were considered as the processing and communication element aboard the cart. All three had Wi-Fi capabilities and were considered for this reason.

Texas Instruments offers a microcontroller in their Launchpad line with onboard wireless and Analog to Digital Conversion (ADC) hardware, the CC3200. While this microcontroller was significantly cheaper than the other two controllers, little library support exists freely for application-level protocols. The complexity of the communication software written for this platform was expected to be much higher than the other platforms.

Arduino offers a microcontroller with Wi-Fi available without the purchase of an additional module called the Arduino Yún. Similar to the Texas Instruments offering, this microcontroller also offers ADC hardware. While more expensive, Arduino has a number of libraries available to developers in addition to numerous support forums available online. The complexity of developing software for this platform is more intuitive than the CC3200, but this platform was more expensive than the CC3200.

The Raspberry Pi is not a microcontroller, but instead a fully functional, mini-computer running a specialized Linux operating system. Developing software for this platform would be much less complex than the CC3200 and about as complex as the Yún. This platform did not include an ADC, which would raise its cost to near or above the Yún. Additionally, since this platform is a full computer, it would also draw the most power. Since the customer desired a

system run off a wirelessly charged battery, this was seen as a large disadvantage.

2.2.3 Load Cell and Analog Circuitry

There existed only one design concept for the weight-sensing system since the project required the use of load cells. Strain Gauge load cells were best suited to this project due to their low cost. The weak signal from the load cell would then need to be amplified to a range appropriate for the microcontroller or ADC chosen.

2.3 Final Design

Because the value of preventing shoplifting was high, the addition of a Wi-Fi network in the store with an attached central server to facilitate communication between the Smartphone App and cart was chosen. While this adds cost to the overall system, the greater ability to prevent theft adds value.

Because of the compressed design schedule, the CC3200 was not a feasible microcontroller for this project despite its lower cost. Instead, the Arduino Yún was chosen. Its onboard ADC and lower power consumption than the Raspberry Pi made it clearly superior given that the Raspberry Pi had little to offer this system that the Yún did not.

Power is provided via a rechargeable USB battery. The Arduino Yún is able to run off the provided 5 Volt input, as well as regulate a 5 Volt output to provide power to the analog circuitry. The battery is charged via two induction coils. One coil would be connected to the store's power and be embedded in a rail that a cart would drive over. The receiving coil would be located on the bottom of the cart, and would charge the battery when the two coils aligned.

A strain gauge load cell is used to detect the weight in the cart. The signal from the load cell is then amplified such that the maximum load of the load cell creates a 5 Volt output. This is then wired into the Yún's primary ADC pin. From there, the values are digitized and sent to the server for further processing.

Chapter 3

3.1 Hardware Design

3.1.1 Amplification Circuit

The amplification circuit plays the role of the messenger between the scale and the Arduino Yun. Once an item has been placed on the scale, the circuit will amplify the signal coming from it and send it to the Arduino, which can absorb a maximum of 5 volts. As a result, the amplifier that is to be selected must be easily controlled and modified in a circuit to output no more than the maximum voltage. Also the amplifier must be able to amplify very minute voltage changes. For these reasons the INA125P amplifier was chosen.



Figure 3.1.1.a

In addition, the amplifier can operate accurately between -40 and 85°C . This means even that the possibly extreme temperatures faced in the parking lot will not affect the systems operations. This was an important aspect of the system to the sponsor. The second most important component in the circuit is the resistor connected between pins 8 and 9. The output voltage and the resistor have an inverse proportional property, that is, the bigger the resistance is the lower voltage the amplifier outputs. The initial circuit design used a 10-ohm resistor as a starting point however, the circuit outputs more than 5 volts at the maximum weight the scale can handle which is 38.5 kilograms. As a consequence, a new resistor needed to be selected to fix this issue, and the selection was two 150 ohm resistors in parallel, making the total resistance 75 ohms. This will ensure that the amplifier output 5 volts at the maximum weight. The circuit schematic is shown in figure 3.1.1.b.

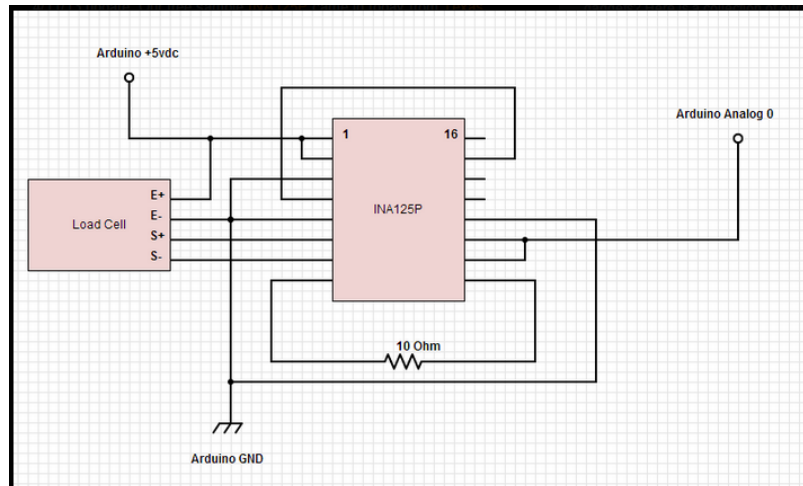


Figure 3.1.1.b

3.1.2 Voltage Regulation

The battery used to power the scale and microcontroller has an internal voltage regulator. The project also makes use of an Arduino Yun, which has a built in 5-volt regulator. A properly regulated voltage is essential because the scale code is very sensitive to small voltage. A small voltage change in either the input or the output can drastically alter the weight readings being sent to the server. If the weight readings are not accurate, the server will have trouble determining if the correct item was placed into the cart.

3.1.3 Wireless charging

The wireless charging design consists of a wireless charging mat that will be placed in the center of the cart collecting rack. The charging receiver will be positioned on the bottom of the cart so the transmitting mat below can charge it. This design uses the transmitter and receiver module, which includes two copper wire coils. Each of the copper wire coils is connected to the wireless charger transmitter or receiver shown in figure 3.1.3.b and 3.1.3.c. For the battery to be charge, both copper wire coils need to be placed on top of one another as shown in figure 3.1.3.a.

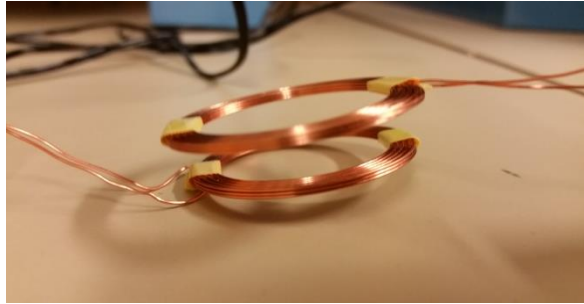


Figure 3.1.3.a

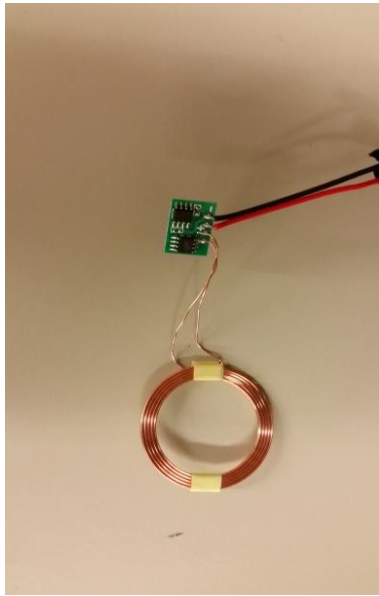


Figure 3.1.3. b

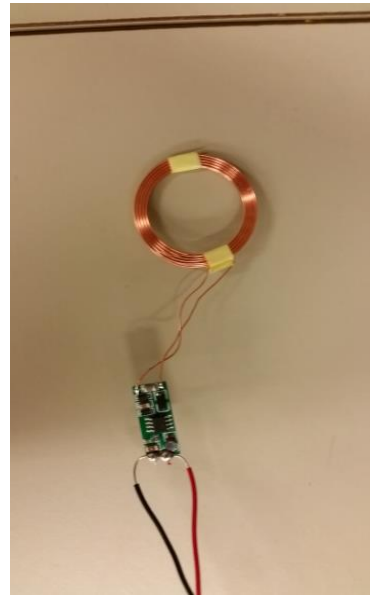


Figure 3.1.3.c

The Transmitter is connected to the wall wart shown in figure 3.1.3.d. The input voltage from wall wart to the transmitter is around 13.5 Volts. The receiver is connected to the battery shown in figure 3.1.3.e. The output voltage comes from the receiver is around 5v.

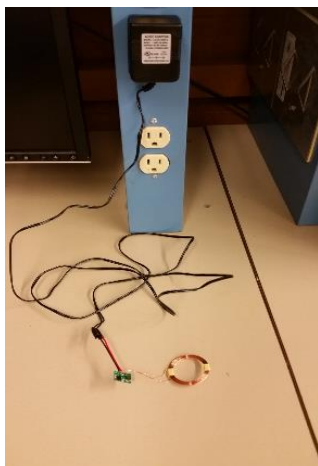


Figure 3.1.3.d



Figure 3.1.3.e

3.1.4 Arduino Yun

The microcontroller used for the project is Arduino Yun shown in figure 3.1.4.a.



Figure 3.1.4.a

The Arduino Yun microcontroller board has built-in Ethernet and Wi-Fi support, a USB-A port, micro-SD card slot, 20 digital input/output pins, a 16 MHz crystal oscillator, a micro USB connection, an ICSP header, and 3 reset buttons.

Both the input voltage and operating voltage is 5V. So, we powered the board through the USB connection with 5V battery. Compared to the Raspberry Pi, the Arduino Yun microcontroller has much lower power consumption. Therefore, it allows us to last longer a single battery charge.

The Arduino Yun microcontroller also has a stable Wi-Fi performance. We can use the Wi-Fi capabilities of the Arduino Yun to communicate with our server. This makes the Yun the best choice for our project.

3.2 Software Design

3.2.1 Central Server

The use of a central server to facilitate the communication between the phone and the cart has many advantages. Rather than attempting to create an entirely new communication protocol, leveraging an existing, well-defined and well-used protocol has many advantages including supporting software libraries.

Because of this library support as well as full program support, communication between the cart, server, and app is done via Hypertext Transfer Protocol (HTTP) an HTTP server is required.

3.2.1.1 Web Server Setup

The server for this system runs Ubuntu Linux as it is free and commonly used to support web servers. After downloading the disc image file and burning it to a DVD, the server will need to boot from the disc. This is done during the initial BIOS boot and varies based on motherboard manufacturer. Once the computer begins to boot from the disc, a step-by-step wizard will walk a user through the installation process. A simple installation was sufficient for this system using the default settings.

Once the Ubuntu Linux operating system has been installed, an HTTP server application must be installed. This is most easily done from a terminal, but requires that the server have Internet access. Ubuntu includes an application called 'apt' that will search for an install software packages from a large repository of software. The following command installs the Apache HTTP server:

```
sudo apt-get install apache2
```

Another important software package is the database software. MySQL is a free database package that can be easily installed with the 'apt' program as well. The following command will install a MySQL database server, as well as perform the initial configuration:

```
sudo apt-get install mysql-server
```

During the installation process, a password for the root user will be created interactively. This will be needed for later connections to the server.

The Apache HTTP server, by default, serves a client's request for static Hypertext Markup Language (HTML) documents stored on the server's hard drive. As the Internet has grown, however, a need has arisen for dynamic HTML documents, generated by code. Apache provides a plugin system allowing other applications to return dynamically generated HTML documents to meet a client's request.

The plugin used in this system is PHP. PHP parses files written in its language, returning HTML based on the script file's output. Additionally, PHP provides libraries to interface with MySQL database servers. Information can then be quickly stored and retrieved from the library. Installing PHP can be done via the following command:

```
sudo apt-get install php5
```

This installation process prepares the server to act as a web server and execute the scripts used to define the responses of the server to requests from the cart's microcontroller and the Smartphone app.

3.2.1.2 Communication with Microcontroller

Because of the microcontroller's limited memory and processing resources, a protocol for communication between the server and the cart was desired. Since HTTP only allows for clients to make requests to servers and not vice versa, the server is unable to request the cart's weight directly. Instead, carts are configured to send the current weight once every second.

Data is sent via HTTP GET variables. The HTTP specification allows for any characters after a '?' character to be passed to the server program.[2.1] Apache interprets what follows as a series of key-value pairs. Keys and values are separated by an equal sign, while key-value pairs are separated by an ampersand. By appending the following string to a Universal Resource Locator (URL), a microcontroller can transmit both its cart number and current weight to the central server:

```
?weight=2305.67&cart=56
```

In this example, the Apache server would pass the string on to PHP, which would separate this string into an array indicating a cart number of 56 with a weight of 2305.67 grams.

After sending a request to the server with this data appended, the cart awaits a response. The server only needs to confirm that the data was sent. It does so by replying with an empty HTTP response. An empty response consists of valid HTTP headers and a Content-Length header of 0. The microcontroller

can determine from the headers that a response was received and needs to take no further action.

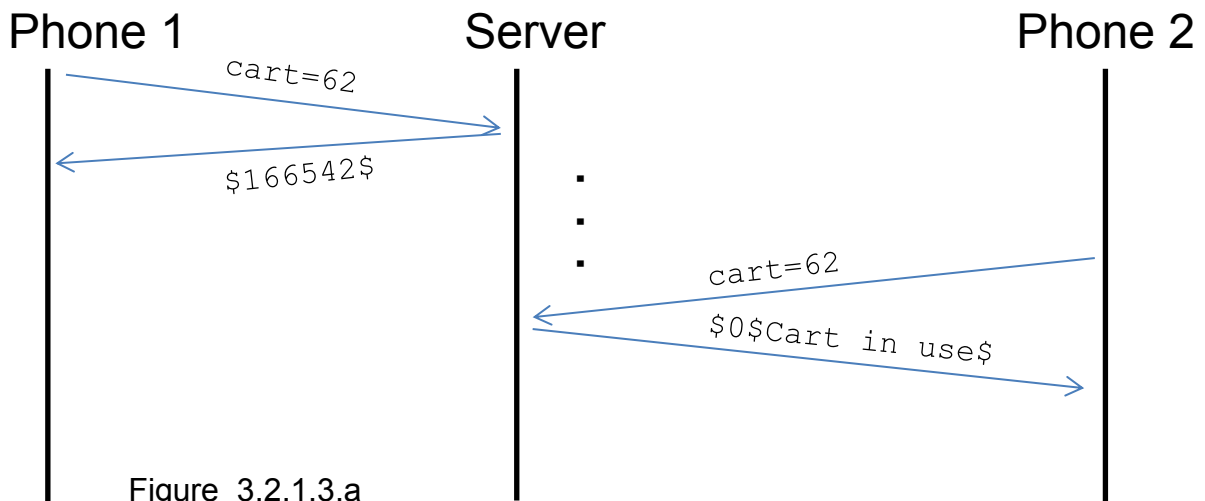
3.2.1.3 Communication with the Phone

The communication between the phone and the server is stateful, which is an issue given HTTP's stateless nature. This is handled by using the database as well as a parameter passed between the phone and the server to indicate the current session.

A phone initiates a session by scanning a QR code located on the cart. This QR code contains the cart's number, as an ASCII string. The phone makes an HTTP POST request to the server with this id to initiate a new shopping session. The server must then confirm that the cart is not in use by another shopper. If the cart is available, the server replies to the phone's request with a new pseudo randomly generated session id enclosed in dollar sign (\$) characters. This session id will be between 1 and 4294967295. If the cart is unavailable or no cart with such a number exists, the server will reply with a 0 enclosed in dollar sign (\$) characters as well as an additional message indicating the source of the error. Examples of the server's response are:

```
$155625$      Valid, session id is 155625
$2556$        Valid, session id is 2556
$0$Cart in use$ Invalid, cart is currently in use
```

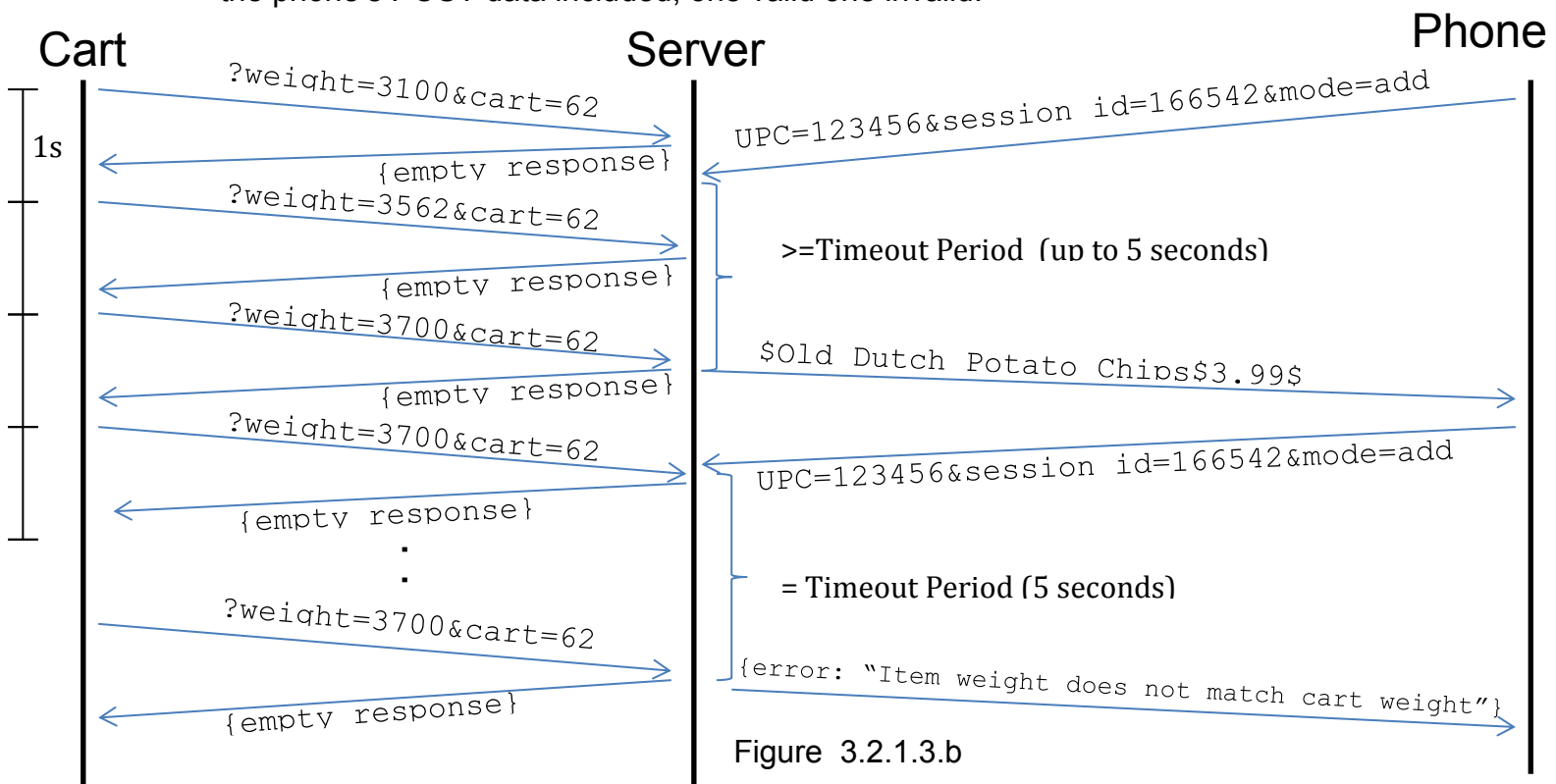
This interaction is included in Figure 3.2.1.3.a.



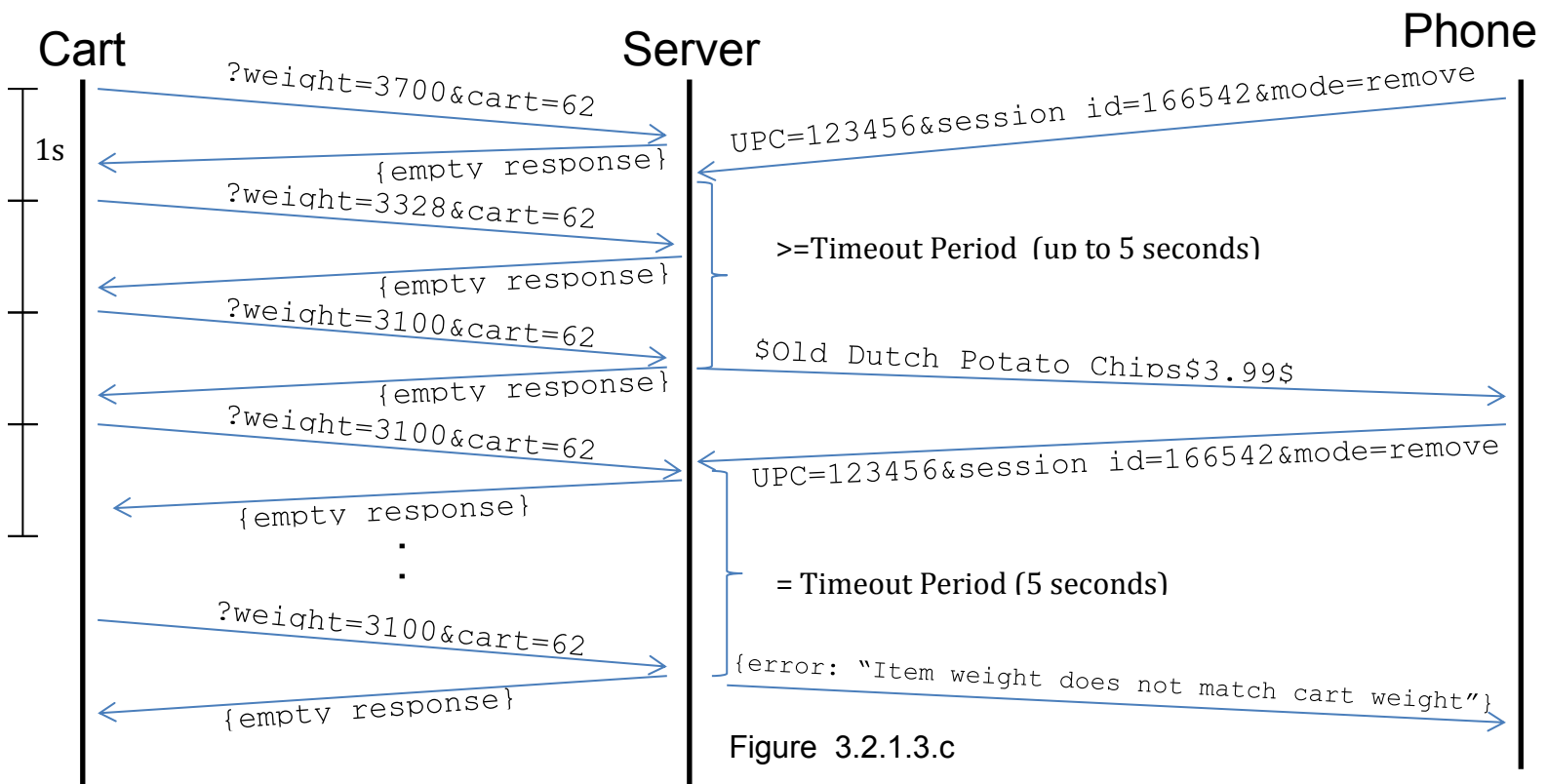
After starting a session, a user will scan items and add them to their cart. To successfully add an item to their cart, the item's UPC code as well as the current session id must be sent to the server. The app must also send a flag indicating that the request originates from the phone as well as a mode parameter equal to the string 'add'. The server then internally compares the weight of the item to the difference between the current weight in the cart and the weight after the last item was added or removed. The server allows up to 5 seconds to elapse before responding so that a user has time to add an item to the cart after scanning it. If the change in weight matches the expected weight of the item to within 10 grams, the server responds to the request with the item's name and price enclosed dollar sign (\$) characters. If not, the server responds with a response starting with an open curly bracket ({} character. Examples of responses are:

\$Mott's Applesauce\$3.99\$	Valid
\$Meijer Paper Towel\$2.49\$	Valid
{error:" Item weight does not match cart weight"}	Invalid

In the first response, the item is Mott's Applesauce, which costs \$3.99 while in the last response the server informs the phone that the weight is not valid. Figure 3.2.1.3.b shows 2 possible interactions between the server and the phone with the phone's POST data included, one valid one invalid.



The only difference when removing items is that the mode parameter must be set to 'remove'. The UPC code, session id, and phone flag must still be passed. The server then compares the item's weight with the difference between the weight at the last addition or removal and the cart's current weight. If the correct amount of weight has been removed, the server replies as above with the item's name and price enclosed in dollar sign (\$) characters. Otherwise, it returns a response starting with an open curly bracket ({) character. Two possible phone-server interactions are shown in Figure 3.2.1.3.c.



3.2.1.4 Server-Side Weight Validation

Validation of cart weight changes based on an item's UPC code is done internally on the central server because of its increased security. To do this successfully, the server's code relies on 3 database tables. The first table holds a list of every cart, by number, and its current weight. This table is continually updated as the cart transmits its weight to the server. The second table contains a list of session ids, as well as whether the session is active or has been closed (either through successful purchase or has been cancelled). Finally, the server

logs every activity (adding or removing items, or paying for the groceries) in a third table.

When a request to add or remove an item is received, the server first checks the UPC code received. If the item is not found in the database, the server will silently fail. Otherwise, the server then stores the item's weight and proceeds to request the weight of the cart after the last activity. Every second, the server also requests the cart's current weight from the database. The current weight and last activity weight are used to calculate the weight change since the last activity. If this is within 10 grams of the scanned item's weight, the server returns a success code to the phone and exits the current one-second loop. If the weight is not within the threshold, the server continues to loop, giving the user some time to place items in the cart. If 5 or more seconds have elapsed, the server returns an error code indicating an invalid weight. Appendix S includes the code related to the weight check for item removal.

3.2.2 Android Application

A key component of this project was to create a user-friendly, secure phone application. The application works closely with the store's server to allow the user to identify which store they are in, communicate which shopper they are within the network, scan items which are automatically added to a virtual shopping list, and allow users to checkout within the application using a credit card or PayPal.

3.2.2.1 Creating A Project and App Overview

The first step in creating a successful app is learning how to create a project and how each component within a project is used. The program that the team used to create the application is called Eclipse.

Eclipse allows a user to create the shell of an android application, which can be filled in whatever way the user desires. The first step is to simply select "new android application". Figure 3.2.2.1.a shows the screen that is displayed after this selection. This screen is where the user names their application, chooses the target SDK, chooses how to compile the app, and the overall visual

theme of the app. Supplemental code for section 3.2.2 can be found in Appendix 3.1.

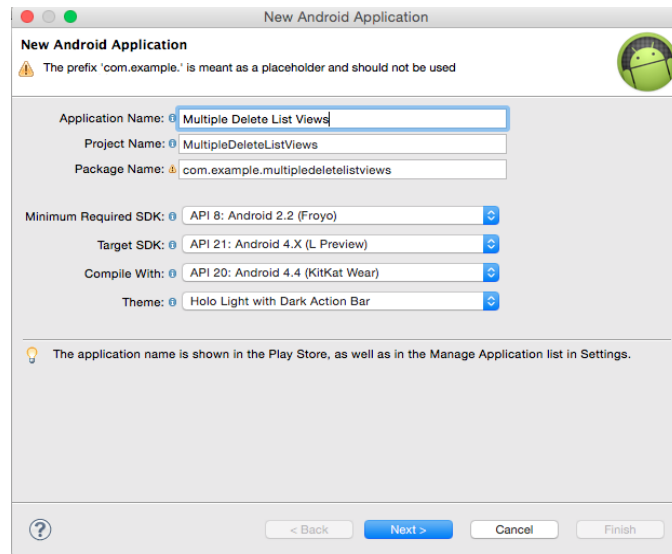


Figure 3.2.2.1.a

After selecting the next button, the user is prompted with the screen shown in figure 3.2.2.1.b, which allows them set the logo for the app. Default logos or custom logos may be used.

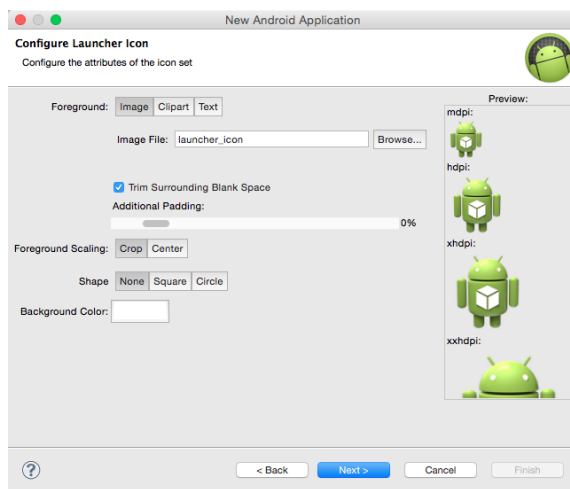


Figure 3.2.2.1.b

After clicking next followed by finish, the bare bones of the app are created. This is displayed in figure 3.2.2.1.c.

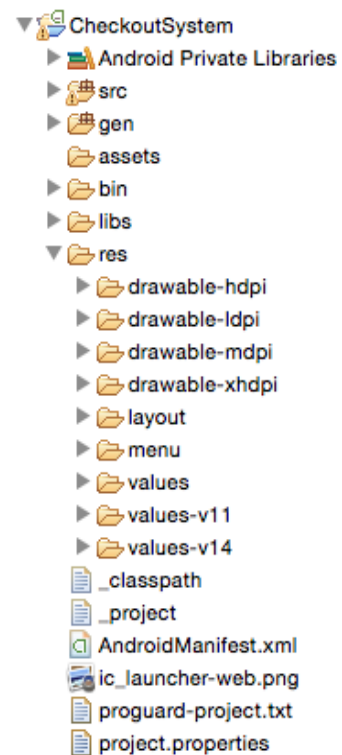


Figure 3.2.2.1.c

Each section of the project has a specific purpose. We found that most of the user interactions took place within the following sections:

AndroidManifest.xml:

The AndroidManifest file organizes the basic characteristics of the app and defines the background components that make up the app. Think of it as the “front page” defining the important aspects of the app.

Src:

This is the directory for source files that the app will be using. The app is divided into “Activities” that are written here. Think of activities as new “windows” for each instance of the app.

Drawablehdp:

Icons and images are stored here and can be used throughout the app.

Layout:

Layout files are essential to the user friendliness aspect of the application. This is the section where the app is visually designed. All buttons, text, and layouts are defined here. This is the part of the app that the end user will be able to see and actively participate with.

Values:

Instead of declaring string or item values at the top of every page, you store them here.

3.2.2.2 Layout

The layout of the application is the most important aspect in making the app user friendly. This section of Eclipse allows the user to make the app stand out and be unique from other apps. The placement of each button, the background, and the text wording and color all play a big impact in making the app unique and user friendly. The code below shows the layout file for our apps opening screen, which allows the user to scan the store’s id.

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/bgimag"
```

```

        android:orientation="vertical" >

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1.61"
            android:gravity="center"
            android:text="Welcome!"

        android:textAppearance="?android:attr/textAppearanceLarge"
            android:textColor="@android:color/white"
            android:textStyle="bold"
            android:textSize="60sp" />

        <Button
            android:id="@+id/btnScan"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Click Here to Log In"
            android:textColor="@android:color/white"
            android:textSize="25sp"
            android:textStyle="bold" />

    </LinearLayout>

```

The first thing to decide when laying out an app is the type of layout you want to use. We chose to use a linear layout, which automatically lines the buttons and text boxes into a horizontal view. Another popular layout it called relative layout, which allows for more flexibility in moving boxes or text into non-linear positions. Android: background allows the background to be set to a picture or preset background. We set a green background using a picture called bgimag in this particular example. Textview creates a text box on the screen. Textbox's content can either be set by android:text or set within the code, which means they can be changed by different actions. This particular box just says, "Welcome!". We can see in the code that there are a number of options used to set text size, color, and positioning. The button is set in very similar fashion to the textview. The biggest difference is that the button can be pressed and it triggers a function. In this example the button starts the function btnscan. Figure 3.2.2.2.b shows what this code looks like visually.

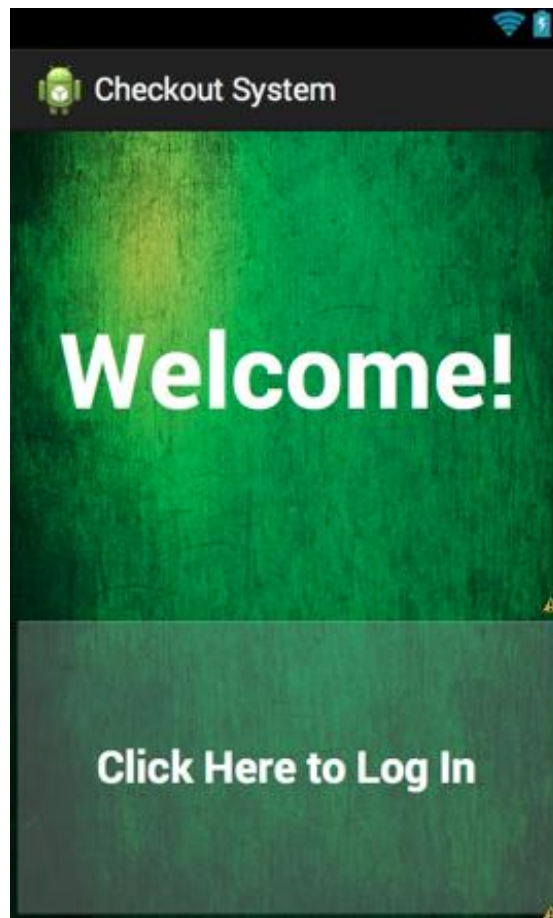


Figure 3.2.2.2.b

This screen demonstrates how we actively kept the end user in mind. The page is not cluttered with text and only has a single button. The button's text is quick and to the point. It is very difficult for a user to be confused when looking at this page. We chose a background that gives personality to the app and the white text makes it very easy to read. The philosophy of keeping the app simple and intuitive was followed throughout the entire design process especially in the design of the app's home screen. The home screen code can be referenced in appendix 3.2.

3.2.2.3 Communicating With The Server

The ability for the android application to communicate with the server allows the app to be much smaller overall in terms of disk space required. All of the information for an item can be kept in one secure place, instead of in the hands of every user. This communication process allows the user to only have

access to received information from the server and also eliminates the need for the grocery cart to ever communicate with the phone of the user. This communication was done using HTTP POST requests. This method is more secure over the alternative, a GET request, where the parameter names and values are simply concatenated onto the end of the URL being accessed. This would allow the user to easily send whatever parameters and values they want to the server, which could lead to crashing or inaccurate cart totals. While POST requests require more overall lines of code, the data being sent is not easily viewable by the user, which results in a much more secure transmission. Using Android Volley, a library included into all current and future versions of Android as of June 2013, a standard POST request can be sent in the form below: [2.1]

```
private void postData(final String param1, final String
param2) {
    RequestQueue rq = Volley.newRequestQueue(this);
    StringRequest postReq = new
    StringRequest(Request.Method.POST,
    "http://hartzweb.com/ece480/upc.php",
    new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        }, new Response.ErrorListener() {
        @Override
        protected Map<String, String> getParams() throws
        AuthFailureError{
            Map<String, String> params = new
            HashMap<String,
            String>();
            params.put("UPC", param1);
            params.put("session_id", param2);
            return params;
        }
    };

    postReq.setRetryPolicy(new
    DefaultRetryPolicy(0,0,DefaultRetryPolicy.DEFAULT_BACKOFF_M
ULT));
    rq.add(postReq); }
```

Whenever the function “postData” is called within this activity, it will send the values stored in the two input parameters, param1 and param2 to the URL specified following the word POST. These parameters are built into a list of key-

value pairs, which is then sent to the server. A transmission from this function would appear similar to {UPC=123456789&session_id=123456}. The section titled “onResponse” handles what the server sends back to the application. In this application, the response contains the name and price of the item just scanned. This data is then parsed and saved as strings in the program to be added to the list. If invalid POST parameter values are sent to the server, the response does not contain any item info, and this response will cause an “Invalid item” error to pop up. This process is used in our application to add items to the shopping list and during the shopper login at the beginning of the application, where the user is linked with the appropriate cart.

3.2.2.4 Building A List

The activity that displays the user’s shopping list allows for them to see items previously scanned, view their total cart cost, as well as modify the list by removing no longer desired items. The building of the list is primarily done within ListMainView.java, which can be found in appendix 3.3. Items are transferred over to this activity from MainActivity.java, which can be found in appendix 3.4, via Intents. This data is added as a key-value pair of the form {“item_name”, price}. The list updates each time a valid item is scanned, but can also be viewed simply by pressing the “View List” button on the main page. When viewing the list, the total cart price is updated in onResume and can be viewed in the status bar at the top of the application. Upon leaving the screen for the list, the updated cart sum is sent over to MainActivity using .putExtra methods, where it can be used in the checkout checkout activities.

In addition to adding to the list, we can also remove items, in case the user decides they no longer want a particular item. This is done using an adapter activity named “ListAdapter”, which can be found in appendix 3.5.[2.3] The functions in this activity are accessed and used within ListMainView in the function “onActionItemClicked” which checks for items that are long-pressed. This allows items to be highlighted for removal. Upon being selected, the class that contains the name and price is removed off of the list, regardless of where the item exists in the list. With each item that is removed, a POST request

(described above) is sent to the server listing the name of the removed item. The response back from the server is then sent back if the updated cart weight approximately matches the previous weight minus the item removed. This is to verify that the item was in fact removed from the cart, and that the user is not just trying to pay for fewer items. If the server's response is of an invalid weight, a popup message lets the user know, and the item is not removed. If valid, the selected item is removed, and the total price of the groceries is updated. Finally, when the user exits the view of the list using the back button, the new price sum is sent over via onPause back to the main activity.

3.2.2.5 Checkout System

Our app makes use of the PayPal Mobile SDK. The PayPal Mobile SDK enables native apps to accept PayPal and credit card payments. PayPal's website has a download for the Android version of this SDK. It was the team's job to figure out how to implement the PayPal SDK into our application. The code also needed to be tailored to fit our specific application. For example, the original code had four buttons all with their own function. We wanted to make the checkout simple so we combined the functions we needed from each button into a single checkout button. Figure 3.2.2.5.a shows the screen we designed, which will launch the PayPal SDK shown in figure 3.2.2.5.b.

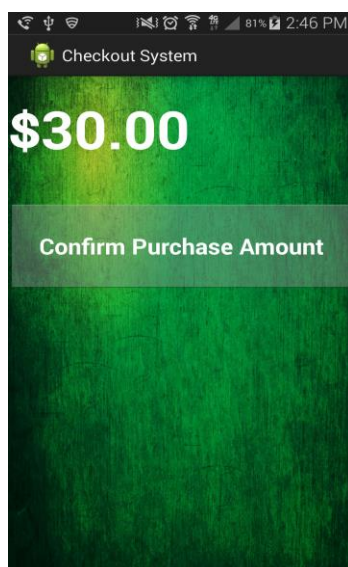


Figure 3.2.2.5.a

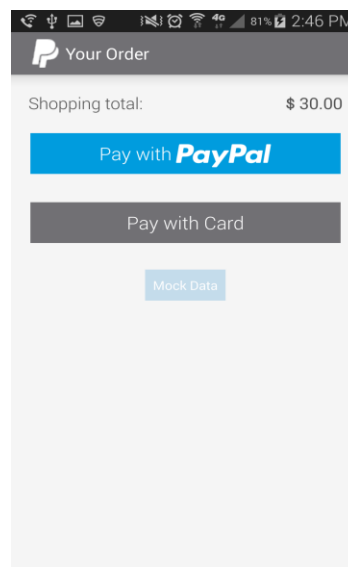


Figure 3.2.2.5.a

One of the big challenges we faced was moving the price total between activities. The total price is calculated in the shopping list activity. We needed to pass this value throughout the activities in the app. This process will be outlined in the section 3.2.2.6. We modified the PayPal code to be able to receive our total price value `cart_sum` and display this for the user to see.

```
@Override
protected void onResume() {
    super.onResume();
    //set's title at in the header

    setTitle("Checkout System");
    cart_sum="0.00";
    // bundle function pulls values from other activites
    Bundle extras = getIntent().getExtras();
    //if a value has been recieved set that value, if not set
    0.00
    if (extras != null) {
        //formatting so cart_sum visually appears as $##.##
        cart_sum = extras.getString("cart_sum");
        price_float=Float.parseFloat(cart_sum);
        formatted_price=String.format("%" + 4 + "." + 2 + "f",
        price_float);
        tvPrice.setText('$'+formatted_price);
    } else {
        formatted_price= "0.00";
        tvPrice.setText('$'+formatted_price);
    }
}
```

The team added and modified a lot of the code to make it suitable for our application. The code can be found in Appendix 3.6. The checkout application gives both flexibility and security. By implementing the PayPal SDK, the user can checkout with a credit card or PayPal. Also PayPal is an extremely secure way to move money. This is a good example of how the team addressed the sponsor's request of making the app secure and user friendly.[2.4]

3.2.2.6 Communicating Between Activities

As outlined in previous sections, an activity is essentially the code that is being run behind every individual screen you see within an Android application. These are by far the bulk of the application itself, and our project has six in total. Since every layout you will see in an application requires at least one activity, learning how to transition between these activities, as well as send useful data

In onCreate, we initialize the shopping list in the ListViewMain activity as a list of key-value pairs with no items inside it. This is because we do not want to continually redefine a new list each time the user views the list, so having the feature of the onCreate function allows things to be done once per run of the application. Next, onResume is run every time the activity is entered. This will happen much more frequently than onCreate, and is used in our application to build the shopping list. Each time an item is added, an Intent is created, and then onResume is run. This is also the function where data regarding a newly added item is received. In order to do this, we must use the method putExtra in conjunction with the use of Intent earlier.

```
Intent myIntent5 = new Intent(MainActivity.this,
    ListMainView.class);
Intent myIntent5 = new Intent(MainActivity.this,
    ListMainView.class);
myIntent5.putExtra("item_added", item_string);
myIntent5.putExtra("item_added_price", price_string_f);
MainActivity.this.startActivity(myIntent5);
```

In order to send data, such as an item name and price sent to the application via the postData function discussed earlier, we must first prepare the data to be sent. In the section of code above, we can see what happens when we want to perform this in our application. Similar to switching between activities, we create a new Intent to say where we are going. Next we use putExtra with two input parameters. The first is the name of the parameter to be sent. In this case it is "item_added". The second is the value of this parameter to be sent. We do this twice with the item and price, and then we start the new activity. To receive this data, we must do the following:

```
Bundle extras = getIntent().getExtras();
    if (extras != null) {
        item_added=extras.getString("item_added");
        item_added_price=extras.getFloat("item_added_price");
    }
```

Creating a variable of type “Bundle” we can store all the parameters sent over from the previous activity in one place. Next, we use a conditional to check that values were actually sent over. Without this if statement, the application could potentially crash. Then we use the method “extras.get(variable type)” to store the transmitted value into our new activity. It is important to match the name in quotes exactly the same way as in the other activity, or no values will be received. Using small segments of code defining necessary Intents, we can both transition between activities, and send data back and forth between them.

3.2.3 Scale Code

The software that was used to program the Arduino Yun to output the results to the server was Arduino 1.5.8. Once the Arduino has taken the voltage from the scale circuit and converted it to a value, the code will convert that number to a weight in grams and send it to the server.

Two libraries must be included to set up the connection between the Arduino and the server. The first library is “Bridge” which will serve as a bridge between the Arduino and Linux. The Arduino will take the weight from the scale and Linux will send it through Wi-Fi to the server.

```
#include <Bridge.h>
#include <HttpClient.h>
```

Then, another variable should be defined to output the number that the Arduino reads once it gets the reading from the scale circuit for an item. Consequently, other variables should be defined to tell the Arduino the frequency at which it should update the results. Also, other variables are defined to set the Arduino to read zero once an item is placed so it does not add the new item to the previous one accidentally.

```
float analogValueAverage = 0;
long timeReadings = 0;
long timeTransmit = 0;
int timeBetweenReadings = 50;
float tare_value = 0.0;
int tare_valid = 0;
```

Another two variables must be defined. The first one is to let the system know that there is a request and that it should wait for a response and the other one is to initialize an HttpClient.

```
int dataWait = 0;
HttpClient client;
```

For the setup, an LED light is turned on or off depending if there is a reading coming from the Arduino. Also, the tare equation is defiend.

```
void setup() {
    Serial.begin(9600);
    tare_value = loadB - (loadB-loadA)/(analogvalB-
analogvalA)*analogvalB;
    tare_valid = 1;
    pinMode(13,OUTPUT);
    digitalWrite(13, LOW);
    Bridge.begin();
    digitalWrite(13, HIGH);
    Serial.print("Bridge Open\n");
}
```

Once the main variables are defined and the setup function is declared, a void loop function must be declared to loop through the process. The first thing to do is to make the Arduino know what pin input it should be reading from. After the Arduino sends a weight of an item, a checking procedure is processed. If more than one second has passed, then the program will send a request and “dataWait” variable is set indicating that there is an outbound request. Then, if the request is ready, it will be processed and the weight of the item is updated. On the other hand, if the program did not count one second, then it will keep updating the weight. The following code summarizes the previous procedure.

[2.5]

```
int analogValue = analogRead(0);
    long unsigned int last_check = 0;
    char buffer[1024];
    int bufferIndex;
    char car;
```

```

float load;
analogValueAverage = 0.99*analogValueAverage +
0.01*analogValue;
if(millis() - timeReadings > timeBetweenReadings){
    load = analogToLoad(analogValueAverage);
    i++;
    if(i >= 10){
        i = 0;
    }
    timeReadings = millis();
}
if( (millis() - timeTransmit > 1000) && !dataWait){
    timeTransmit = millis();
    load = analogToLoad(analogValueAverage);
    String url =
String("http://192.168.1.2/ece480/system/?weight=")+String(load)+
String("&cart=")+CART_NUMBER;
    Serial.print(url);
    Serial.print("\n");
    client.getAsynchronously(url);
    dataWait = 1;//request
}
if(dataWait && client.ready()){

    Serial.print("\n");
    dataWait = 0;
    bufferIndex = 0;
    while(client.available() && bufferIndex < 1022){
        car = client.read();
        Serial.print(car);
        buffer[bufferIndex] = car;
        bufferIndex++;
    }
    buffer[bufferIndex] = 0;
    Serial.print(String("Received: "));

```

```
    Serial.print(buffer);  
    Serial.print("\n");  
}  
}
```

Finally, a function is defined to output the load. The function will return the results that the Arduino reads from the scale circuit and multiply that by a linear slope, which converts the weight into grams.

```
float mapfloat(float x)  
return 43.7939*(x);
```

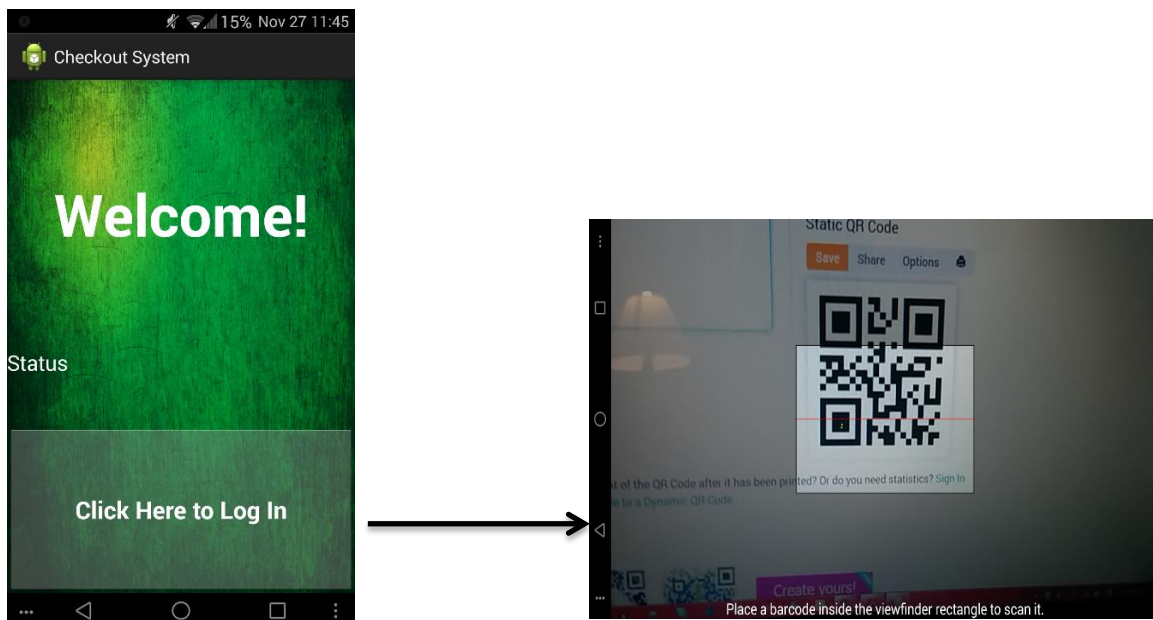
Chapter 4

4.1 Final Product Evaluation

While previous sections have detailed specific components of the application. It is important to detail how a typical interaction with it would work. This section outlines the order in which you might see each screen, as well as a short explanation of what each button/option will do. One of our main goals with the application itself was to make it both appealing to the eye, and user-friendly, without compromising features.

Initial Login

This is the first screen the user will see upon opening the application. At this point the only way to proceed is to log in. When this button is clicked, the user's camera will open, and require them to scan a valid QR code, which will be on the shopping cart itself. This links the user to the store they are in and a specific shopping cart, and gives them a unique 6-digit ID number. This also allows the server to be able to keep all lists separate, so that there is no chance of accidentally having to pay for someone's else's item. While we only have one cart at the moment as a prototype, this could easily be implemented on a larger scale. The following picture shows what the user would see upon pressing the login button. They must simply place the QR code within the on-screen box, and it will scan the code immediately.



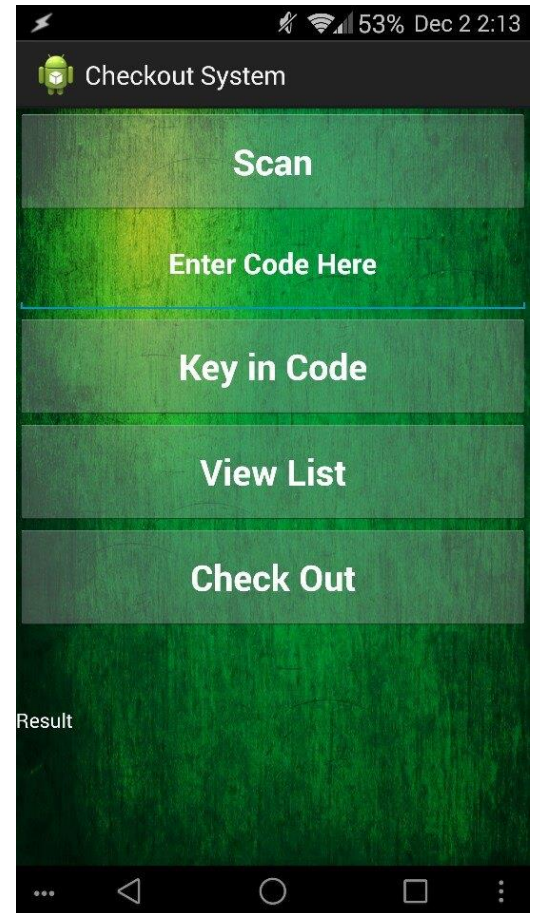
Main Screen

This is what the user sees upon successfully logging in. They will only be given access to this page once the server has given the green light in regards to the QR code previously scanned. In this screen we have several options. The first of which is to scan in a new item using the “Scan” button. Upon pressing this, a view similar to the previous image will pop up, and the user will place their item’s UPC within the viewfinder. Once successfully scanned, a toast message will show up at the bottom of the screen and say “Adding Item” (pictured) to let the user know that the code has been sent to the server, who is now verifying the items weight. The result message at the bottom allows for the user to view the numerical value of the code scanned in. This number is viewable at the bottom of almost every type of UPC you will see. This can be used to troubleshoot in case an invalid code was scanned, or if the correct code was not correctly read. An invalid code will result in another toast message telling the user that this was an invalid item, and therefore nothing will be added to the list.

The “Key In Item” button has the exact same function of the scan button except the user will type in a 4-digit code instead of scanning an item. This button will be used for items that do not have UPCs such as produce.

If the user simply wants to check their shopping list without adding an item, then can press “View List” and this will bring them to a page (outlined in the next section) that shows their items.

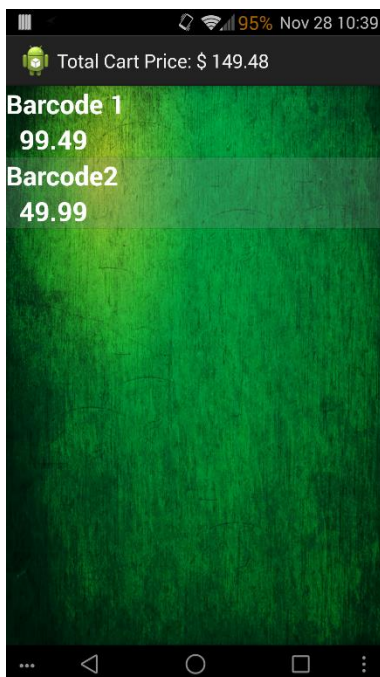
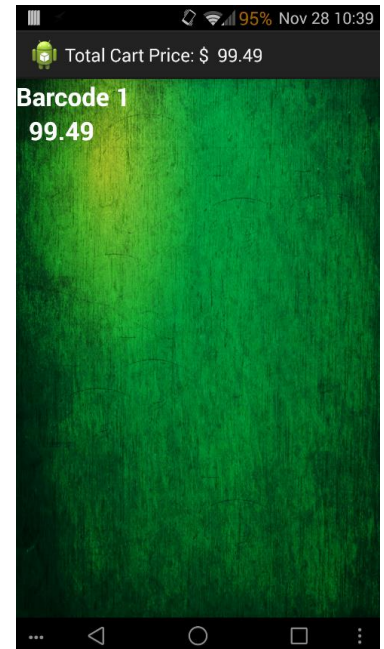
Finally, if the user decides to exit out of the application, another toast message will popup saying “Press the back button again to exit” which will force them to press back a second time before closing their session. This makes the application friendlier because it is harder to accidentally close out the application



while using it. It is very important that the system allows for user error and prevents user error in any way possible. All menus were designed to be as intuitive as possible and allow the user to undo anything they did not mean to do.

Shopping List

Upon a successful scan and response from the server, the newly scanned item (in this case, “Barcode 1”) will be added to your list, and the screen’s focus will automatically bring you to the list itself. The user is brought directly to this page so there is no confusion if the item was added. Here we see the item’s name, with the price beneath it. The title bar also updates with the current total cart price, so the user knows how much they need to pay. The user can return from this screen at any time using the back button on their phone.

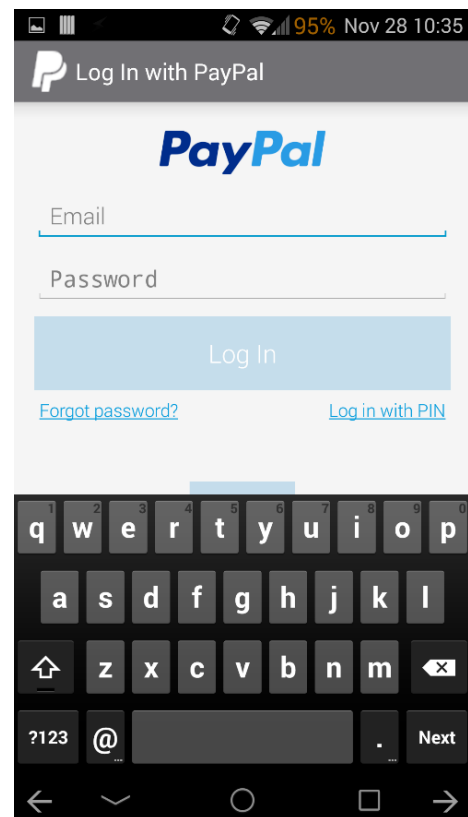
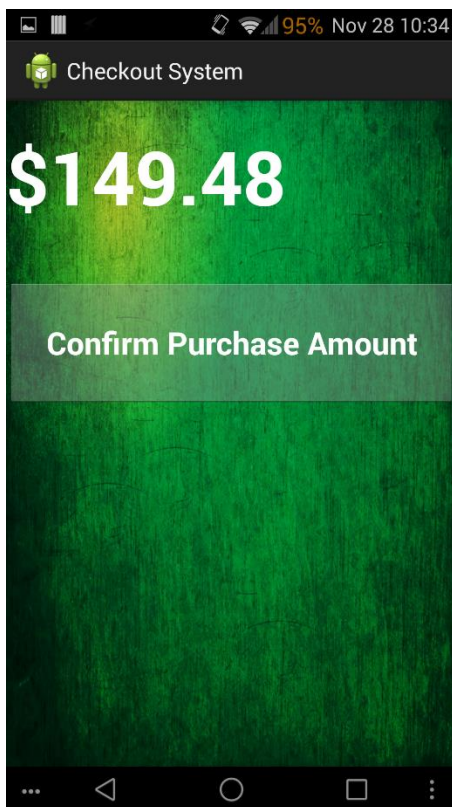


The user can continue to add items at this point to the list, and the cart total will update each time they return to the screen. If the user long-clicks on a particular item, it will become highlighted, and the user will receive a prompt to delete any items they might no longer desire. This deletion will send a POST request to the server, telling it to check for an updated cart weight. This is done to verify that the item was actually removed from the cart itself; otherwise it will remain on the list. The server checks that the new weight is equal to the remaining sum of the previous weight minus the removed item’s weight. This allows users to delete any items that

they may have unintentionally added or decided they no longer want.

Checking Out

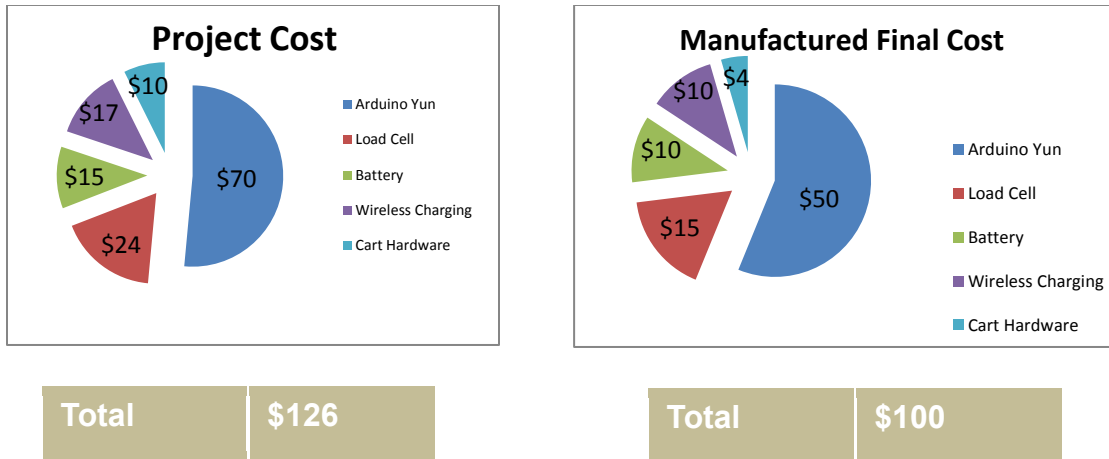
Once back on the main page, the user can select the “Check Out” option to finish shopping and pay for items. A screen will pop up with the final shopping cart total and allow the user to verify that this is in fact the correct amount. Upon pressing “Confirm Purchase Amount” the user will be brought to a PayPal login, where they can log in and send the payment over to the grocery store’s account.



It was very important to the team to design an app that can be used by individuals of all technical levels. Every button is straightforward and does exactly what a user would expect it to do. All the menus are very simple and the button's text and size make it very legible. The only place where users can mess up is by adding an incorrect item or exiting the app unintentionally. Both of these cases were accounted for and creative solutions were implemented to address these potential issues.

Chapter 5

5.1 Final Cost



The project cost is based on the parts used in the final design. The manufactured final cost is an estimated cost if we were to implement the product large scale. If the carts were to be produced in a manufacture setting, the cost per grocery cart would be much lower. Our system is very cost effective for stores to implement.

Although a substantial initial investment is required for the system, the money saved in personal costs will far surpass the cost of purchasing the system. The system was also designed to fit in standard grocery carts, so it is very simple for stores to implement the *point of sale grocery cart*. The costs savings and flexibility that the system provides to the store and it's customers make it a very appealing product with a lot of marketplace potential.

5.2 Schedule

Week	Tasks Accomplished
9/15-9/21	Finalize Pre-proposal Begin assembling of initial load cell design.
9/22-9/28	Build and test load cell circuit

9/29-10/5	Calibrate scale code and begin server setup
10/6-10/12	Research on creating an application using eclipse Implement wireless charging
10/13-10/19	Create initial application project
10/20-10/26	Write code to implement scanner into application
10/27-11/2	Code and modify editable list
11/3-11/9	Integrate shopping list, passing data to and from
11/10-11/16	Integrate login system into application
11/17-11/23	Finalize application and hardware layout. Update and finalize server code.
11/24-11/30	Discuss and collaborate regarding final report. Finalize application for demonstration. Finalize physical layout for load cells and circuitry.
12/1-12/5	Prepare presentation for design day.

5.3 Future Improvements

Many features could be added to the proposed design that could make it more valuable and efficient. A second load-cell could be easily added to the design to increase both the quantity the cart can handle and quality in terms of having less error probability.

Since smartphones usually do not last more than a day, a charging stand could be added on the cart that can charge customer's phone. As a result, the customers will not worry about their phones running low on battery.

An image-processing feature could be developed and added to the smartphone application. This feature will be able to identify items by taking a picture of them using the customer's smartphone. This will add more options to the shopping experience, in a way that customers will have a second option to identify items in case some errors occurred during the regular scanning process.

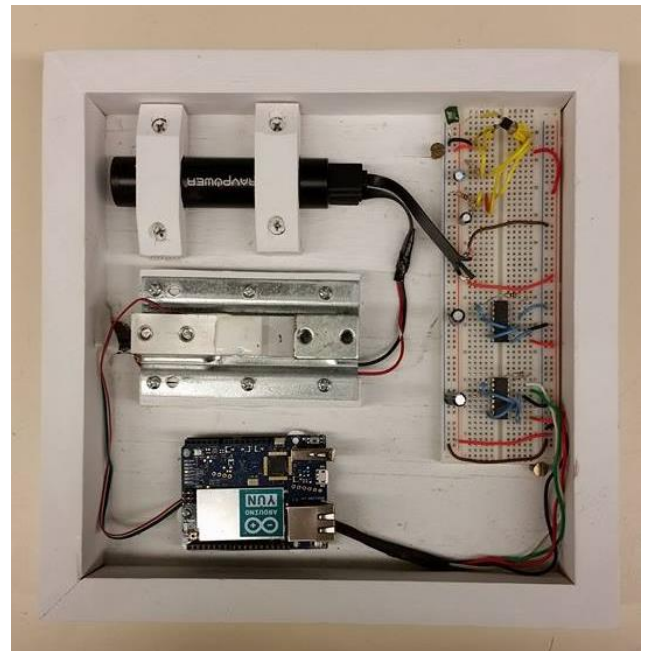
To save customer's time, a 21-age verification feature could be also added. Another feature that could be developed is location detection. If the customer has an issue or needs help, a button could be pressed on the phone that will alert and tell the cashier of the position of the customer. It is also good to

mention that this could be done wirelessly, that is, the cashier does not have to walk but simply call the customer and offer help.

Having an option of adding coupons to the checkout will be helpful to the customer. Once the customer has finished shopping, a message could appear on the phone screen asking if there are any coupons that the customer would like to use. After checking out the user could be automatically emailed a copy of their receipt to the PayPal email used to purchase the items.

5.4 Conclusion

The team successfully met the objectives and requirements of the design and completed a prototype that could be advertised to the market. The design can measure items with a resolution of .2 grams. Also, the Arduino Yun can take measurements from the scale with the help of the amplification circuit and send these weights to the server. In addition, customers can download the friendly-user application on their smartphones and use it right away. The application is able to scan barcodes, create a virtual shopping list, talk with the server, and checkout.



The team took into consideration the robustness of the different components in the design. The design should work properly even at extreme temperatures as the components were selected carefully. Moreover, the different parts are expected to have a high life expectancy making the cart last for years without the need to be replaced.

A Cost-effective solution was one of the main concerns for the team. The team was able to balance cost and system effectiveness perfectly. The system is very secure because all of the important data is stored on the server and no in the app, which is easily acceptable.

Above all the team was able to create an intuitive, user-friendly shopping experience that will allow shoppers to skip the long lines once and for all.

Appendix 1



Ben Lauzon – Manager

The technical role that Ben played on the project was developing the android phone application, designing the scale hardware and writing the scale code. The bulk of Ben's time was used to create the android application. With no previous app developing or Java experience he spent a lot of time doing background research on how to create a successful, user-friendly application through online forums, tutorials, and articles. Ben created the applications self-generating grocery list with deletion capabilities and the app's checkout system. He also helped write the code that allowed the application to communicate with the server and helped to debug the code in all aspects of the application throughout the semester. Each section of the app was originally created as separate projects to help with the debugging process. Ben and Matt spent a lot of their time combining these apps so that the end product looked as if it was created as a single project. Ben and Matt also spent a substantial amount of time making the application visually appealing and easy to use.

Ben also worked on the hardware design of the scale circuit. He contributed to the selection of the load cell and all of the parts used in the amplification circuit. He and Husain designed, tested, and optimized the circuit. The finished product was able to amplify the millivolt output of the load cell so that the microcontroller could detect minimal to drastic weight changes.

Ben's other large contribution to the project involved helping write the code that converted the voltage output of the amplification circuit to a reading in grams. He also helped write a section in the code that would zero out the initial weight of the scale.

Ben contributed to nearly all aspects of the project and used any free time he had to help other team members in any way that he could. It was very important

for him to know how every aspect of the project worked, not just the parts he was specifically working on.



Matt Rasmussen – Webmaster

The bulk of Matt's time spent on this project was developing the Android application. Matt began this at the very early stages, and was therefore able to educate other members such as Ben the basics of Eclipse and how some of the first sections of code worked. This allowed Ben to work alongside Matt to maximize the progress made on the application, seeing as it was one of the biggest portions of the project. Since no one in the group had any prior experience in developing a mobile application, or even java experience, a lot of learning was required by Matt and Ben in order to complete this project.

Matt spend a large amount of time on his own following tutorials and learning the language, as well as techniques that could be later used to benefit this project.

Within specific parts of the application, Matt initially spent a great deal of time building the scanning portion of the application. This is where Matt learned his way around eclipse and became familiar with the layout of java code.

Upon completing the scanning portion, Matt then moved his focus to sending values to the web server. This was done in conjunction with Steven, who set up the server. Steven was able to set up a page that logged contacts to the server from the application, which allowed Matt to test the communications portion to ultimately get it to send the desired parameters.

Next Matt worked with the list code built by Ben to make it properly work with this application. Giving it the ability to build the list, total the cart value, and remove items was something Matt spend a great deal of time ironing out. Upon completing a new section of code Matt and Ben would work together to integrate and tailor it to suit the group's needs.

While Matt spent the bulk of his time developing the Android application, since he was at nearly every group meeting, he also provided help debugging the

load cell code, selecting components/products used in the project, and providing general help as needed.



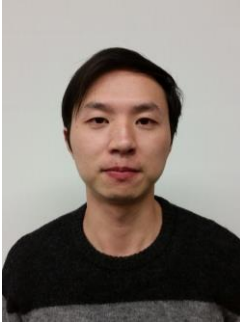
Shuangfei Liu – Document Prep

Shuangfei's technical role was being responsible for documents, testing voltage, working on wire connecting in the circuit, and web design. He helped prepare and presentation slides, booklet, and sections of required assignments. Shuangfei also helped to assemble and disassemble the load cells. He also did voltage testing on the wireless charging circuit.

He also helped Taoping and Husain with the poster about the project. He helped by adding more information into the poster and changing the overview of the data. Shuangfei also did the research about the wireless charging with Taoping and Husain, and made sure it was the cheapest one and would acceptable for the project. The wireless charging included a transmitter and a receiver circuit. The team preferred to choose a 12V DC transmitter regulated circuit that would output 5 V. Since these wireless charging units are expensive, the decision they made was based on the cost and usefulness for the receiver.

Another big contribution he did is web design. He spent a lot of time designing the team webpage. He collected all the materials and documents and put them into the webpage. He chose the layout and style of the webpage and wrote the HTML code. Every time new materials or information about the project were written, he would add and collect them into the website. Also, he wrote a tutorial about making a simple website using html. In addition, Shuangfei was also assisting the team for any work such as, finding materials, information, and placing online orders. He always tried himself to do the best for the project. Some parts of the project such as the microcontroller and the programming for the

server and application were not very familiar to him, but he still tried to figure out how they work.



Taoping Zhu - Document Prep

The technical role of Taoping in this project was researching the design of wireless charging with Husain. He and Husain searched for some useful information online, which could help the team to design the wireless charging parts. During the research for the transmitter and receiver, they choose the 12VDC transmitter regulated circuits and the 5V output voltage receiver. For ordering these parts of wireless charging circuits, they also compared prices from different websites and chose the best. Taoping and Husain also searched for the best battery, which could supply power for the Arduino Yun microcontroller and be charged from the wireless receiver circuit.

After the wireless charging parts and battery delivered, Taoping and Shuangfei tested the voltages coming from the battery and wireless charging transmitter to make sure it could be connected to the microcontroller without causing damage. To make the wireless charging circuits be tested additional wires were cut and added to the charging circuit. Finally they soldered these wires to connect with the wireless charging circuits

Taoping also helped the team prepare presentation slides, papers, and some other assignment. Shuangfei and Taoping created the web page of the team. Without any webpage design experience at start, He and Shuangfei spent a lot of time to learn how to make a webpage. Taoping was mainly responsible to gather documents into the website folder, and give ideas about designing the webpage. The team project webpage was designed through Dreamweaver. They spent a lot of time on designing the style of the webpage and writing html code. When other teammates needed some assistance, Taoping tried his best to help them in anyway possible.



Husain Aleid – Presentation Prep

Husain's main technical roles were circuit building, parts selection, scale troubleshooting and calibration. Husain and Ben cooperated on building the amplifier circuit and chose the appropriate resistor that make the amplifier output no more than 5 volts.

It was Husain's and Taoping's responsibility to search for a wireless charging circuit that includes both a transmitter and a receiver circuits. After intensive research, they decided to pick a 12VDC transmitter regulated circuit which will be placed near the receiver circuit that will output 5 volts. Also, they took into consideration the price of these circuits as they tend to be expensive. Thus, their decision was based on the cost and the maximum distance at where the receiver can output.

Husain and Taoping also had to search for a battery that could be charged from the wireless charging circuit and supply power to the Arduino Yun. After finding many batteries, they selected a 3000 mA battery that should last for at least three hours, again cost was taking into consideration. Also, Husain selected the voltage regulator that will take the output voltage from the battery –which happens to be 5.17 volts- and regulate it to 4.99 volts, in that case, potential damage to the Arduino due to high voltage is prevented. Then, he selected the resistor and the capacitor that must be used to output the desired voltage.

For software, Husain helped Ben write the calibration codes for the scale to make sure the program output the correct weight of the items. He also helped Matt researching some eclipse programs that can read barcodes once they are loaded on the smartphone. At the end of the design, few issues with the

calibration reoccurred and Husain with the suggestions of other members worked to fix that. Husain also researched eclipse programs that will help create the shopping list. He helped Ben find and implement an initial code that lists multiple item prices.



Steven Hartz – Lab Manager

Steven contributed to the design of many of the subsystems of the shopping cart during the conceptual phase. As the designs were refined and grew more detailed, he specialized in the subsystems in which he had previous experience and expertise. He is responsible for the network protocol used to communicate between the cart and server as well as the protocol used to communicate between the phone and server.

The implementation of the cart-server protocol above the HTTP layer was written by Steven, both on the microcontroller and the web server. While he did not write the phone-server implementation within the app, he did write the protocol implementation on the server as well as develop numerous development tools to aid the app developers' networking implementation. He is also responsible for the entirety of the server's PHP code, including all the weight checking and shopping session management.

Steven acted in a consolatory role in the development of the microcontroller code that converts ADC output values into real weights. In this role, he helped develop initial code, which was later revised, and offered design suggestions to those developing the code.

Wi-Fi network setup was also performed by Steven. This included DHCP management and assigning a static IP address to the server so that it could be reliably contacted by both the cart and app. He was also responsible for the server setup, from operating system installation through the selection and installation of HTTP server, database server, and PHP packages.

Appendix 2

- [2.1] <https://tools.ietf.org/html/rfc7230>
- [2.2] <http://developer.android.com/training/volley/index.html>
- [2.3] <http://www.vogella.com/tutorials/AndroidListView/article.html>
- [2.4] <https://developer.paypal.com/docs/integration/mobile/mobile-sdk-overview/>
- [2.5] <http://arduinotronics.blogspot.com>
- [2.6] <http://khurramitdeveloper.blogspot.com/p/android-barcode-scan-using-zxing-library.html>

Appendix 3

3.1

```
// Get the initial time for 1 second loop time.
$scanTime = time();

// Get the item's data or return an error is no such item exists.
$q = "SELECT * FROM items WHERE upc = '". $upc ."' LIMIT 1";
$item = $db->GetRow($q);
if($item == false){
    die(json_encode(array('error'=>'Internal error: database')));
}
if(empty($item)){
    return false;
}

// Get most recent activity data, including cart weight at the time.
$q = "SELECT * FROM session_logs WHERE session_id = " . $sessionID . "
ORDER
                                BY log_id
DESC LIMIT 1";
$temp = $db->GetRow($q);
$old_weight = $temp['weight'];

// Determine which cart is associated with this session
$q = "SELECT cart_id FROM cartSessions WHERE session_id = " .
$sessionID;
$cartID = $db->GetOne($q);

// Flag to continue looping, time for 5 second timeout.
$keepLooping = true;
$startTime = microtime(true);
do{
    // Get information about the cart, most importantly current
    weight
```

```

    $q = "SELECT * from carts WHERE cart_id = " . $cartID;
    $cart = $db->GetRow($q);

    // Calculate difference between cart weight and expected cart
weight
    $difference = abs($cart['current_weight'] - ($old_weight -

    $item['weight']));

    // If a second has elapsed
    if(strtotime($cart['weight_updated']) - $scanTime > 1){
        //10g threshold, defined elsewhere
        if( $difference < THRESHOLD ){
            // Array of data is built to use database library
function
                $data = array('session_id' => $sessionID,
                                'mode'      => 'remove',
                                'item_id'   => $item['item_id'],
                                'weight'    => $cart['current_weight']
                                );
                $db->AutoExecute('session_logs',$data,'INSERT');
                $keepLooping = false;
            }
        }
        if($keepLooping){
            usleep(1e6); //Sleep 1 second.
        }
    }while($keepLooping && (microtime(true) - $startTime <
WEIGHT_TIMEOUT));

    if($keepLooping){
        //We never saw the correct weight, issue invalid code.
        echo json_encode(array('error'=>
                                'Item weight does not match cart
weight')));
    }else{
        echo '$'.$item['name'].'$'.$item['price'].'$';
    }
}

```

3.2

```

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/bgimag"
    android:orientation="vertical" >
<Button
    android:id="@+id/btnScan"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Scan"
    android:textColor="@android:color/white"
    android:textSize="25sp"
    android:textStyle="bold" />

```

```

<Button
    android:id="@+id/button1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="View List"
    android:textColor="@android:color/white"
    android:textSize="25sp"
    android:textStyle="bold" />
<EditText
    android:id="@+id/editText1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:ems="10"
    android:hint="item to be added to shopping list"
    android:textColor="@android:color/white">
    <requestFocus />
</EditText>
<Button
    android:id="@+id/button2"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Add to List"
    android:textColor="@android:color/white"
    android:textSize="25sp"
    android:textStyle="bold" />
<Button
    android:id="@+id/button3"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:text="Show me the money"
    android:textColor="@android:color/white"
    android:textSize="25sp"
    android:textStyle="bold" />
<TextView
    android:id="@+id/tvStatuslbl"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="0.03"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textColor="@android:color/white" />
<TextView
    android:id="@+id/tvStatus"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:textColor="@android:color/white"
    android:textAppearance="?android:attr/textAppearanceLarge" />
<TextView
    android:id="@+id/tvResultlbl"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="0.03"
    android:textColor="@android:color/white" />

```

```

        <TextView
            android:id="@+id/tvResult"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="2.95"
            android:textColor="@android:color/white"
        />
    </LinearLayout>

```

3.3

```

public class ListMainView extends Activity {
    // Declare Variables
    ListView list;
    ListViewAdapter listviewadapter;
    public static List<Shopping> shoppinglist = new
        //Intialize List
        ArrayList<Shopping>();
    String[] Item;
    String[] Price;
    String item_added;
    float item_added_price;
    String formatted_price;
    public static double price_sum;
    public static String formatted_price_sum;
    public static List pricelist = new ArrayList();
    public static String price_string;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // Get the view from listview_main.xml
        setContentView(R.layout.listview_main);
        Item = new String[]{" "};
        Price = new String[]{" "};
        // Locate the ListView in listview_main.xml
        list = (ListView) findViewById(R.id.listview);
        // Pass results to ListViewAdapter Class
        listviewadapter = new ListViewAdapter(this,
            R.layout.listview_item, shoppinglist);
        // Binds the Adapter to the ListView
        list.setAdapter(listviewadapter);
        list.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
    }
}

```

```

        // Capture ListView item click
        list.setMultiChoiceModeListener(new MultiChoiceModeListener() {
@Override
public void onItemCheckedStateChanged(ActionMode mode,
    int position, long id, boolean checked) {
    // Capture total checked items
    final int checkedCount = list.getCheckedItemCount();
    // Set the CAB title according to total checked items
    mode.setTitle(checkedCount + " Selected");
    // Calls toggleSelection method from ListViewAdapter Class
        listviewadapter.toggleSelection(position);
    }
@Override
    public boolean onActionItemClicked(ActionMode mode, MenuItem
    item) {
        switch (item.getItemId()) {
            case R.id.delete:
                // Calls getSelectedIds method from ListViewAdapter Class
                SparseBooleanArray selected = listviewadapter
                    .getSelectedIds();
                // Captures all selected ids with a loop
                for (int i = (selected.size() - 1); i >= 0; i--) {
                    if (selected.valueAt(i)) {
                        Shopping selecteditem = listviewadapter
                            .getItem(selected.keyAt(i));
                        postData(selecteditem.Item, selecteditem);
                        Toast.makeText(getApplicationContext(), "Removing
                            Item...", Toast.LENGTH_SHORT).show();
                    }
                }
            mode.finish();
            return true;
            default: return false;
        }
    }
@Override
    public boolean onCreateActionMode(ActionMode mode, Menu
    menu) {
        mode.getMenuInflater().inflate(R.menu.activity_main, menu);

```

```

        return true;
    }

    @Override
    public void onDestroyActionMode(ActionMode mode) {
        listviewadapter.removeSelection();
    }

    @Override
    public boolean onPrepareActionMode(ActionMode mode, Menu
        menu) {
        return false;
    }
});

@Override
protected void onResume() {
    super.onResume();
    Bundle extras = getIntent().getExtras();
    if (extras != null){
        item_added=extras.getString("item_added");
        item_added_price=extras.getFloat("item_added_price");
        formatted_price=String.format("%" + 7 + "." + 2 +
            "f", item_added_price);
        Item[0]=item_added;
        Price[0]=formatted_price;
        price_sum+=item_added_price;
        price_string= String.valueOf(price_sum);
        formatted_price_sum=String.format("%" + 7 + "."
            + 2 + "f", price_sum);
        pricelist.add(item_added_price);
    }
    if (formatted_price_sum != null){
        setTitle("Total Cart Price: $" + formatted_price_sum);
    }else{
        setTitle("Total Cart Price: $0.00");
    }
    if(Item[0] != "" && Price[0] != ""){
        for (int i = 0; i < Item.length; i++) {
            Shopping shopping = new Shopping(Item[i], Price[i]);

```

```

        ListMainView.shoppinglist.add(shopping);
    }
}

@Override
protected void onPause() {
    super.onPause();
    Bundle extras = new Bundle();
    Intent intentprice = new Intent(ListMainView.this,
        MainActivity.class);
    extras.putString("cart_sum", price_string);
    intentprice.putExtras(extras);
    startActivity(intentprice);}
}

private void postData(final String item_name, final Shopping
selecteditem) {
    RequestQueue rq = Volley.newRequestQueue(this);
    StringRequest postReq = new
StringRequest(Request.Method.POST, "http://192.168.1.2/system2/",
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            if(response.length() != 0){
                if(response.charAt(0) == '$'){
                    Toast.makeText(getBaseContext(), "Item Removed",
                        Toast.LENGTH_SHORT).show();
                    listviewadapter.remove(selecteditem);
                    float
                        b=Float.parseFloat(selecteditem.Price);
                    price_sum-=b;
                    price_string= String.valueOf(price_sum);
                    formatted_price_sum=String.format("%" + 7 +
                        "." + 2 + "f", price_sum);
                    setTitle("Total Cart Price: $" +
                        formatted_price_sum);
                }else{
                    Toast.makeText(getBaseContext(), "Invalid
                        Cart Weight", Toast.LENGTH_SHORT).show();
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            System.out.println("Error [" +error+ "]");
        }
    }) {
        @Override

```

```

        protected Map<String, String> getParams() throws
            AuthFailureError {
            Map<String, String> params = new HashMap<String,
                String>();
            params.put("item_name", item_name);
            params.put("mode", "remove");
            params.put("phone", "1");
            if(session_id.length() == 0){
                session_id="YOU ARE BAD";
            }
            params.put("session_id", session_id);
            return params;
        }

    };
    postReq.setRetryPolicy(new
DefaultRetryPolicy(0,0,DefaultRetryPolicy.DEFAULT_BACKOFF_M
    ULT));
    rq.add(postReq);
}

}

```

3.4

```

public class MainActivity extends Activity {
    TextView tvResult;
    TextView tvSend_status;
    //EditText addEdit;

    public static String price_sum;
    public static String formatted_price_sum;
    public static float cart_sum;
    public static String session_id ;
    Bundle extras1;

    //    /** Called when the user clicks the Send button */
    //    public void sendMsage(View view) {
    //        // Do something in response to button
    //        Intent intent = new Intent(this, GetR.java);
    //        EditText editText = (EditText)
    findViewById(R.id.edit_message);
    //        String message = editText.getText().toString();
    //        intent.putExtra(EXTRA_MESSAGE, message);
    //        startActivity(intent);
    }
}

```

```

//      }

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    setTitle("Checkout System");

    tvResult = (TextView) findViewById(R.id.tvResult);
    //tvSend_status = (Textview)
findViewById(R.id.Send_status);
    Button scanBtn = (Button) findViewById(R.id.btnScan);
    Button listBtn = (Button) findViewById(R.id.button1);
    //Button addBtn = (Button)
findViewById(R.id.button2);
    Button payBtn = (Button) findViewById(R.id.button3);

    //addEdit=(EditText) findViewById(R.id.editText1);

    scanBtn.setOnClickListener(buttonhandler);
    listBtn.setOnClickListener(buttonhandler);
    //addBtn.setOnClickListener(buttonhandler);
    payBtn.setOnClickListener(buttonhandler);

    //if(session_id == null){
    extras1 = getIntent().getExtras();
    Intent intent_receive=getIntent();
    session_id =
intent_receive.getStringExtra("session_id");
    //}

    //Bundle extras_create = getIntent().getExtras();

    //if(extras_create != null){
    //    session_id =
extras_create.getString("session_id");
    //}

```

```

        //Toast.makeText(getApplicationContext(), "You have now
logged in", Toast.LENGTH_SHORT).show();

        /*addEdit.setOnClickListener(
            new View.OnClickListener()
            {
                public void onClick(View view)
                {

Log.v("EditText",addEdit.getText().toString());

                }

            })*/;
    }
    @Override
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        setIntent(intent);
        Bundle b=getIntent().getExtras();
    }

    View.OnClickListener buttonhandler=new
View.OnClickListener() {
    public void onClick(View v) {
        switch(v.getId()) {
            //Case statements handle which button has been pressed
            case R.id.btnScan:
                try {
                    Intent intent = new Intent(
                        "com.google.zxing.client.android.SCAN");
                    intent.putExtra("SCAN_MODE",
"QR_CODE_MODE,PRODUCT_MODE");
                    startActivityForResult(intent, 0);

                } catch (Exception e) {
                    e.printStackTrace();
                    Toast.makeText(getApplicationContext(), "ERROR:" + e,
                        1).show();
                }
            }
        }
    }
}

```

```

        break;
    case R.id.button1:
        Intent myIntent2 = new Intent(MainActivity.this,
            ListMainView.class);
        MainActivity.this.startActivity(myIntent2);
        break;
    case R.id.button3:
        Intent myIntent4 = new Intent(MainActivity.this,
            SampleActivity.class);
        myIntent4.putExtra("cart_sum",
            Float.toString(cart_sum));
        MainActivity.this.startActivity(myIntent4);
        break;
    }
}

};

//Retrive UPC results from ZXing
public void onActivityResult(int requestCode, int
resultCode,
    Intent intent) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            //Sends the data to the server

            postData(intent.getStringExtra("SCAN_RESULT"),
                session_id); //posts
            upc,session_id,mode=add,
                and phone=1
        } else if (resultCode == RESULT_CANCELED) {
            tvResult.setText("Scan cancelled.");
        }
    }
}

long lastPress;
@Override
public void onBackPressed() {
    long currentTime = System.currentTimeMillis();

```



```

        forloop_1:
        for(int i=0; i<response.length(); i++){
            if (i >= 1){
                if(response.charAt(i) == '$'){
                    k=i;
                    break forloop_1;
                }else{
                    name_list.add(response.charAt(i));
                }
            }
        }
        forloop_2:
        for(int i=k+1; i<response.length(); i++){
            if(response.charAt(i) == '$'){
                break forloop_2;
            }else{
                price_list.add(response.charAt(i));
            }
        }

        String item_string=name_list.toString().replaceAll(", |\\[|\\]",
        "");
        String price_string=price_list.toString().replaceAll(",
        |\\[|\\]", "");
        float price_string_f = Float.parseFloat(price_string);
        Intent myIntent5 = new Intent(MainActivity.this,
        ListMainView.class);
        myIntent5.putExtra("item_added", item_string);
        myIntent5.putExtra("item_added_price",price_string_f);
        myIntent5.putExtra("session_id",session_id);
        MainActivity.this.startActivity(myIntent5);
    }else{
        Toast.makeText(getApplicationContext(),"Invalid Item Scanned",
        Toast.LENGTH_LONG).show();
    }
}

}, new Response.ErrorListener() {
    @Override

```

```

        public void onErrorResponse(VolleyError error)
        {
            System.out.println("Error [" + error + "]);
        }
    }) {
        @Override
        protected Map<String, String> getParams()
        throws AuthFailureError {
            Map<String, String> params = new
HashMap<String,
                String>();
            params.put("UPC_code", param1);
            params.put("session_id", param2);
            params.put("phone", "1");
            params.put("mode", "add");
            return params;
        }
    };

    postReq.setRetryPolicy(new
DefaultRetryPolicy(0,0,DefaultRetryPolicy.DEFAULT_BACKOFF_MULT));
    rq.add(postReq);
}

```

3.5

```

public class ListViewAdapter extends ArrayAdapter<Shopping> {
    // Declare Variables
    Context context;
    LayoutInflater inflater;
    List<Shopping> shoppinglist;
    private SparseBooleanArray mSelectedItemsIds;
    public ListViewAdapter(Context context, int resourceId,
        List<Shopping> shoppinglist) {
        super(context, resourceId, shoppinglist);
        mSelectedItemsIds = new SparseBooleanArray();
        this.context = context;
        this.shoppinglist = shoppinglist;
        inflater = LayoutInflater.from(context);
    }
    private class ViewHolder {

```

```

        TextView Item;
        TextView Price;
    }

    public View getView(int position, View view, ViewGroup
parent) {
        final ViewHolder holder;
        if (view == null) {
            holder = new ViewHolder();
            view = inflater.inflate(R.layout.listview_item,
null);
            // Locate the TextViews in listview_item.xml
            holder.Item = (TextView)
                view.findViewById(R.id.Item);
            holder.Price = (TextView)
                view.findViewById(R.id.Price);
            view.setTag(holder);
        } else {
            holder = (ViewHolder) view.getTag();
        }
        // Capture position and set to the TextViews
        holder.Item.setText(shoppinglist.get(position).getItem());
        holder.Price.setText(shoppinglist.get(position).getPrice());
        return view;
    }

    @Override
    public void remove(Shopping object) {
        shoppinglist.remove(object);
        notifyDataSetChanged();
    }

    public List<Shopping> getShopping() {
        return shoppinglist;
    }

    public void toggleSelection(int position) {
        selectView(position, !mSelectedItemsIds.get(position));
    }

    public void removeSelection() {
        mSelectedItemsIds = new SparseBooleanArray();
        notifyDataSetChanged();
    }

```

```

    }

    public void selectView(int position, boolean value) {
        if (value)
            mSelectedItemsIds.put(position, value);
        else
            mSelectedItemsIds.delete(position);
        notifyDataSetChanged();
    }

    public int getSelectedCount() {
        return mSelectedItemsIds.size();
    }

    public SparseBooleanArray getSelectedIds() {
        return mSelectedItemsIds;
    }
}

```

3.6

```

public class SampleActivity extends Activity {
    private static final String TAG = "paymentExample";
    TextView tvPrice;
    public static float price_float;
    public static String formatted_price:
    /* - Set to PaymentActivity.ENVIRONMENT_PRODUCTION to move real
money.
    - Set to PaymentActivity.ENVIRONMENT_SANDBOX to use your test
credentials

    - Set to PayPalConfiguration.ENVIRONMENT_NO_NETWORK to kick the
tires
    without communicating to PayPal's servers.
*/
    private static final String CONFIG_ENVIRONMENT =
PayPalConfiguration.ENVIRONMENT_NO_NETWORK;
    public static String cart_sum="0" ;
    // note that these credentials will differ between live & sandbox
environments.
    private static final String CONFIG_CLIENT_ID = "credential from
developer.paypal.com";

    private static final int REQUEST_CODE_PAYMENT = 1;
    private static final int REQUEST_CODE_FUTURE_PAYMENT = 2;
    private static final int REQUEST_CODE_PROFILE_SHARING = 3;

    private static PayPalConfiguration config = new
PayPalConfiguration()
        .environment(CONFIG_ENVIRONMENT)
        .clientId(CONFIG_CLIENT_ID)

```

```

        // The following are only used in
        PayPalFuturePaymentActivity.
        .merchantName("Hipster Store")

        .merchantPrivacyPolicyUri(Uri.parse("https://www.example.com/privacy"))

        .merchantUserAgreementUri(Uri.parse("https://www.example.com/legal"));
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.pp_activity_main);
        tvPrice = (TextView) findViewById(R.id.tvPrice);
        Intent intent = new Intent(this, PayPalService.class);
        intent.putExtra(PayPalService.EXTRA_PAYPAL_CONFIGURATION,
        config);
        startService(intent);
    }
    public void onBuyPressed(View pressed) {
        /*
         * PAYMENT_INTENT_SALE will cause the payment to complete
        immediately.
         * Change PAYMENT_INTENT_SALE to
         * - PAYMENT_INTENT_AUTHORIZE to only authorize payment and
        capture funds later.
         * - PAYMENT_INTENT_ORDER to create a payment for
        authorization and capture
         * later via calls from your server.
         *
         * Also, to include additional payment details and an item
        list, see getStuffToBuy() below.
         */
        PayPalPayment thingToBuy =
        getThingToBuy(PayPalPayment.PAYMENT_INTENT_SALE);
        /*
         * See getStuffToBuy(..) for examples of some available payment
        options.
         */
        Intent intent6 = new Intent(SampleActivity.this,
        PaymentActivity.class);
        intent6.putExtra(PaymentActivity.EXTRA_PAYMENT, thingToBuy);
        startActivityForResult(intent6, REQUEST_CODE_PAYMENT);
    }
    @Override
    protected void onResume() {
        super.onResume();
        //set's title at in the header
        setTitle("Checkout System");
        cart_sum="0.00";
        // bundle function pulls values from other activites
        Bundle extras = getIntent().getExtras();
        //if a value has been recieved set that value, if not set 0.00
        if (extras != null){
            //formatting so cart_sum visually appears as $##.##
            cart_sum = extras.getString("cart_sum");
            price_float=Float.parseFloat(cart_sum);
            formatted_price=String.format("%" + 4 + "." + 2 + "f",
        price_float);

```

```

        tvPrice.setText('$'+formatted_price);
    }else{
        formatted_price= "0.00";
        tvPrice.setText('$'+formatted_price);
    }
}

private PayPalPayment getThingToBuy(String paymentIntent) {
    return new PayPalPayment(new BigDecimal(cart_sum), "USD",
"Shopping total:",
        paymentIntent);
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
    if (requestCode == REQUEST_CODE_PAYMENT) {
        if (resultCode == Activity.RESULT_OK) {
            PaymentConfirmation confirm =
data.getParcelableExtra(PaymentActivity.EXTRA_RESULT_CONFIRMATION);
            if (confirm != null) {
                try {
                    Log.i(TAG, confirm.toJSONString());
                    Log.i(TAG,
confirm.getPayment().toJSONObject().toString());
                    /*
                     * TODO: send 'confirm' (and possibly
confirm.getPayment() to your server for verification
                     * or consent completion.
                     * See
https://developer.paypal.com/webapps/developer/docs/integration/mobile/verify-mobile-payment/
                     * for more details.
                     *
                     * For sample mobile backend interactions, see
                     * https://github.com/paypal/rest-api-sdk-python/tree/master/samples/mobile\_backend
                     */
                    Toast.makeText(
                        getApplicationContext(),
                        "PaymentConfirmation info received from
PayPal", Toast.LENGTH_LONG)
                        .show();
                } catch (JSONException e) {
                    Log.e(TAG, "an extremely unlikely failure
occurred: ", e);
                }
            }
        } else if (resultCode == Activity.RESULT_CANCELED) {
            Log.i(TAG, "The user canceled.");
        } else if (resultCode ==
PaymentActivity.RESULT_EXTRAS_INVALID) {
            Log.i(
                TAG,
                "An invalid Payment or PayPalConfiguration was
submitted. Please see the docs.");
        }
    } else if (requestCode == REQUEST_CODE_FUTURE_PAYMENT) {

```



```

    PayPal", Toast.LENGTH_LONG)
        .show();

    } catch (JSONException e) {
        Log.e("ProfileSharingExample", "an extremely
unlikely failure occurred: ", e);
    }
}

} else if (resultCode == Activity.RESULT_CANCELED) {
    Log.i("ProfileSharingExample", "The user canceled.");
} else if (resultCode ==
PayPalFuturePaymentActivity.RESULT_EXTRAS_INVALID) {
    Log.i(
        "ProfileSharingExample",
        "Probably the attempt to previously start the
PayPalService had an invalid PayPalConfiguration. Please see the
docs.");
}

}

}

@Override
public void onDestroy() {
    // Stop service when done
    stopService(new Intent(this, PayPalService.class));
    super.onDestroy();
}
}

```