UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

Development of a Cashless Vending Machine

by

JC Lock
16016548

Mechatronic Project 488

Final Report

Department of Mechanical and Mechatronic Engineering,
Stellenbosch University,
Private Bag X1, Matieland 7602.

Supervisor: Prof. G-J van Rooyen

October 2013

Declaration

I, the undersigned, hereby declare that the work contained within this report is my own, original work.

Signature: .....
JC Lock

Date: .....

Acknowledgements

I would like to thank my project supervisor, Prof. G-J van Rooyen, for his support and guidance throughout this project and J.P. Meijers for his help with some of the electronic design aspects of this project.

Lastly I would like to thank my parents, Koos and Elzahn Lock, my brother and all my family and friends for their continued support throughout my life.

MECHATRONIC PROJECT 488: SUMMARY

Student: JC Lock
Co-worker: N/A

Title of Project
Development of a Cashless Vending Machine.
Objectives
Program and build a model vending machine which accepts payments made via a cellphone.
Which aspects of the project are new/unique?
The use of simple cellphone technologies which most students currently have built into in their cellphones, such as NFC and QR Codes.
What are the findings?
A working test model was built which accepts faux money paid with either QR Codes or NFC. All the necessary security measures, i.e. encryption and challenge codes, were added as well as a central web server that tracks user transactions.
What value do the results have?
The results show that the machine is working reliably. Some improvements can be made, but overall the vending machine is working as expected.
If more than one student is involved, what is each one's contribution?
Only one student was involved in this project.
Which aspects of the project will carry on after completion?
This project has been successfully completed and will therefore not continue.
What are the expected advantages of continuation?
N/A
What arrangements have been made to expedite continuation?
N/A

Student	Date	Lecturer

Abstract

Currently, Stellenbosch University only has cash-based vending machines available. This means that a physical cash transaction has to take place before the vending machine can dispense its products. In a world moving away from using cash, this payment approach becomes a problem since less people are likely to carry around cash on their person. Therefore, it is important that vendors and manufacturers keep up with the trend and implement the latest technologies in their products and payment methodologies.

This project focused on introducing a vending machine system that will allow customers to pay for their products using only their cellphones. The project was designed to use existing technologies, services and protocols wherever possible. The system is based mainly on the Python programming language, with some Java code implemented in the Android application.

The two main technologies implemented are Quick Response Codes (QR Codes) and Near Field Communication (NFC). An Android application was created to facilitate payments made via NFC. A cloud-based server was also created and is used by the vending machine and Android application to validate transactions. To demonstrate that the system works, a model vending machine was designed and built that uses the cashless payment system.

The results of this project are favourable and show that it is possible to make a cashless vending machine. However, more work still needs to be done to fully commercialise and optimise this system and make it more user friendly.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

AWS	Amazon Web Services
BJT	Bipolar Junction Transistor
EC2	Elastic Compute Cloud
EMF	Electromotive Force
GPIO	General Purpose Input Output
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
i^2c	Inter-Integrated Circuit
LLCP	Logical Link Control Protocol
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
OS	Operating System
QR Code	Quick Response Code
RFID	Radio Frequency Identification
RSA	Ron Rivest, Adi Shamir and Leonard Adleman
SNEP	Simple NDEF Exchange Protocol
SPI	Serial Peripheral Interface
SU	Stellenbosch University
USSD	Unstructured Supplementary Service Data
UART	Universal Asynchronous Receiver Transmitter
VM	Vending Machine
ZXing	Zebra Crossing

Variables

I	Current	[A]

P	Power	[W]
V	Voltage	[V]
R	Resistance	[Ω]
ω	Angular Velocity	[rad/sec]

Variable Subscripts

b	Base
p	Raspberry Pi
r	Relay
o	Supply Voltage
e	Back-EMF

Constants

β	Transistor Current Amplification
k_e	Back-EMF Constant

Chapter 1

Introduction

1.1 Problem Statement

The Vending Machines (VMs) currently used at Stellenbosch University (SU) exclusively make use of cash transactions. These VM systems are currently the de facto standard throughout the world, but they do have one drawback: they require a hard cash transaction to take place. In a world moving away from cash transactions and towards online payments, e-transactions and mobile payments, this may become a problem to potential customers. With that in mind, a need has been identified at SU for a VM that accepts cashless transactions.

1.2 Existing Solutions

Currently there are cashless payment solutions being used by the general public. These include debit and credit cards, Radio Field Identification (RFID) cards, Unstructured Supplementary Service Data (USSD) based systems and, more recently, Near Field Communication (NFC) payments. These alternatives are further discussed in this section.

1.2.1 Credit and Debit Cards

A familiar and widely-used alternative to cash payments are the debit and credit cards most modern adults possess. This is especially true in developed countries with mature and reliable financial institutions.

The advantages these cards hold over the other cashless options are that they are easy to obtain and that they have become very reliable and simple to use. A disadvantage is that debit and credit systems may become costly and complicated to implement. This is very true for the simple system developed in this project.

1.2.2 Radio Field Identification

RFID is a technology which was first patented in 1983 [Walton (1983)]. Since then, the technology has grown and matured into a very reliable identification and payment platform. Examples of where this is used are the payments made to new parking meters with a contactless card.

The advantage of this technology is its great convenience: a customer only needs to tap the card against a receiver and it is not required that a password be entered.

However, this leads to some security concerns. For example, if the card gets stolen or cloned, the thief can use the money on the card for his own benefit. Fortunately, these cards most commonly work with pre-paid money. Therefore, provided that there was not too much money loaded onto the card, the theft victim will not suffer a financial loss greater than if cash were stolen.

1.2.3 Unstructured Supplementary Service Data

USSD is a communication standard used by cellphones to exchange data with a service provider's servers. If the service provider allows it, USSD may be used by a customer to make financial transfers.

An example of this is the M-Pesa mobile money service in Kenya, which is based on USSD. It allows a customer to pay for goods ranging from milk to bread and even the monthly rent. It is currently regarded as the most advanced and popular mobile payment platform in the world [Jack, William and Suri, Tavneet (2011)].

An advantage of implementing such a system is that it has been proven to be reliable and is usable by almost any cellphone. The disadvantage is that it requires third party vendors, such as the mobile service providers, to provide systems and services. This may add unnecessary overhead costs to the system.

1.2.4 Near Field Communication

NFC payments have recently come to the fore as a prominent method of making cashless transactions. In Europe and North America there have been significant advances in making this payment method a more attractive payment option. Google has been making large contributions with the addition of NFC protocols to its Android platform [Google Android Team (2009)].

Some examples of NFC based payments are the London public transport system, which makes provision for NFC payments [Weinstein, Lauren Sager (2009)], as well as some retail vendors which accept payments made via Google's Wallet application.

1.3 Goal of the Final System

Hard cash still remains the largest contributor to global financial transactions, standing at 59% of the 37 billion transactions that took place in 2012 [Humphrey, David B (2004), Pasquali, Valentina and Bedell, Denise (2013)]. However, mobile and card transactions are expected to surpass cash as the leading payment method by 2015 [Wollop, Harry (2010)].

To this end, mobile payments, i.e. payments made with a cellphone, was chosen as the medium to facilitate cashless payments for this VM. Therefore, the final goal of this project is to deliver a VM that can be used on SU's campus and will allow anyone to buy products from the VM using only their cellphones.

1.4 System Objectives

The system objectives are:

- The system must make provision for both NFC and Quick Response Code-based payments.
- The system must make use of a web server based in the cloud that must be accessible by anyone across the world.
- A cellphone application must be created that will allow transactions to be completed using NFC.
- A demonstration model VM must be designed and constructed.
- All the data transfers between the user's cellphone and the server must be encrypted.
- Extra layers of security must be added to prevent theft and product loss.

1.5 Report Structure

In this report, Chapter 2 gives background information on all the technology, concepts and programs used in this project. Thereafter, Chapter 3 discusses the overall system design, which is followed by a discussion on the detailed design of the software and hardware aspects of the entire system in Chapters 4 and 5. Afterwards in Chapter 6, the system test results are discussed and analysed, which is followed by a discussion on the complete system. Finally, the project conclusion is given in Chapter 7.

Chapter 2

Background Study

This chapter contains background information on the software, services and algorithms used in this project. They are divided up into Quick Response Codes (QR Codes), Near Field Communication (NFC), the web server and the encryption and encoding algorithms used.

2.1 Quick Response Codes

QR Codes are two dimensional bar codes that were initially used in Japanese car factories to allow computers to track the progress of an item on a production line [Soon, Tan Jin (2008)]. The technology has since evolved and matured and is today widely used in the media industry for storing data, such as a web address or a phone number. See Fig. 2.1 for an example of a QR code.

QR Codes can store up to 7089 alphanumeric characters [Soon, Tan Jin (2008)], which are accessible by scanning the code with either a laser or a digital camera. Scanning a QR Code requires a camera that can produce a digital image at a resolution that is at least twice that of the QR Code. This image is then processed by a QR Code library, e.g. the ZXing library (see Sec. 2.1.1 for more detail on the ZXing library), which decodes the picture and extracts the data embedded inside the code. Cellphones are commonly used today because of their portability,

Figure 2.1: Example of a QR Code.

increasingly powerful hardware and QR Code technology's simplicity. However, an image with an embedded QR Code can be decoded by any computer with the necessary hardware and libraries installed, e.g. a webcam and the ZXing library.

2.1.1 Zebra Crossing Library

The Zebra Crossing Library (ZXing) is a QR Code coding and decoding library [ZXing Team (2013)]. It is commonly built into smartphone applications to decode QR Codes embedded inside a static image or a video stream. A desktop version of the library, called ZBar, is also available and works in a similar manner.

The Barcode Scanner application is made by the team that made the ZXing library. It is freely available on multiple cellphone platforms, such as BlackBerry OS, Apples's iOS and Google's Android. There have been at least 50 million downloads of the Barcode Scanner application on the Android platform alone, and it currently lies 98^{th} in the top 100 of Google Play's most downloaded list [Google Play (2013)]. This shows the extent to which QR Code technology and the ZXing library has evolved to be used by millions of people.

2.2 Near Field Communication

NFC is a relatively new communication standard in the world of wireless technology. It allows two NFC-enabled devices to wirelessly transmit data by bringing them close to one another, typically around 4 centimeters.

Most cellphone manufacturers, with the major exception of Apple, have added NFC hardware to their flagship models, and more recently to some of their budget models [NFC World (2013)]. The technology has also been ported to other platforms, such as the desktop computer and Arduino. This adds a new dimension to wireless inter-device communication and makes projects such as this more feasible.

2.2.1 libnfc

Libnfc is an open-source library for Linux systems [Libnfc Team (2013b)]. It allows a desktop computer to communicate with an NFC device based on the Phillips PN53 series of NFC chips [Libnfc Team (2013a)]. Recent versions have made provision for the use of a PN532 breakout board that can be used by a Raspberry Pi. It is currently in version 1.7 and is classified as a 'mature' library by the open-source community.

nfcpy

Nfcpy is an open-source Python interface for the libnfc library, developed and maintained by Stephen Tiedemann [Tiedemann, Stephen (2013)], that allows for peer-to-peer communication between a cellphone and desktop-based NFC controller. This is done by using the NFC Data Exchange Format (NDEF), the Simple NDEF Exchange Protocol (SNEP) and the Logical Link Control Protocol (LLCP). These standards and protocols have been set by the NFC standard governing body, the NFC Forum [NFC Forum (2013)], to simplify and standardise data exchange between different platforms, making the user experience more pleasant.

2.2.2 Android

Google's Android operating system (OS) is currently the most popular cellphone OS world wide, with an estimated 80% market share [Etherington, Darrel (2013)].

Since version 2.3 'Gingerbread', Android has had NFC capabilities [Google Android Team (2009)]. Google has since then been promoting the use of NFC as an alternative payment option. Other platforms, such as Blackberry's OS and Windows Phone, have also added NFC to their latest phones, but they do not have the market penetration that Android currently has. It was also found that application development on the Android platform is relatively simple and free. It was therefore decided that the NFC payment application will be based on the Android platform.

2.2.3 RFID and Stellenbosch University Cards

NFC and Radio Frequency Identification (RFID) technology work in a similar manner: when two devices (e.g. cellphone, RFID card, etc.), equipped with an antenna tuned to a mutual frequency, for example 13.56 MHz, come into close proximity, they transmit some form of data to one another.

However, there are some important difference between the two technologies. For example, an NFC system is an active system, meaning that the device's antenna is always powered. NFC devices also have peer-to-peer capabilities, meaning that two devices can communicate with one another by both sending and receiving data. RFID systems, on the other hand, only allow for one-way communication [Chandler, Nathan (2012)] (e.g. the current SU's student entry control system).

2.3 Web Server

The web server is responsible for handling all the data transfers and transactions that take place when a customer buys a product. In this section, some background

information will be given on the key features of the web server that was implemented for this project.

2.3.1 Django Web Framework

Django is a Python web server framework which focuses on easy setup and simple design. Some of its features are that it fully handles Hypertext Transfer Protocol (HTTP) requests, integrates with SQL databases, supports Hypertext Markup Language (HTML) template design, makes provision for the execution of Python scripts, and has an offline server debugging function available. Django is also expandable to commercial size servers that are accessible across the globe. For example, large websites, such as Instagram and Pinterest, are based on the Django framework [Django Software Foundation (2013*a*)].

To make it easier to program, read and debug, the original Django developers designed Django to split its websites into multiple, so-called 'applications'. These applications typically contain a single web-page with its own script and database handling functionality. These applications can communicate with one another, meaning that a script from application X may execute a script in application Y.

Django was initially developed by web programmers Adrian Holovaty and Simon Willison, from the newspaper Lawrence Journal World [Django Software Foundation (2013*b*)]. It was first released in 2005 under the Berkeley Software Distribution (BSD) license and is completely free to use.

2.3.2 Elastic Compute Cloud

Elastic Cloud Compute (EC2) is a cloud computing service offered by Amazon Web Services (AWS) [Amazon Web Services (2013)]. It allows a user to rent a cloud-based virtual machine from AWS on which to run their own applications. These applications can be web-based, which means that these virtual machines are accessible across the world.

2.3.3 Apache

Apache is a popular web server application (an estimated 53.4% of the world's servers are running on it [Netcraft (2013)]) that is available on most OS platforms [Apache Software Foundation (2013*b*)]. A notable feature of Apache is that it was designed to be configurable. This makes it easy to run various web frameworks, such as Python's Django, PHP's cgiapp and C++'s Poco, off of it.

It was initially released by Robert McCool in 1995 under the Apache Licence. It is currently being maintained by the Apache Software Foundation [Apache Software Foundation (2013*a*)].

2.4 Encryption

Encryption is the act of encoding some data into a form that is intended to be unintelligible to anyone other than the intended recipient. This is done to ensure that only authorised parties can access sensitive data. This is often done with an encryption key and an algorithm which specifies how the data was encrypted and how it can be decrypted. Encryption is often used where sensitive information is being transmitted between two parties, e.g. personal e-mails, bank passwords, etc.

Two encryption schemes were considered for this project. They are symmetric and asymmetric encryption and they are discussed in this section.

2.4.1 Symmetric Encryption

In symmetric encryption, both parties, i.e. the sender and receiver, have to agree to a common encryption key prior to data transmission [Tyson, Jeff (2001)]. A famous example of symmetric encryption is the Enigma cipher machine used by Nazi Germany during the Second World War [Stripp, Alan (1999)].

Unfortunately, due to the increase in knowledge and understanding around this type of encryption and the increase in modern computing power, various code cracking methods, such as known and chosen plain-text attacks [Biham, Eli (1994)] have been developed since 1945 that can break the most commonly used symmetric encryption methods. However, methods such as One-Time Pad (OTP) encryption are still being used today. OTP encryption is still considered to be unbreakable, if done properly [Rijmenants, Dirk (2011)]. However, it is difficult to securely exchange the keys between the communicating parties [Rijmenants, Dirk (2011)].

2.4.2 Asymmetric Encryption

Another encryption scheme is asymmetric encryption, or public-key encryption. It involves the use of a public and private key pair that can be used to securely encrypt and sign a data package on the sender's side and to decrypt and verify the data and its origin on the receiver's side [Public-Key Encryption (2005)].

These private and public keys are mathematically related to one another according to the algorithm in use (e.g. ElGamal or RSA. See sections 2.4.2 and 2.4.2 for more information). The public key is used to encrypt data and may be publicly distributed. However, in practise the keys are only distributed to trusted parties to increase security. The private key allows one to decrypt data encrypted with the public key.

An advantage of asymmetric encryption is that even if the public key is leaked, it is still very difficult, and sometimes impossible, to derive the private key. See Appendix E for a detailed explanation on asymmetric encryption.

ElGamal

The ElGamal algorithm was developed by Taher ElGamal in 1984 [ElGamal, Taher (1985)]. It is an alternative to the popular Ron Rivest, Adi Shamir and Leonard Adleman (RSA) encryption algorithm. Its security stems from the 'difficulty of computing discrete logarithms in a large prime modulus' [Leon, Jeffrey S. (2008)].

An advantage that ElGamal has over RSA is that due to the mathematics behind the algorithm, it is almost certain that a different ciphertext will be generated each time a string is encrypted. However, a large drawback of this algorithm is that the key needs to be at least twice as long as the plain-text string that is being encrypted [ElGamal, Taher (1985)].

RSA

The RSA encryption algorithm is a widely-used encryption standard. Its security is based on 'the difficulty of factoring large integers' [Leon, Jeffrey S. (2008)].

An advantage of the RSA algorithm is its encryption and decryption speed. Also, the encrypted data's length is determined by the size of the encryption key, provided the key is long enough. Therefore, the encryptor has some measure of control over how long the produced ciphertext is going to be.

RSA was developed by Ron Rivest, Adi Shamir and Leonard Adleman in 1978 and has been widely used since 1993.

2.4.3 PyCrypto

PyCrypto is a Python cryptography toolkit which contains various encryption algorithms and key schemes, such as ElGamal, MD5 and RSA. It is currently registered under the Python License and is available in the public domain. It is maintained by the PyCrypto Team [PyCrypto Team (2013)].

2.4.4 Base64 Encoding

Base64 encoding is a scheme which represent arbitrary data as alphanumeric characters. This encoding scheme is often applied where the output of encrypted text is a collection of random characters. ASCII characters are normally preferred because they are easier to read by humans and simpler to transmit via HTTP.

Here is an example of a base64 encoded string:

Original text:
Hi, I'm a base64 encoded string!
Base64 encoded output:
SGksIEknbSBhIGJhc2UgNjQgZW5jb2RlZCBzdHJpbmch

Chapter 3

System Design

In this chapter, the overall system design is discussed. The chapter is divided up into nine sections with each section focusing on a different aspect of the complete system. These nine sections are:

1. The VM central controller.
2. The Near Field Communication controller.
3. The Quick Response Code camera.
4. The product dispensing mechanism.
5. The VM unit.
6. The web server.
7. The encryption scheme used.
8. The security schemes used.
9. The transaction process for each payment option.

Each section explains which technology or service was used in the final component design. Where two or more available services or technologies were available, a brief discussion and explanation is given as to why the particular technology or service was used in the final design of the VM.

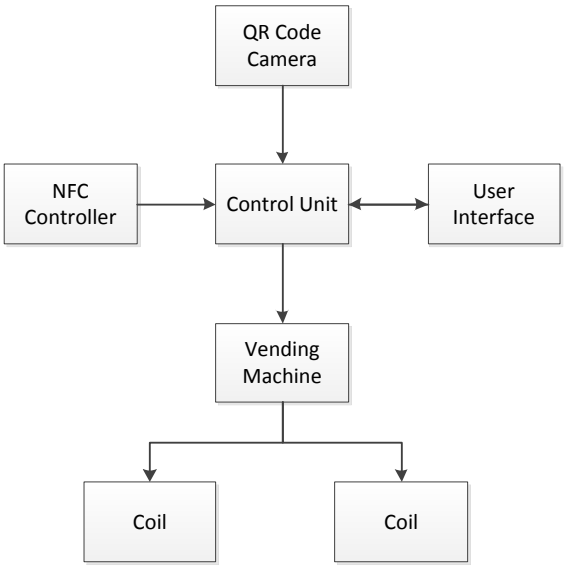

Figure 3.1: System overview from the control unit's perspective.

3.1 System Overview

Fig. 3.1 gives a diagrammatical layout of the complete system. It shows the data interactions between the different sub-components of the complete system.

The components used in these subsystems are discussed in the subsequent sections of this chapter.

3.2 Central Control Unit

To be able to handle the data processing that Quick Response (QR) Code decoding and Near Field Communication (NFC) requires, a fairly powerful central controller is required. The two controllers that were considered for this project is the Raspberry Pi microcomputer and the Arduino Uno microcontroller. These controllers are discussed in this section. Their specifications are given in Table 3.1.

3.2.1 Arduino Uno

The Arduino Uno is popular open-source microcontroller. It is based on an 8-bit Atmel ATmega328 ARM microprocessor.

Because of its open-source design, there are a multitude of peripheral devices and expansion boards (known as 'shields'), along with all their libraries and drivers, available locally. The Arduino's programming language of choice is a modified

Table 3.1: Comparison between Arduino Uno and the Raspberry Pi Specs [Arduino (2013), eLinux (2013)].

Specification Detail	**Arduino Uno**	**Raspberry Pi**
Operating Voltage	5 V	3.3 V
Processor	Atmel ARM @ 16 MHz	ARM 6 @ 700 MHz
GPIO Pins	14	28
Memory	32 kB Flash	512 MB RAM
Communication	i^2c, UART, SPI	SPI, UART, USB, i^2c, Ethernet
Price	R310.00	R400.00
Video Output	N/A	HDMI

version of C and comes with its own Integrated Development Environment. This, along with its relatively low cost and specifications, made the Arduino Uno an attractive option for this project.

3.2.2 Raspberry Pi

The Raspberry Pi is a Linux-based microcomputer, designed and manufactured by the Raspberry Pi Foundation in the UK for the purpose of educating and familiarising young children with programming. However, its low price and respectable specifications makes it a strong choice as a control unit for the VM.

The Pi was designed with the focus on Python as its main programming language, which makes running scripts and controlling the board relatively simple. It also runs on a modified version of Debian Linux, called Raspbian.

3.2.3 Design Choice

The Raspberry Pi was chosen as the central controller of this project and controls the hardware connected to it via a Universal Serial Bus (USB) connection, or one of its General Purpose Input Output (GPIO) pins.

The Pi was chosen ahead of the Arduino, because of its video stream processing capabilities and that libnfc can be installed on it. This allows the Pi to decode QR Codes and to communicate with an NFC-enable cellphone through an NFC add-on chip. Furthermore, the Pi can interface with desktop peripheral hardware, such as a mouse and keyboard, which made development and prototyping much simpler.

After development and optimisation has been completed for this project, work can begin on porting it to a micro-controller, such as an Arduino. However, for the development phase of this project, the Pi is a much better suited platform.

3.3 NFC Controller

The NFC controller that was used is the PN532 NFC shield from Adafruit Industries [Adafruit Industries (2012)]. It is based on the Phillips PN532 chip.

The main reason it was selected instead of other NFC controllers was that it has a large support base in the open-source community and is fully compatible with the libnfc open-source NFC library [Libnfc Team (2013*a*)]. The manufacturer also provides comprehensive documentation and guides on how to set up and configure the controller [Townsend, Kevin (2012)] for a Raspberry Pi.

The main purpose of this component is to add the option of sending or receiving data through an NFC connection. This component is also capable of reading Radio Frequency Identification (RFID) cards, such as student or staff cards, since NFC and RFID transmit similar types of data. This adds the option of paying for the products with any NFC-capable smart phone running Google's Android operating system, or with a Stellenbosch University (SU) staff or student card.

3.4 QR Code Camera

To decode QR Codes, the VM needs to take pictures of a code so that it can be decoded by a QR Code library, such as the ZBar library. A PlayStation 2 EyeToy was chosen and added to the system to facilitate this. It was chosen because its drivers are freely available for Linux systems [ov511 (2013)], it interfaces easily with the USB ports on the Pi and one was readily available for use.

There is currently a camera add-on available for the Pi, but this is relatively expensive (approximately $30 [Adafruit (2013)] versus the Pi's cost of $35 and the PS2 EyeToy's cost of $10) and unavailable in locally at the time of writing.

3.5 Product Dispensing

3.5.1 Coils

To be able to effectively dispense bought products to the user, a coil mechanism is used. These mechanisms are familiar and simple methods of dispensing goods.

These coils are designed and made in such a manner that one rotation of the coil will drop one product. The turning motion is made by attaching a DC motor to the base of the coil.

3.5.2 DC Motors

The motors attached to the base of the coils are two 12 V DC motors from Faulhaber [Faulhaber (2013*a*)]. Although these motors are rated for 12 V, it is possible to run them from a lower voltage. This will cause the motor to turn slower, and therefore be easier to control. The motors are switched on by a 12 V relay switch controlled by the Raspberry Pi.

3.5.3 Relay Switch

A relay is an electronic switch, which requires a voltage across it to open or close it. With this, it is possible to control when the DC motors turn (after a successful transaction) and when they are standing still.

The relays used here are 12 V, because they were readily available, but the Raspberry Pi can only deliver a maximum voltage of 5 V. Therefore, it was required that the relay will be permanently connected to a 12 V DC supply. The relays are then switched by a 2N2222 transistor, which is controlled directly from the Pi's GPIO pins (see Sec. 5.1 on p.32 for a detailed discussion on the switch). This allows the Pi to directly control the motors. Due to the circuit's construction, the Pi is protected from the relatively high voltages and currents involved in the working of the motor and relay.

3.6 Vending Machine Unit

The VM unit houses all the components (i.e. the Raspberry Pi, the NFC Shield, webcam, switches, motors and the product coils). See Appendix A for detailed manufacturing drawings and a picture of the VM.

3.7 Web Server

A web server in the computing cloud is used to process and authenticate the transactions that take place in the VM. This is done by using a user database that is populated with the customers' login details that they supply when they sign up for the VM service. The database stores the customers' information, such as the customers' username, password and their remaining balances. Part of the transaction process is to check if a customer has enough credits loaded onto his account.

Arguably, the transaction authentication could have been done by the VM itself. However, using a web server allows the system to be expanded beyond a single VM and standardises the transaction process for all of the VMs connected to the server.

The server communicates with a customer through the customer's cellphone, which sends data to the server over the internet via Hypertext Transfer Protocol (HTTP) requests and Universal Resource Locators (URLs). How the web server is used to process and authenticate each payment method is described in Sec. 3.10.

3.8 Encryption Scheme Design

To prevent the system from being hacked, an asymmetric encryption scheme was implemented. The main reasons for this choice is that the keys are easily distributed, the encryption scheme is relatively secure and there is powerful software available to encrypt the data exchanges.

The keys were generated using the PyCrypto module and were hard-coded into the VM and the web server during development. If required, the old keys can be replaced remotely over the internet or by a technician working on the VM. The Android application has its own key, which is key is distributed with the application.

The keys are hardcoded to keep the keys more secure and to avoid having to dynamically update and redistribute the keys after each transaction.

Two different schemes were implemented for the Android NFC application and the QR Code payment option. These two schemes are discussed in this section.

3.8.1 Android NFC Application

For the Android NFC application, a 1024-bit key based on the RSA encryption algorithm was used. A 1024-bit key was chosen because it gives a good balance between security and encryption speed. It was decided to base the application's encryption on RSA, because it is already included in the Android Development Kit and is therefore the simplest to implement and distribute with the application.

3.8.2 QR Code

For the QR Code payment option, a 384-bit key based on the ElGamal algorithm is used. The reason for choosing this key size was to produce smaller, less complex output which leads to a more readable QR Code. A simpler, more readable QR Code takes less time to scan. A smaller key size also reduces the time it takes to encrypt a data string.

It was also found that the RSA module in PyCrypto has a minimum key-length of 1024 bits, which is too long. The ElGamal module allows for key sizes of any bit length, as long as it is longer than the string being encrypted. Therefore, the ElGamal algorithm with a key size of 384-bits was selected.

3.9 Security Scheme

In addition to the asymmetric encryption used on the data transfer between the server and its clients, two more layers of security were added. The effectiveness of these additional layers of security still rely upon the encryption scheme staying intact. However, these extra security layers help to obscure the plain text message being encrypted, thereby making it harder for would-be hackers to realise they have cracked the encryption. It also prevents the same code from being used more than once without being paid for. These security schemes are discussed in this section.

3.9.1 Random Character String

The code transmitted to the server is a random 16-character hexadecimal string. The product code is 4 hex characters long and is embedded inside this 16-character string. The product code is saved on the database and can therefore not be random.

A hexadecimal string was chosen because it is a common data representation format and it will make it harder for a hacker to see that they have successfully broken through the security. For example, if the hackers manage to break through the security, they will see a string like this one: A2FB32E1CFF7. If the product code were alphanumeric, for example 'coke_350ml', the hackers will immediately see that they have broken through. This scheme does not make the code uncrackable, but it does make brute-force attacks harder to execute properly.

3.9.2 Challenge and Response Code

To prevent customers from repeatedly using the same QR Code to buy a product, a second layer of security was added. This layer is a challenge-response scheme.

Such a scheme works by having party A generate a challenge (this can be a string or a number) and send it to party B. Party B then takes this challenge and puts it through a previously agreed-upon process, e.g. puts the string though a one-time hash or adds the numbers together. This is the response. Party B then sends the back the response to party A, which then checks to see if it is a valid response to party A's original challenge.

In the case of the VM, a 16 character hex string is being generated. It was decided to use 4 characters of the 16-character string, excluding the 4-character product code, as the VM's challenge. After receiving this challenge, the server then takes out the agreed-upon 4 characters and embeds it inside the server's response code. When the VM scans the customer's response QR Code, the VM checks to see if the 4 character response was part of its original code the VM generated. This ensures that each code only dispenses one product, because the 4 characters used are randomly generated and are therefore only valid for one transaction.

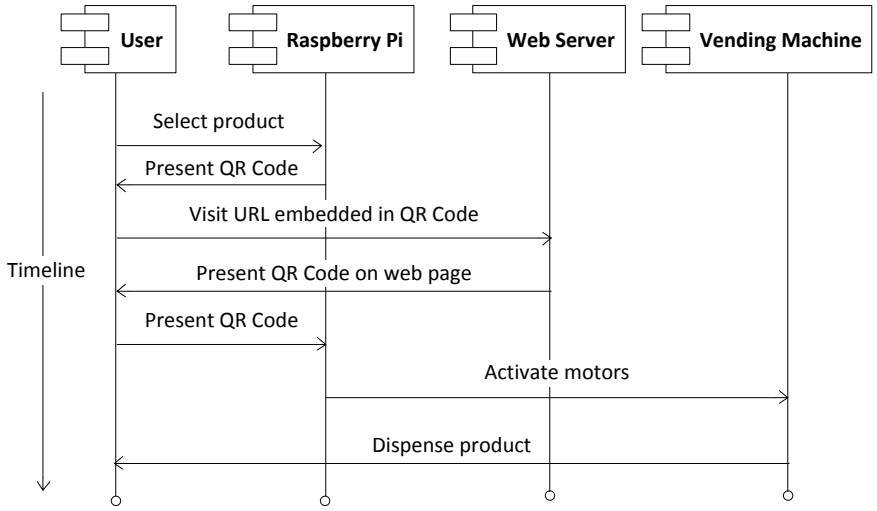

Figure 3.2: The VM QR Code transaction process.

3.10 Transaction Design

Three different transaction processes were designed to accommodate for QR Code NFC and RFID payments. However, all three of the payment options have some common elements to them.

Firstly, each product in the VM is assigned a unique, 4-letter hexadecimal number. This is done to comply with the security code scheme discussed in Sec. 3.9. When a customer selects a product to buy, this unique code is used identify which product to dispense. To simplify the demonstration system, only two codes are used. The second common element between the payment options is that all the transactions require authentication from the central server. At this stage, this excludes RFID card payments. The reason for this is explained in Sec. 3.10.3.

3.10.1 QR Code Transactions

The QR Code transaction process and the interactions between the different system components is described diagrammatically in Fig. 3.2.

Fig. 3.2 shows that the transaction begins with the customer selecting a product to buy from the VM Graphical User Interface (GUI). The VM then generates a QR Code which the customer scans. The QR Code contains a URL that contains a link to the web server, the encrypted product data, the VM's signature and the challenge-response code (the challenge-response scheme is described in Sec. 3.9.2 on p.16). This redirects the customer to the web server and allows the server to process the transaction.

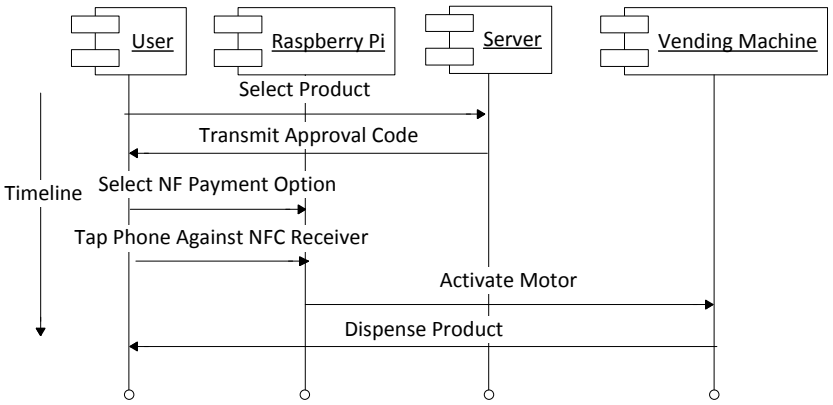

Figure 3.3: The vending machine NFC transaction process.

After the transaction has been processed and approved, the web server displays a QR Code, embedded with an encrypted approval message, the server's signature and the response to the VM's challenge, on the screen of the customer's cellphone. The customer then displays this QR Code to the VM's camera. After the code has been scanned and processed, the VM activates the correct motor which dispenses the customer's desired product.

3.10.2 NFC Transactions

The NFC transaction process is given in Fig. 3.3.

Fig. 3.3 shows that the transaction process starts with the customer selecting a product from the Android NFC application. The application then contacts the server using a URL containing the encrypted transaction data, as well as the customer's username and password, both encrypted.

Using this data, the server processes the transaction. If the transaction is authenticated, the server returns an encrypted approval code to the application. The customer then taps his phone on the VM's NFC receiver and the application transmits the authentication code to the VM via the NFC protocol. The VM then activates the correct motor and dispenses the customer's desired product.

3.10.3 SU Card Transactions

The SU card payment option was added to the VM as a proof of concept. At present, it can identify an RFID card that has been hard-coded into the VM. Therefore, no communication with the server takes place during the SU Card transaction process and there is no user authentication besides the hard-coded RFID card numbers and there is no user account to credit with the product cost.

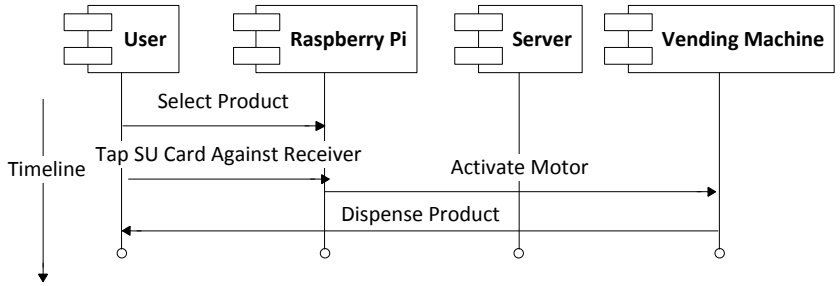

Figure 3.4: The vending machine SU Card transaction process.

The reason for not adding the SU card numbers to the server's database is that the SU access control office already keeps a comprehensive, up-to-date database. It would therefore be more feasible to integrate their database into the system and credit the customer's student or staff account during the transaction process.

Fig. 3.4 shows the transaction process for SU cards as it is used in system's the current format.

Fig. 3.4 shows that the transaction process begin when the customer selects a product from the GUI. The customer then taps his SU card against the VM's NFC receiver. The VM then reads the card's Unique Identification Number (UID) and checks if it is coded into the VM. If the VM finds the card number, it activates the correct motor and the customer receives his product.

Chapter 4

Software Detail Design

In this chapter, the software design aspects of this project are discussed in detail. This chapter is divided up into three sections, with each section explaining what the program being discussed is responsible for, what third party programs it uses and how it interacts with the rest of the system.

These three sections are the web server, the VM's control program and the Android application that was created.

4.1 Web Server Program Design

The web server that was made for this project is based on the Django web framework for Python. The Django server was configured to run on top of an Apache web server located on an Amazon Web Service (AWS) Elastic Compute Cloud (EC2) cloud computer instance.

This section stipulates the design of the complete server and how the different components interact with one another. See Appendix F for details on setting up and configuring Django on Apache and EC2. The source code is available from github.com/jayceelock/server_site and in the project file.

4.1.1 Django Server

The Django server is responsible for all of the scripting and database work that the server performs. The server is divided up into a total of five applications. Each of these is responsible for either displaying a single web page or to handle data requests from the Near Field Communication (NFC) Android application (see Sec. 4.3 on p.28 for more details on the Android application). The website server structure is given in Fig. 4.1.

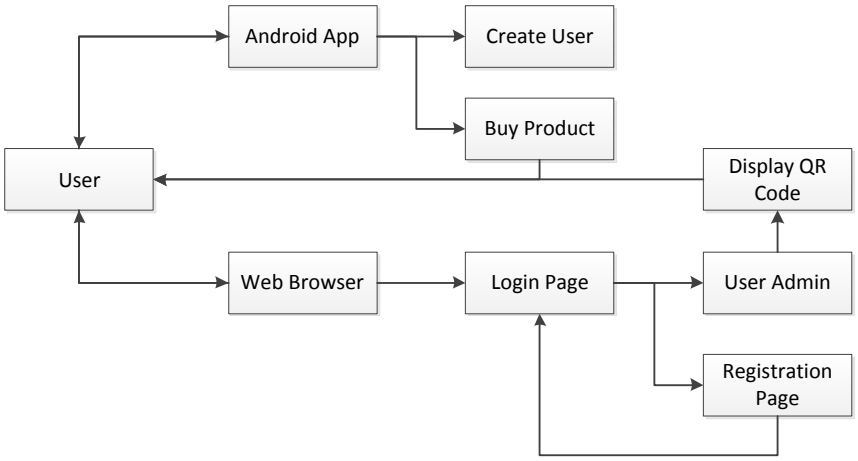

Figure 4.1: The web server application structure.

display_qrcode

This application forms the core of the Quick Response Code (QR Code) payment handling part of the server. See Fig. 4.2 for the process flow of this application.

Fig. 4.2 shows that the application first extracts the code containing the product code and VM's signature from the Uniform Resource Locator (URL) that the customer requested with his cellphone's web browser. The application then proceeds to decode the product code from its base64 encoded format.

After successfully decoding the data, the application proceeds to decrypt the data and signature with the ElGamal algorithm using the server's private key and the VM's public key. Afterwards, following the security code scheme described in Sec. 3.9, it extracts the product code and challenge from the decrypted string.

The application then checks to see whether the signature comes from a valid source (i.e. one of the VMs using this system), whether the product code is valid (i.e. the product is in the database) and whether the customer has enough credits available. If either of these checks return false, an error message is shown to the customer explaining what went wrong and what the customer should do next.

If the checks were passed, the application proceeds to subtract the product cost from the user's remaining balance. Following the security code scheme, the application then generates the correct return code, encrypts and signs it with the vending machine's public key and the server's private key, and encodes it in base64. After this is completed, the application embeds this data into a QR Code, which is displayed on the customer's cellphone screen.

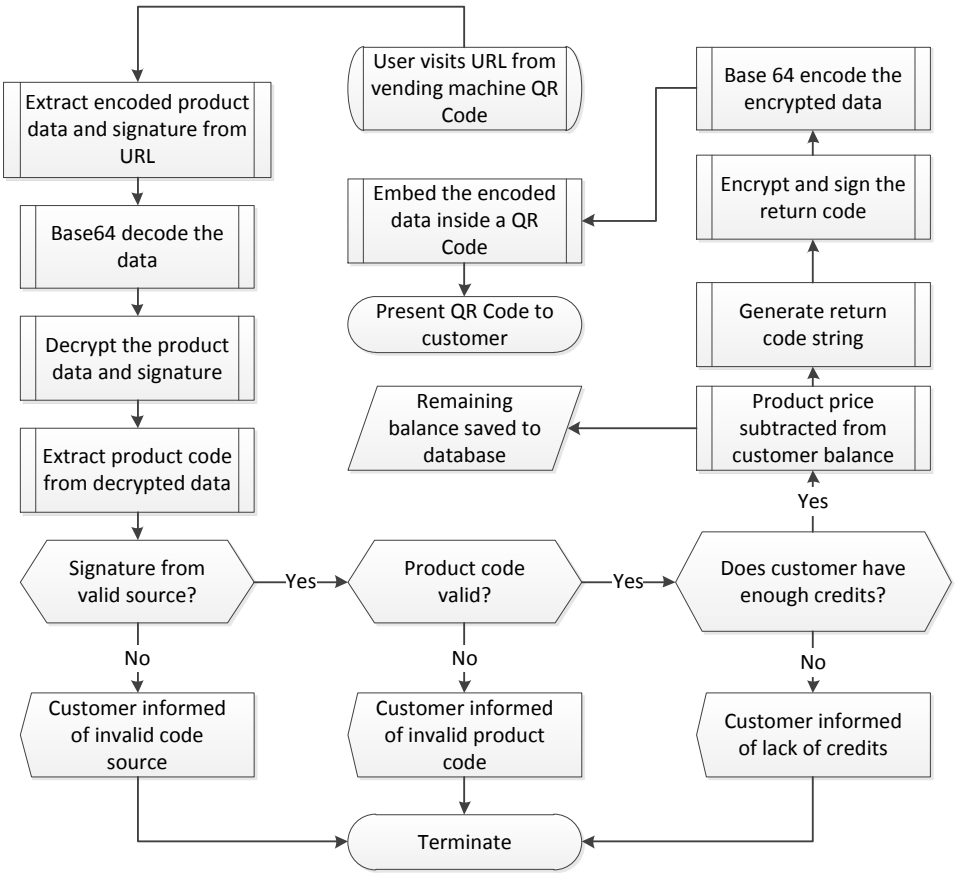

Figure 4.2: The display_qrcode application process flow.

load_money

This application allows the customer to load money onto his account. At the moment it makes use of faux money, meaning that the money loaded has no real-world value. The customer can load a maximum of R1000.00 at a time onto his account. See Fig. 4.3 for the process flow of this application.

nfc

This application forms the core of the NFC payment handling part of the server. See Fig. 4.4 for a detailed process flow diagram.

Fig. 4.4 shows that the server first extracts the encrypted customer login details and product code from the NFC application's URL request. These codes are then decoded and decrypted using the RSA algorithm and the server's private key.

The user login details are then checked and verified using the server's user

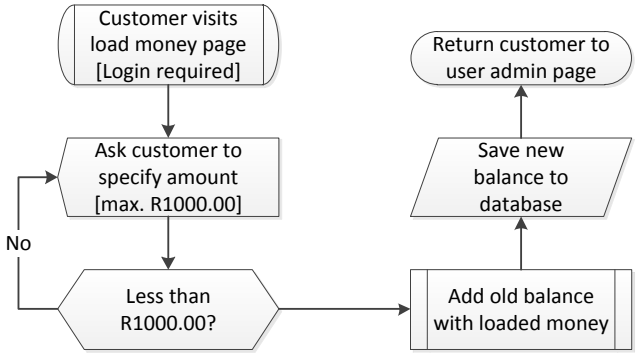

Figure 4.3: The load_money application process flow.

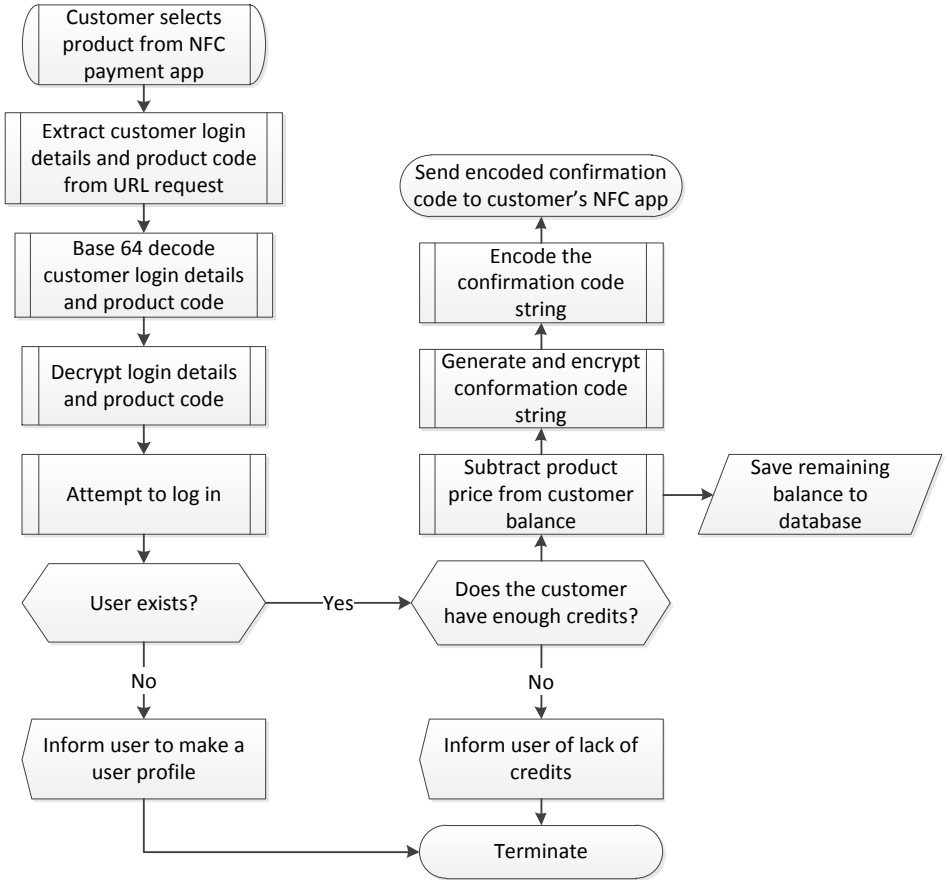

Figure 4.4: The NFC application process flow.

database. If this check fails, the customer is given an error message and is asked to create a user profile. If the check is passed, the server then checks to see if the customer has enough money loaded onto his account. If this check fails, the customer is informed of his lack of funds and is instructed to load more money. If the check is passed, the server subtracts the product cost from the customer's balance and encrypts and encodes a confirmation code, according to the security code scheme specified in Sec. 3.9. This code is then sent to the NFC application.

nfc_add_user

This application allows a customer using the NFC application to create a user profile for himself. The server extracts the customer's login details from the URL request that the NFC application sends to the server. These details are then saved to the database and is immediately available to be used by the new customer.

register

This application allows a new customer to register a new user profile with an internet browser. This application is only accessible by a web browser and not by the NFC application. However, a user registered with this application will be able to use the same login details for the NFC application, and vice versa.

The application presents the user with a simple registration page which asks for a user name, email and password. Using Django's built-in form support, the server handles the POST request that is generated when the customer presses the 'Continue' button. When this is done, the customer's login details contained within the POST request is extracted and saved into the user database.

4.2 Vending Machine Program

The vending machine's program runs the VM. It is responsible for allowing the customer to select a product, to create a QR Code that redirects the customer to the web server, to scan the customer's response QR Code, to scan the customer's NFC/RFID request and to dispense the product after a successful transaction.

The whole program is based on Python and designed to be used by a Raspberry Pi microcomputer. To simplify the program, it is split into separate subprograms. These subprograms, called scripts, are discussed in this section. The complete program structure and its inter-script interfaces can be seen in Fig. 4.5. The source code is available on github.com/jayceelock/vending_prog and in the project file.

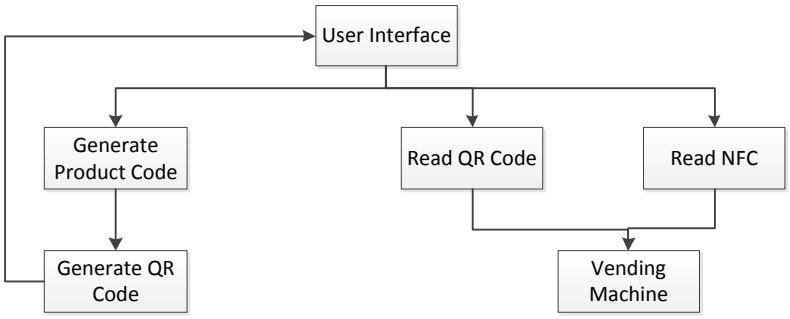

Figure 4.5: The vending machine's program structure.

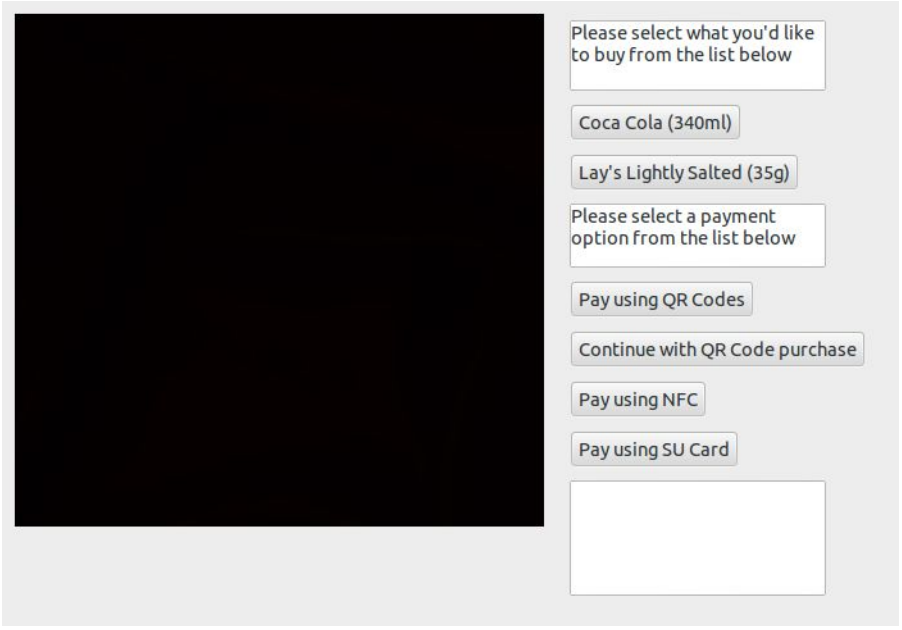

Figure 4.6: A screenshot of the user interface.

4.2.1 User Interface

To allow the customer to select a product, a Graphical User Interface (GUI) was created. The GUI was made using the WX Python GUI toolkit [WX Python Team (2013)]. See Fig. 4.6 for a screenshot of the GUI.

As can be seen from Fig. 4.5, the GUI script is responsible for calling the encryption script and the QR Code generation script. It is also responsible for displaying the QR Code, to handle transactions from the Android NFC application and to activate the correct motor inside the vending machine.

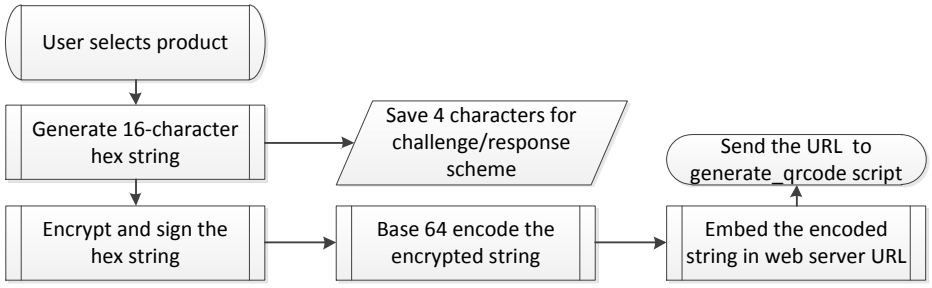

Figure 4.7: The GUI process flow.

4.2.2 Generating a Product Code

After the customer selects which product to buy from the GUI, the encrypt_elgamal script is called. This script is responsible for generating the random hex character string, in accordance with the security scheme described in Sec. 3.9, encrypting, signing and encoding the string in base64 and embedding the random string inside a URL that points to the web server. This URL is then sent to the generate_qrcode script described in Sec. 4.2.3. See Fig. 4.7 for a detailed process flow diagram.

4.2.3 Generating a QR Code

After the encrypt_elgamal script has been run, the generate_qrcode script is called. This script is responsible for embedding the URL received from the encrypt_elgamal script into a QR Code. This is done by using a qrcode module for Python, called qrcode [Loop, Lincoln (2013)].

4.2.4 Reading a QR Code

After the customer has received his verification QR Code from the server, the customer may presses 'Continue with purchase' button. When this is done, the read_qrcode script is run. This script is responsible for reading the customer's QR Code via a webcam, extracting the data from the scanned image, decrypting the data and verifying the transaction. See Fig. 4.8 for the process flow diagram.

As seen from Fig. 4.8, as soon as the 'Continue with purchase' button is pressed, a ZBar image processor is created which scans a webcam video feed for a QR Code. The image processor runs until the user cancels it or it scans a QR Code.

After the ZBar processor has scanned a QR Code, it sends the retrieved data to be decrypted and verified with the ElGamal algorithm, using the VM's private key and the server's public key. If the signature is valid and the data contains a valid response and product code, the script activates the correct motor and the customer receives his product.

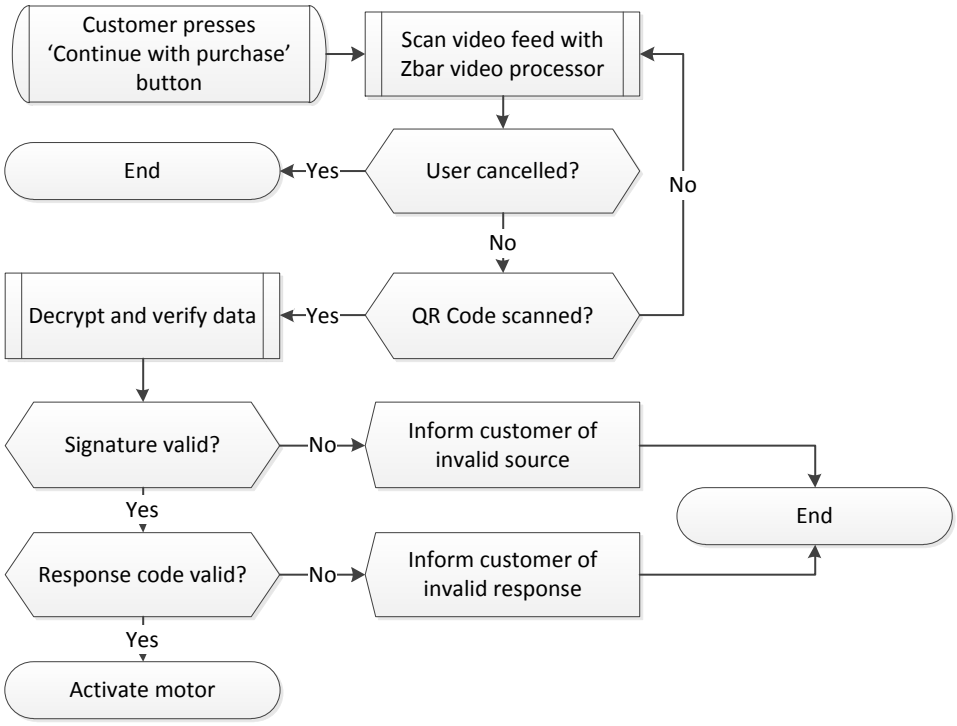

Figure 4.8: The generate_qrcode script process flow.

4.2.5 Near Field Communication

If the customer opts to purchase a product with the Android NFC application, the NFC script is run. This script is based on an example script included in the nfcpy package and is written by nfcpy's creator, Stephen Tiedemann [Tiedemann, Stephen (2013)]. This example script, called snep_test_server.py, connects the Pi to the NFC controller chip, polls the chip for a Simple NFC Data Exchange Format Exchange Protocol (SNEP) message, and when a SNEP message is read, it extracts the data in the message and presents it to the programmer for further processing and manipulation. This script allows the VM to extract SNEP messages from any source that follows the NFC Forum's standards [NFC Forum (2013)]. For this project, an Android application was written specifically for this purpose.

After the data is extracted from the NFC source, the script decrypts the data using the RSA algorithm and the VM's private key. The vending machine then checks the NFC message's response code. If the response code is valid, the script then activates the correct motor to dispense the product to the customer.

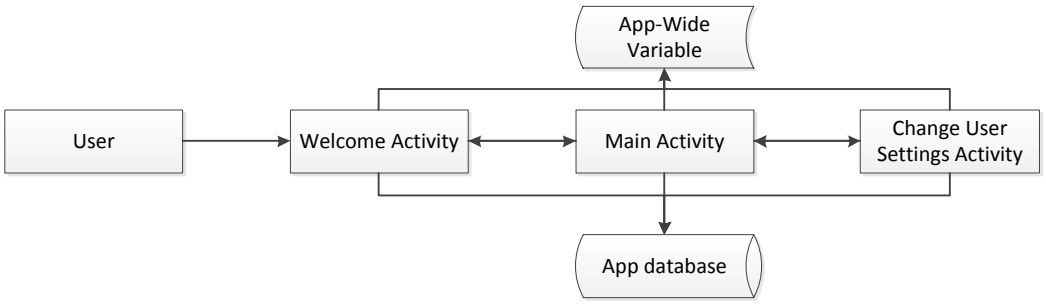

Figure 4.9: The Android NFC application structure.

4.2.6 Radio Field Identification

This module allows for a purchase to take place using a Radio Field Identification (RFID) card, such as a SU card. It is based on an example script from the libnfc module. The transaction process and its shortcomings are discussed in Sec. 3.10.3

It works by using the libnfc module to scan for a RFID card. When a card is detected, it extracts the card's Unique Identification (UID) number. If the UID number is present in the VM's code, the script activates the correct motor and the customer receives his product.

4.3 Android Application

An Android NFC application was made for this project. It allows a customer to buy a product from the application's product menu and to complete the purchase by tapping his phone against the VM's NFC receiver.

The application is divided up into three activities (Android's technical term for what is essentially a different window of the application). These activities, their design and significance are discussed in this section. See Fig. 4.9 for a diagram of the application's inter-activity interactions.

The source code is available from github.com/jayceelock/nfcpay6 and in the project file.

4.3.1 Welcome Screen

This activity is called when the application is opened. This activity's process flow can be seen in Fig. 4.10. Fig. 4.11 shows a screenshot of the welcome screen.

As seen from Fig. 4.10, the activity first checks to see whether it is the first time the user opens the application. If it is not, the activity goes on to the Main

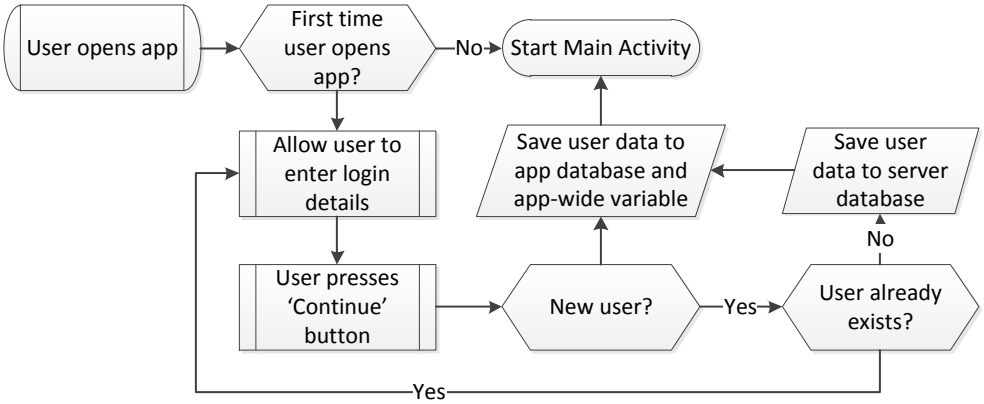

Figure 4.10: The Android NFC application welcome screen.

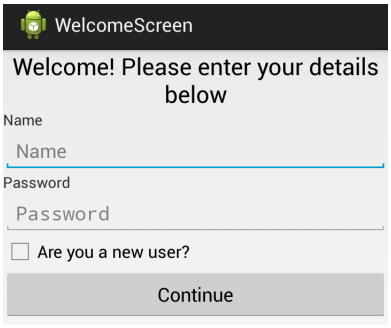

Figure 4.11: A screenshot of the application's welcome screen.

Activity. Otherwise, it allows the user to either sign in with an existing profile or create a new one.

When the user signs into an existing profile, the login details he provides is stored in a database inside the application. When this is done, it makes the login information persistent throughout the lifetime of the application. These details are also saved into an application-wide variable. This variable is used to make the application more efficient by not having to read the database as often. This variable is only active while the application is running in the foreground or background. The login details saved here are used later by the application's other activities.

If the user checks the 'New user' checkbox, the application the data is still saved to the database and the app-wide variable, but the applications also encrypts and embeds the user's login information into a URL. The application then sends this URL to the server, which then decrypts and saves the data to the user database. See Sec. 4.1.1 for more detail on the server activity.

After this has been done, the Main Activity is launched.

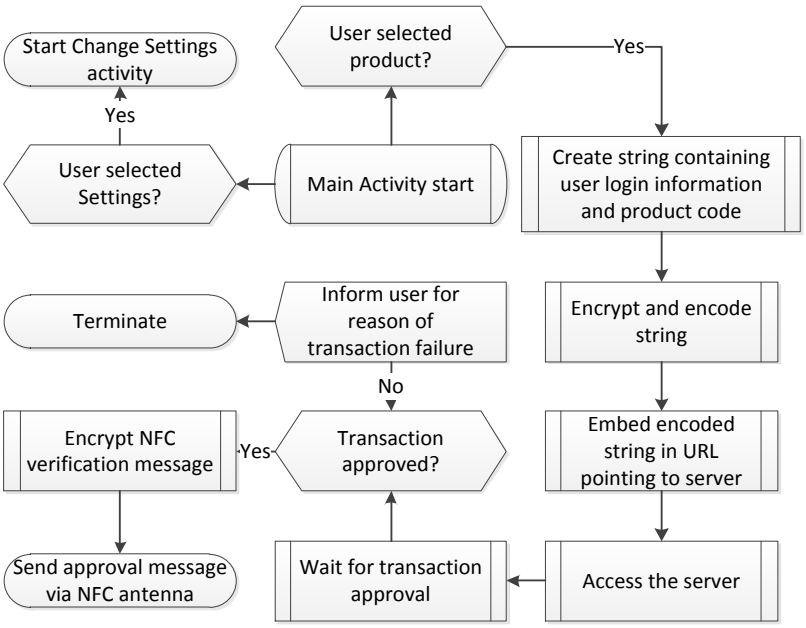

Figure 4.12: The process flow of the Main activity.

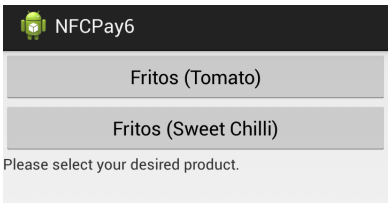

Figure 4.13: A screenshot of the application's main activity.

4.3.2 Main Activity

The main activity is the activity which is responsible for encrypting and sending the purchase requests to the web server, receiving and decrypting the purchase approval codes and sending the NFC messages to the VM's NFC receiver. See Fig. 4.12 for a process flow diagram and Fig. 4.13 for a screenshot of this activity.

This activity is based on example code for NFC usage from Google [Google (2011)] and the encryption and base64 encoding section is based on an encryption tutorial by tibi [tibi (2011)].

As seen from Fig. 4.13, the activity presents the user with a list of products. When the user selects a product to buy, the activity forms a data string by adding the product code, the user's login name and password together. This data string is then encrypted with the server's public key and then encoded to base64. This

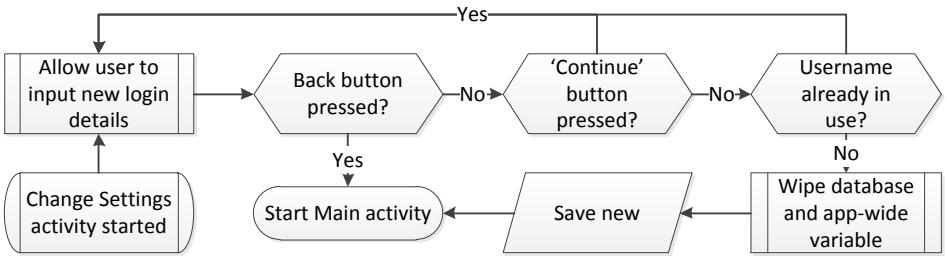

Figure 4.14: The process flow of the Change Settings activity.

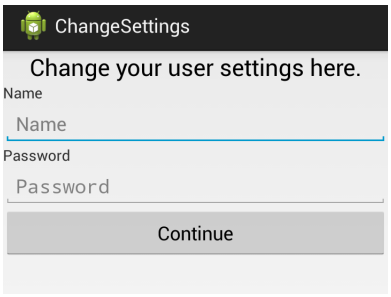

Figure 4.15: A screenshot of the application's change settings activity.

encoded string is then embedded inside a URL which points to the web server.

The activity then requests this URL in the application's background, which prompts the server to process the transaction (see Sec. 4.1.1 for more details on the server's NFC processes). The server then tells the activity if the transaction has been approved or denied. If it has been denied, the user is informed what was wrong. If it is approved, the activity activates the cellphone's NFC antenna which transmits an encrypted approval message to the VM's NFC receiver.

4.3.3 Change User Settings

The Change User Settings activity allows the user to change his login details that are saved in the database and the application-wide variable. See Fig. 4.14 for a process flow diagram for this activity. Fig. 4.15 shows a screenshot of this activity.

When the application enters this activity, it prompts the user to enter his user name and password. When the user presses the 'continue button', the old database entry and application-wide variable is overwritten with the new values the user entered. The details stored on the server's database is not affected, however.

Chapter 5

Hardware Detail Design

This chapter gives a detailed discussion on the hardware design and configuration of this system. This includes the relay switching circuit, the motor and coil design and the NFC chip configuration.

See Appendix D for a schematic layout of the hardware and their connections. All the datasheets and drawings referred to in this chapter are cited and are available in the project file.

5.1 Relay Switch Circuit

As explained in Sec. 3.5.3 on p.14, a relay switch [Mantech (2013)], in conjunction with a 2N2222 Bipolar Junction Transistor (BJT) [ST Electronics (2013)], is used by the Raspberry Pi to control the motor. See Fig. 5.1 for the circuit diagram.

For the following equations, P_r and V_r is the power dissipation and voltage

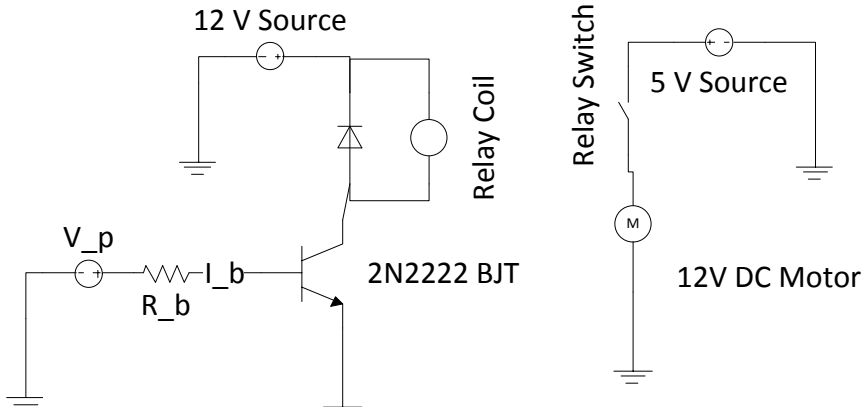

Figure 5.1: 12 V relay transistor switch.

across the relay coil, β is the BJT current amplification, V_p and I_p is the voltage and current supply from the Raspberry Pi's GPIO pin and I_b and R_b is the BJT's base current and resistor.

The relevant parameters for these components are [Mantech (2013), ST Electronics (2013)]:

$$P_r = 0.36 \text{ W}$$

$$V_r = 12 \text{ V}$$

$$\beta \approx 10$$

$$V_p = 3.3 \text{ V}$$

From the relay's power dissipation and voltage, its current draw is found by

$$I_r = \frac{P_r}{V_r}$$

which gives a current draw of

$$I_r = 0.03 \text{ A}$$

Taking the BJT's current amplification as roughly 10, as recommended by the BJT's datasheet [ST Electronics (2013)], the current draw from the Pi to the base of the BJT is given by

$$I_b = \frac{I_r}{\beta}$$

This gives a current draw of

$$I_b = 3 \text{ mA}$$

The maximum allowable current draw from a GPIO pin is 16 mA [elinux.org (2013)], though this is not recommended as the Pi does not have any current limiting or over-current protection hardware. A current draw of 5mA per pin is recommended by the Pi's manufacturers [elinux.org (2013)]. Therefore, a current draw of 3 mA is completely safe.

To limit the current draw from the Pi, a base resistor must be added between the Pi's GPIO pin and the BJT's base. With a current draw of 3 mA and a voltage of 3.3 V, the resistor size is found by using Ohm's law:

$$R_b = \frac{V_p}{I_p}$$

which gives

$$R_b = 1.1 \text{ k}\Omega$$

To prevent voltage spikes from the Pi's GPIO's from supplying to much current, a further 10% was added to the resistor size. This gives a resistor size of

$$R_b = 1.2 \text{ k}\Omega$$

which draws a current of

$$I_b = 2.75 \text{ mA}$$

which is still enough to activate the relay's coil and turn the motor on.

5.2 NFC Chip

The Near Field Communication (NFC) chip used, is the Adafruit NFC shield for an Arduino Uno microcontroller [Adafruit Industries (2012)]. See Sec. 3.3 on p.13 for more details on the controller.

The shield was designed and built to be used by an Arduino Uno microcontroller. Therefore, some modification to the chip's connections had to be made before it was able to communicate with the Pi. By default, the chip is made to communicate with an Arduino using the Inter Integrated Circuit (i^2c) communication protocol. However, the component manufacturers have added connection pads to the chip that, when shorted, allows the chip to communicate via Serial Peripheral Interface (SPI) or Transistor Transistor Login (TTL). The libnfc library is currently only configured to allow communication via an Universal Asynchronous Receiver Transmitter (UART). Therefore, the chip was modified to communicate via its TTL interface, which is compatible with the Pi's UART interface.

See Appendix G for detailed explanation on this configuration process and also how the libnfc library was built and compiled.

5.3 Vending Machine Unit

A small VM unit was constructed for demonstration purposes. It is designed to house all the VM components, i.e. the two DC motors, the two coils, the Raspberry Pi central controller, the NFC shield and the webcam.

Four 10 mm vent holes were added at the sides to improve air circulation and to provide external wire access, while a larger 60 mm vent hole was also added to allow for an external 60 mm desktop computer fan to be added at a later stage.

The unit is made from 1.6 mm thick mild steel and the components were cut and bent by Fabrinox, Paarl and welded together by the Electrical and Electronic Engineering Department of Stellenbosch University's workshop.

See Fig. A.1 in Appendix A for a picture of the complete VM unit with the two DC motor inserted. See Appendix A for detailed design drawings of the VM unit.

5.4 Webcam

As discussed in Sec. 3.4, a Sony PS2 Eye Toy webcam was attached to the Raspberry Pi. This allows the VM to scan a live video feed for a QR Code. It is already compatible with the Raspberry Pi and therefore requires minimal configuration to begin working.

It is important to note that a Raspbian version from early 2013 was used on the Raspberry Pi. It was found that the latest Raspbian distributions have an unknown bug in its Video4Linux video drivers which causes the EyeToy to work incorrectly. Furthermore, the camera needs to be plugged into a Universal Serial Bus (USB) port on the Pi itself and not into an external USB hub. This is done to prevent hardware timing issues that are introduced when a USB hub is used.

5.5 Motor and Coil

The VM dispenses products by activating a motor that turns a coil loaded with a product. This turning motion causes a product to move along the coil in an Archimedean screw-like manner until it falls off at the end of the coil.

In the demonstration VM, two coils are used. The coils are made from 2.5 mm thick galvanised iron wire. The coil diameter is 50 mm and has a longitudinal pitch of 20 mm.

It is important to only activate the motor for one turn of the coil to ensure that only one product is dispensed per completed transaction. This is currently being done by only activating a motor for a set amount of time. To determine this time, it was required that the speed of the motor be determined for a given supply voltage. The motors that are used are 12 V DC motors made by Faulhaber [Faulhaber (2013a)] coupled with a 14:1 speed gearbox [Faulhaber (2013b)].

For the following calculations, V_o, I and R are the motor supply voltage, current and terminal resistance, V_e is the back-Electromotive Force (EMF) voltage, ω is the motor speed and k_e is the Back-EMF constant.

The following equation gives the relationship between a DC motor's supply voltage and its back-Electromotive Force (EMF) [MicroMo (2008)].

$$V_o = IR + V_e$$

The back-EMF is proportional to the rotational speed of the motor. Therefore, the V_e term can be taken as

$$V_e = k_e\omega$$

which gives the following equation

$$V_o = IR + k_e\omega \tag{5.1}$$

The IR term is taken as constant, since the torque load on the motor stays constant. From the motor's datasheets and usage tests, the I, R and k_e terms were determined to have the following values:

$$I = 0.35 \text{ A}$$

$$R = 0.41\ \Omega$$

$$k_e = 2 \text{ mV/rpm}$$

Using these values, and setting the input voltage as 3.3 V in equation 5.1, the motor's speed was determined to be 113 RPM. Therefore, to complete a single revolution, the motor may only be activated for 0.53 seconds.

This is not the most effective method, however, because the errors in the system accumulate over time. In time, these errors may lead incorrect product dispensing.

To improve this, a proximity sensor may be added onto the coil and onto the VM wall. This sensor can be connected to and be controlled by the Raspberry Pi and would be activated every time the coil completes one revolution. When the sensor is triggered, the Pi will then know that exactly one revolution has been completed and stop the motor. This will prevent any errors from accumulating over time and make the VM more reliable.

Chapter 6

System Tests

In this chapter, the tests conducted to determine how well the system measures up to the original goals and objectives, set out in Sec. 1.4 on p.3, are discussed.

The tests conducted are the system resource usage for the different payment options and user stories where scenarios are created and the response of the system is given. The test results are also displayed and discussed.

6.1 System Resource Usage

Tests were conducted to check and compare the system load caused by each of the different payment methods. The processor load, memory and disk usage were tested by using the DStat package [Wieers, Dag (2013)].

The DStat software ran alongside the software being tested, which also used some resources. However, this could not be avoided. The resource usage is present for all three the tests and therefore does not unfairly skew the results.

6.1.1 Quick Response Code Test

Processor Usage

Fig. 6.1 shows the total processor usage during the complete Quick Response Code (QR Code) transaction, as described in Sec. 3.10 on p.17.

It can be observed from Fig. 6.1 that there are significant spikes near maximum capacity when the Pi starts generating a QR Code and when the return QR Code is scanned. This was expected, as image and video processing is taxing on the Pi's hardware and the Pi has no dedicated graphics processor. The data encryption also adds to the processing load. A possible solution is to overclock the processor, but it is already overclocked to 900 MHz and pushing the processor harder may damage the Pi and void its warranty.

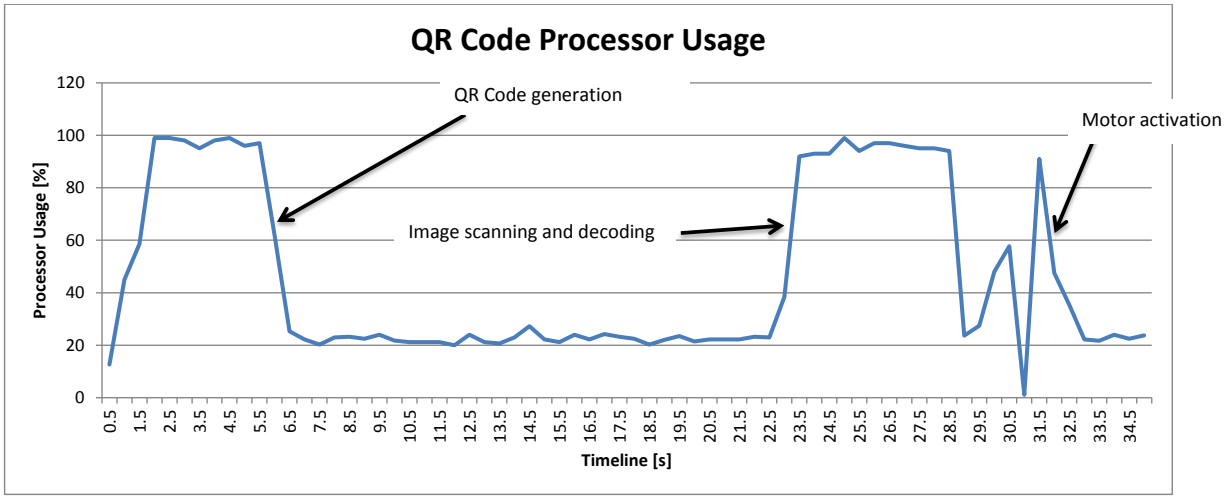

Figure 6.1: QR Code processor usage test results.

It can also be observed that there is a significant time lag between QR Code generation and the QR Code scanning stages. This is when the user is awaiting authentication from the server. The only way that this lag can be improved is to move the server location from the USA to South Africa. It is not expected that doing so will make a significant difference, however.

Disk Write Activity

Fig. 6.2 shows the disk writing activity during the QR Code transaction. It shows several spikes during the transaction. This can possibly be attributed towards swap file activity by the Pi's operating system and the image encoding and decoding processes. Optimising the code may improve this.

Memory Usage

Fig. 6.3 shows the memory usage during the QR Code transaction process. It shows a relatively flat graph in the beginning, but goes up toward the end. It also shows that the memory used does not return to its start value. This may indicate that memory leaks may occur over prolonged use.

The memory performance is tested and discussed in more detail in Sec. 6.2.

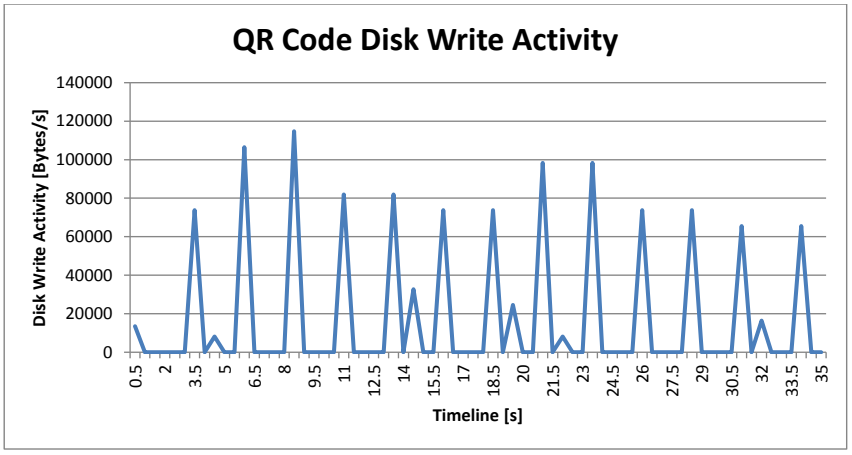

Figure 6.2: QR Code disk write test results.

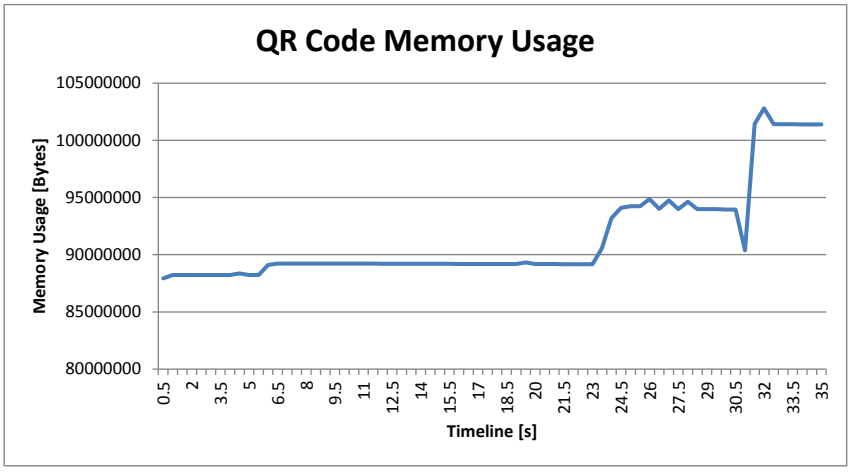

Figure 6.3: QR Code memory usage test results.

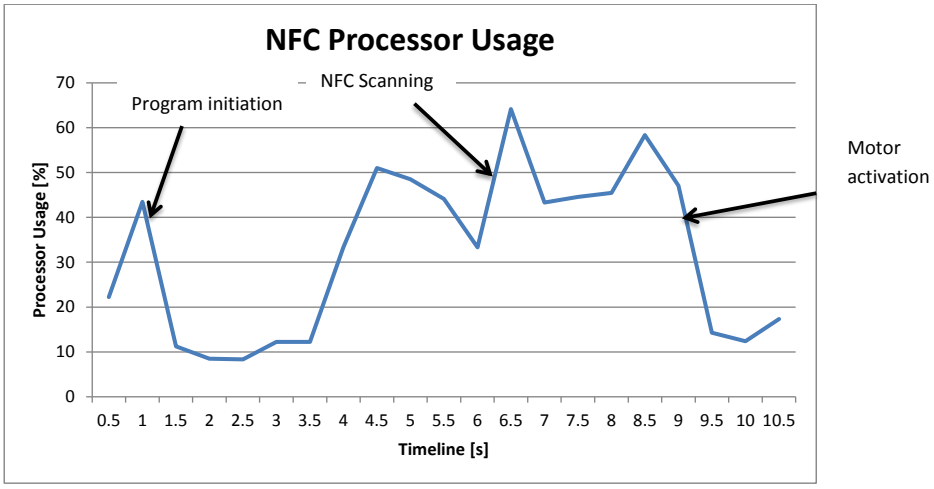

Figure 6.4: NFC processor usage test results.

6.1.2 Near Field Communication Test

Processor Usage

Fig. 6.4 shows the processor activity during the complete Near Field Communication (NFC) transaction process. It shows an average load of approximately 50% during the NFC scanning process. It can also be observed that the transaction takes under a third of the time it takes for the QR Code option. This is a significant improvement over the QR Code payment option, but code optimisation and improvements to the Android application may improve this further.

Disk Write Activity

Fig. 6.5 shows the disk activity during the NFC transaction process. It shows far less spikes during the transaction than the QR Code option. These spikes may again be attributed toward swap file activity from the Pi's operating system. The spikes may be fewer because of the decreased processing load and that it takes less time to complete one transaction.

Memory Usage

Fig. 6.6 shows the memory usage during the NFC payment process. It shows a relatively flat line, which indicates that the NFC transaction process is not memory intensive. This is again a significant improvement over the QR Code option.

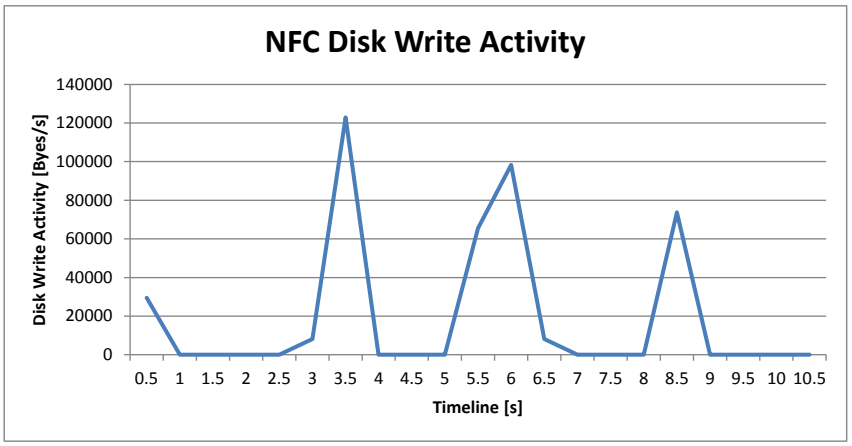

Figure 6.5: NFC disk write test results.

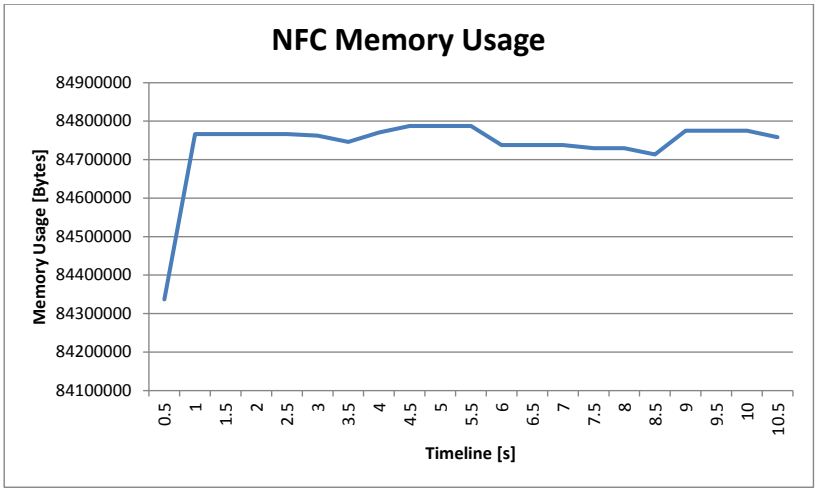

Figure 6.6: NFC memory usage test results.

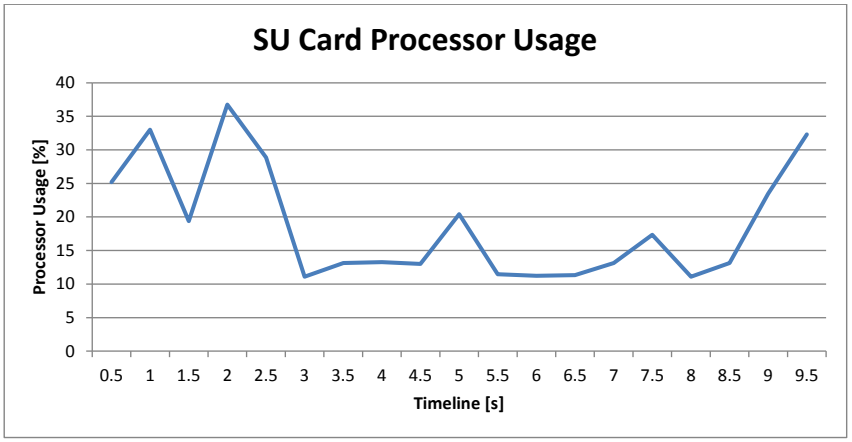

Figure 6.7: SU Card processor usage test results.

6.1.3 Stellenbosch University Card Test

Processor Usage

Fig. 6.7 shows the processor usage during the Stellenbosch University (SU) card transaction process. It shows only a single spike at 35% while it is scanning for a card. This is far less that the QR Code option and a small improvement over the NFC option. This increase in performance over the NFC option may be due to the fact that NfcPy (the module used during the NFC transaction process) is built on top op libnfc and incorporates additional communication protocols, making it a bulkier process than the SU card option. The SU transaction process makes use of only libnfc and will therefore most likely be less processor intensive.

Disk Write Activity

Fig. 6.8 shows the disk writing activity of the SU transaction process. It looks similar to the NFC transaction disk activity from Fig. 6.5. This may be because they work in a similar manner and both use libnfc to scan for NFC devices.

Memory Usage

Fig. 6.9 shows the memory usage during the SU card transaction process. With the exception of the start-up spike, the graph shows a very flat line. This indicates that the memory usage relatively low and is very stable.

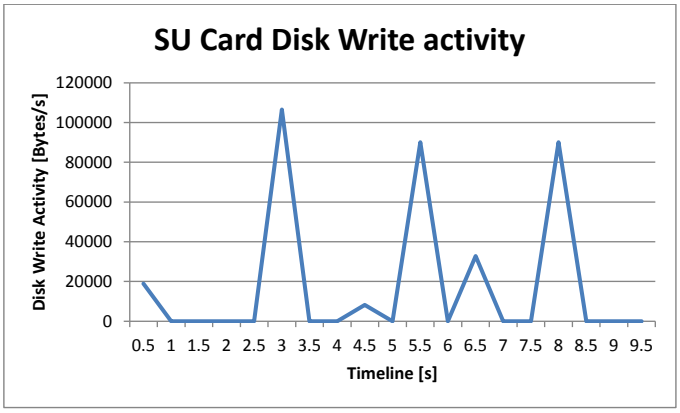

Figure 6.8: SU Card disk write test results.

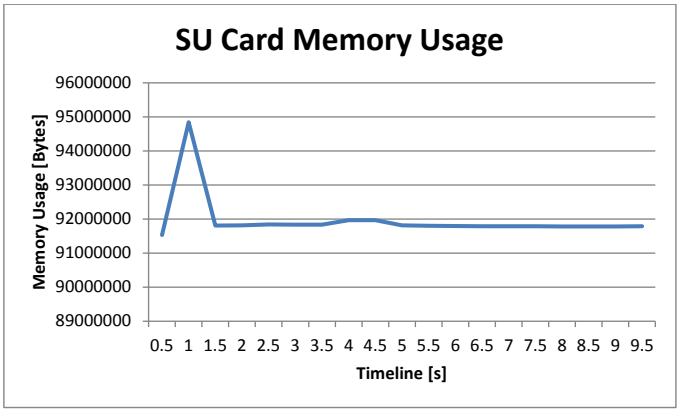

Figure 6.9: SU Card memory usage test results.

6.1.4 Conclusions

The resource usage of the QR Code, NFC and SU card transaction processes were analysed. It was found that the QR Code option is the most memory and processor intensive and causes significant disk activity to take place.

The NFC and SU card transactions cause similar system loads with regards to memory usage and disk activity, possibly due to their common code base. However, the SU card transaction process demands approximately 15% less processing power than the NFC option.

All three transaction processes can be improved and streamlined. This can be done by porting the Python code over to a compiled code format, such as C or C++. Furthermore, some hardware modifications, such as processor overclocking, may also decrease the system load.

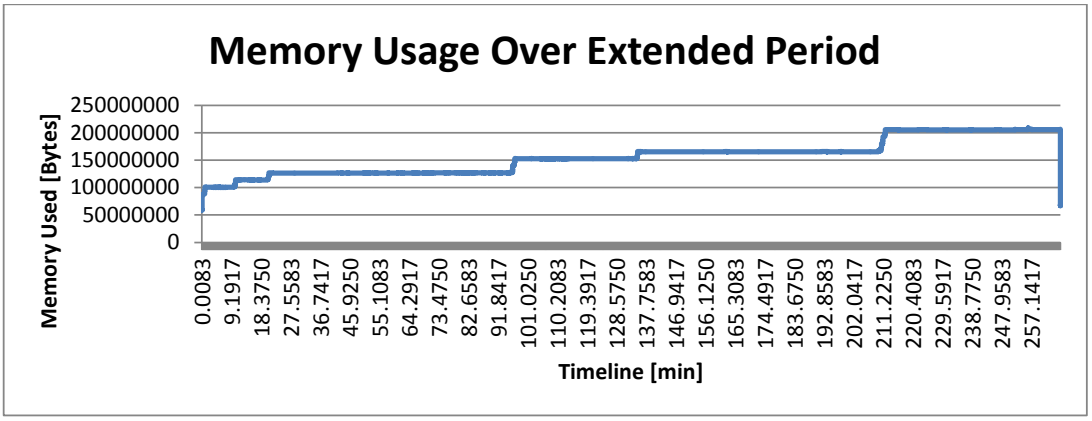

Figure 6.10: Memory usage over extended period of time.

6.2 System Usage Over Extended Period

This test was performed to measure the memory usage of the VM program over an extended period of time. In this case, the VM program was left running for approximately 4.5 hours. During this time, 8 QR Code transactions took place.

The reason for only using QR Code transactions is because it was found that the QR Code transaction process is the most resource intensive (see Sec. 6.1.1). Fig. 6.3 also showed that the process does not release the memory it used during the transaction back to the system. This is called 'memory leaking'.

A graph of the memory usage over time can be observed in Fig. 6.10.

Fig. 6.10 shows that the memory usage doubles from 100 MB to 200 MB in less than 5 hours and 8 QR Code transactions. The sudden decrease in the end is because the VM program was shut down.

It is suspected that the reason for this memory leakage is that the image processing step does not properly clear the itself from memory after it has finished scanning a QR Code. Every subsequent QR Code scanning process adds to this build-up of uncleared memory. This will continue until the VM eventually runs out of memory and freezes. This may have unforeseen consequences for the VM.

The memory leakage here is a serious issue and must be fixed before the system can ever be commercialised. To fix it, the code must be thoroughly inspected and tested for bugs. This may require a possible rewrite of the image processing code.

6.3 User Tests

The functionality of the system was tested by exposing it to different user scenarios and seeing what the outcome was. The testing process, scenarios and results are

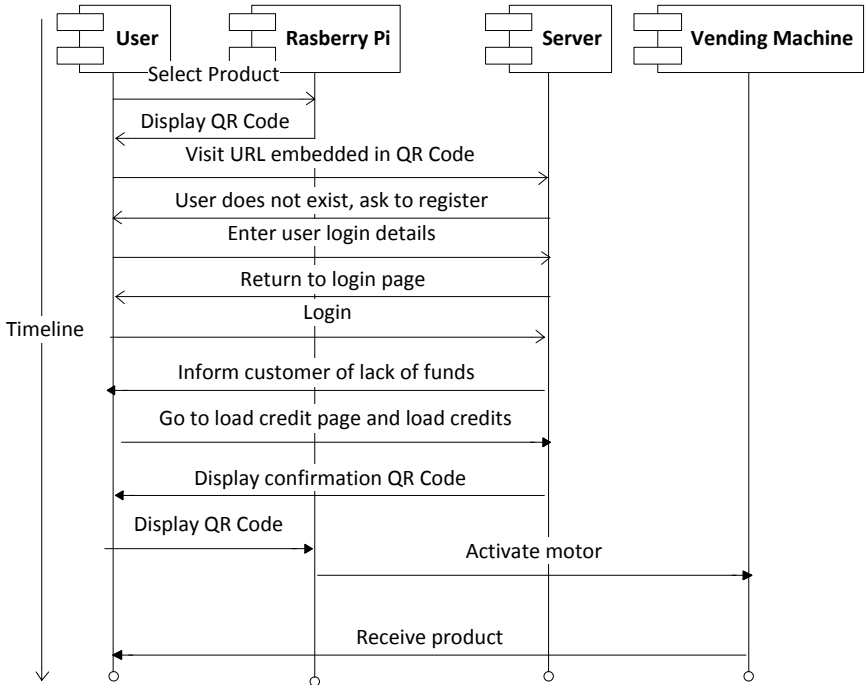

Figure 6.11: The first user scenario.

discussed in this section.

6.3.1 User Test 1

In this user story, the customer decides to pay with a QR Code. He does not yet have a user account on the database and has no credits loaded. Fig. 6.11 shows the user story diagram.

Scenario Walkthrough

The user first selects which product to buy from the Raspberry Pi's User Interface (GUI). The Pi then generates a QR Code that the user scans with any barcode scanning application. The Universal Resource Locator (URL) embedded in the QR Code then takes the customer to a web page located on the server.

Not having a user account, the user clicks on the link to create a new user profile. After entering valid user information, the customer is returned to the login page where he can log in with his new login credentials. After logging in, the server displays to the customer that he does not have any credits loaded onto his account. The customer then goes to the load credit page where he loads some money. When this is done, the user can press the ling that prompts the server to

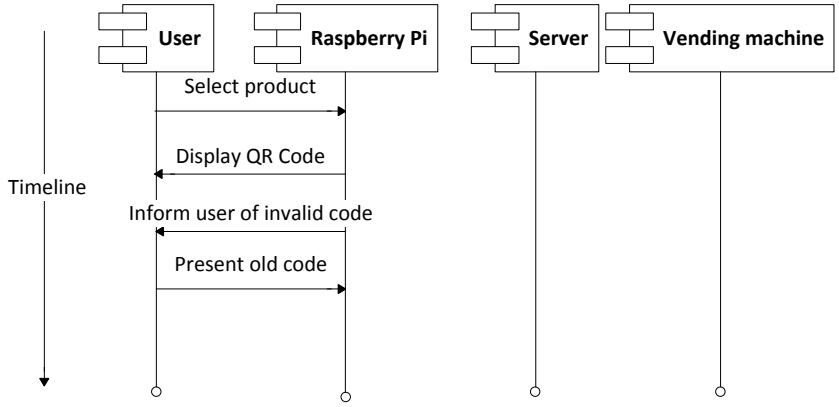

Figure 6.12: The second user scenario.

display the confirmation QR Code on the customer's cellphone screen, verifying the transaction.

The customer then presses the continue button on the GUI, which starts scanning the web cam feed for the customer's QR Code. After this is done, the Pi activates the correct motor and dispenses the product.

Results

In this scenario, a customer successfully loaded credits onto his account and bought a product with the new credits. There are many steps involved in the process, however, and this leads to a relatively long transaction process.

It was also found that the server is sometimes unable to correctly decrypt the code it receives from the customer's cellphone. It is unclear why this happens. It is possible that the barcode scanning application on the cellphone incorrectly decodes the VM's QR Code. This issue will have to be fixed before the system can be commercialised.

6.3.2 User Test 2

In this scenario, a customer tries to use a QR Code from a previous transaction without paying. Fig. 6.12 shows the user story.

Scenario Walkthrough

To begin, the customer selects a product to buy, after which the Raspberry Pi displays a QR Code. However, instead of scanning the QR Code and paying for a new product, the customer tries to use an old QR Code that the server sent him

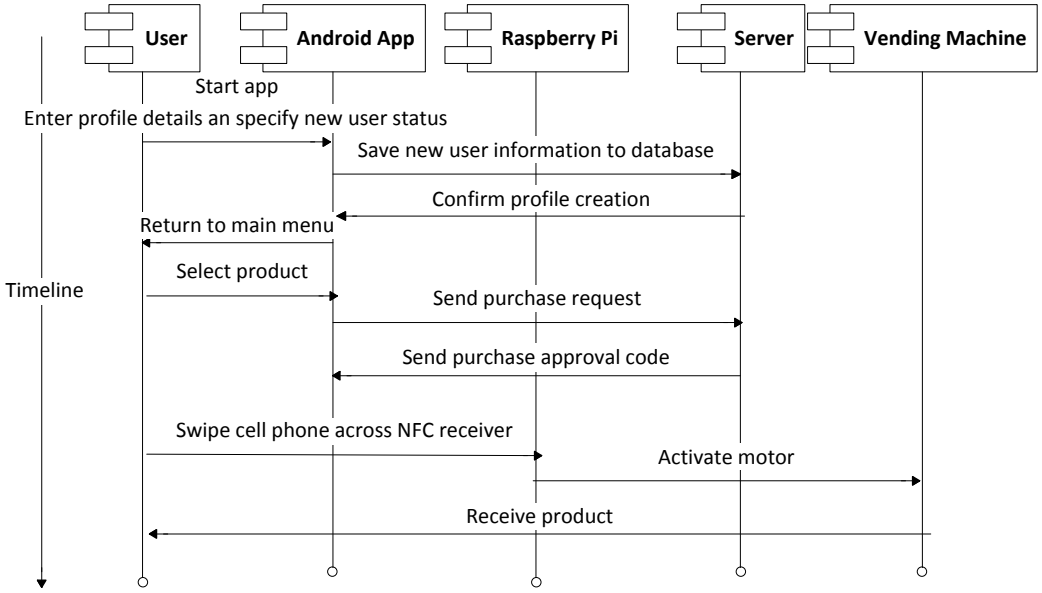

Figure 6.13: The third user scenario.

after a successful transaction previously. The customer tells the Raspberry Pi to scan his old QR Code. The Raspberry Pi does this, but the old QR Code does not contain a valid response to the Raspberry Pi's challenge. Therefore, the transaction is not accepted and the customer is informed of his invalid QR Code.

Results

The results from this scenario shows that the challenge-response system works as it was designed to: it does not allow an old QR Code to be used again without it being paid for.

6.3.3 User Test 3

In this scenario, the user opts to buy a product using the Android NFC application. However, the application must first be downloaded, installed and a new user profile must be created. The user story can be seen in Fig. 6.13.

Scenario Walkthrough

After the user installs and opens the application, he is presented with a login screen. The user enters his login credentials and checks the box that confirms that he is a new user. The application then contacts the server with a user creation request.

Provided the username is not already used, the server creates and saves the new user profile.

The application then takes the user to the main menu where the user can pick a product to buy. After picking a product, the application contacts the server with a purchase request. If the transaction is successful, the server sends the application a confirmation code which the application transmits via the cellphone's NFC antenna.

The user then swipes his phone across the VMs NFC receiver. The VM then activates the motor and the user receives his product.

Results

In this scenario, a customer has successfully used the Android NFC payment application to create a new user profile and buy a product from the VM.

The process is more streamlined than the QR Code option and consists of fewer steps. This makes the NFC transaction process take place in a shorter period of time. After the customer has opened the application once, he will not have to enter his login details again. This will make the process slightly quicker in the future.

Chapter 7

Conclusion

7.1 System Performance

In the final system, both Quick Response Code (QR Code) and Near Field (NFC) technology, as well as Stellenbosch University (SU) cards, have been implemented as payment methods. It was found that the NFC and SU cards options have the shortest transaction time between product selection and when the product is dispensed. However, the NFC option is currently limited to cellphones that have NFC antennas and are based on the Android Operating System and the SU card option currently has limited user verification functionality and cannot keep track of a customer's current balance.

It was found that the QR Code payment option typically has a slower transaction time. This is mainly due to some hardware bottlenecks of the Pi. These bottlenecks include the Raspberry Pi's moderate processing power, which increases the encryption time and the time it takes the Pi to process and decode a live video stream from the webcam. These bottlenecks may be improved with hardware modifications and processor overclocking. Extra work can also go into optimising the code to make it run more efficiently. This can be done by converting the code into compiled machine code, such as C or C++.

Furthermore, a server has been made that runs in the computing cloud. This server interacts with a customer's cellphone through the Android NFC application or a cellphone's internet browser. All data transactions between the server and the customer's cellphone are encrypted with an asymmetric encryption scheme with extra layers of security added.

Lastly, a working demonstration VM was designed and made. It houses all of the VM components and allows a customer to buy one of two products using his cellphone.

7.2 Future Work

The VM system that has been designed in this project has been designed to specification. With that being said, there is potential to expand upon the work that has been completed thus far. This section discusses some improvements and additions that can be made to this system.

7.2.1 Commercialisation

The VM currently uses faux money which has no real-world value. Therefore, the system in its current state is not ready to be deployed across SU's campus.

To commercialise the VM, third-party payment providers, such as PayPal or PayPoint, can be integrated into the system. This will allow customers to buy credits to use on the VM system by using their credit cards. All the transactions take place electronically and will therefore be integratable with the current system.

Furthermore, the memory leakage issue discussed in Sec. 6.2 will need to be addressed and fixed.

7.2.2 Polish

This project focused on the practical and functional aspects of the complete system and little attention was given to the aesthetics of the complete system. Therefore, some extra work can go into making the web server and Android NFC application more user-friendly and improving their interfaces.

7.2.3 Integration With Current Systems

One important aspect that must be focused on to increase the odds of making this project more commercially successful, is its integration with current VMs, i.e. adding a cashless option to current VMs that are restricted to cash.

7.2.4 More Payment Options

Some work can go into expanding the number of payment options available to the user. Some options that may be considered are Bluetooth, Instant Messaging (such as WhatsApp, BlackBerry Messenger and MXit) and Unstructured Supplementary Service Data (USSD).

Appendix A

Vending Machine Drawing

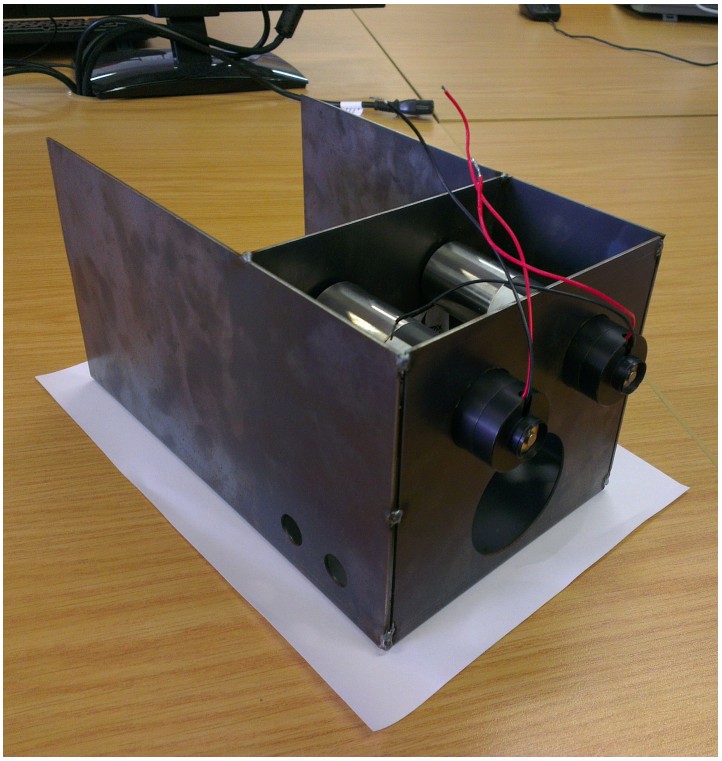

Figure A.1: The complete vending machine unit.

Appendix B

Techno-Economic Report

B.1 Budget

The planned total cost of completing this project was R532 000, with R12 000 going toward manufacturing and components and R520 000 going toward the designer's salary.

The final cost for the design and manufacture of the complete system is R534 596. The cost breakdown for this can be seen in Table B.1. The actual system value is R14 596.

Not all of the components used in this project were bought new. Some were loaned from Stellenbosch University or elsewhere. Therefore, the actual cost of completing the project is R520 887.

Possible areas where saving can be made is to use smaller, cheaper motors and a smaller, less expensive computer monitor. A smaller monitor will cost approxi-

Table B.1: Cost breakdown of the project and system.

Component	**Amount required**	**Bought?**	**Cost when new**
12DC Motor	2	No	R4000
NFC Controller	1	Yes	R780
PS2 Eye Toy Webcam	1	No	R200
12V Relay	2	No	R20
2N2222 BJT	2	No	R5
Vending machine unit	1	Yes	R107
Raspberry Pi	1	No	R400
HDMI Monitor	1	No	R5000
Coils	2	No	R30
Web server	1	No	Free

mately R1000 and if the system is integrated with a cash-based vending machine, the cost for the DC motors will disappear. This gives a total system cost of R 3 596.

B.2 Time Management

This project was planned to be completed within 10 months. It was completed ahead of schedule. A Gannt chart showing the project timeline can be seen in Figure C.2 in Appendix C.

B.3 Technical Impact

This project produced a vending machine which accepts payments made via cell phone. Allowing customers to pay using only their cell phones makes an already convinient service even more convenient.

The prototype system was designed with expansion and modification in mind an is not necessarily restricted to vending machines. For example, the system will accept any cell phone payment made at any vendor if it is properly modified and configured.

B.4 Return on Investment

As discussed in Section 1.3 of the main report, cashless transactions are expected to surpass cash transactions by the year 2015. It is therefore very important for vendors to keep up with the trend and allow customers to pay with cash, but still provide a cashless payment facility.

This project has delivered a working cashless vending machine, but the system can be expanded upon and adapted to be integrated with traditional, cash-based vending machines. This integration will require more research and development, however, but it will be worth it to give modern shoppers a choice between paying with or without cash.

To get to this point, an estimated R2.3million will be required. R1.1million will go toward paying the developers' salaries, R1million toward setting up data servers and paying their overhead costs, R100 000 for components and manufacturing costs and a final R100 000 for buying at least three standard cash-based vending machines to test the final system.

When the final system is complete, it presents a few opportunities for investors to make money. Firstly, the systems can be manufactured by the development

company and sold to vending machine manufacturers. Second, it can be manufactured under license by the vending machine manufacturers, with the development company receiving royalties. Lastly, the system can be patented and the patent can be sold to another company for a premium.

If the system is manufactured in-house, it will cost approximately R3 596 to make. If these systems are then sold to vending machine manufacturers at R7000 per unit and 210 units are sold per year, the original investment will be remade approximately 4 years, not accounting for inflation. This gives a return of 18.9% per year.

If these systems are manufactured under license by vending machine companies, and royalties are received in the amount of R2300 per unit made with 210 units made per year, it will take 6 years for the investors to remake their money. This gives a rate of return of 12.2% per year.

Lastly, if the patent is sold, it will have to be sold for at least R2.9million to make up for the initial investment. It is highly unlikely that a company will spend this amount of capital on an unproven system. If it is sold for R3million, it will immediately give investors a return on their investment. However, the long-term revenue generated by the other two options far outweigh this R3million and it is therefore recommended that the first two options be considered.

Appendix C

Updated Gannt Chart

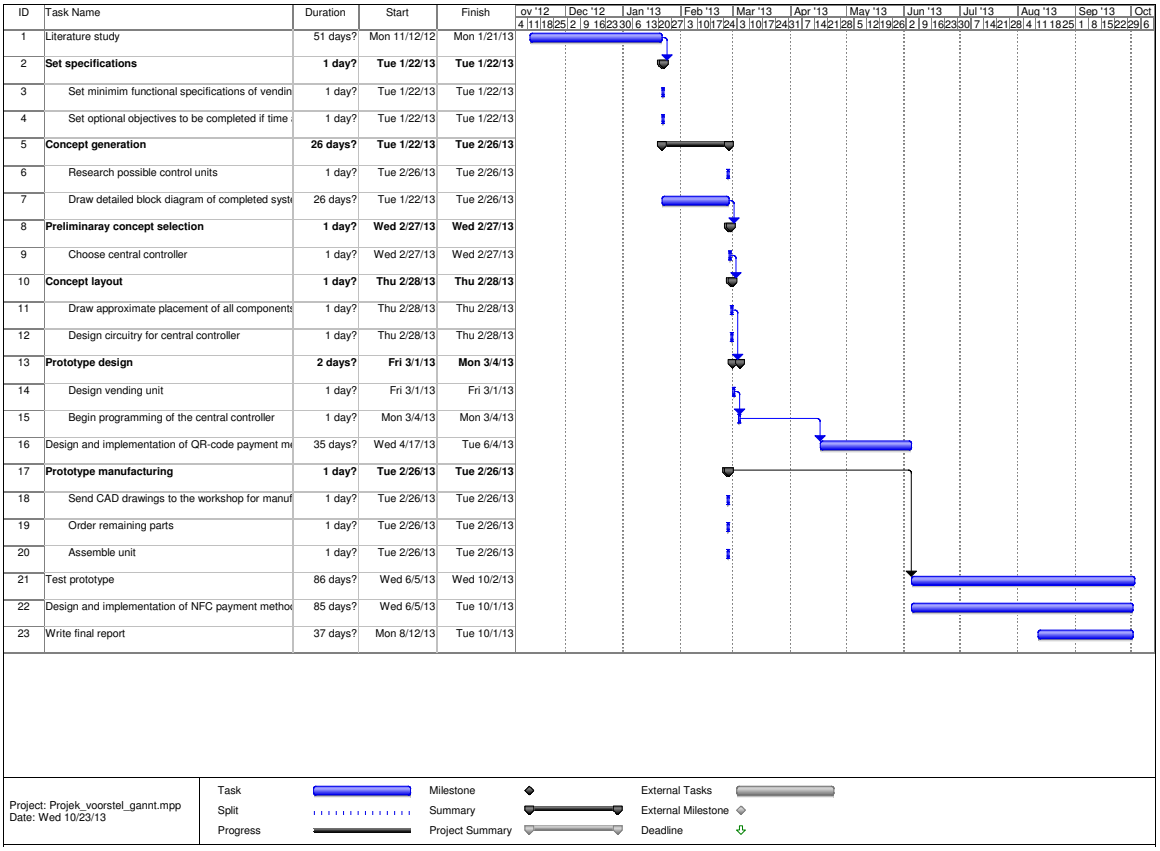

Figure C.1: Original Gannt Chart showing the proposed project timeline.

ID	Task Mode	Task Name	Duration	Start	Finish
1					
2		Start Project	0 days	Mon 2/4/13	Mon 2/4/13
3		Configure Raspberry Pi	6 days	Mon 2/4/13	Mon 2/11/13
4		Develop web server	120 days	Mon 2/11/13	Fri 7/26/13
5		Project Proposal	0 days	Fri 3/1/13	Fri 3/1/13
6		Develop QR Code payment option	100 days	Mon 3/11/13	Fri 7/26/13
7		June Recess	0 days	Sat 6/1/13	Sat 6/1/13
8		Create vending machine GUI	36 days	Fri 6/7/13	Fri 7/26/13
9		Configure NFC hardware and software	37 days	Fri 6/7/13	Sat 7/27/13
10		Progress Report	0 days	Mon 7/29/13	Mon 7/29/13
11		Create Android NFC app	45 days	Mon 7/29/13	Fri 9/27/13
12		Design vending machine unit	6 days	Mon 7/22/13	Mon 7/29/13
13		Manufacture vending machine unit	3 days	Tue 7/30/13	Thu 8/1/13
14		Design and make relay switches	2 days	Fri 8/2/13	Mon 8/5/13
15		Deploy web server to cloud	6 days	Mon 9/2/13	Mon 9/9/13
16		Integrate vending machine program with cloud server	15 days	Tue 9/10/13	Mon 9/30/13
17		Rough Report Hand-in	0 days	Fri 9/27/13	Fri 9/27/13
18		Fine-tune and debug system	19 days	Fri 9/27/13	Wed 10/23/13
19		Perform system tests	19 days	Fri 9/27/13	Wed 10/23/13
20		Final Report Hand-in	0 days	Thu 10/24/13	Thu 10/24/13

Figure C.2: Gannt Chart showing the actual project timeline.

Appendix D

System Hardware Schematic

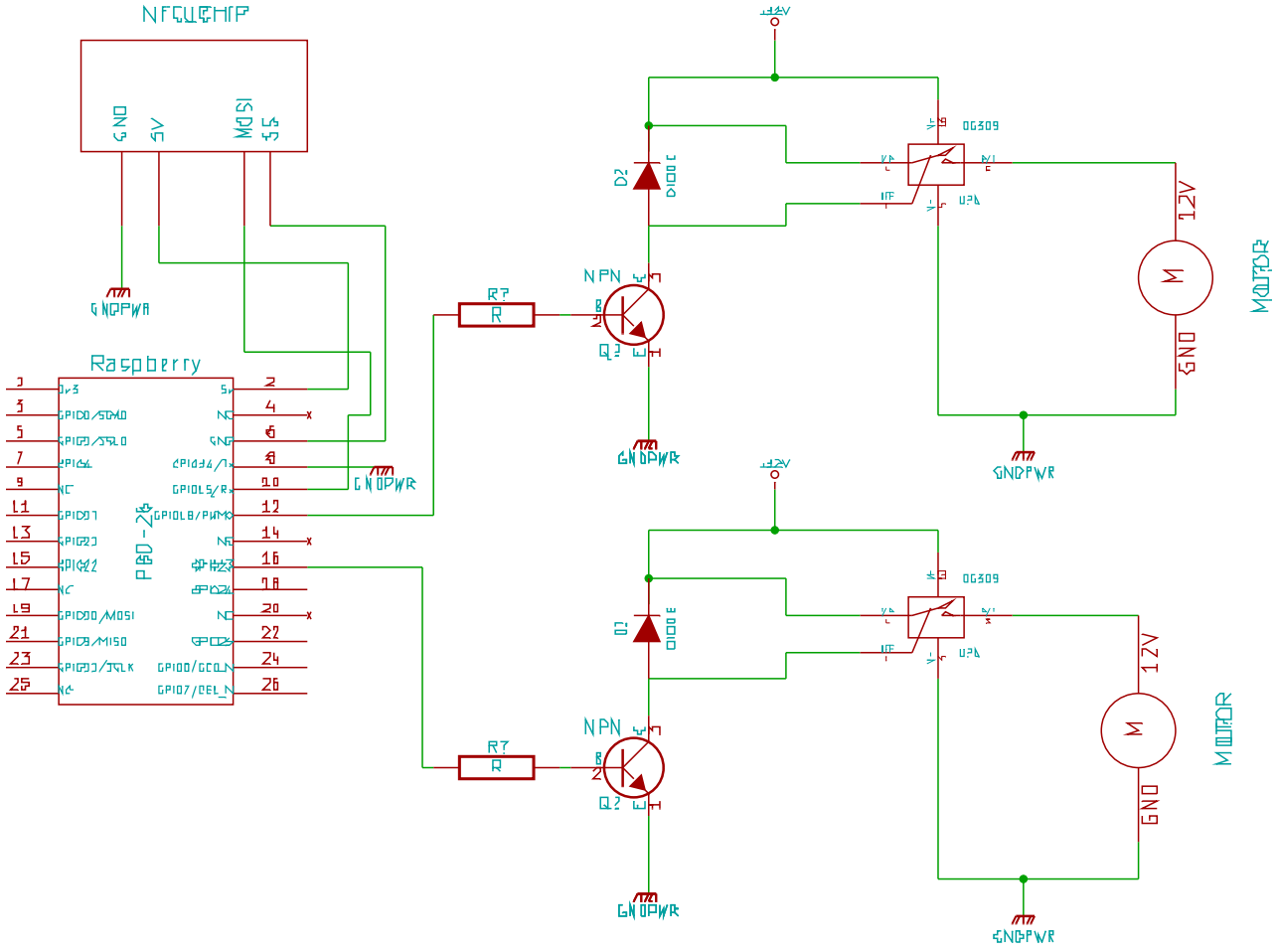

Figure D.1: Schematic of the complete system.

Appendix E

Asymmetric Encryption

Asymmetric encryption is most easily explained with a postal analogy:

Imagine that two people, Alice and Bob, want to send each other secret messages through the public mail. In other words, Alice wants to send Bob a secret message and she expects a secret reply from Bob, and vice versa.

In an asymmetric scheme, Bob can lock his letter to Alice with a padlock to which only she has a key (she keeps this on her person at all times and does not show it to anyone, which includes Bob). This open padlock represents the public key half of Alice's key. This means that anyone can send Alice a secure message with a public key, which is easy to get from Alice, and only Alice can unlock the message with her private key half of the key pair. Similarly, Alice can lock her letter with Bob's padlock which only Bob can open.

The great advantage that this has over symmetric encryption is that the decryption keys never have to be exchanged between parties. This neutralises the risk of a middle-man attack, analogous to a nosy postal worker called Eve who likes to read other people's mail, who then intercepts the message and steals the key. Also, if for example Bob has been careless and allowed Eve to see his key, his messages to Alice will be compromised. However, the messages from anyone else (including Bob) to Alice will remain as secure as it was before Bob lost his key.

The data source can also be signed and verified by using this key scheme. Referring once again to the postal analogy: To show Alice that it was indeed Bob who sent her the message, and not Eve for example, he can send an extra message along with the original message. This extra message is locked with Bob's key that he shares with no one (i.e. his private key). However, Bob has sent out his public key to everyone who wants it. These keys can *only* be used to unlock the messages locked with Bob's own private key. Therefore, if Alice, who received one of Bob's keys, can unlock this extra message with that key, she knows that as long as Bob has not given anyone his private key, it can only be his message. The reverse is also true if Alice wants to prove to Bob that it was indeed she who sent him a message.

Appendix F

Server Configuration

F.1 Elastic Compute Cloud

The AWS EC2 server provides the platform on which the Apache server runs. The EC2 server instance was configured to run Ubuntu 12.10, 'Quantal Quetzal'. This was done because most of the server development was done on Ubuntu 12.10, and the server code will therefore require minimal adaptation to be able to run on the EC2 server.

After the server instance was created, the following packages and programs were installed for the server to be able to run properly:

- Apache2: Installs the Apache server framework.
- libapache2-mod-wsgi: An Apache module that allows Apache to work with Python Web Server Gateway Interface (WSGI) scripts.
- python-pip: Allows Ubuntu to install Django from the Python Package Index (PyPI) [Python Package Index (2013)].
- Django: Installs the all the Django packages that will be used by the server.

Because the server's database uses SQLite3, which is part of the standard Ubuntu 12.10 distribution, no external database programs were needed to be installed.

After this was completed, the server is fully capable of serving Django web pages.

F.2 Apache Configuration

The Apache server framework provides the foundation on which the Django server runs. It had to be configured to be able to run the Python scripts that Django

contains. To do this, the steps described in Nick Polet's blog post was followed [Polet, Nick (2013)]. It describes in detail how to configure Apache to serve a Django website.

For Apache to be able to serve Django sites, it had to be configured to run the Web Server Gateway Interface (wsgi.py) script located within the Django server folder. This was done by adding the following code to the Apache's httpd.conf configuration file:

```
WSGIScriptAlias / /home/ubuntu/srv/server_site/server_site/wsgi.py
WSGIPythonPath /home/ubuntu/srv/server_site

<Directory /home/ubuntu/srv/server_site/server_site>
<Files wsgi.py>
Order deny,allow
Allow from all
</Files>
</Directory>
```

The following line was also needed to be added to Apache's apache2.conf file:

```
Include httpd.conf
```

Appendix G

NFC Chip Configuration and libnfc Setup

To make the Raspberry Pi serially communicate with the NF Chip, the 'SEL1' pads were shorted (see Fig. G.1). With this done, the chip can serially communicate with the Raspberry Pi's UART interface.

The connections between the Pi and the NFC controller are given in Table G.1.

G.1 libnfc Setup on the Raspberry Pi

Before libnfc could be built and configured, a communication line between the NFC controller and the Pi needed to be opened. To do this, the Pi's UART needed to be freed up. By default, the Raspberry Pi uses its UART to serially write out its

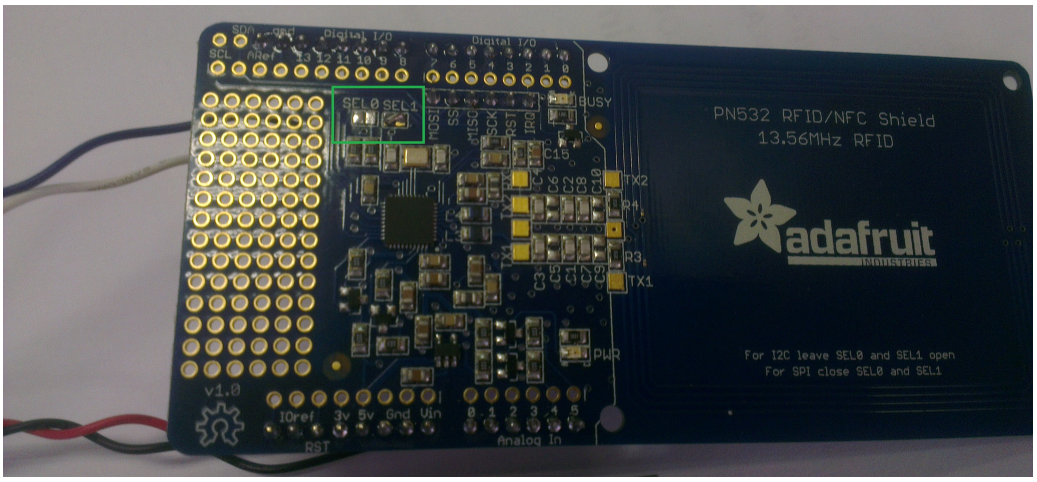

Figure G.1: The location of the SEL1 pads (highlighted in green).

Table G.1: Connections between the Raspberry Pi and the NFC Controller chip.

NCF Controller Pin	**Raspberry Pi Pin**
5V pin	5V (pin 4)
Ground pin	Ground pin (pin 6)
SS	UART0 TXD (pin 8)
MOSI	UART0 RXD (pin 10)

booting information. Therefore, to allow the Raspberry Pi to communicate with the NFC controller via its UART0 interface, it was necessary to modify some of its configuration files. To do this, the Adafruit tutorial was followed [Townsend, Kevin (2012)].

The file '/boot/cmdline.txt' and '/etc/inittab' had to be edited to contain the following lines of code:

cmdline.txt

```
dwc_otg.lpm_enable=0 console=tty1
```

inittab

```
#Spawn a getty on Raspberry Pi serial line
#T0:23:respawn:/sbin/getty -L ttyAMA0 115200 vt100
```

After this was done, the libnfc package could be configured and installed to work with the Pi's UART interface.

Thereafter, the Pi was ready to receive NFC messages from the NFC shield.

Appendix H

ECSA Outcomes Self Evaluation

Outcome	Location	Description
1. Problem Solving		
1.1 Problem identification	Section 1.1	The problem is formulated and explained here.
1.2 Set solution criteria	Section 1.3 Section 1.4	The project goal and system objectives are explained here.
1.3 Identify solutions	Section 1.2 Chapter 2	Existing solutions, such as USSD and NFC are discussed here.
1.4 Solve problem	Chapter 4 Chapter 5	The system design is discussed in these chapters.
2. Application of scientific engineering knowledge		
2.1 Use fundemental physics used to solve problems	Section 5.1 Section 5.5	Basic electro-technicques and mathmatical laws are used here to design a relay switch circuit and a vending machine coil.
2.2 Integrate different systems		This project required a physical system (the vending machine unit and its hardware) to be integrated with software systems (the server and the vending machine control program). This required careful planning and execution of the plan.

3. Engineering Design		
3.1 Acquire background knowledge	Chapter 2	This chapter gives the background information on all the tools and concepts that were required to design a successful system.
3.2 Generate concepts	Section 1.2	Here different payment options are listed.
3.3 System design	Chapter 3	This chapter discusses the complete system design. Project design choices are made and motivated here and unfamiliar concepts are explained.
3.4 Detail Design	Chapter 4 Chapter 5	These chapters discuss the detail design of the software and hardware components of the project. This includes mathematical motivations for the design choices made. All hardware and software configuration is also discussed.
3.5 Design evaluation	Chapter 6	This chapter discusses the perfomance of the completed system. Evaluations of each of the payment methods are made and discussed. User tests are also performed and discussed.
3.6 Plan and manage project	Appendix C	Here a Gannt chart is given of the project planning. This plan was followed throughout this report.
3.7 Assess impacts of the design	Appendix B	Here is a socio-economic report that discusses the potential economic impact that this system may have.
3.8 Use CAD as design tool	Section 5.3 Appendix A	Designed and manufactured a working vending machine unit using CAD software

4. Investigations, experiments and data analysis		
4.1 Plan and conduct tests	Chapter 6	
4.2 Select software	Section 6.1	The Dstat load monitoring software was identified, tested and selected for system tests.
4.3 Analyse data	Sections 6.1 - 6.10	For every test conducted, the data was analysed using MS Excel and presented on a reader-friendly graph.
4.4 Draw conclusions based on data	Chapter 6	A conclusion to each test was given and motivated.
4.5 Communicate purpose, process and outcomes	Chapter 6	The purose of each test, the steps taken during the tests and the results of each test was discussed and motivated in this chapter.
5. Engineering methods, skills and tools, including IT		
5.1 Select appropriate programming language	Section 4.1 Section 4.2	Python was selected to create a web server, the vending machine's GUI and to control the vending machine.
5.2 Successfully create computer and cellphone applications and programs	Chapter 4	This chapter discusses a web server, a vending machine program and an Android app thatw as created for this project.
5.3 Correctly configure and use hardware	Section 5.2 Section 5.4	The NFC Chip and webcam had to be configured to be compatible with the Raspberry Pi.
5.4 Successfully design system using engineering design principles		

5.5 Display knowledge of manufacturing methodologies and constraints	Appendix A	A manufacturable vending machine shell was designed and manufactured. Bending and, laser-cutting and welding were used in this process.
5.6 Display competency in processing data	Chapter 6	MS Excel software was used to process the test data and display the results on a graph.
6. Proffesional and technical communication		
6.1 Witten communication	This report Previous reports	Graphs and tables are used where appropriate. Correct language used thoughout the report. Acronyms and terms are properly defined and consistently used. Referencing to outside sources are also used throughout the report.
6.2 Oral communication		Not applicable in this report.
9. Independent learning ability		
9.1 Learn new programming langues		Skills in using Python and Java were required and acquired in this project. Android, Django and EC2 also required some familiarisation and learning.
9.2 Familiarise self with new Operating Systems.		This project required that one be able to work on Linux-based computers. It also required basix knowledge on how Android applications work.
9.3 Familiarise self with new hardware and sensors.		Knowledge on how a Raspberry Pi and the NFC protocols work was required.
9.4 Learn new concepts		An understanding of how public key encryption works was required. Additional security measures also had to be studied and implemented.

9.5 Critically evaluate self	Section 6.10 Chapter 7	Faults were found in the finished system and recommendations were given on how they could be fixed. Future work and improvements that can be made are also given and discussed.

List of References

Adafruit (2013 September). Raspberry Pi Camera Board. http://www.adafruit.com/products/1367.

Adafruit Industries (2012 November). Adafruit NFC Controller Shield. http://www.adafruit.com/products/789.

Amazon Web Services (2013 October). Amazon Web Services Home. http://aws.amazon.com/.

Apache Software Foundation (2013 October*a*). Apache. http://httpd.apache.org.

Apache Software Foundation (2013 October*b*). Apache Platforms. http://bit.ly/19Sk6BD.

Arduino (2013 September). *Arduino Uno*. http://bit.ly/19HAvVP.

Biham, Eli (1994). New types of cryptanalytic attacks using related keys. *Journal of Cryptology*, vol. 7, no. 4, pp. 229–246.

Chandler, Nathan (2012 March). What's the difference between RFID and NFC. *How Stuff Works*. http://bit.ly/1eIXfvW.

Django Software Foundation (2013 September*a*). List of Django sites. http://www.djangosites.org/.

Django Software Foundation (2013 September*b*). Why does Django exist? http://bit.ly/16sjRi3.

ElGamal, Taher (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. *Information Theory, IEEE Transactions on*, vol. 31, no. 4, pp. 469–472.

eLinux (2013 September). Raspberry Pi Specifications. http://elinux.org/RPi_Hardware.

elinux.org (2013 September). Low Level Peripherals. http://bit.ly/184ue8S.

Etherington, Darrel (2013 August). Android Nears 80% Market Share In Global Smartphone Shipments, As iOS And BlackBerry Share Slides, Per IDC. *Tech Crunch*. http://tcrn.ch/15uSuxy.

Faulhaber (2013 September*a*). 12V DC Motor Spec Sheet. http://bit.ly/18qSsuy.

Faulhaber (2013 September*b*). Planetary Gearhead Spec Sheet. http://bit.ly/18qSsuy.

Google (2011 September). StickyNotes. http://nfc.android.com.

Google Android Team (2009 November). Android Gingerbread New Features. http://bit.ly/15jP3Nh.

Google Play (2013 September). Barcode Scanner. http://bit.ly/15vE2uD.

Humphrey, David B (2004). Replacement of cash by cards in US consumer payments. *Journal of Economics and Business*, vol. 56, no. 3, pp. 211–225.

Jack, William and Suri, Tavneet (2011). Mobile money: the economics of M-PESA. Tech. Rep., National Bureau of Economic Research.

Leon, Jeffrey S. (2008 March). The ElGamal Public Key Encryption Algorithm. http://bit.ly/16KMuB8.

Libnfc Team (2013 September*a*). Compatible Hardware. http://bit.ly/1fpsqh5.

Libnfc Team (2013 September*b*). Libnfc. http://bit.ly/19BgRvX.

Loop, Lincoln (2013 September). qrcode. http://bit.ly/H5oHFD.

Mantech (2013 September). *NT72 Spec Sheet*. http://bit.ly/1eIXQOb.

MicroMo (2008). DC Motor Calculations. *MicroMo*. http://bit.ly/1aPYnZo.

Netcraft (2013 June). June 2013 Web Server Survey. http://bit.ly/16KMyRn.

NFC Forum (2013 September). NFC Forum. http://bit.ly/1hY82iY.

NFC World (2013 October). List of NFC-Capable Cellpphones. http://bit.ly/GRYElg.

ov511 (2013 September). ov511 Known Cameras. http://bit.ly/1fpt4ew.

Pasquali, Valentina and Bedell, Denise (2013 January). Payments Volumes Worldwide. http://bit.ly/14JRPeu.

Polet, Nick (2013 April). Deploying Django on AMazon EC2 server. *Nick Polet's Blog*. http://nickpolet.com/blog/1/.

Public-Key Encryption (2005). Public Key Encryption. *Virtual Private Network (VPM)*.

PyCrypto Team (2013 September). PyCrypto. `https://launchpad.net/pycrypto`.

Python Package Index (2013 September). Python Package Index. `https://pypi.python.org/pypi`.

Rijmenants, Dirk (2011). THE COMPLETE GUIDE TO SECURE COMMUNICATIONS WITH THE ONE TIME PAD CIPHER. *Cipher machines and cryptology*, vol. 5.

Soon, Tan Jin (2008). QR code. *Synthesis Journal*, pp. 59–78.

ST Electronics (2013 September). *2N2222 Transistor Spec Sheet*. ST Electronics. `http://bit.ly/1h3D2xA`.

Stripp, Alan (1999 September). How the Enigma Works. *NOVA*. `http://to.pbs.org/16Ptpla`.

tibi (2011 12). RSA String Encryption, security. *Virtual Private Network (VPM)*.

Tiedemann, Stephen (2013 September). Nfcpy. `https://launchpad.net/nfcpy`.

Townsend, Kevin (2012 November). Adafruit NFC/RFID on Raspberry Pi. `http://bit.ly/15VpO4A`.

Tyson, Jeff (2001 April). How Encryption Works. *How Stuff Works*. `http://bit.ly/1cw3Jej`.

Walton, C.A. (1983 05). Portable radio frequency emitting identifier. Available at: `"http://bit.ly/1b7Zxkk"`

Weinstein, Lauren Sager (2009 September). Tfl's contactless ticketing: Oyster and beyond. *Transport for London, London, UK*.

Wieers, Dag (2013 October). DStat. `http://bit.ly/180InAA`.

Wollop, Harry (2010 April). Cash to be used in fewer than half transactions by 2015. *The Telegraph*. `http://bit.ly/1bdUiTs`.

WX Python Team (2013 September). WX Python Home Site. `http://www.wxpython.org/`.

ZXing Team (2013 October). ZXing Home. `https://code.google.com/p/zxing/`.