Stellenbosch University

Electrical & Electronical Engineering Department

Design and Implementation of an NFC Mobile Payment and Token Printing device

Student
Izak Jean Britz

Study Leader
Prof. G-J van Rooyen

4 November 2013

Report submitted in partial fulfilment of the requirements of the module Project (E) 448 for the degree Baccallaureus in Engineering in the Department of Electrical and Electronic Engineering at the University of Stellenbosch

Declaration

I, the undersigned, hereby declare that the work contained in this report is my own original work unless indicated otherwise.

Signature	*Date*
....	......../......../.............

Acknowledgements

I would like to say thanks for Prof. G-J van Rooyen for guiding me during the project and providing a comfortable workspace in the MIH Media Lab.

I also want to say thanks for MTN and Trintel Telecomms who supported this project with their financial assistance.

Lastly I want to say thanks to my parents for their support throughout the semester when I needed it most.

Summary

Mobile payment and ticketing solutions have recently received considerable attention due to its potential for replacing the traditional wallet with your mobile device. Consumers today carries so many important items in their wallets – like their credit cards or bus pass – that it has become a huge risk for them if it is lost or stolen. The solution to this problem is to replace your wallet with your mobile device. This is now possible using Near-field Communication (NFC) technology. In this project, we will develop a low–cost NFC payment and claiming system that makes use of commercially off–the–shelf (COTS) components. By taking advantage of open–source software packages and add-on packages, the users of this system can access secure web interface for managing their transactions, invoices and claims on the internet. Using NFC–complaint mobile devices it will create a fast and hassle-free method of making payments or to retrieve claims by hand.

Opsomming

Die oplossings van mobiele betaling metodes en die mobiele uitreik van kaartjies het die afgelope paar jaar groot aandag getrek oor die potensiaal wat dit het om jou beursie te vervang met jou selfoon. Verbruiker hou al soveel waardevolle items in hul beursie vandag – soos hul kreditkaart of buskaartjie – dat dit 'n gevaar inhou vir die verbruiker om hul te verloor of om beroof te word van jou beursie. Die oplossing tot hierdie probleem is om die beursie te vervang met jou selfoon. Dit is nou moontlin met Near-field Communication (NFC) tegnologie. In hierdie project, ons gaan 'n lae-koste NFC betaling system en 'n system vir die uitreik van kaartjies wat gebruik maak van kommerisile beskikbare komponente. Deur voordeel te trek uit oop-sagteware pakkette en bykomende pakkete, gebruikers van die sisteem kan beveiligde toegang bekom deur die internet om hul transaksies, fakture en hul koeponne to bestuur. Deur gebruik te maak van selfone wat NFC-funkies ondersteun kan betaling en die uitreik van in 'n vinnige en eenvoudige metode geskied.

Contents

List of Figures

List of Tables

Nomenclature

3G	Global System for Mobile communications
3G	Third generation of mobile telecommunications technology
ADC	Analog–to–Digital Converter
API	Application Programming Interface
AVR	A modified Harvard architecture 8-bit RISC single chip microcontroller
COTS	Commercial Off–The–Shelf
CRUD	Create, Read, Update and Delete
CSV	Comma–Separated Values
DC	Direct Current
DRY	Don't Repeat Yourself
EER	Enhanced Entity–Relationship
GPRS	General Packet Radio Service
GUI	Graphical User Interface
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
LCD	Liquid–Crystal Display
LED	Light–Emitting diode
LLCP	Logical Link Control Protocol
MIME	Multi-purpose Internet Mail Extensions
MVC	Model–View–Controller

NDEF	NFC Data Exchange Format
NFC	Near–Field Communication
PCB	Printed Circuit Board
REST	REpresentational State Transfer
RFID	Radio–Frequency IDentification
SDK	Source Development Kit
SNEP	Simple NDEF Exchange Protocol
SPI	Serial Peripheral Interface bus
SRAM	Static Random–Access Memory
UART	Universal Asynchronous Receiver and Transmitter
URI	Uniform Resource Identifier
XML	Extensible Mark-up Language

Chapter 1

Introduction

Mobile payment and ticketing solutions have recently received considerable attention due to its potential for changing the manner of how business can be done. Consumers today carries so many cards in their wallets that it has become a huge risk for them if it is lost or stolen. Commercial banks have already developed secure ways of doing online bank transactions using the internet, but there is still a need for a secure way of doing payments by hand. The next step is to seek for a solution using your cellphone, because people nowadays consider their cellphones just as important as their wallets. The answer to this problem led to the development of Near–field Communication (NFC).
In this project, we propose a low–cost NFC payment and ticketing system that makes use of commercially off–the–shelf (COTS) components. These components have the advantage of being affordable and easy to be assembled. The support frameworks for open software and hardware is currently so extensive that it is possible to create the proposed system using it in conjunction with open-source software and NFC technology.
MasterCard PayPass has already developed a working mobile payment system. It uses Near–Field Communication (NFC) to "make secure payments fast and convenient by simply tapping the phone on any PayPass-enabled terminal at checkout" [1].

1.1 Project description

Mobile payment and ticketing solutions have recently received considerable attention due to its potential for replacing the traditional wallet with your mobile device. Consumers today carries so many important items in their wallets – like their credit cards or bus pass – that it has become a huge risk for them if it is lost or stolen. The solution to this problem is to replace your wallet with your mobile device. This is now possible using Near-field Communication (NFC) technology. In this project, we will develop a low–cost NFC payment and claiming system that makes use of commercially off–the–shelf (COTS) components. By taking advantage of open–source software packages and add-on packages, the users of this system can access secure web interface for managing their transactions, invoices and claims on the internet. Using NFC–complaint mobile devices it will create a fast and hassle-free method of making payments or to retrieve claims by hand.

1.2 Objectives and limitations

1.2.1 Objectives

There are three main objectives for this project:

1. Building a controller module which includes:
   - An NFC–interface that is compatible with an NFC–complaint mobile device to transfer data wirelessly when they are held closely together.
   - An 3G cellular module for making remote request to a server and then to receive the given response from the server.
   - An thermal printer module for printing receipts as proof of payment or printing a ticket or voucher when the mobile device user is claiming it.
   - An LCD screen that indicates the operating state of the controller module.
   - An rechargeable battery with adequate capacity to supply power for one hour when the controller module operates in remote areas.
2. Developing a suitable network model that facilitates user authentication and process transactions when requests are being sent from either the NFC–compliant mobile device or the controller module.
3. Developing an mobile device application, which is able to do the following:
   - Manages user credentials (i.e. username and password) in a secure manner.
   - Sends data messages to the controller module when the NFC–compliant mobile device interacts with it.

1.2.2 Limitations for the scope of the project

In practice this system would work with a cash register to initiate the payment process, but this feature is not included in this project. Only an user input of an integer will be used to initiate a payment process.
Payments will not involve actual money to perform a payment operation, only integers will be used to represent a value used for creating transactions.
The project does deal with electronic tickets and vouchers represented on a web server, but the process of buying a ticket or a voucher is not featured. Examples of tickets and vouchers will be created on the web server to use when testing the system to claim these tickets or vouchers.

1.3 Use cases

This project is aimed at consumers, therefore it needs to be located where consumers go and be able to operate when they need it. Two scenarios have been identified when the functions of this project can come into good use:

Two use cases will be investigated. They will be referred to as activities in the project. They are described as follows:

1. The system acts as an access point when the specific activity involves a ticket being purchased beforehand or a voucher that has been bought by a third party. In both cases the ticket or voucher could be claimed by printing it physically.

2. The system acts as an point-of-sale (POS) activity. In this event the controller module acts as a payment system when a purchase has been made with a pseudo-teller. The purchase is being paid when the user with the NFC-enabled Android device makes contact with the controller module's NFC-interface. When the user credentials is satisfactory and there is efficient funds in the user's account then the transaction will be successful by printing a receipt.

Chapter 2

Background information

This chapter will start by briefly discussing the concepts that were needed to design a suitable system. Certain protocols were required to be studied to shed some perspective on their behaviour when it is used. Some standards had to be study to choices and the technologies and protocols that were necessary for developing this project. Some light will also be shed on the COTS hardware components and the available open-source software that were required for this project.

2.1 Concepts, standards and protocols

2.1.1 Representational state transfer (REST)

Representational state transfer (REST) "is an architectural style for distributed hypermedia systems" [2]. An architectural style is a set of architectural constraints. For example a protocol that uses REST as a style is Hypertext Transfer Protocol (HTTP). These required constraints for REST are listed in Table 2.1 with a quick explination how this constraint is applicable to the web.

REST architecture is necessary to understand for developing applications on the internet,

Table 2.1: List of the REST architectural constraints.

Constraint	**Why is it applicable for the web?**
Client-server model	"Separation of concerns is the principle behind the client-server" [2], therefore it is necessary
Stateless	To enforce separation of concerns a "request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server." [2]
Cacheble	It has the potential for "improving efficiency, scalability, and user-perceived performance by reducing the average latency of a series of interactions." [2]
Uniform Interface	It decouples the user interface from the actual data storage

because overall web architecture is based on the constraints of REST.

2.1.2 Hypertext transfer protocol (HTTP)

Hypertext transfer protocol is a “request-response protocol in the client-server computing model” [3]. It is developed according to the REST constraints.
HTTP has the ability to manipulate resources on the web, by locating the specific resource using an Uniform Resource Identifier (URI). Then it can make a request using a predefined HTTP header format including one of the methods listed in Table 2.2. This effectively also execute the corresponding Create-Retrieve-Update-Delete (CRUD) operation on the server’s database where the resource’s data are being stored. HTTP responses are rendered in a stateless manner on the server-side, which is send back to the client. HTTP responses always has header with a response code included and a Multi-purpose Internet Mail Extensions (MIME) formatted body. Most common MIME-format associated with HTTP is Hypertext Markup Language (HTML). HTTP methods, irrespective of their function, does not have the same behavioural attributes. You get the unsafe, safe, idempotent and cacheble attributes. The safe attribute involves in operations where data is only data or the HTTP header is being retrieved and does not alter the state of the server. Methods with the unsafe attribute “ are intended for actions that may cause side effects either on the server, or external side effects such as financial transactions or transmission of email” [3]. In the case of the POST method it “is not necessarily idempotent, and therefore sending an identical POST request multiple times may further affect state or cause further side effects (such as financial transactions)” [3]. Methods has the idempotent attribute means that if multiple identical request are made the effect on the state of the server must be the same as a single request.

Table 2.2: Mapping of HTTP methods to basic database operations.

HTTP	**Type of method**	**Database operation**
POST	Unsafe	Create
GET	Safe, Idempotent, Cacheable	Retrieve
PUT	Idempotent	Update
DELETE	Idempotent	Delete

2.1.3 Near–field communication (NFC)

“Near Field Communication (NFC) is a standards-based, short-range (a few centimeters) wireless connectivity technology that enables simple and safe two-way interactions between electronic devices, allowing consumers to perform contactless transactions, access digital content, and connect electronic devices with a single touch.” [4]. Any NFC-compliant device (also known as an NFC Forum device) can operate in three different modes namely: Reader/writer mode; card emulation mode and peer–to–peer mode. We are interested in using

the peer–to–peer mode for this project.
Peer–to–peer mode basically involves two NFC–compliant devices. One device is the initiator (the device that starts to transfer data) and the other one is the target (the device that listens passively to any data being transferred). Information between the devices are exchanged according to the NFC Data Exchange Format (NDEF). NDEF allows to exchange contacts, bluetooth pairing information or any Multi-purpose Internet Mail Extensions (MIME) data between the two NFC-compliant devices. [5] For Peer–to–peer mode to be set–up to communicate in a bi-directional fashion two layers needs to be implemented, namely Logical Link Control Protocol (LLCP) and the Simple NFC Exchange Protocol (SNEP).

NFC data exchange format (NDEF)

“NDEF is a lightweight, binary message format that can be used to encapsulate one or more application-defined payloads of arbitrary type and size into a single message construct. Each payload is described by a type, a length, and an optional identifier” [6] NDEF messages are used for transmitting data via NFC using Peer–to–Peer mode.

2.1.4 Basic authentication

“Basic authentication (BA) implementation is the simplest technique for enforcing access controls to web resources” [7]. This authentication method provides minimal security, because it sends the user’s credentials in a binary-encoded format known as BASE64 [8]. which can easily be decoded with algorithms available to any internet user. This protocol is generally used for testing purposes, because it is very easy to implement.

2.2 Hardware

2.2.1 Arduino

Arduino is an open–source Printed Circuit Board (PCB) development board. The Arduino products usually consists of an A modified Harvard architecture 8-bit RISC single chip micro-controller or better known as an AVR. The Arduino Uno comes packaged with a programmable AVR with General purpose Input-Output (GPIO) ports already wired to stackable headers on the edges of the Arduino Uno’s PCB. The Arduino Uno supports UART, SPI and I^2C communication protocols and has a built-in USB interface (using the ATmega8u2 chip) to program the AVR using the Arduino IDE. This board can either be powered by the USB or by using a power jack. Some attributes of the Uno are shown in Table 2.3:

2.2.2 Arduino shields

”Shields are boards that can be plugged in top of the Arduino PCB extending its capabilities.”[9] Most of the Arduino products has a standardised pin configuration, which enables other shields to be stacked on the Arduino. Shields are then able to communicate with the Arduino via SPI, UART, I2C or GPIO when they are stacked.

Table 2.3: Arduino Uno Rev 3 attributes.

Operating voltage	5V
Digital I/O pins	14
Analog input pins	6 (can be used as digital inputs)
Flash memory	32 kB
SRAM	2 kB
EEPROM	1 kB
Clock speed	16 MHz

Itead 3G shield

This shield has a Simcom SIM5216 3G module populated on. Only the UART–interface is available for the Arduino Uno to communicate with. The 3G module uses 3.3V to operate, but it has a voltage translation circuit placed between the 3G module and the AVR to make it compatible with the AVR's 5V GPIO pins. There is also a power pin connected to D8 pin of the Arduino Uno. This allows the AVR to turn the power on of the 3G module by pulling pin D8 low and then high again. The shield comes with 4 LED's, which effectively has two functions described in Table 2.4:

Table 2.4: Itead 3G Shield LED status information.

Power	3 leftmost LED's indicate the various voltages available	
	LED state	***3G module state***
	Always on	Searching network / Call connect
Status	200ms on, 200ms off	Data Transmit
	800ms on, 800ms off	Registered to network / Idle
	Off	Power off / Sleep

Itead 1602 LCD with keypad shield

This shield provides an LCD display with a back light include. The keypad populated on the shield uses a resistor ladder to distinguish which button has been pressed.

Adafruit PN532 NFC Controller Shield

This shield has an PN532 chip populated on it that is capable of using the UART, SPI or I^2C protocol just by specifying two select pins populated on the shield.

2.2.3 Thermal printer

The printer module comes with a command set that can be used to send commands for printing characters on paper. Only an UART-interface is available and at a baud rate of 19200 bps. This module also needs its own power supply, because it uses more that 1 A to print characters on the paper.

2.2.4 Android device

We used the Asus Nexus 7 with Android version 4.3 installed on it. It has very useful capabilities like a USB debugging interface that makes it easier to detect bugs in the code.

2.3 Software

2.3.1 Django

Django is an open source web application framework based on the Python language. It follows the MVC architecture style. It states that "It lets you build high–performing, elegant Web applications quickly. Django focuses on automating as much as possible and adhering to the DRY principle."[10] Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes re-usability and "pluggability" of components, rapid development, and the DRY principle. Python is used throughout, even for settings, files, and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

2.3.2 Android SDK

The Android Source Development Kit (SDK) provides all the necessary functions required to develop mobile applications. The SDK can be integrated the Eclipse Integrated Development Environment (IDE) which provides a very rich environment for developing applications on Android Also the SDK provides a built-in feature of creating virtual devices that emulates of an Android device. This allows the application developer to test their code on different Android devices with different Android versions to test for compatibility issues.

2.3.3 Arduino processing

Arduino has simplified the software coding of their products by developing libraries based on C/C++ to make it easier for their users to develop projects. For instance they developed a library (called SoftwareSerial) to implement UART communication via software. It has two main functions set-up and loop. The set-up function is where all the pin modes and voltage levels are being set and where typically the hardware and software serials baud rate are set. The loop function can be considered as an infinite while loop, just like a state-machine. The Arduino IDE does not have an extensive debugging facility, but most library has incorporated it by printing certain statements to the hardware serial.

2.3.4 AT commands

AT commands are a ASCII-based command set. Typically used in telecommunication to perform certain operations on a cellular module. When commands are executed from a computer terminal the response is usually in ASCII format e.g. "OK". Most of the commands are there to change internal registers on the 3G module, but there are commands that does execute external operations such as making a phone call, sending an SMS or sending a HTTP request to a remote server. Those commands that change internal register values usually have four types of commands, which are distinguishable by a specific syntax. We will demonstrate the syntax using a common network registration command in Table 2.5:

Table 2.5: AT+CREG command example usage.

Type	**Command**	**Response**	**Comments**
Test	AT+CREG=?	+CREG: (0-2)	It shows the possible values that the register accepts
Read	AT+CREG?	+CREG: 0,1	Queries the network registration status.
Write	AT+CREG=1	OK	Enable unsolicited result code +CREG
Execution	AT+CREG	OK	Set default value (i.e. AT+CREG=0)

Chapter 3

System Design

This chapter analyses how the problem statement is solved and shows how a realisable data flow system can be created to make it work in practice. General terminology is given to describe the different data objects that the system has to deal with and also numerous functional requirements that the system has to obey. The complete data flow system consists of a controller module, a web server and a mobile device. The controller module delegates with web requests and processes data by using a state machine. While both the web server and mobile device data flows implement the MVC design pattern. The symmetrical implementation between web server and mobile device will be discuss together, because both of them implement the REST API. Lastly we discuss two use cases of the system.

3.1 General terminology

General terminology is given to explain custom entities used in the system. There are two types of entities namely objects and users. Due to the nature of system having a payment facility, currency are only integers. It neccessary to distinguish between objects and users, because the user plays a dynamic role in the system, while objects are more passive. Objects cannot change unless some user interaction is taking place. For the use cases the controller module there are two modes to be defined for each case.
There are four different objects in this project:

- Claim – This object resembles a ticket or a voucher that can be retrieved via an NFC transaction between the consumer and the controller device.
- NFC device – This object stores the unique identity code of an NFC-compliant device to keep track of every device used in the system. This object is put in place to allow users to use any NFC-compliant device they want, but to prevent two or more NFC-compliant devices to use the same user account.
- Transaction – This object creates a record for each transaction being processed on the system.
- Invoice – This object can only be created by a vendor. Its function is to create check condition to check whether the consumer has paid enough to complete the payment procedure.

- Location – This object stores the neccessary information to locate information of a mobile operating tower using OpenCellID API [11].

The system distinguishes between three different types of users:

- Consumer – This user wants to be able to purchase items from a vendor and/or claim a ticket or voucher by printing it at an access-point. In the system this user would operate the mobile device.
- Vendor – This user manages the controller module and facilitates invoices and transactions when a payment is made via NFC. In the system this user makes the connection between the consumer and the web server.
- Superuser – This user is created for debugging purposes and has access to every aspect of the system. The consumer and the vendor can contact him if there is any problem on the system. This user is then responsible to rectify the problem on the system.

Modes of operation for the controller module:

- Access–point - In this mode the controller module acts as a ticketing system.
- Point–of–sale - In the mode the controller module acts as a payment system.

3.2 Functional specifications

By translating the objectives of the project into technical terms to show how it could be reached we generated a list of functional specifications:

3.2.1 Web server

- All users must be able to register and create an account by specifying a suitable username and password.
- The username must be used as a unique reference throughout the whole system.
- The web interface must be accessible for all authenticated users to edit their profile and/or reset password.
- The web interface must give all users the ability to list their invoices, devices and transactions.
- Only consumers must be able to list claims or delete them, while superusers must exclusively add and edit them.
- Consumers are not allowed to create, edit or delete invoices except for vendors and superusers
- Vendors can only create a transaction, but cannot edit or delete them. Only a superuser can.
- All users must only have one reference to a Location object, while many Locations can point to a specific user.
- All users must not be obligated to log their location by referencing a single Location object.

3.2.2 Mobile device

- Consumers must be able to create an entry of the device's unique identifier and store it in a Device-object
- Consumers must be able to log in onto the mobile application.
- Authorized consumers must then be able to view claims, invoices, devices and transactions in a suitable GUI.
- The mobile application must facilitate consumers in making mobile payments by interacting with the controller module via the NFC-interface.
- The mobile application must also facilitate consumers if they want to claim tickets or vouchers by interacting with the controller module via the NFC-interface.

- Consumers must only have one reference to a Location object, while many Locations can point to a specific user.
- Consumers must not be obligated to log their location by referencing a single Location object.

3.2.3 Controller module

- This module has to initialise the NFC-module and 3G-module to be able to operate.
- This module must implement a state machine by listening for any data transmitted via the NFC-interface and then to generate the appropriate HTTP-requests and handling HTTP-responses from and to the web server.
- The LCD must be used to indicate the operating state of the module.

3.3 Data flow

3.3.1 Overall

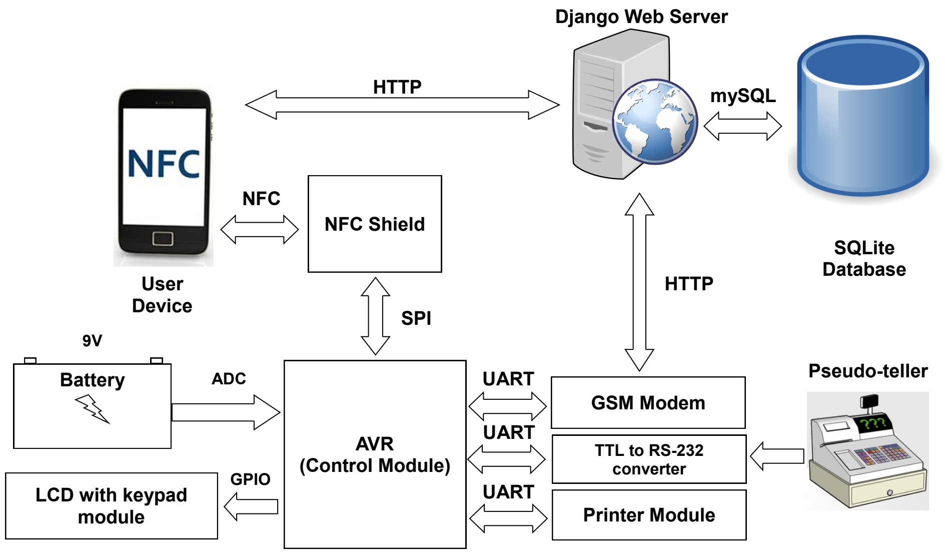


Figure 3.1: Overall system flow diagram.

Figure 3.1 can be divided into three sub-components. That is the mobile device, the controller module and the web server. The HTTP protocol is used for communication between all three sub-components, because it is a well establish and reliable protocol. The NFC technology is used between the mobile device and controller module to enhance a secure connection, because it is close-range and difficult for hackers to intercept. The LCD module is used to indicate the state and to provide feedback for the controller module. The printer module is used for printing tickets/vouchers that has been claimed and for printing receipts for payments.

3.3.2 Web server

The web browser follows the MVC design pattern. The web browser has the following features:

- Web browser interface for users to login and view their transactions, claims, invoices and their location if they have allowed it.
- REST API for the mobile device and controller module to interface with the web server and its database.

3.3.3 Mobile application

In our case the mobile device needs to have an user interface which in turn can interact with the web server's REST API. In [12]'s presentation its shows developers three design patterns to integrate RESTful web services with an Android application We used the first pattern called Service API. It also uses the built-in ContentProvider, which creates a mySQLite database on the Android device. When the mobile application interacts with the web server's REST API it creates equivalent tables on the Android application, which increase the consistency of the database between the web server and the Android application. For clarification:

- Content Provider – helps implementing a local dynamic caching scheme by creating a local SQLlite database on the Android device.
- Service Helper – handles requests from main application and process data in the background, while making asynchronous calls to RESTful services if it is neccessary.

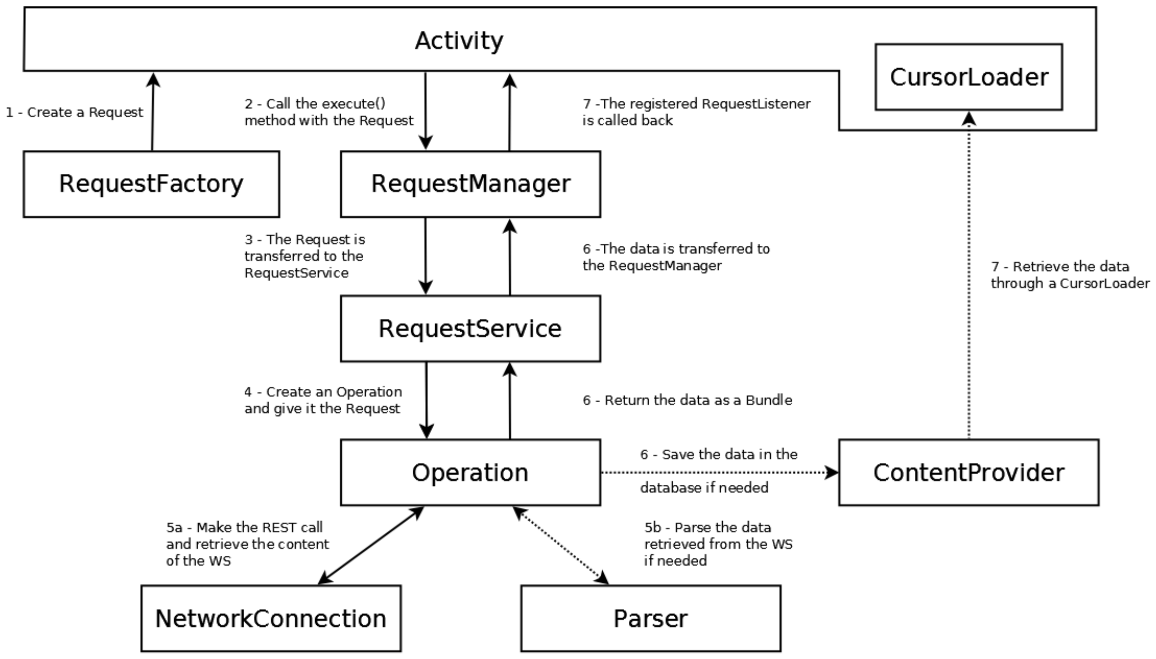

Activity
CursorLoader
1 - Create a Request
2 - Call the execute() method with the Request
7 -The registered RequestListener is called back
RequestFactory
RequestManager
3 - The Request is transferred to the RequestService
6 -The data is transferred to the RequestManager
7 - Retrieve the data through a CursorLoader
RequestService
4 - Create an Operation and give it the Request
6 - Return the data as a Bundle
Operation
6 - Save the data in the database if needed
ContentProvider
5a - Make the REST call and retrieve the content of the WS
5b - Parse the data retrieved from the WS if needed
NetworkConnection
Parser

Figure 3.2: Flow diagram for the implementation of the Service API.

In Figure 3.2 the diagram shows step–by–step of how the MVC design pattern corresponds with the Service API.
The Service API enforces the principle of separation of concerns. That means that the Activity cannot make a direct call to the ContentProvider or make a network request directly the the REST API. If the Activity wants to request something it has to pass it to the RequestManager first.
We used an third-party library to implement the Service API called DataDroid. We adapted the example code to be able to interact with our REST API. This external code is hosted on a remote repository listed in Appendix E.

3.3.4 Controller module

The controller module uses a state machine, because the Arduino Processing language is implying it already. In Figure 3.3 the conceptual state diagram shows how the control module should operate.

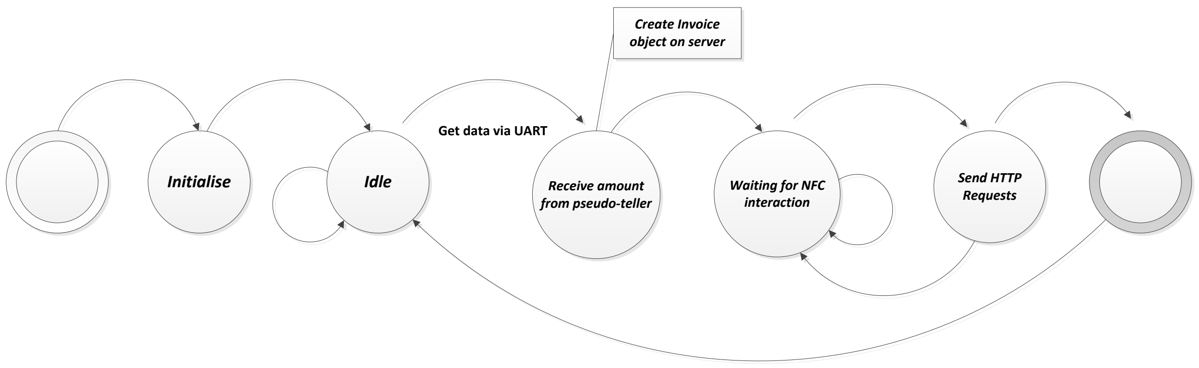

Create Invoice object on server
Initialise
Idle
Get data via UART
Receive amount from pseudo-teller
Waiting for NFC interaction
Send HTTP Requests

Figure 3.3: State diagram for the controller module.

3.4 Use cases

As mentioned in section 1.3 there are two use cases that will be investigated in the project. Figure 3.4 shows how the purchase activity will operate and how the communication would work between each of the sub-components.

In Figure 3.4 it is clear that the whole process consists out of two sub-sequential processes. First process would be the process coming from the vendor's side by specifying the amount the consumer has to pay and registering this amount as an invoice onto the system and the other one would be where the consumer makes the payments via the NFC-interface. The payments does not have to be from only one consumer, it could be from different users that has a consumer account on the system. This feature is making payments more intuitive. For example if you and a bunch of friends is paying to see a movie, each person can pay separately, thus minimizing the fuss with owing money to each other.

The process was design that all the crucial data processing (i.e. making bank transactions) would happen on the server, because both the consumer and vendor could not be trusted with such tasks and it would therefore compromise the integrity and transparency of the whole system.
In Figure 3.5 it shows how the ticketing activity will be initiated and executed.
In this diagram it is assumed that the controller module acts as the access-point, because this is the device which would print the valid ticket to be used when needed. Again the process of checking whether the ticket/voucher has been claimed is executed on the server. This is done so to compensate for scenarios where the consumer's device is out-of-sync with the server and then falsely shows certain tickets/vouchers which has been claimed as being not claimed.

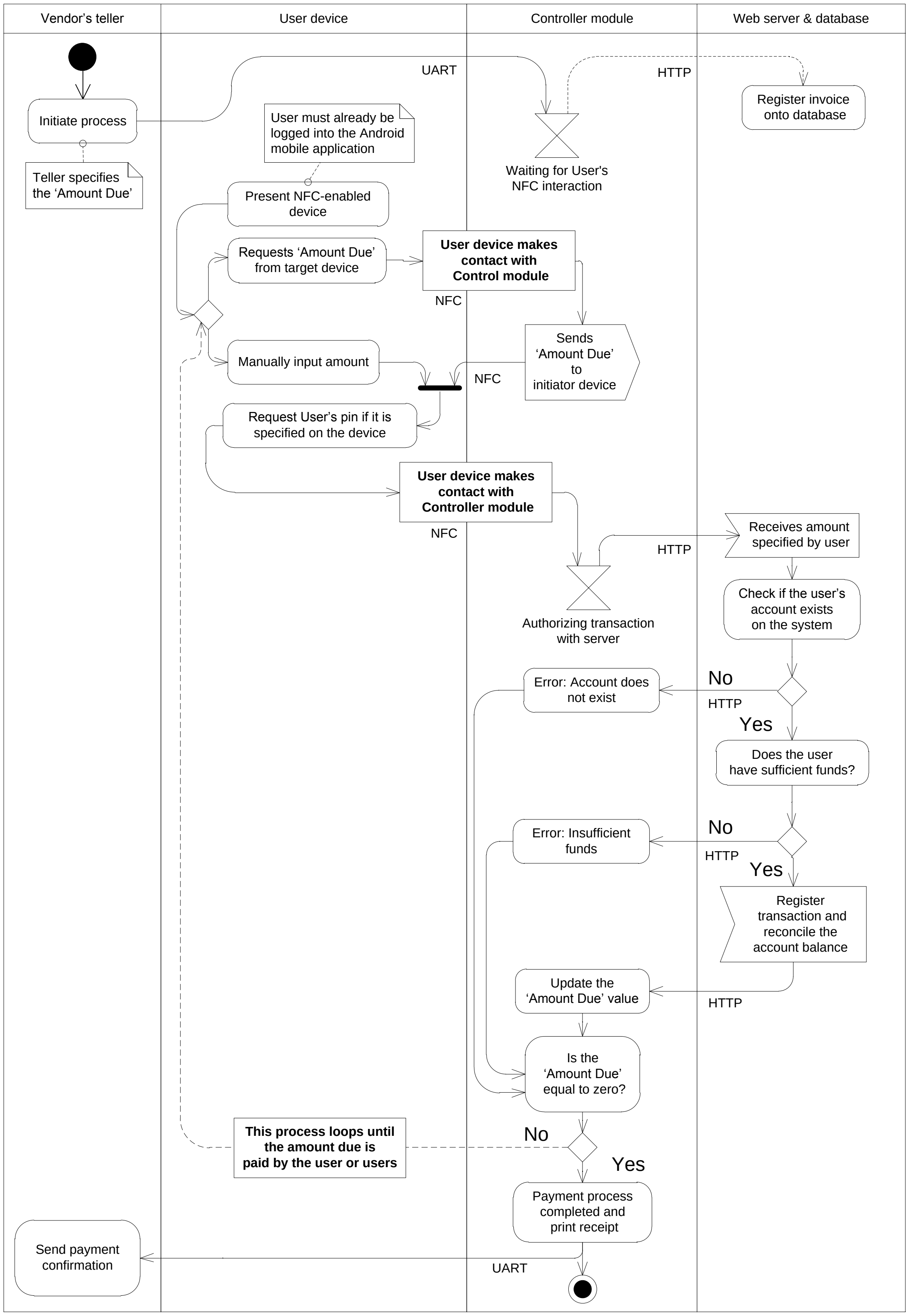

Vendor's teller
User device
Controller module
Web server & database
Initiate process
Teller specifies the 'Amount Due'
UART
HTTP
Register invoice onto database
User must already be logged into the Android mobile application
Waiting for User's NFC interaction
Present NFC-enabled device
Requests 'Amount Due' from target device
User device makes contact with Control module
NFC
Sends 'Amount Due' to initiator device
Manually input amount
NFC
Request User's pin if it is specified on the device
User device makes contact with Controller module
NFC
Receives amount specified by user
HTTP
Authorizing transaction with server
Check if the user's account exists on the system
Error: Account does not exist
No
HTTP
Yes
Does the user have sufficient funds?
Error: Insufficient funds
No
HTTP
Yes
Register transaction and reconcile the account balance
Update the 'Amount Due' value
HTTP
Is the 'Amount Due' equal to zero?
This process loops until the amount due is paid by the user or users
No
Yes
Payment process completed and print receipt
Send payment confirmation
UART

Figure 3.4: Activity diagram when purchasing an item.

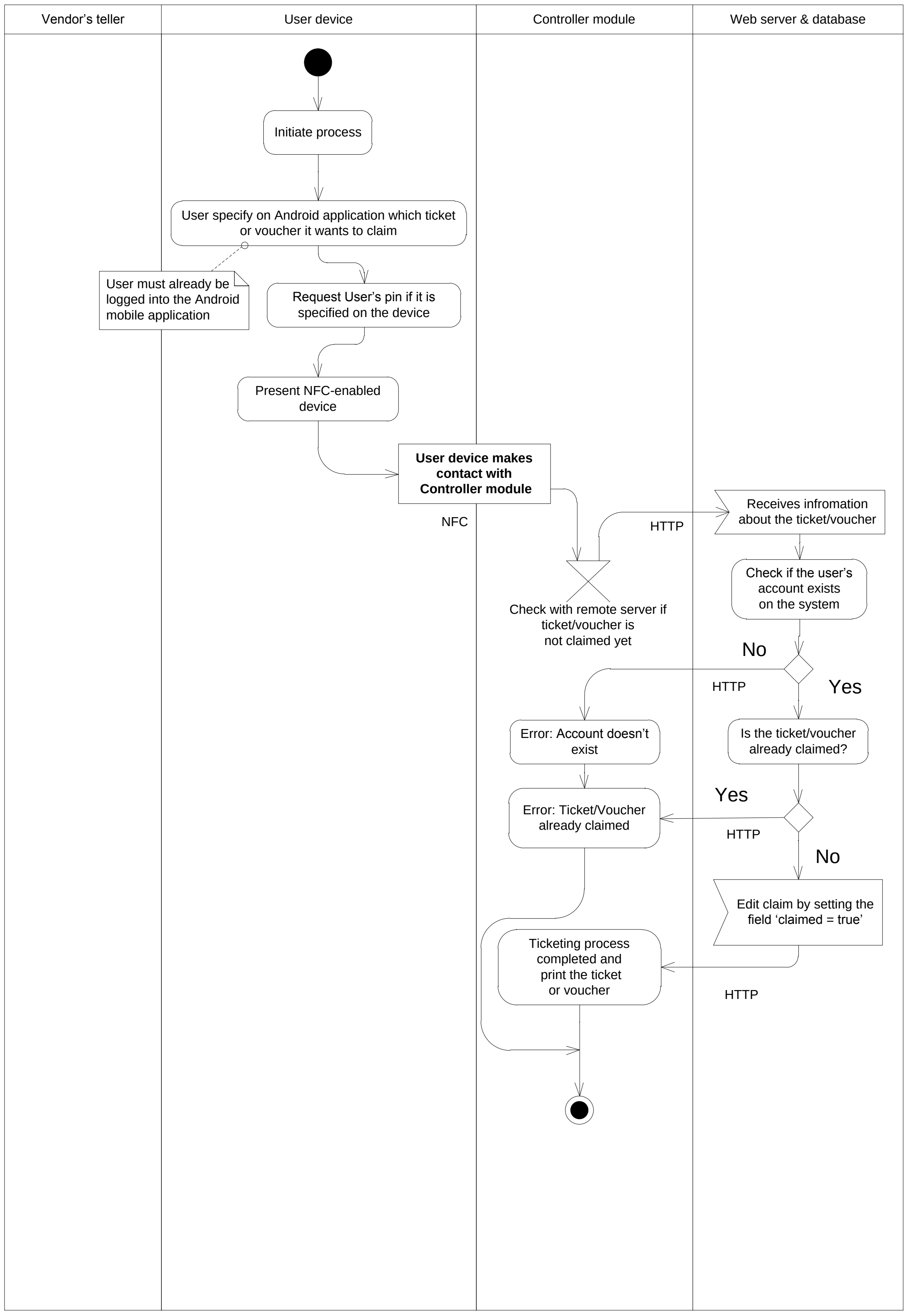

Vendor's teller
User device
Controller module
Web server & database
Initiate process
User specify on Android application which ticket or voucher it wants to claim
User must already be logged into the Android mobile application
Request User's pin if it is specified on the device
Present NFC-enabled device
User device makes contact with Controller module
NFC
HTTP
Receives infromation about the ticket/voucher
Check with remote server if ticket/voucher is not claimed yet
Check if the user's account exists on the system
No
HTTP
Yes
Error: Account doesn't exist
Is the ticket/voucher already claimed?
Yes
Error: Ticket/Voucher already claimed
HTTP
No
Edit claim by setting the field 'claimed = true'
Ticketing process completed and print the ticket or voucher
HTTP

Figure 3.5: Activity diagram when claiming a ticket or voucher at the access point.

Chapter 4

Detail Design

The detail design will be divided into two sections. Firstly the software design of the web server and mobile application will be discussed. The controller module software and hardware design together in the subsequent section. The majority of of choices made for the software design was influenced by the way the web server has implemented the MVC design pattern.

4.1 Software

4.1.1 Web server

The Django 1.5 web framework was chosen as the suitable software package to be used, because it followed the Don't Repeat Yourself (DRY) principle, which saved us time creating a web server from scratch. It also follows the MVC design pattern, which made it easier to implement the REST API. Its built-in generic components like user authentication; handling of HTTP transactions; rendering of web pages and forms; and the testing facility posed as a great advantage to us who want to create a web server as quickly as possible. It also included a generic permission structure, which allowed us to enforce control over the access of each view on the web server.
Django makes it easy to implement a custom user model (see Appendix E for an example) that was needed for this project, because extra attributes were needed to be added such as the available money each user has and the location of the user if it has specified it. The Django package did not come with all the features needed for the project, therefore third-party libraries needed to be included to supply the required features needed for the web server. Django Registration package was used to add a registration facility for if users wanted to create an account on the web server. The Django REST framework was included because it supplies generic class-based view functions to implement an REST API for the mobile device to communicate with the web server. Users could be given certain permissions, that were automatically generated. The web interface design was semi-automated by using the built-in generic class views, which allowed the developer to incorporate the existing permission structure by specifying some parameters within the views.

Models

The Django model section represents the mySQLite database operating behind the scenes. When we refer to a 'model' we also refer to the equivalent table that exists in the database. This is a critical part of the web server, as it provide the basic format of how data is formatted and sorted in this project. Therefore it is needed to supply a description about the most important tables created in the database and their associated fields. This will eventually provide some context when we discuss the views of the web server in the next section. It is important to note that mySQLite requires that every model must have a primary key field to maintain referential integrity.

In the user model the distinction between each type of user defined in section 3.1 is made by the 'Group' object included in the base structure of the Django generic user model class. Consequently this 'Group' object altered the database structure to be a more simplified database structure as it can be observed in Figure 4.1. This 'Group' object in the user model forced us to implement the functional specifications given in section 3.2.1 rather in the view section of the web server. In Table 4.1 only the addition fields in the 'User' object are described:

Table 4.1: Addition fields introduced into the 'User' object.

Field	**Type**	**Description**
contactnumber	text	Users could specify their contact number in case an email address was not suitable for the situation.
acct_balance	integer	This value is needed when payments are being processed. It indicates the hypothetical amount of money this user has.
location	integer	This is a foreign key reference to the 'Location' object. It is used to track the last physical location the user has engaged with a claiming operation or a payment operation.

The 'NFCDevice' object is created to provide security in system, but the security is only implement in the view section. In the model section it is only an object that records each NFC-compliant device that has been used in the system. The 'NFCDevice' will only be briefly described in Table 4.2, but its really value is shown in the next section.

Table 4.2: Description of the 'NFCDevice' object.

Field	**Type**	**Description**
user_id	integer	Reference the user to which this 'NFCDevice' object is being associated with.
uid	text	The unique identifier number embedded in every NFC-compliant device.

For the one use case illustrated in Figure 3.5 that involves the ticketing operation, it was neccessary to create an object containing information about the ticket. The object created for this purpose is called a ‘Claim’, because it can refer to a ticket or a voucher. In Table 4.3 each field in this object is accompanied with a short description.

Table 4.3: Model description of the ‘Claim’ object.

Field	**Type**	**Description**
user_id	integer	Foreign key. Identifies to which user this claim is belonging to.
type	text	Indicates whether this claim is a ‘voucher’ or a ‘ticket’.
expiry_date	date	Indicates the date until this claim object would still be considered valid.
claimed	boolean	Indicates whether this object already have undergo the claiming operation.
amount	integer	Represents the hypothetical value of this object.

For the NFC payment use case illustrated in Figure 3.4 three models are created for this operation, namely ‘Invoice’ and ‘Transactions’. They are related with a one-to-many relationship, because this allows more than one user to make a payment to the same invoice. In other words an ‘Invoice’ can be associated with many ‘Transactions’, which in turn means many users can be associated with one ‘Invoice’. These two objects is only used during the payment process and creates a mechanism for implementing checks and balances.
The equivalent Enhanced Entity–Relationship (EER) diagram of the database is displayed in Figure 4.1:

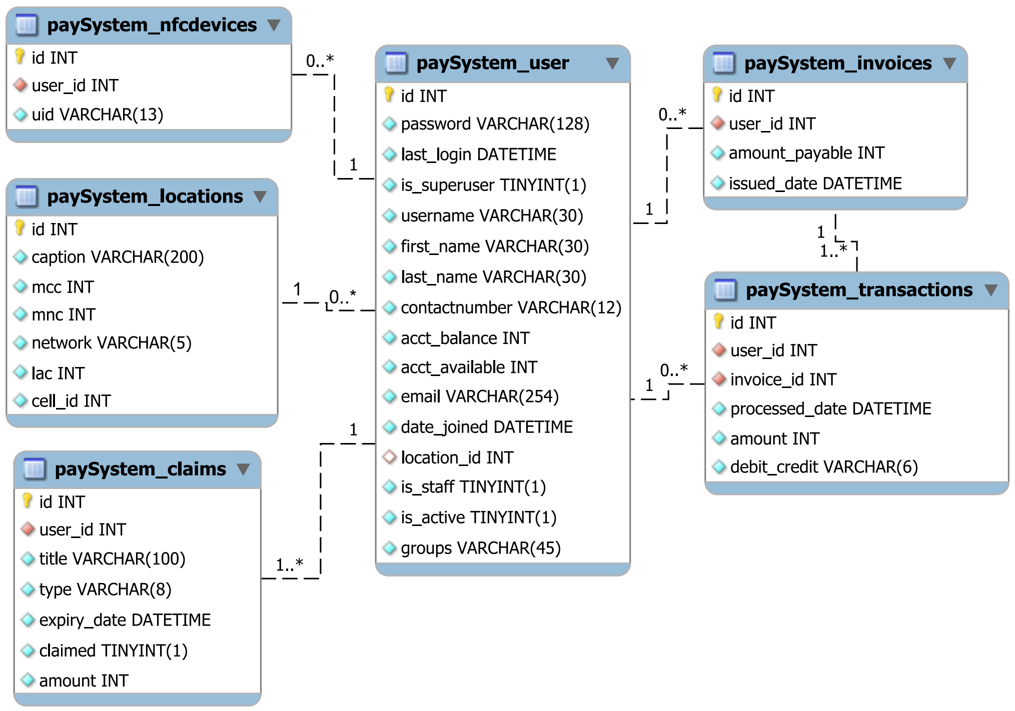


Figure 4.1: The EER model of the database used in this project.

Views

In Django the view and controller sections of the MVC design pattern is tightly coupled. The views can be group into three seperate groups:

- Views that represent the REST API (Figure 4.3 represents the URI to locate each resource and the permission imposed on the specfic resource.)
- Views used to render web pages for the website
- Views used by the Arduino

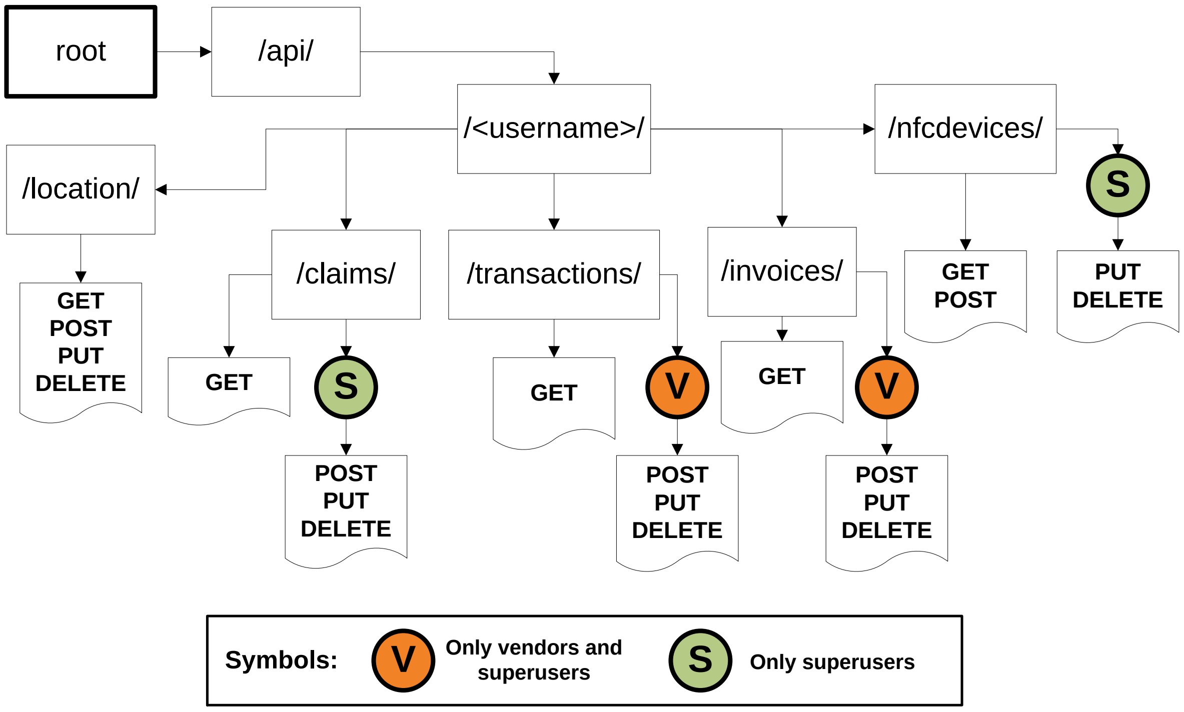


Figure 4.2: URI of the website with user permissions indicated.

Figure 4.3: URI for the REST API with user permissions indicated.

URL configurations

In Figure 4.4 the site map is visualised by using a flow chart. Each branch is equivalent to a generic URI path, which invokes a specific view assigned to the path. It is important to notice that every branch the users needs to be authenticated (or logged in), except for the branch that starts with `root/accounts/`.

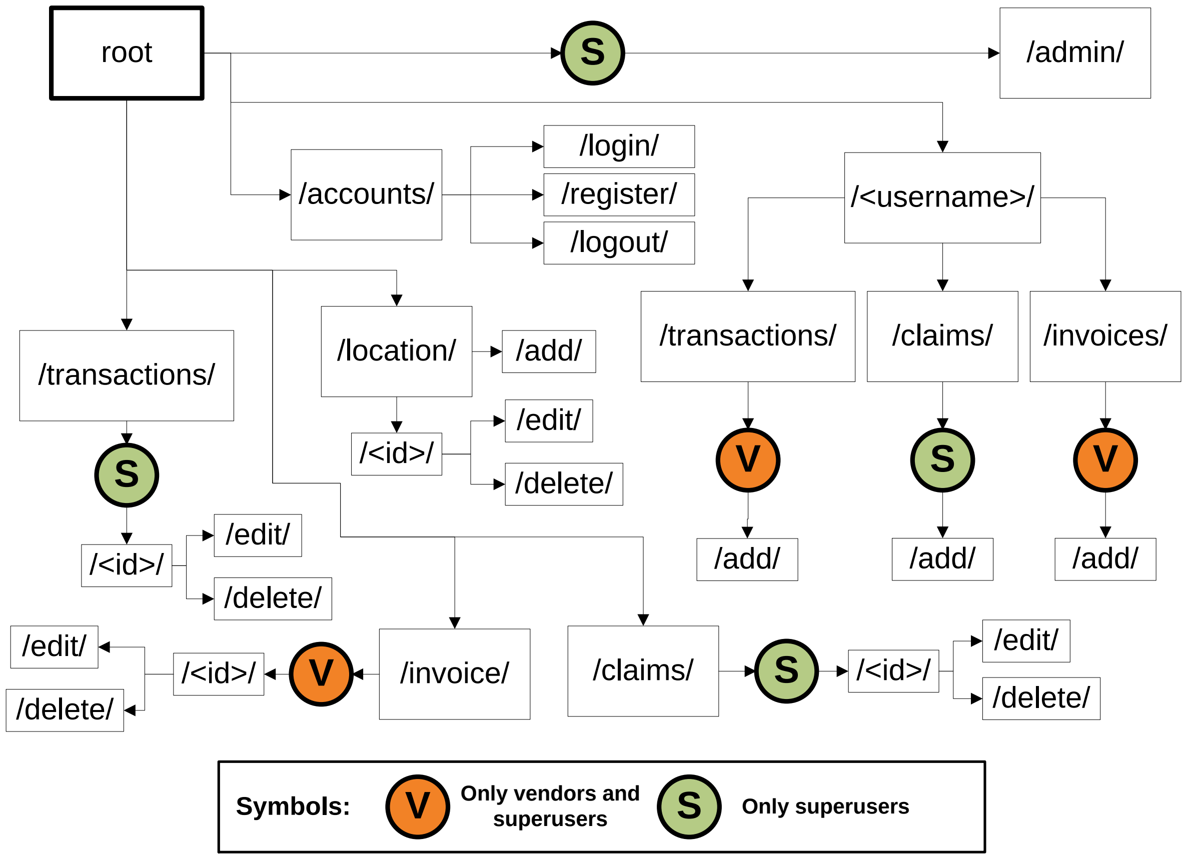


Figure 4.4: URI of the website with user permissions indicated.

4.1.2 Android device

In this project we used the Asus Nexus 7 with Android version 4.3. This section only discusses the software design choices that was taken during the development of the NFCPaySystem mobile application. Numerous functions that are packaged with the Android SDK was needed for developing the mobile application. One of these functions provided us the necessary code to use the NFC–interface embedded in the Android device. The rest of the functions were used to implement the Service API discussed in [12]. The Android SDK did not provide a framework to implement the Service API used for interacting with our web server's REST API. Therefore we used a third-party library DataDroid that included an example mobile application that showed a way of interacting with a REST web service. Lastly we will discuss the user authentication aspect of the mobile application.

Sending custom requests to the AVR via NFC

For the NFC–interface there was a need to specify a custom protocol with a simple syntax when data is to be transmitted from the Android to the AVR. Comma–separated values

(CSV) was used as the standard for the syntax. Keeping in mind the AVR's limited SRAM, this seemed the most viable option, because CSV only uses a comma to make a distinction between each value specified by the syntax. This custom protocol is only used when requests are made by transmitting NDEF messages via NFC between the Android and the AVR. Each request can be recognised by the AVR by reading the first byte of the NDEF record payload. There are only two different requests – a claiming request or a payment request. Table 4.4 explains the custom protocol for the Android to invoke a request on the AVR when they are interacting via NFC in Peer–to–peer mode. Table 4.5 and Table 4.6 specifies the syntax for the NDEF text record for a claiming request and a payment request respectively.

Table 4.4: Description of the protocol between the Android device and the AVR when sending CSV formatted data via NFC.

First byte	**Description**	**NDEF message length**
'T'	This request calls for the procedure to claim a ticket.	33 bytes (maximum)
'V'	This request calls for the procedure to claim a voucher.	33 bytes (maximum)
'P'	This request calls for the making a payment during a purchasing. activity	11 bytes (maximum)

For any claim the syntax only allows a maximum of 33 bytes (with commas included) for the NDEF text record. The first byte shown in Table 4.5 is used to make a distinction between a ticket or a voucher.

Table 4.5: NDEF text record syntax for sending a claim request to the AVR.

Model attribute	**Type**	**User ID**	**Claim ID**	**Title**	**Expiry date**	**Amount**
Size	1 byte	2 bytes	2 bytes	10 bytes	10 bytes	4 bytes
Range/Format	T/V	1 - 99	1 - 99	Text	yyyy-mm-dd	1-9999

To clarify the syntax shown in Table 4.5 an example of a raw NDEF text record for requesting a ticket is supplied:

```
T05,66,TheBeeGees,2013-12-01,0740
```

For any payment made the syntax only allows a maximum of 11 bytes (with commas include) for the NDEF text record.
To clarify the syntax shown in Table 4.6 an example of a raw NDEF text record for requesting a ticket is supplied:

```
P23,05,1000
```

Table 4.6: NDEF text record syntax for sending a payment request to the AVR.

Variable	**Type**	**Invoice ID**	**Amount**
Size	1 byte	2 bytes	4 bytes
Range	P	1 - 99	1-9999

Basic authentication

By default convention with any application that has users the first Android activity called when starting the NFCPaySystem application is the login activity. In this project basic authentication protocol is used to authenticate users. When the user has successfully logged in only the basic authentication header and their username are temporary stored on the device using the SharedPreference built-in class. For every HTTP request made to the web server's REST API the basic authentication header string is included in the HTTP header. Due to time constraints the more secure OAuth 2.0 protocol was not implemented and tested in time. For testing the NFCPaySystem mobile application, when it was needed to authenticate the specified username to be able to access the the user's REST API resources, this protocol was sufficient.

4.2 Controller module or AVR design

For the controller module of the project we chosen the Arduino Uno as the AVR.

4.2.1 AVR hardware

This section will cover primarily the design of the Arduino Uno (controller module) with its associated shields. Here is the list of compatible shields or modules to be used with the controller module. In Table **??** each shield/module used in conjunction with the AVR is listed and the function that it provides the AVR with:

Table 4.7: List of modules and/or shields used to provided extended features to the AVR.

Shield / module	**Function that it provides the AVR**
IteadStudio 3G shield	This shield gives the AVR the ability to make HTTP requests to remote web servers, that will be needed with the claiming and payment operations discussed in section 4.1.2.
Adafruit NFC controller shield	This shield provides the AVR with an NFC-interface for the AVR to be used to receive NDEF messages from other NFC-compliant devices.
IteadStudio LCD w/ Keypad shield	This shield provides an LCD display for the Arduino with a keypad to enable user input.
Thermal printer module	This module provides the AVR with the ability to print paper receipts when payments are being made or an proof of claim when a valid claim request has been made.

Pin configurations

The AVR is the most important component of the controller module, because the AVR determines which protocols and features is available to be used for creating the controller module. In Table 4.8 the pin configuration with respect to the AVR is tabled to show how each component with its shield is connected.

Table 4.8: Pin assignments with respect to the Arduino Uno supplying the AVR.

Pin	**D0**	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**
Description	RX	TX	RXD	TXD	Back light	RS	RXD
Shield/module connected to	*	*	Printer	Printer	LCD	LCD	3G
Pin	**D7**	**D8**	**D9**	**D10**	**D11**	**D12**	**D13**
Description	TXD	Power	Reset	SS	MOSI	MISO	SCK
Shield/module connected to	3G	3G	3G	NFC	NFC	NFC	NFC
Pin	**A0**	**A1**	**A2**	**A3**	**A4**	**A5**	
Description	Buttons	DB4	DB5	DB6	DB7	Status Indicator	
Shield/module connected to	LCD	LCD	LCD	LCD	LCD	LED	

Power considerations

The Arduino Uno has the NCP1117-chip as the power regulator. It is able to output a current in excess of 1 A. The Itead LCD shield was set aside during the integration phase, because it used approximately 30 mA when operating in standby mode (without the back light turn on).

Battery

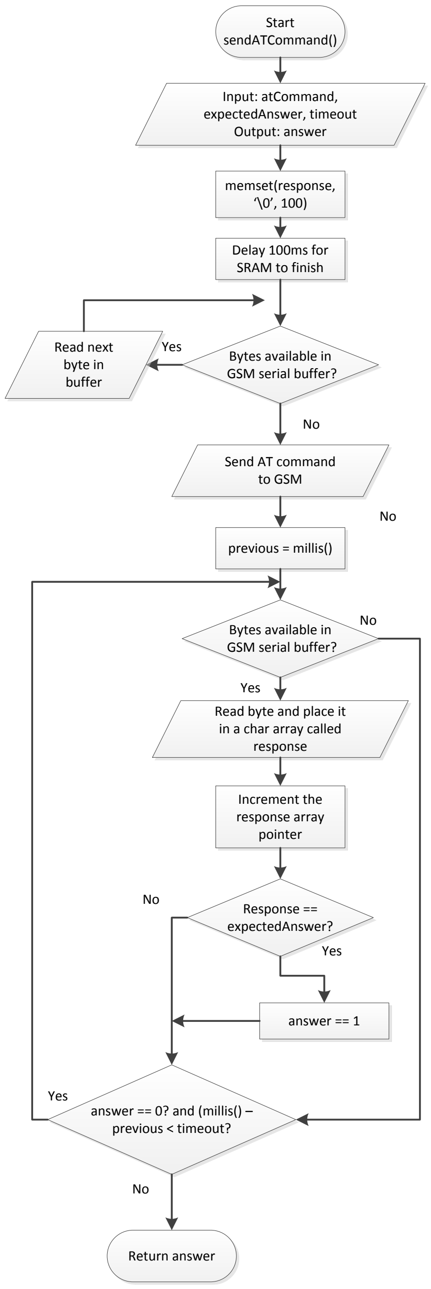
Battery voltage is being regulated by the AVR using an internal voltage sensor. Calculations to determine the capacity needed for a battery was not made yet, because the system was not yet fully integrated before the deadline.

4.2.2 AVR software

Initial set-up segment

For any hardware component it is needed to initialise some settings on it before any processing can be done with it. The hardware SPI protocol was used for communicating with Adafruit NFC controller shield. The Itead 3G shield only has an UART interface available. The software UART protocol was used, because the hardware UART is only made available for the Arduino Uno to use for uploading sketches from the computer. The software UART serial becomes very inaccurate with baud rates higher than 19200 bps, therefore we used the lowest baud rate possible to provide communication between the AVR and the 3G shield. The 3G module used AT commands to perform specific operations that included checking the network status and starting an HTTP transaction.

The 3G module does not always receive an answer that it has expected, therefore is was needed to develop an algorithm that had a time-out variable specified with each AT command been sent to the 3G module.



Start sendATCommand()
Input: atCommand, expectedAnswer, timeout Output: answer
memset(response, '\0', 100)
Delay 100ms for SRAM to finish
Read next byte in buffer
Yes
Bytes available in GSM serial buffer?
No
Send AT command to GSM
No
previous = millis()
No
Bytes available in GSM serial buffer?
Yes
Read byte and place it in a char array called response
Increment the response array pointer
No
Response == expectedAnswer?
Yes
answer == 1
Yes
answer == 0? and (millis() – previous < timeout?
No
Return answer

Figure 4.5: Algorithm used to send AT commands to the 3G module.

Custom HTTP responses

Given that AVR had limited SRAM memory available, custom views were developed on the web server. The views that are associated with the controller module used basic HTTP responses with a content body that only consisted of a single byte. because the HTTP response is only text and thus needs a significant amount of memory and string manipulations to parse the response into an object that can be understood by the controller module. Solving this problem of memory consumption and processing power, we use send an HTTP response with a body of only 8 bytes. This makes it much easier for the controller module to acknowledge and parse it.

Android compatibility

We need the controller module to be compatible with Android as stated in section 1.2.1. To achieve this we are required to use the Peer–to–Peer mode, which means that two extra software layers namely the LLCP and the SNEP needs to be implemented first for the controller module to be able to interact with the Android Beam feature.

Chapter 5

Measurements and Tests

5.1 Testing of Itead 3G shield to send an HTTP GET request to the test web server

5.1.1 Objective

The objective of this test is to determine if the Itead 3G shield can successfully send an HTTP GET request to the web server and then receive an HTTP response from the web server.

5.1.2 Hypothesis

The AVR is capable of sending strings of characters resembling HTTP GET requests to the Itead 3G shield by communicating over a software UART connection. The appropriate AT commands listed in [13] can then be used to transmit an HTTP GET request in ASCII format to the web server. The 3G shield is then capable of receiving the returning HTTP response and send it over UART to the AVR to be displayed onto a terminal screen. For this hypothesis to be valid, the following assumptions are made:

1. A web server is already running and is able to receive HTTP requests sent over the MTN network.
2. An SIM card registered on the MTN network, with a minimum of R20 in airtime available, is inserted into the SIM card holder populated on the shield.
3. The Itead 3G shield, with its antenna connected, is connected to two GPIO pins of the AVR, which has been programmed to implement a software UART-interface. See Table 4.8 for the pin connections.

5.1.3 Test design

This test involves a web server and an Arduino Uno with the Itead 3G shield stacked onto it. Firstly the web server needs to be started by following these instructions:

1. Open a web terminal program (e.g. PuTTY) and start a network session at `ml.sun.ac.za:22`
2. When the connection has been established, enter your user credentials.
3. After you are logged in locate the root directory of the Django project.
4. Then execute the following command: python manage.py runserver 0.0.0.0:8000

Secondly a test sketch needs to be uploaded onto the Arduino Uno. See Appendix E for a link to the test script.
For uploading the sketch follow these steps:

1. Connect the Arduino Uno to a 9V DC power supply that can deliver at least a direct current of 800mA.
2. Connect an USB–cable between the Arduino Uno and the computer with Arduino IDE version 1.0.2 installed on.
3. Upload the "SIM5216_http_get.ino" sketch onto the Arduino Uno.
4. Open a terminal program (e.g. Serial monitor) and set the baud rate to 115200 bps (8-N-1).
5. Wait for 3G module to power up and register to the MTN network (The status LED on the Itead 3G shield will indicate whether the 3G module is registered and ready to be used.

For executing this test successfully, no user intervention is required. Wait for the HTTP transactions to commence and see the resultant response in the terminal window. Note: This test is executed repeatedly with a 2 second delay between each loop.

5.1.4 Result

When the Arduino and the Itead 3G shield was powered with both the USB–cable and the power supply plugged in they have drawn approximately 800mA, assuming that the USB port contributes 500mA at most. The Itead 3G Shield started its power on sequence and its status LED was blinking at a rate of 1.25Hz. When the HTTP response was being transmitted the status LED was blinking at a rate of 5Hz and then back to a rate of 800ms when it has receive an HTTP response. During the data transmitting period the Itead 3G shield has shown periodical bursts of current which peaked at 800mA excluding the current that is delivered from the USB port.
The two tables below represents the responses shown in Arduino IDE and PuTTY terminal respectively. Ten HTTP GET requests has been executed between the web server and 3G module. In Table 5.1 only the first request and response output is shown, because all of the others are exactly the same, except for the date that will differ.

Table 5.1: Arduino IDE's Serial Monitor output summary for HTTP GET request.

Request	**Response**
3G : Registered to network	+CHTTPACT: DATA,
3G : Ready for AT command	172
*** Start HTTP Transaction ***	http/1.0 200 ok
+CHTTPACT: REQUEST	date: fri, 01 nov 2013 23:29:52 gmt
AT+CHTTPACT="ml.sun.ac.za",8000	server: wsgiserver/0.1 python/2.7.5
GET /test/ HTTP/1.1	content-type: text/html; charset=utf-8
Host:ml.sun.ac.za:8000	
Content-Length:0	
<html><body><h3>Test</h3>	
</body></html>	*** End HTTP Transaction ***
	Memory Free : 1386

Table 5.2: PuTTY output summary for HTTP GET request.

```
login as: jibritz
jibritz@ml.sun.ac.za's password:
Last login: Tue Oct 29 20:29:46 2013 from 10.10.11.102
[jibritz@sheldon ~]$ cd devel/skripsie/
[jibritz@sheldon skripsie]$ python manage.py runserver 0.0.0.0:8000
Validating models...

0 errors found
November 02, 2013 - 01:28:50
Django version 1.5.2, using settings 'skripsie.settings'
Development server is running at http://0.0.0.0:8000/
Quit the server with CONTROL-C.

[02/Nov/2013 01:29:52] (197.79.11.48) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:02] (41.117.83.222) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:11] (197.65.238.124) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:19] (197.79.5.112) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:28] (41.117.205.102) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:37] (41.118.86.27) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:45] (41.118.7.161) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:30:54] (41.117.172.244) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:31:03] (41.118.128.52) "GET /test/ HTTP/1.1" 200 39
[02/Nov/2013 01:31:11] (197.65.57.218) "GET /test/ HTTP/1.1" 200 39
```

We can note from the timestamps in Table 5.2 that each request was completed at an average duration of 6.7 seconds if we subtract two seconds for the delay introduced on the Arduino.

5.2 Testing of Itead 3G shield to send an HTTP POST request to the test web server

5.2.1 Objective

The objective of this test is to determine if the Itead 3G shield can successfully create an invoice object on the web server by sending an HTTP POST request to the web server and then receive the returning HTTP response from the web server which will indicate whether this test was successful.

5.2.2 Hypothesis

The AVR is capable of receiving user input from the computer and then concatenated these inputs with strings of characters resembling an HTTP POST request. This HTTP POST request is then stored on the internal memory of the AVR. The appropriate AT commands listed in [13] can then be used to transmit this HTTP POST request stored on the AVR in ASCII format to the web server. The 3G shield is then capable of receiving the returning HTTP response and send it over UART to the AVR to be displayed onto a terminal screen.

Assumptions

For this hypothesis to be valid, the following assumptions are made:

1. A web server is already running and is able to receive HTTP requests sent over the MTN network.
2. An SIM card registered on the MTN network, with a minimum of R20 in airtime available, is inserted into the SIM card holder populated on the shield.
3. The Itead 3G shield, with its antenna connected, is connected to two GPIO pins of the AVR, which has been programmed to implement a software UART-interface. See Table 4.8 for the pin connections.

5.2.3 Test design

The test requires a web server and an Arduino Uno with the Itead 3G shield stacked onto it. Firstly the web server needs to be started by following these instructions:

1. Open a web terminal program (e.g. PuTTY) and start a network session at `ml.sun.ac.za:22`
2. When the connection has been established, enter your user credentials.
3. After you are logged in locate the root directory of the Django project.
4. Then execute the following command: python manage.py runserver 0.0.0.0:8000

Secondly a test sketch needs to be uploaded onto the Arduino Uno. See Appendix E for a link to the test script. For uploading the sketch and executing this test successfully, follow these steps:

1. Connect the Arduino Uno to a 9V DC power supply that can deliver at least a direct current of 800mA.
2. Connect an USB–cable between the Arduino Uno and the computer with Arduino IDE version 1.0.2 installed on.
3. Upload the ”SIM5216_http_post.ino” sketch onto the Arduino Uno.
4. Open a terminal program (e.g. Serial monitor) and set the baud rate to 115200 bps (8-N-1).
5. Wait for 3G module to power up and register to the MTN network (The status LED on the Itead 3G shield will indicate whether the 3G module is registered and ready to be used.
6. Two parameters are required to create an invoice:
   - User ID: Each user has an unique id field. (Maximum of 2 characters)
   - Amount: Specifies the required 'amount_payable' field of the invoice object (Maximum 4 characters)
7. Wait for 3G module to process the HTTP transaction.

5.2.4 Test data

The nature of this test requires some user input. Table 5.3 lists the test values used in this test.

Table 5.3: Test input used for sending HTTP POST request test.

Trial	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
User ID	1	2	3	1	2	3	1	2	3	1	2	3
Amount	500	500	500	125	231	999	9999	4321	8888	3	21	333

5.2.5 Result

When the Arduino and the Itead 3G shield is powered with both the USB–cable and the power supply plugged in it uses approximately 800mA, assuming that the USB port contributes 500mA at most. The Itead 3G Shield started its power on sequence and subsequently its status LED was blinking at a rate of 1.25Hz indicating that the 3G shield is registered to a network. When the HTTP response was being transmitted the status LED was blinking at a

rate of 5Hz and then back to a rate of 1.25Hz when it has receive an HTTP response. During the data transmitting period the Itead 3G shield has shown periodical bursts of current which reached a maximum of 800mA. This measurement is excluding the current that is being supplied from the USB port.
The two tables below represents the responses shown in Arduino IDE and PuTTY terminal respectively. Ten HTTP POST requests has been executed between the web server and 3G module. In Table 5.4 only the first request and response output is shown, because all of the others are exactly the same, except for the date has differed.

Table 5.4: Arduino IDE's Serial Monitor output summary for HTTP POST request.

Request	**Response**
3G : Registered to network	+CHTTPACT: DATA,
3G : Ready for AT command	134
==== HTTP POST EXAMPLE ====	http/1.0 200 ok
Create an invoice	date: sat, 02 nov 2013 12:46:17 gmt
Enter user's id number:1	server: wsgiserver/0.1 python/2.7.5
Enter the amount the user have to pay:500	content-type: text/html; charset=utf-8
** Start HTTP Transaction ***	V
+CHTTPACT: REQUEST	*** End HTTP Transaction ***
AT+CHTTPACT="ml.sun.ac.za",8000	
Sends Request:	Memory Free : 1353
POST /arduino/invoice/add HTTP/1.1	
Host:ml.sun.ac.za:8000	
Content-Type:application/x-www-form-urlencoded	
Content-Length:29	
user=1&amount_payable=500	

We can note from the timestamps and the JSON output located in Appendix D that each request was successful.

5.3 Sending NDEF messages via NFC from the Android device to the Adafruit PN532 NFC Controller Shield

The objective of this test is to send a string of characters over an NFC connection successfully. To achieve this objective NFC connection has to operate in Peer–to–peer mode on both devices, to be able to send that from the Nexus 7 Android device to the Adafruit PN532 NFC Controller Shield connected to the AVR.

5.3.1 Hypothesis

The AVR is capable of implementing a hardware SPI-communication with the Adafruit PN532 NFC Controller Shield. The PN532–chip populated on the Adafruit PN532 NFC Controller Shield is capable of implementing an LLCP software layer with an SNEP layer (required for Peer–to–peer transactions) to manage data transfers between the Android device and the NFC shield. When the Android device and the NFC shield is held closely together, both devices will spontaneously initiate an LLCP connection between them and then follow the communication conventions set in SNEP to engage in a data transaction.

Assumptions

For this test it is assumed that the Adafruit NFC shield to be set-up as the target device and the Android device is set-up to be the initiator device. It is also assumed that the Android device has an mobile application installed that is capable of sending NDEF messages via its NFC–interface.

5.3.2 Test design

Firstly this test requires an Arduino Uno with the Adafruit NFC shield connected to it with via the SPI-interface. Secondly a test sketch needs to be uploaded onto the Arduino Uno. See Appendix E for a link to the test script. For uploading the sketch and executing this test successfully, follow these steps:

1. Connect the Arduino Uno to a 9V DC power supply that can deliver at least a direct current of 800mA.

2. Connect an USB–cable between the Arduino Uno and the computer with Arduino IDE version 1.0.2 installed on.

3. Upload the ”nfc_ndef_pull.ino” sketch onto the Arduino Uno.

4. Open a terminal program (e.g. Serial monitor) and set the baud rate to 115200 bps (8-N-1).

5. Wait for PN532-chip to power up and state that it is ready on the Arduino Serial monitor it.

6. Start the Android device and upload the NFCDemo mobile application project from the Eclipse IDE.
7. Type in each string of characters listed in Table 5.5 and hold the two NFC device closely together. Tap with you finger on the Android touch-screen when the following message appear: “Touch to beam”.

5.3.3 Test data

Table 5.5: Test input used for sending NDEF messages between an Android device and Arduino Uno.

Trial	**Value**
1	Hello world
2	abcdefghijklmnopqrstuvwxyz
3	Hallo! Hoe gaan dit?
4	NFC is so awesome!

5.3.4 Result

When the AVR with the Adafruit NFC shield is supplied with power the power LED on the NFC shield lights up and shows on the power supply that it uses a current of 320mA. When the Serial Monitor in the Arduino IDE is opened it states that the PN532-chip was found and that the module is ready to for interacting with other NFC–device by displaying: “*** LOOP ***”
Every trial has succeed. There were occurrences where the PN532 timed out, but there were no occurrences where the NDEF message has failed numerous time before it was successfully transmitted.
The output from the Arduino IDE’s Serial Monitor is available in Appendix D

5.4 Sending a claim request via NFC from the Android device to the controller module

5.4.1 Objective

The objective of this test is to send a string in the format shown in Table 4.5 over an NFC connection successfully. To achieve this objective an NFC connection has to operate in Peer–to–peer mode on both devices for it to be able to send NDEF messages from the Android device to the AVR. The Itead 3G shield, connected to the AVR, needs to make an GET request to the web server to process the claim. Then when the response arrives from the web server, the AVR should indicate if the claim operation was successful or not.

5.4.2 Hypothesis

The AVR is capable of communicating with the Adafruit PN532 NFC Controller Shield. If the AVR, with the Adafruit PN532 NFC Controller Shield connected, is capable of receiving data via NFC from the Android device, then it is capable of extracting the NDEF record from the NDEF message that contains the payload data. If the AVR has received the NDEF message from the Android device, then the AVR is able to process and extract the 'user_id' and 'claim_id' variables contained in the NDEF message. If the Itead 3G shield, connected to the AVR, is idle, then the AVR is capable of sending an HTTP GET request to the web server. When the AVR receives the HTTP response from the web server it is able to determine if the operation was successful or not.

Assumptions

For this test to be executed successfully the following is assumed:

- TheAdafruit PN532 NFC Controller Shield is already set-up as the target device and the Android device is set-up to be the initiator device.
- In the database on the web server there exists claim items that are not claim (in other words 'claimed=false') and belongs to the specific user using the NFCPaySystem application.
- Each claiming operation, that involves in claiming one item only, is considered as a trial for this test.

5.4.3 Test data

Five different claim objects shown in Table 5.6 have been created on the web server that will be used to provide data input for the claiming operation.

Table 5.6: Test claims generated for the claiming operation.

Trial	User ID	Title	Type	Expiry Date	Claimed	Amount
1	2	Rugby Game	TICKET	2013-10-26	false	250
2	2	Look&Listn	VOUCHER	2013-11-01	false	500
3	2	The Nutcracker	TICKET	2013-11-22	false	340
4	2	Woolworths	VOUCHER	2013-11-30	false	2000
5	2	PicknPay	VOUCHER	2013-12-01	true	3000

5.4.4 Test design

This test requires an Arduino Uno with the Adafruit PN532 NFC Controller Shield connected via the SPI-interface and the Itead 3G shield connected via the software implementation of UART. It also requires the Nexus 7 Android device and the web server.
Firstly the web server needs to be started by following these instructions:

1. Open a web terminal program (e.g. PuTTY) and start a network session at `ml.sun.ac.za:22`.
2. When the connection has been established, enter your user credentials.
3. After you are logged in locate the root directory of the Django project.
4. Then execute the following command: python manage.py runserver 0.0.0.0:8000.

Secondly a test sketch needs to be uploaded onto the Arduino Uno. See Appendix E for a link to the test script. For uploading the sketch, follow these steps:

1. Connect the Arduino Uno to a 9V DC power supply that can deliver at least a direct current of 800mA.
2. Connect an USB–cable between the Arduino Uno and the computer with Arduino IDE version 1.0.2 installed on.
3. Upload the ”Claiming.ino” sketch onto the Arduino Uno.
4. Open a terminal program (e.g. Serial monitor) and set the baud rate to 115200 bps (8-N-1).
5. Wait for both the 3G module and the PN532-chip to power up and state that they are ready on the Arduino Serial monitor it.

Thirdly the Android device has to be set-up by following these steps:

1. Start the Android device and upload the NFCPaySystem mobile application project from the Eclipse IDE.

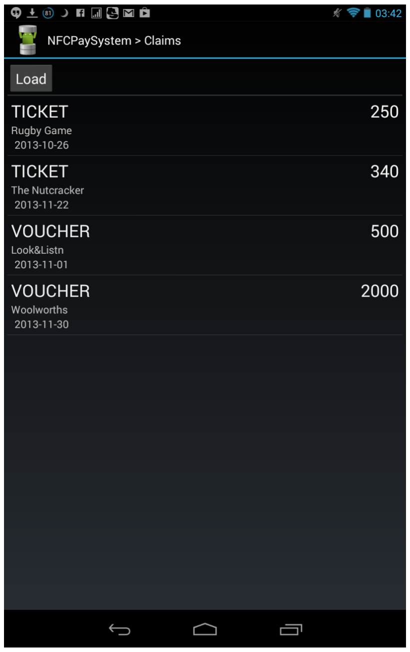
2. Type in user credentials and tap ‘Log in’.
3. On the Android device tap with your finger on the ‘Claims’ list item.
4. Press ‘Load’ if there are no items listed yet.

For executing this test successfully, follow these steps:

1. Tap and your finger on the list items shown on the screen until it shows the message: “Hold near the Controller Module to proceed”
2. Hold the Android device near the Adafruit PN532 NFC Controller Shield and tap again on the screen when it this message appear “Touch to beam”.
3. Wait for the AVR to make the HTTP transaction and receive a response.
4. Repeat this process for each list item shown on the Android device.
5. After each item has been processed, press the “Load” button again to see whether the operation was successful.

5.4.5 Result

When the AVR with the Adafruit PN532 NFC Controller Shield is supplied with power the power LED on the shield lights up and shows on the power supply that it uses a current of 320mA. When the Serial Monitor in the Arduino IDE is opened it states that the PN532-chip was found and that the module is ready to for interacting with other NFC–device by displaying: “Ready”
Every trial has succeeded. Except that for each trial the AVR had to be restarted. The AVR reaches an unknown breakpoint after each trial, which could not be identified. There were occurrences where the PN532 timed out, but there were no occurrences where the claim operation has failed numerous times before it was successfully transmitted.

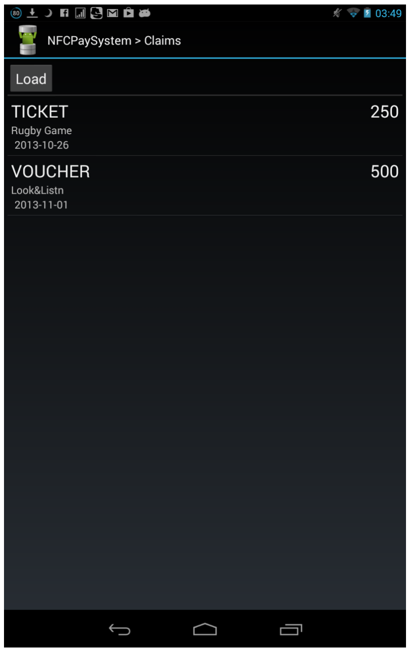


NFCPaySystem > Claims

Load

TICKET Rugby Game 2013-10-26	250
TICKET The Nutcracker 2013-11-22	340
VOUCHER Look&Listn 2013-11-01	500
VOUCHER Woolworths 2013-11-30	2000

Figure 5.1: User interface of Android device before the test.



NFCPaySystem > Claims

Load

TICKET Rugby Game 2013-10-26	250
VOUCHER Look&Listn 2013-11-01	500

Figure 5.2: User interface of Android device after the test.

In Figure 5.1 it shows the four test claims that were created before the test started. The fifth test claim is not listed, because it has been claimed already.
In Figure 5.2 it shows only two claims left, because they have expired and cannot be claimed. The other two claims were valid and the 'claimed' field was set to true like we expected.

Chapter 6

Conclusions

6.1 Objectives achieved

This project has not achieved all of the objectives that was specified in in section 1.2, but did achieved some of them.
Three main objectives were set for this projective, namely:

1. Building a controller module.
2. Developing a suitable network model.
3. Developing an mobile device application.

Building a controller module

For transferring data wirelessly between an NFC-interface and an NFC-compliant mobile device has been achieved and results are shown in the tests done.
For the 3G cellular module to be able to make both HTTP GET and POST request to a remote server and then receiving a response has been achieve in this project.
For a mobile device user to claim a ticket or a voucher has been achieved, except for the part where the thermal printer module has to print a proof when the ticket has been claimed.
For an LCD screen to be used to indicate the operationg state of the controller module has not been achieved.
For using a rechargable battery to supply power for one hour has not been achieved.

Developing a suitable network model

For developiong a suitable database structure for this project has been achieved. For processing claims when a request is being made from the controller module has been achieved and tested.
For creating an invoice using the controller module has been achieved.
For facilitating a payment procedure has not been achieved yet.

Developing a mobile application

For manageing user credentials on the application has been achieved, but it was not achieved in a secure manner.

For sending data messages to the controller module when it interacts with an NFC-compliant device has been achieved.

6.2 Possible improvements

Possible improvements for this project is to enhance the security aspects when payments are being process. In the case of the mobile device, Android SDK version 4.4 has released API support for using mobile devices in Card Emulation mode[14].
Giving greater attention on the system integration part of the controller module.

In terms of user authentication, this project can take the advantage of OAuth 2.0 to implement better security policies when dealing with payment transactions of consumers and vendors.

References

[1] MasterCard Inc. (2013) Mastercard®PayPass™ | Global. [Online]. Available: http://www.mastercard.com/us/paypass/phonetrial/whatispaypass.html

[2] R. T. Fielding, "Architectural Styles and the Design of Network–based Software Architectures," Ph.D. dissertation, Univ. of California, Irvine, 2000. [Online]. Available: http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[3] Wikipedia. (2013) Wikipedia — Hypertext Transfer Protocol. [Online]. Available: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[4] NFC Forum. (2013) NFC Forum : Frequently Asked Questions. [Online]. Available: www.nfc-forum.org/resources/faqs

[5] NFC Forum. NFC Data Exchange Format. [Online]. Available: http://www.eet-china.com/ARTICLES/2006AUG/PDF/NFCForum-TS-NDEF.pdf

[6] NFC Forum. (2013) NFC Forum Technical Specifications. [Online]. Available: http://www.nfc-forum.org/specs/spec_list/

[7] Wikipedia. (2013) Basic access authentication. [Online]. Available: http://en.wikipedia.org/wiki/Basic_access_authentication

[8] ——. (2013) Base64. [Online]. Available: http://en.wikipedia.org/wiki/BASE64

[9] Arduino. (2013) Arduino – Arduino Shields. [Online]. Available: http://arduino.cc/en/Main/ArduinoShields

[10] Django. (2013) The web framework for perfectionists with deadlines. [Online]. Available: https://www.djangoproject.com/

[11] 8Motions. (2013) OpenCellID. [Online]. Available: http://www.opencellid.org/api

[12] V. Dobjanschi. (2010, May) Developing Android REST Client Applications. [Online]. Available: http://dl.google.com/googleio/2010/android-developing-RESTful-android-apps.pdf

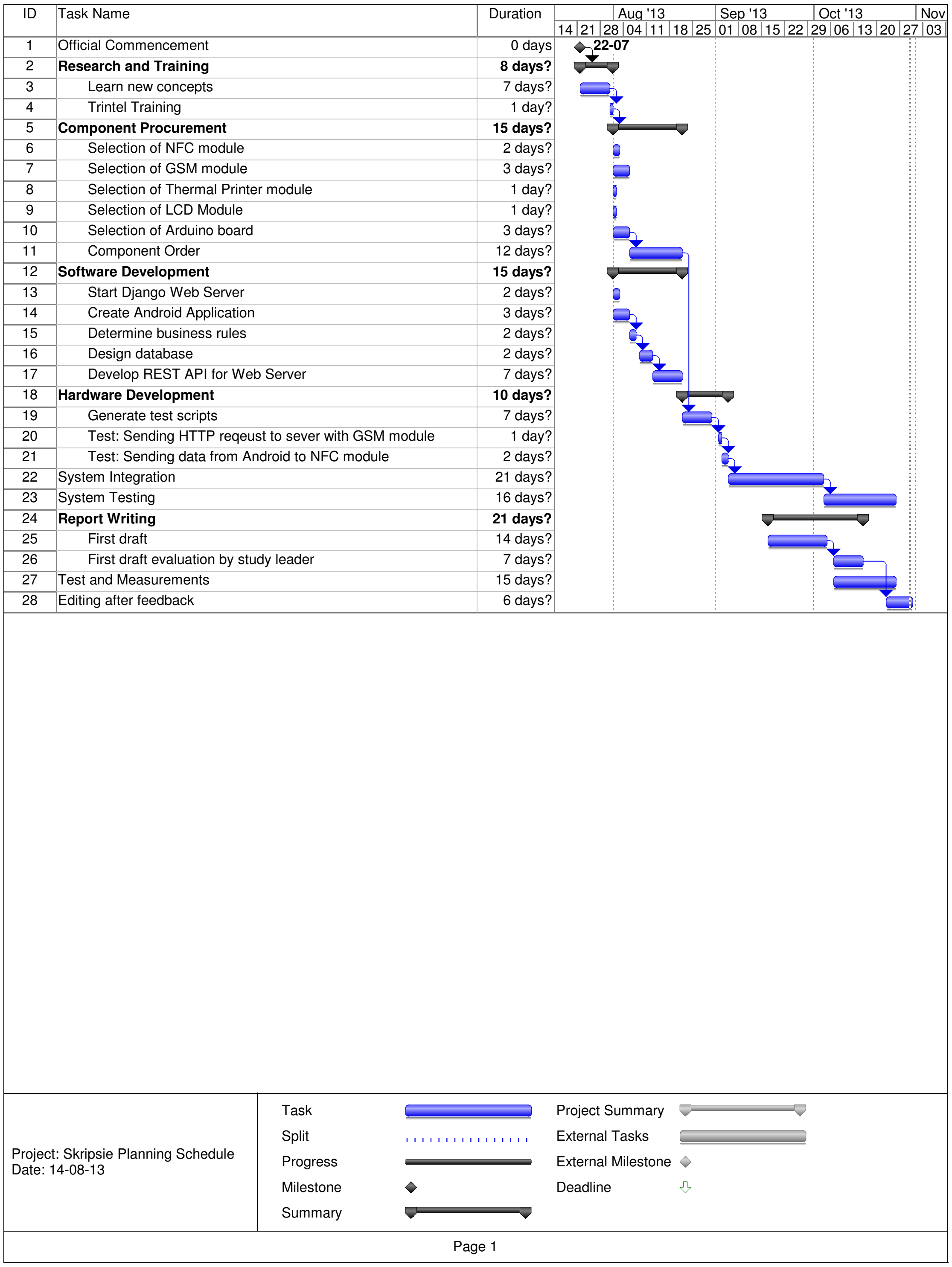
[13] *SIM5216J AT Command Manual*, SIMCOM, Inc., Shanghai, 2011, version 1.02.

[14] S. Clark. (2013) Google gets around the carriers with host card emulation for nfc payments. [Online]. Available: http://www.nfcworld.com/2013/10/31/326619/google-gets-around-carriers-host-card-emulation-nfc-payments/

Appendices

Appendix A

Project planning schedule



ID	Task Name	Duration
1	Official Commencement	0 days
2	**Research and Training**	**8 days?**
3	Learn new concepts	7 days?
4	Trintel Training	1 day?
5	**Component Procurement**	**15 days?**
6	Selection of NFC module	2 days?
7	Selection of GSM module	3 days?
8	Selection of Thermal Printer module	1 day?
9	Selection of LCD Module	1 day?
10	Selection of Arduino board	3 days?
11	Component Order	12 days?
12	**Software Development**	**15 days?**
13	Start Django Web Server	2 days?
14	Create Android Application	3 days?
15	Determine business rules	2 days?
16	Design database	2 days?
17	Develop REST API for Web Server	7 days?
18	**Hardware Development**	**10 days?**
19	Generate test scripts	7 days?
20	Test: Sending HTTP reqeust to sever with GSM module	1 day?
21	Test: Sending data from Android to NFC module	2 days?
22	System Integration	21 days?
23	System Testing	16 days?
24	**Report Writing**	**21 days?**
25	First draft	14 days?
26	First draft evaluation by study leader	7 days?
27	Test and Measurements	15 days?
28	Editing after feedback	6 days?

Project: Skripsie Planning Schedule
Date: 14-08-13

Task	Project Summary
Split	External Tasks
Progress	External Milestone
Milestone	Deadline
Summary	

Page 1

Figure A.1: Project planning schedule using a Gantt chart.

Appendix B

Project specification

NFC is becoming a pervasive technology in the payment and personal authentication industries. Once a transaction has taken place, some physical notification is required – a receipt for a payment or entry ticket to a stadium. A product requirement exists that compromises an NFC–transceiver (SPI) interface, thermal printer (module) and cellular (3G) module for remote management of access control as well as user authentication and notification. The student will be required to build the electronics to control the 3G module and printer module as well as the NFC module and communicate via UART to the 3G modem.

Appendix C

Outcomes compliance

Table C.1: ECSA outcomes criteria.

Outcome	**Reference**	
	Sections	Pages
1. Problem solving: Demonstrate competence to identify, assess, formulate and solve convergent and divergent engineering problems creatively and innovatively.	3	21 - 29
2. Application of Scientific and Engineering Knowledge: Demonstrate competence to apply knowledge of mathematics, basic science and engineering sciences from first principles to solve engineering problems.	4.1 - 4.2	30 - 41
3. Engineering Design: Demonstrate competence to perform creative procedural and non-procedural design and synthesis of components, works, products and processes.	3 - 4	21 - 41
4. Investigations, experiments and data analysis: Demonstrate competence to design and conduct investigations and experiments.	5	42 - 53
5. Engineering methods, skills and tools, including information technology: Demonstrate competence to use appropriate engineering methods, skills and tools, including those based on information technology.	3.3; 4.1	24 - 26; 30 - 37
6. Professional and technical communication: Demonstrate competence to communicate effectively, both orally and in writing, with engineering audiences and the community at large.	All	All
9. Independent learning ability: Demonstrate competence to engage in independent learning through well developed learning skills.	1 - 2	12 - 20

Appendix D

Test logs

D.1 Test output of Itead 3G shield for sending an HTTP POST request to the test web server

This JSON output can be located on the web server at `root\api\<username>\invoices\`:

<username> is jibritz

```
[{
"id": 1,
"amount_payable": 500,
"issued_date": "2013-11-02T14:46:17.394"
},
{
"id": 4,
"amount_payable": 125,
"issued_date": "2013-11-02T14:46:47.132"
},
{
"id": 7,
"amount_payable": 9999,
"issued_date": "2013-11-02T14:48:01.877"
},
{
"id": 10,
"amount_payable": 3,
"issued_date": "2013-11-02T14:48:28.925"
}]
```

<username> is jannie.vendor

```
[{
"id": 3,
"amount_payable": 500,
"issued_date": "2013-11-02T14:46:39.015"
},
{
"id": 6,
"amount_payable": 999,
"issued_date": "2013-11-02T14:47:51.511"
},
{
"id": 9,
"amount_payable": 8888,
"issued_date": "2013-11-02T14:48:20.460"
},
{
"id": 12,
"amount_payable": 333,
"issued_date": "2013-11-02T14:48:45.054"
}]
```

<username> is jean.britz

```
[{
"id": 2,
"amount_payable": 500,
"issued_date": "2013-11-02T14:46:31.027"
},
{
"id": 5,
"amount_payable": 231,
"issued_date": "2013-11-02T14:46:56.967"
},
{
"id": 8,
"amount_payable": 4321,
"issued_date": "2013-11-02T14:48:12.452"
},
{
"id": 11,
"amount_payable": 21,
"issued_date": "2013-11-02T14:48:36.856"
}]
```

D.2 Test output of sending NDEF messages via NFC

This output is from the Arduino IDE's Serial Monitor

```
*** NFC pull NDEF message ***
Found chip PN532
Firmware ver. 1.6
Supports 7

***        LOOP         ***

NDEF record: 1
  NDEF Record
    TNF 0x2 Mime Media
    Type Length 0xA 10
    Payload Length 0xB 11
    Type 74 65 78 74 2F 70 6C 61 69 6E  text/plain
    Payload Hello world
    Record is 24 bytes
rxNDEFPayload() timeout

***        LOOP         ***

NDEF record: 1
  NDEF Record
    TNF 0x2 Mime Media
    Type Length 0xA 10
    Payload Length 0x19 25
    Type 74 65 78 74 2F 70 6C 61 69 6E  text/plain
    Payload abcdefghijklmnopqrsuvwxyz
    Record is 38 bytes
rxNDEFPayload() timeout

***        LOOP         ***

rxNDEFPlayload() failed

***        LOOP         ***

rxNDEFPlayload() failed

***        LOOP         ***

rxNDEFPlayload() failed
```

```
***         LOOP          ***

rxNDEFPlayload() failed

***         LOOP          ***

rxNDEFPlayload() failed

***         LOOP          ***

NDEF record: 1
  NDEF Record
    TNF 0x2 Mime Media
    Type Length 0xA 10
    Payload Length 0x14 20
    Type 74 65 78 74 2F 70 6C 61 69 6E  text/plain
    Payload Hallo! How gaan dit?
    Record is 33 bytes
rxNDEFPayload() timeout

***         LOOP          ***

NDEF record: 1
  NDEF Record
    TNF 0x2 Mime Media
    Type Length 0xA 10
    Payload Length 0x12 18
    Type 74 65 78 74 2F 70 6C 61 69 6E  text/plain
    Payload NFC is so awesome!
    Record is 31 bytes
rxNDEFPayload() timeout
```

Appendix E

Source code

E.1 Git repositories

All the source code used in this project is stored in a git repository hosted on Github

Web server	`https://github.com/jeanbritz/DjangoSkripsie`
Android device	`https://github.com/jeanbritz/AndroidSkripsie`
Arduino module	`https://github.com/jeanbritz/ArduinoSkripsie`
Arduino test sketches	`https://github.com/jeanbritz/ArduinoTestScripts`

E.2 Third-party libraries

Third-party libraries are used to supply neccessary functions, but was beyond the scope of the project due to time constraints.

Django packages	
Django REST framework:	`https://github.com/tomchristie/django-rest-framework/`
Django Registration:	`https://github.com/nathanborror/django-registration`
Crispy forms:	`https://github.com/maraujop/django-crispy-forms`
South:	`https://github.com/dmishe/django-south`
Android	
DataDroid:	`https://github.com/foxykeep/DataDroid/`
Arduino	
SoftwareSerial:	`https://github.com/arduino/Arduino/tree/master/libraries/SoftwareSerial`
SPI:	`https://github.com/arduino/Arduino/tree/master/libraries/SPI`
NFC Link & SNEP Layer for Arduino:	`https://github.com/Seeed-Shield/NFC_Shield_DEV`

E.3 Supplemental code snippets

Django	
Django custom user model:	`https://github.com/iqbalhasnan/django-custom-user-class-based-view`
Arduino	
Available SRAM memory:	`http://playground.arduino.cc/Code/AvailableMemory`
Sending AT command:	`http://www.cooking-hacks.com/documentation/tutorials/arduino-3g-gprs-gsm-gps#step15`
NFC NDEF pull message:	`https://github.com/Seeed-Shield/NFC_Shield_DEV/blob/android_beam/example/nfc_ndef_pull/nfc_ndef_pull.ino`
Android	
Basic authentication :	`http://blog.leocad.io/post/basic-http` `authentication-on-android`

Appendix F

Demonstrations Video

Here is a link to a demonstrations video to show how this project can be used as an NFC ticketing system and/or an NFC payment system: `http://goo.gl/qoGFFw`