The Implementation of Electronic Purse (E-Purse) System in the Banking Sector

(A case study of Guaranty Trust Bank (Nigeria) PLC)

BY

MOHAMMED, AYANNIYI

PGD/MCS/2006/2007/1213

DEPARTMENT OF MATHEMATICS/COMPUTER SCIENCE, FEDERAL UNIVERSITY OF TECHNOLOGY MINNA.

SEPTEMBER, 2008

The Implementation of Electronic Purse (E-Purse) System in the Banking Sector

(A case study of Guaranty Trust Bank (Nigeria) PLC

BY

MOHAMMED, AYANNIYI

PGD/MCS/2006/2007/1213

A PROJECT SUBMITTED TO THE DEPARTMENT OF MATHEMATICS/COMPUTER SCIENCE FEDERAL UNIVERSITY OF TECHNOLOGY MINNA, IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF POST GRADUATE DIPLOMA IN COMPUTER SCIENCE.

SEPTEMBER, 2008

CERTIFICATION

This is to certify that this project, titled: "The Implementation of Electronic Purse (E-Purse) System in the Banking Sector has been read and confirmed by the undersigned as meeting the requirements of the Department of Mathematics/Computer Science, Federal University of Technology, Minna.

---------- ----------

MR BOLARINWA G. DATE

PROJECT SUPERVISOR

---------- ----------

DATE

HEAD OF DEPARTMENT

---------- ----------

EXTERNAL EXAMINER DATE

DEDICATION

This project is dedicated to God for His affections and enablement, to my wife Mrs Suwaibat Mohammed and my Parents.

ACKNOWLEDGMENT

I wish to express my sincere gratitude to God who in His infinite mercy made it possible for me to complete this programme.

Special thanks to my Supervisor, Mr. Bolarinwa Gbolahan who, despite his busy schedule, could afford to read the manuscripts, gave guidance in the course of writing this project and subsequently gave his approval. I also wish to thank my Head of Department Dr. N. I. Akinwande, my course coordinator Alhaji Ndanusa A. and all my Lecturers in the department for their immense contributions towards the success of this program.

I also appreciate the encouragement of my wife Mrs Suwaibat Mohammed whose support has been so enormous. Appreciations to my Parent, brothers and sister for their endurance throughout the period that the program lasted.

My sincere gratitude goes to Engineer Dayo Faleye for taking pains to read through the papers and made appreciable contributions.

I wish to thank all my course mates, brothers, sisters, and friends who contributed in various ways towards the successful completion of this work.

ABSTRACT

Electronic Purse (E-Purse) also known as **Smart-cards** acts as an electronic payment alternative to bank notes. Automated currency solutions offer greater levels of convenience to consumers, incremental sales opportunities and reduce operating costs to merchants and new service revenue streams to banks.

Smart card applications reduce operating costs and control fraud in electronic benefits programs, carrying patient information for healthcare applications and providing a secure vehicle for delivering government benefits such as social insurance and welfare programs.

Smart cards (e-purse) are used in a variety of other applications that require greater processing power and/or more secure storage. College students use smart cards (e-purse) to pay for cafeteria and bookstore purchases and to access health, recreation and other services; commuters use smart cards (e-purse) to pay tolls and parking fees; and parents use smart cards (e-purse) to pay for child care.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Chapter 2

LITERATURE REVIEW

Chapter 3

NEW SYSTEM DESIGN

Chapter 4

SYSTEM IMPLEMENTATION

Chapter 5

Chapter 1
INTRODUCTION

1.1 The Electronic Purse (E-Purse) Introduction

An Electronic purse is a Smartcard that holds an electronic equivalent of cash. The electronic purse can be used for the payment of goods and services i.e. the electronic purse is a substitute for bank notes.

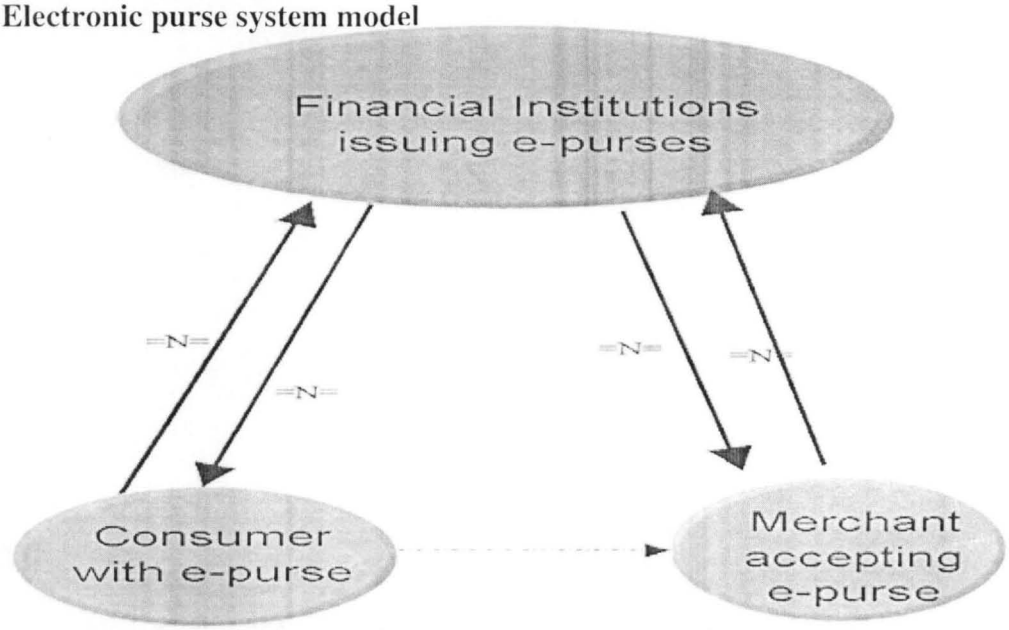
Interest is fast developing in a multipurpose prepaid smart card commonly known as the **electronic purse**. In contrast to a debit card, the electronic purse is intended to facilitate a variety of small-value retail transactions and so it is a clear substitute for currency. It might function as follows.

Monetary value would be loaded onto the card, with a corresponding debit to the cardholder's account at a financial institution. In a retail transaction, monetary value would be transferred from the purchaser's card into the merchant's terminal in an off-line mode. The value of consumer purchases made with electronic purses would accumulate in the merchant's terminal and would be transferred to the merchant's account at a financial institution from time to time through on-line transactions.

More technically, in many purse systems the financial institution issuing the card would earmark or put a hold on an amount of funds in the cardholder's account that is equivalent to that recorded on the smart card, and would in effect be providing a guarantee of the value shown on the card. When a transaction is cleared through the

payments system, a debit would be made to the cardholder's special suspense account, and a credit would be made to the merchant's bank account. This type of electronic purse would essentially function as an off-line debit card (Figure 1).

Figure 1- Electronic purse system model



In most proposed systems, the cardholder would be able to "replenish" the monetary value on the card at Issuer's bank. Such electronic purses would have to be equipped with personal identification in order to keep track of individual transactions. This feature would also assure the cardholder of enhanced security.

Other purse systems are being designed that share physical currency's characteristic of anonymity; they would be an even closer substitute for cash. Any institution issuing this type of electronic purse would have to establish a general suspense account for the amount outstanding in its issued purses. In such systems, monetary value could be transferred directly between cards without the action of an intermediary.

1.2 Definition Of Terms

Smartcard is a new concept in the Nigeria Payment and Settlement System. It is therefore, necessary to start this overview with definition of some of terms used.

The following terms are in common use:

- Smartcard
- Electronic Purse
- Cardholder
- Issuer Bank
- Acquirer Bank
- Merchant
- Merchant/Transport Card
- Bank Teller Terminal (BTT)
- Point of Sale Terminal (POS)
- Hotlist

1.2.1 SmartCard

The term 'Smartcard' applies to all a technology that has become a generic term for all sorts of advanced card technology. A card is in accordance with International Organization for Standardization (ISO), a piece of nominal dimension:

85.6mm x 53.9mm x 0.76 mm

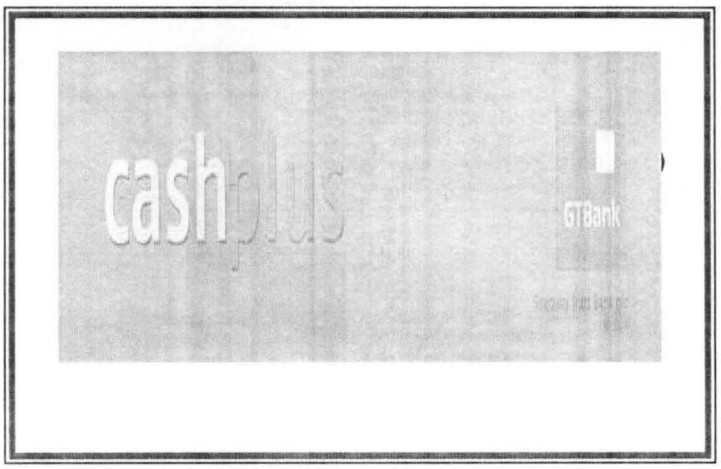
The term 'Smart' applies to a card that carries a semi-conductor chip. The chip contains basic features of a computer and thus has a small processor, storage facility,

an operating system and set(s) of stored instructions called programs. Thus, Smartcard is a card of ISO dimensions which has in-built logical ability. Smartcard range from small capacity memory only chips (mainly used for Phone cards) to large (16k bytes) memory capacity microprocessor chips and more recently, microprocessor chip with an added high speed arithmetic unit for use on public key cryptographic calculations such as Random Security Algorithm(RSA) and Data Encryption Standards.

There are basically two types of Smartcards viz: Disposable and Re-loadable cards. The **disposable card** is that is disposed off after user, example is Phone cards, while **re-loadable card** allows users to replenish the value of their card as often as they wish.

1.2.2 Electronic Purse

An electronic purse is a device (smartcard) that holds an electronic equivalent of cash. The electronic cash is commonly accepted for payment for goods and services at designated Point of Sale (POS) outlets. Values are loaded onto the card in a bank and unloaded (deducted/debited) at the Point of Sale outlets in payment for goods and services. The purse can only be used while its balance is positive.



1.2.3 Cardholder

A cardholder is the bank customer who having purchased and loaded a card in a bank will use the card in payment for goods and services.

1.2.4 Issuer Bank

Issuer bank is the bank that issues the card to its customers. It obtains the card from the Smartcard company, embosses them to identify users and thereafter, issues them to the customer.

1.2.5 Acquirer Bank

This is the bank that negotiates and enters into agreement with merchants (shop owners) to accept Smartcard(Electronic Purse) in payment for goods and services.

1.2.6 Merchant

A merchant is a service provider/trader, who has Point of Sale (POS) terminals installed in their shops, offices or outlets for Smartcard (electronic purse) transactions.

1.2.7 Merchant Card or Transport Card

A merchant card or transport card is issued to the merchant by an acquiring bank. The card is issued to sum up periodic transactions. All transactions are uploaded from the merchant POS terminal to a Bank Teller Terminal (BTT) via a merchant

card or modem if on-line. The merchant card is also used to transfer the hotlist to the merchant POS terminal.

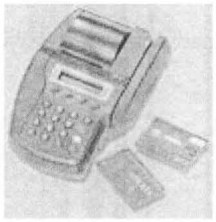
1.2.8 Bank Teller Terminal (BTT)

The Bank Teller Terminal is used to interact with cardholders and merchant cards. It will be located in the bank branches and will be operated by bank employees, who will provide services to cardholders and merchants. The Bank Teller Terminal will have an on-line connection to the Server.

The functions provided by the BTT are as follows:

- Exchange of value on card for cash.
- Loading value onto card.
- Transaction amount entry.
- Off-line PIN validation.
- Card identification and validation.
- Card balance checking and inquiry
- Print or view customer card transaction history.
- Key maintenance.
- Destroy hotlisted cards inserted in the terminal.
- Receipt printing.
- Transaction reports, summary and details.
- Upload of transactions from a merchant card.
- Download hotlist to merchant card.

- Issue card to cardholder, i.e. cardholder selects own PIN.



- **Bank Teller Terminal (BTT)**

1.2.9 Merchant Point of Sale Terminal

The merchant Point of Sale terminal is used to store payment transactions once value has been removed from a cardholder's card as payment for goods and services. Each Point of Sale device within the system will have a unique identification number. The Point of Sale terminal will support the following on-line and off-line functions:

- Exchange of value for goods and services
- Exchange of value on card for cash
- Receipt printing
- Transaction amount entry
- Off-line PIN Validation
- Destroy hotlisted card inserted on the machine
- Card identification and validation
- Upload hotlist from merchant card
- Card balance checking and inquiry
- Printing customers' card transaction history

- Key maintenance
- Transaction reports, summary and details
- Upload transaction files via modem
- Download of hotlist via modem.

- Point of Sales (POS)

1.2.10 Hotlist

Hotlist is a list of lost or stolen cards made available to bank's branches connected to the hub. When a card is reported lost or stolen, the card is hotlisted. The uploaded hotlist is then transferred to the merchant terminals through the merchant card.

1.3 History of Guaranty Trust Bank Nigeria Plc.

Guaranty Trust Bank plc was incorporated in July 1990, as a private limited liability company wholly owned by Nigerian individuals and institutions. We were licensed as a Commercial Bank in August 1990 and commenced operation in February 1991.

In September 1996, Guaranty Trust Bank plc became a publicly quoted company and won the Nigerian Stock Exchange President's Merit award that same year and again in the years 2000, 2003,2005 and 2006. The Bank was also runner-up for the quoted company of the year award in 2005. In February 2002, we obtained a Universal Banking license and were appointed a settlement bank by the Central Bank of Nigeria (CBN) in 2003.

Our quest to continue adding value to the businesses of our stakeholders has seen us emerge as a pacesetter and industry leader over the years. This is evident in our introduction of real time online banking in 1990, mobile, telephone and internet banking in 2002, Slip free banking in 2006 and the first fully interactive self service call centre; GT Connect in 2006.

Mission Statement

We are a high quality financial service provider with the urge to be the best at all times whilst adding value to all stakeholders.

Vision Statement

We are a team driven to deliver the utmost in customer services.

We are synonymous with innovation, building excellence and superior financial performance; and creating role models for society

RATINGS

Guaranty Trust Bank financial capacity to meet obligations as they fall due has been recognized by several rating agencies Agusto & Co, (one of the foremost credit rating agencies in Nigeria) has reaffirmed the Bank's triple A (Aaa) risk rating every year

for the last four years. Guaranty Trust Bank plc is one of only two banks in Nigeria with such a rating.

Fitch, one of the foremost international rating agencies assigned the Bank a double A minus (AA-) risk rating in recognition of the its strong domestic franchise, good quality assets and sound earnings record. This is the highest rating ever received by any Nigerian or West African based bank.

Standard & Poor's, another international rating agency assigned the Bank a double B minus (BB-) risk rating. The Bank is the only Nigerian financial institution with such a rating, which is the same as the Agencies Sovereign rating for Nigeria.

BELIEFS & CULTURE

The Bank maintains a culture of excellence, and goes to great lengths to ensure that customers are satisfied at all times. Our values are hinged upon professionalism, integrity and superior service delivery.

We maintain an informal but competitive environment where people call each other by their first names from entry level through to the Managing Director- no Sirs or Madams. This informal culture is not common practice in Nigeria, but true to its convictions, the non-regimented and open environment brings out the best in Guaranty Trust Bank employees.

The Bank also maintains an open door policy. This reinforces the informal atmosphere and breeds a feeling of equality. Everyone is accessible and approachable, working in open offices along side their colleagues.

The work environment is built saliently on Total Quality Management, and a thirst for excellence in every thing we do. Quality is an integral part of the Bank and its improvement is not just in the hands of a few but in the hands of every member of staff. Delivering quality is the way we know how to sustain our competitive advantage.

1.4 Problem Definition

The following problems are being encountered with the present Nigerian payment system.

- Some of the present payment system involves too much cash carrying for exchange of goods and services.
- Lack of trust on some of the payment instrument.
- Insecure payment instrument.
- Faking, forging and cloning of the payment instruments.
- Movement of large sum of cash exposes the carriers to imminent danger in the hand of hoodlums.

1.5 Objectives

The objectives of this project work are:

- To provide alternative to the present payment system that eliminates too much cash carrying for exchange of goods and services.
- To provide a secured payment instrument.
- To make payment for goods and services easier.

- To provide a reliable and trusted instrument.

1.6 Limitation

- This study is concerned with the mode of payment through electronic purse (smartcard) within Guaranty Trust Bank Nigeria Plc only.
- The payment currency is limited to only Naira and is within Nigeria.
- Of all Smartcard payment system, only electronic purse is considered.
- Due to the cost of acquiring the Bank Teller Terminal (BTT) and Point of Sales machine (POS), Guaranty Trust model were used for the test running of the software.

1.7 Methodology

This project work is done by studying and reviewing the present payment system in Nigeria, other products that are related to the electronic payment system.

Chapter 2

LITERATURE REVIEW

2.1 The Nigerian Payment System

A country's payment system comprises of all items used in payment for goods and services in the country. It is dynamic and changes over time, depending on the level of economic activities, the sophistication of the financial and banking system and level of financial literacy.

As financial transactions in Nigeria are predominantly cash-based, with attendant risks and high handling costs. The need for a more secured and convenient means of payments has led, many banks in recent years to introduce alternatives. Presently, the Nigerian payment system comprises:

- Money (Coins & Bank Notes)
- Personal Cheques
- Certified Cheques
- Bank Drafts
- Bankers Payment
- Mail Transfers
- Electronic Funds Transfer
- Automated Teller Machines
- Plastic Money/Card (Credit Cards, Smartcard)

The most recent of these payment systems is the Smartcard technology used in certain parts of the world today, with Nigeria not left out this time around. Smartcards are already gaining grounds in Nigeria through Valucard, Smartpaycard, ESCA, and Paycard.

2.2 Evolution of Smart Cards

The smart card technology originated in both France and Japan and much of the impetus for its development has come from the national governments of these two countries. In France, the Director General of Telecommunications, faced with modernizing the national telephone system in 1974, decided that using smart cards would be a good way to update its pay phone system. It was also felt that this new technology could be a key factor in responding effectively to an expected strong growth in demand for home banking and shopping services. Furthermore, the French banks were interested in smart cards because of an earlier explosive increase in the issuance of cheques and because of a problem with fraud related to their Automated Teller Machines which operated, generally speaking, in an off-line mode. Finally, the French government had invested substantially in computer research and development in the second half of the 1970s, contributing still further to the development of the smart card. In Japan, the national government placed special emphasis on the role of computer technologies in its national economic development programs. The subsequent emergence of a large computer-chip manufacturing industry in Japan also contributed to the development of smart card applications in that country.

Smart cards have found extensive applications in United State of America; with an estimated half a billion cards in use worldwide as of 1994 (Ravensbergen 1994). The diffusion of smart card payments applications has been particularly rapid in Europe and East Asia, reflecting in part the relatively higher cost of telecommunications services in those countries than in North America. At the moment, most prepaid cards are employed for single purpose transactions, although interest in the electronic purse application is growing rapidly. Many electronic purse pilot projects have been announced over the past year, both in North America and overseas.

Still more recently, there has been interest in developing payments systems for use on the Internet and other personal computer networks (Crone 1995; Holland and Cortese 1995). While some of these systems simply involve the use of bank and credit card numbers for purchases on the Internet, other systems are to include the "virtual" equivalent of an electronic purse card.

2.3 Operation Of CashPlus Product

2.3.1 CashPlus Product

The Smartcard scheme will offer a re-loadable, EMV complaint, electronic purse, branded **CashPlus**, aimed at cash substitution, by providing an electronic means of payment for goods and services at the Point of Sale.

The electronic purse will be PIN protected and receipt will be produced after each credit and debit of the purse. The last ten transactions will be stored on the card.

2.3.2 System Components

A number of separate components operate together to create the complete **CashPlus** E-purse Payment System. The components include:

- Cards
- Merchant POS terminal
- Banks Teller Terminals
- **CashPlus** Management System

2.3.3 CashPlus Security

- **Personal Identification Number (PIN)**

PIN is a set of codes (numbers) selected by any cardholder as security lock/key for access to his card either at points of loading or payment. If it is forgotten, only the issuing bank can undo the codes and give the cardholder another opportunity to select new set of codes as security to his card.

- **Personalization**

Cards will be electronically and physically personalized by issuing bank (GTBbank) at the bank's head office. Electronic personalization will involve printing the card number, cardholder name and expiry date on front of the card.

- **Information Privacy**

Information stored in any card cannot be made available to anyone except with the knowledge/consent of the owner. This is the essence of PIN, which serves as the gateway to access the card for whatever data manipulation.

- **Hotlist**

This is a document containing series of cards reported lost or stolen. This list instructs POS terminals to disregard information on such cards and to destroy such cards. The hotlist is distributed to all merchants, who in turn load the refreshed list into the POS.

- **Risks**

Risks to all participating partners have been identified, isolated and minimized to the least conceivable.

2.3.4 CashPlus Operations

The operations of the **CashPlus** Management System include the following:

- Cardholder/Merchant Registration
- Card Issue
- Card Balance
- Cash Load/Unload
- Change PIN/Unlock PIN
- Upload Transaction from BTT
- Upload Transaction from Merchant/Transport Card
- Hotlist Management
- Reports
- Users Configuration
- Bank's Liability Management

Cardholder/Merchant Registration

This module handles the registration of both New Cardholder and Merchant. Information on the Cardholder/Merchant form are extracted and input through this module.

Card Issue

Card will be issued through the cardholder's branch on application by him/her. The branch will forward the application, including customer information, the branch code and account to which the card is linked to the bank's head office. On personalization, the cards are then delivered to the branch. The customers will collect the cards at their branches where they will be required to enter their own PIN, which replace the default PIN created on the cards when they were personalized.

Card Balance

It is possible for cardholder to check his balance through the branch Bank Teller Terminal, merchant Point of Sale and card reader device.

Cash Load

Cardholders may be able to load at all branches of their bank if the branches are on-line. Once a card has been issued, the cardholder may load value onto his/her card, using the Bank Teller Terminal in the bank branch. The cardholder will insert the card into the PIN pad attached to the Bank Teller Terminal; the customer service

officer will enter the amount to be loaded. The bank's liability is adjusted less the cash loaded amount. When the transaction is completed, the value is loaded onto the card and a receipt is printed.

Cash Unload

Cardholder may decide to get cash from his card at the bank's branch. The card is inserted, the officer enter the amount to unload, after verifying, the cardholder will enter the Personal Identification Number (PIN). The system check whether the PIN entered is correct, if yes, the value is unloaded and a receipt is printed. The bank's liability is adjusted with the cash unload amount.

Change PIN/Unlock PIN

It is possible for cardholder to change his Personal Identification Number (PIN) at any time provided he knows the old PIN. If he lost his PIN there is an option called Unlock PIN that is used to introduce new PIN without asking for old PIN.

Upload Transactions from BTT/Transport Card

All transactions of Bank Teller Terminal and Point of Sale are uploaded to the central database through the Upload transactions option. This update the master files of all records and tables affected.

Hotlist Management

Issuing banks will place cards reported lost or stolen on a hotlist. Placing card on hotlist ensures that the card will be destroyed if used. Branch Bank Teller Terminal download hotlist online at least once in a week while the Merchants' Point of Sale received Hotlisted Cards through the Transport card.

Reports

The following reports are possible with the **CashPlus** card Management System.

- Daily Transaction Report
- Cardholder statement report
- Hotlisted Cards
- Active Cards
- Expired Cards

Administration

New users are first introduced to the system through the Users configuration modules. Menu are also assign to new users. Modification and deletion is done here.

Liability Management is also under this module.

2.4 Benefits Of CashPlus

2.4.1 Benefits for Bank

- Improved payment system through replacement of cash
- Reduced cash handling costs
- Improved transaction times at branches

- Reduced security risk for staff
- Improved customer service image.

2.4.2 Benefits for Merchants

- Safer and more secure payment system to customers
- A better way of documenting transactions as each of them is receipted automatically and added up at end of each business period.
- Reduction in the volume of cash handled at the shops.
- Outlet managers will have better accountability and cash pilferage will be eliminated.
- Offers opportunity to design special loyalty schemes for customers.
- Increase in customer base.
- Retention of market share and improved market positioning
- Faster transaction time
- Cost saving from reduced infrastructure equipment (Note counting machine, Note binder, Counterfeit detector, etc.)
- Enhanced corporate image
- Match competitor's offering.

2.4.3 Benefits for Customers

- Increase security
- Reduced handling of cash
- Greater convenience

2.5 Barriers to the Development of E-Purse Payment Systems

In spite of the strong interest worldwide in the electronic purse, the adoption of this innovation has been comparatively slow. There are a number of reasons for this.

- First, the market for the electronic purse, like other innovations that involve the creation of networks between the suppliers of services and their customers, needs to attain a critical mass before the purse can be used effectively. Prior to this stage, there will be considerable uncertainty among both consumers and merchants as to the potential usefulness of the product. Clearly, the benefits to consumers will rise as the new means of payment becomes acceptable to merchants, while the benefits to merchants will rise with greater usage by consumers.

- Second, lack of good telecommunications network affects the connectivity of the smart cards networks.

- The average consumer may also prove to be slow to accept the electronic purse, because cash is still such a basic feature of daily life. Many people may consider an electronic purse more complicated to use than cash and may have concerns about the risk of card or equipment failure or about difficulties in recharging an electronic purse. Others may also be bothered by the potential loss of privacy, particularly in the case of multifunction applications, where a variety of personal information might be stored on a single card.

- Lack of experience with Electronic payment systems
- Lack of suitable merchants

Chapter 3
NEW SYSTEM DESIGN

3.1 Methodology

The methodology adopted for the proposed system is the System Life Cycle approach as shown below:

3.1.1 Preliminary Survey/Study

The purpose of this survey is to establish whether there is a need for a new payment system and how cash carrying Nigeria society can be turned to cashless society.

3.1.2 Feasibility Study

The purpose of the feasibility study is to investigate the alternative to the present Nigerian payment system i.e. electronic purse payment system and how it is being operated in other countries, the cost and benefits analysis.

3.1.3 Investigation & Fact Recording

Detailed study on the present and proposed system is conducted. This is a more detailed and comprehensive than feasibility study.

3.1.4 System Analysis

Analysis of the full description of the present payment system and of the objectives of the proposed system is done here.

3.1.5 System Design

Next to the System Analysis is the Design phase i.e. the designing of the proposed system called Electronic Purse/Smartcard Payment System. A system specification is produced.

3.1.6 Implementation

This system is implemented using a modern day development tools and a robust database engine. Microsoft Visual Basic is used for the front-end, SQL Server 2000 for the backend and Crystal Report for the reporting tool.

3.1.7 Maintenance & Review

After the system is implemented and operational, it is examined to see if it has met the objectives set out in the specifications.

3.2 Input Design

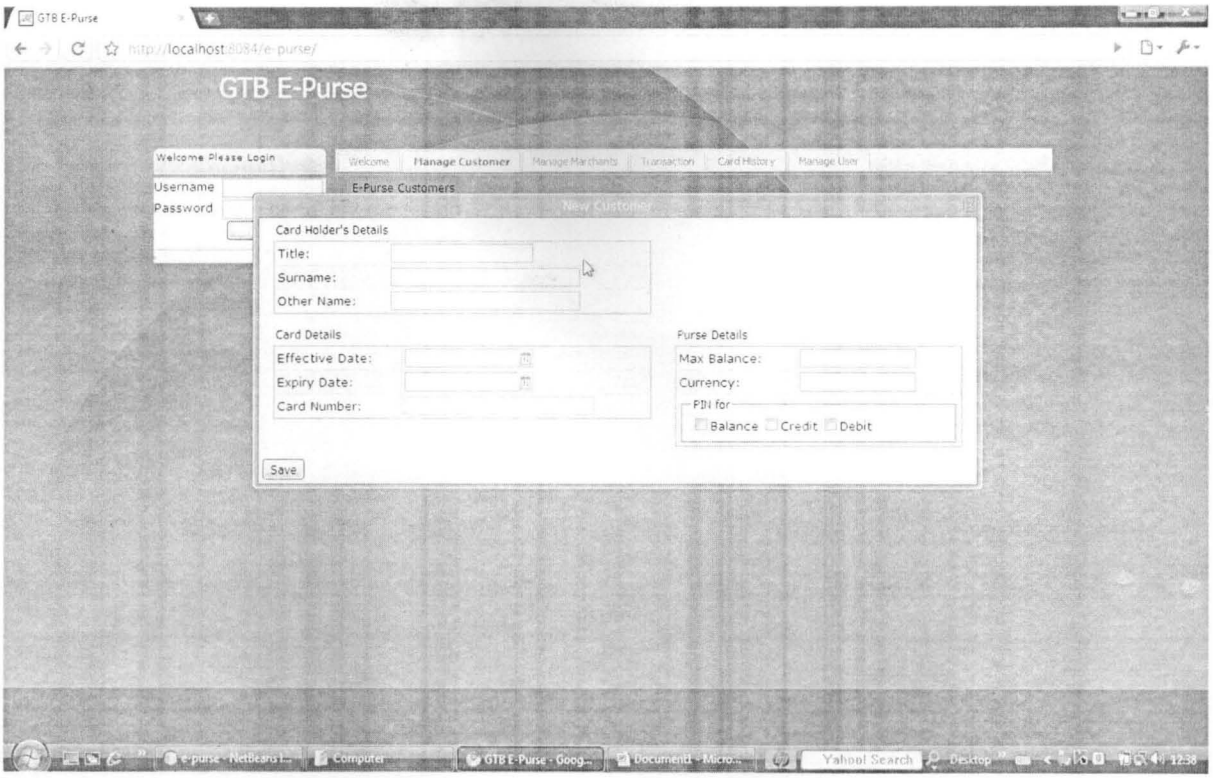
This is the means where data is transferred into the central processor. The adopted method is by using keyboard to key data directly to the system.

Basically there are 4 input Forms.

- Cardholder Details Form
- Branch/Merchant Form
- Cash Load Form
- Cash Unload Form

i) Cardholder Details Form

Purpose: This is used to capture data of new cardholder.

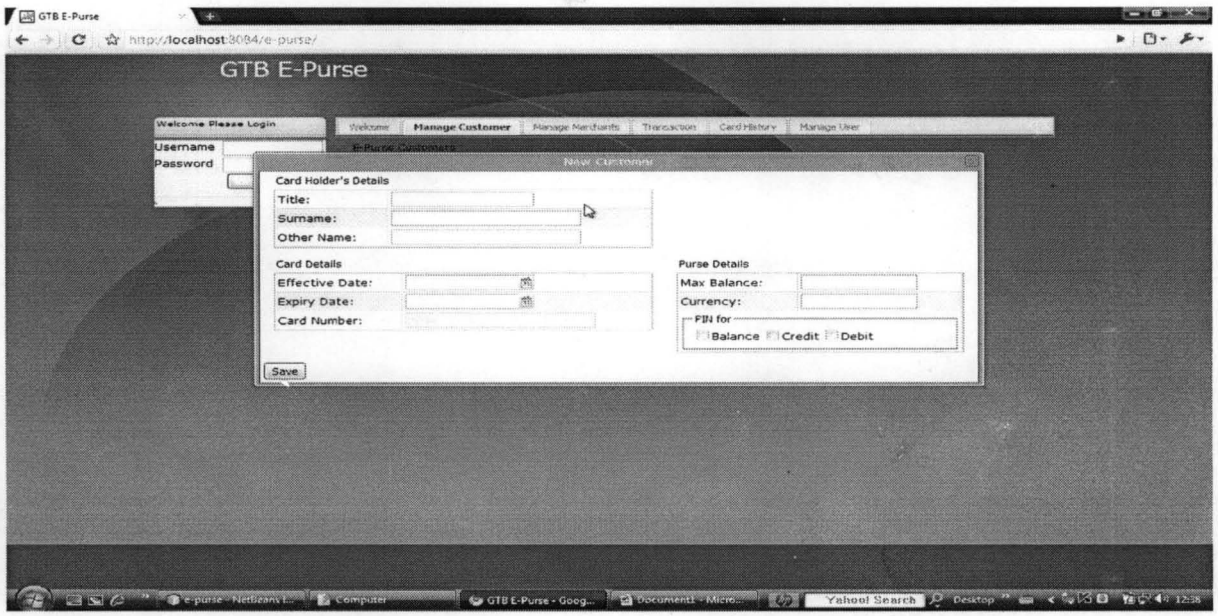


ii) **Branch/Merchant Form**

Purpose: For the registration of new branch or merchant.

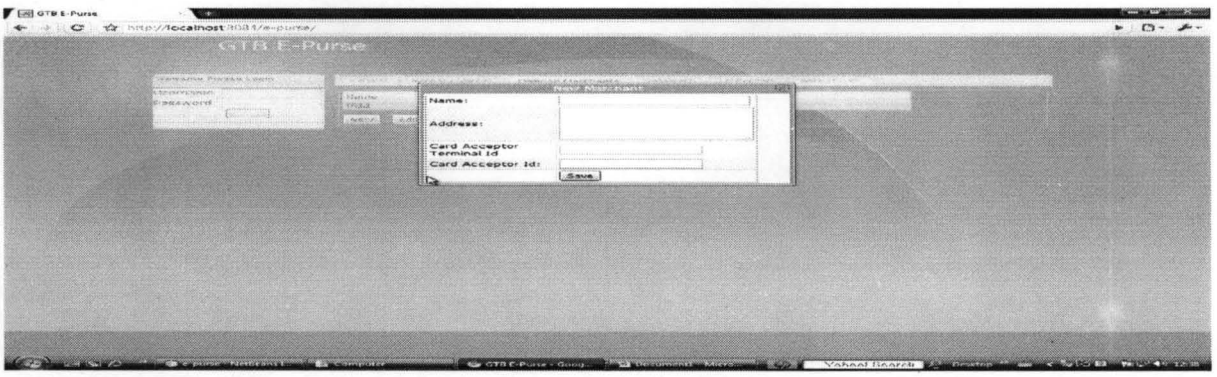
i) Cardholder Details Form

Purpose: This is used to capture data of new cardholder.



ii) Branch/Merchant Form

Purpose: For the registration of new branch or merchant.

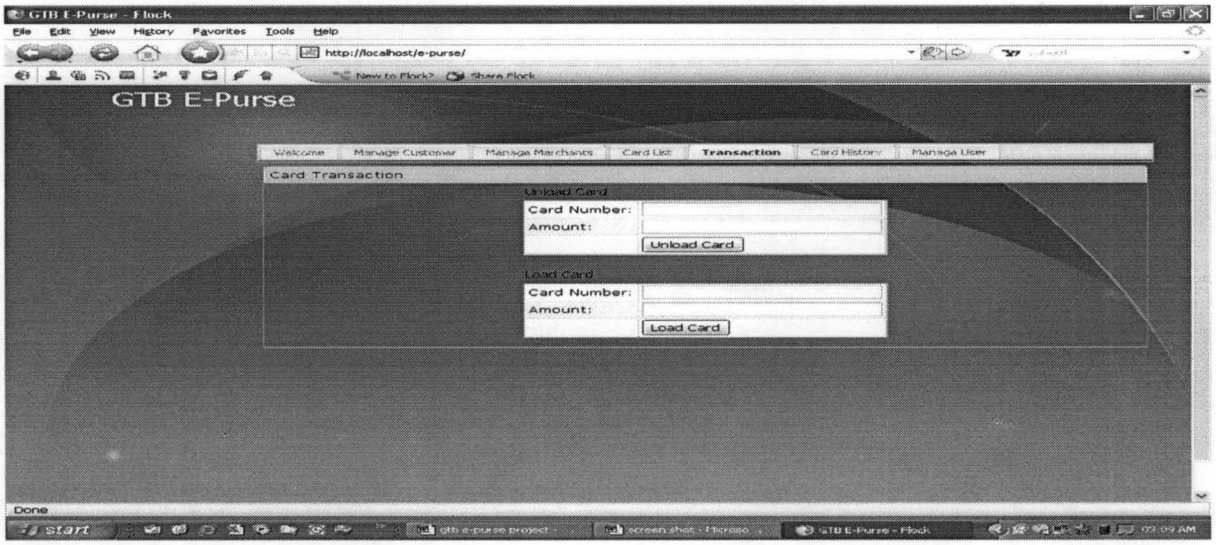


iii) Cash Load Form

Purpose: Use for loading of cash to the card.

iv) Cash UnLoad Form

Purpose: Use for unloading of cash to the card.



3.3 Output Design

Output is the means by which computer communicate result to the users for decision making. This can be through Monitor(soft copy) or Printer (hard copy).

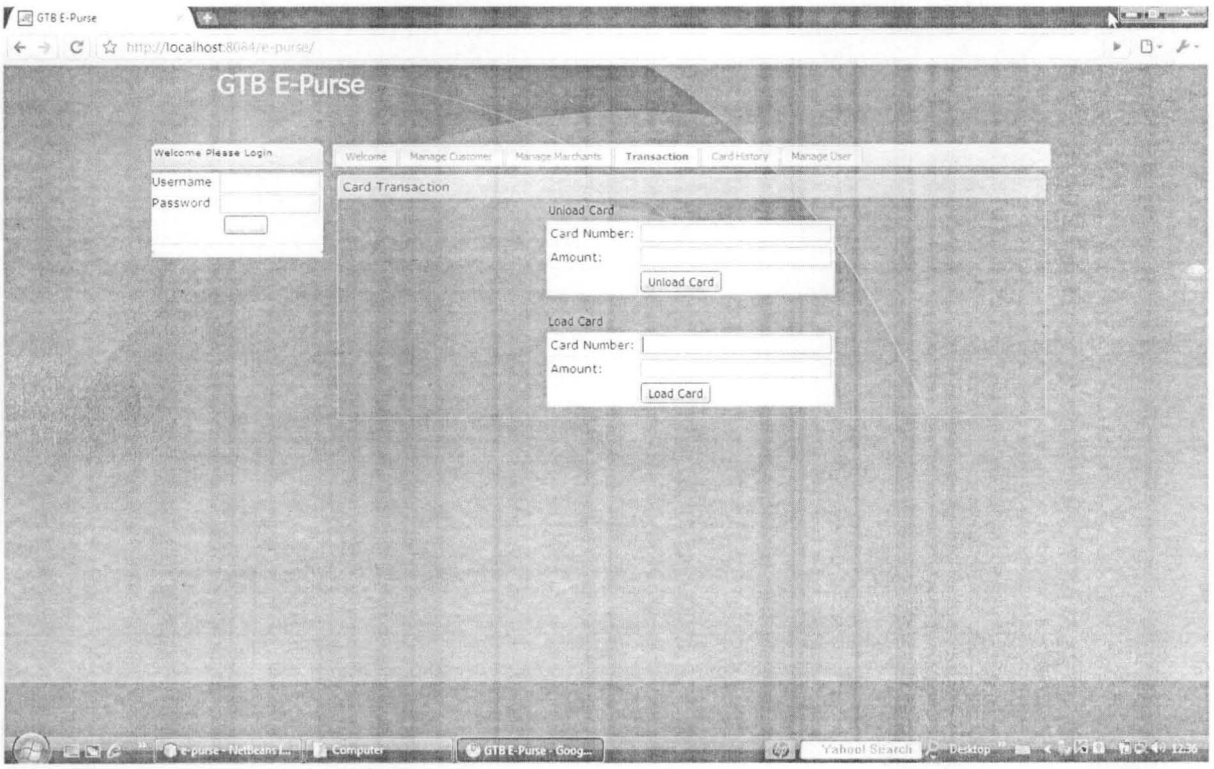
Reports available:

- Daily Transaction Report
- Card Holder Statement
- Hotlisted Cards
- Active Cards
- Expired Cards

i) Daily Transaction Report

This is a daily transaction report. It gives details of all transactions (Cash Load and Cash unload) for a day. It contains the following columns:

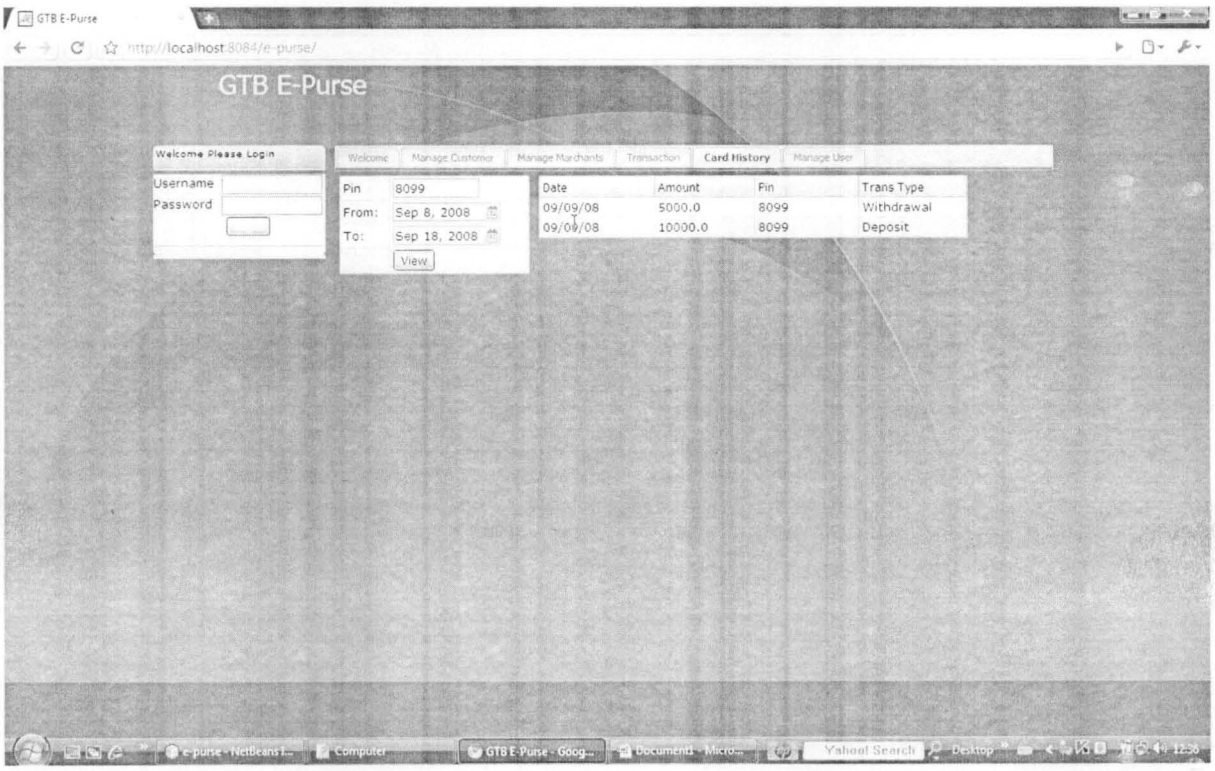
- Transaction ID
- Card Number
- Card Acceptor Terminal ID
- Card Acceptor ID
- Amount
- Transaction Type
- Date/Time of transactioni



ii) Card Holder Statement

This report shows all the transactions the cardholder has done. It consists of the following:

- Date/Time of transaction
- Transaction type
- Branch/Merchant Name
- Debit
- Credit



iii) Hotlisted Cards report

This shows the list of all hotlisted cards. It contains the following columns:

- Card Number
- Card Holder Name
- Hotlisted Date

iv) Active Cards

This is list of all active cardholders.

- Card Number
- Cardholder Name
- Expiry Date

v) Expired Cards

This is the list of all expired cardholders.

- Card Number
- Cardholder Name
- Expiry Date

3.4 File Design

This describe how the data is to be structured and physically stored on backing storage device.

DATABASE : EPURSE

FILENAME: EPURSE_DATA.MDF

DATABASE ENGINE: SQL SERVER 2000

STORAGE MEDIA: HARD DISK

- TABLE NAME: **CardHolderDetails**

  PURPOSE: This holds the details of the Cardholders.

  PRIMARY KEY : ***CardNumber***

Column Name	**Data type**	**Length**	**Description**
Title	Varchar	10	Title
FirstName	Varchar	20	First Name
MiddleName	Varchar	20	Middle Name
LastName	Varchar	20	Last Name
IDType	Varchar	15	Identification Type
IDString	Varchar	15	Identification String
Language	Varchar	10	Language
Account	Varchar	13	Account Number
EffectiveDate	Datetime	8	Effective Date
ExpiryDate	Datetime	8	Expiry Date
CardNumber	Varchar	13	Card Number
NCurrency	Varchar	16	Nigerian Currency
MaxBalance	Numeric	9	Maximum Balance
PinBalance	Varchar	1	Pin Balance
PinCredit	Varchar	1	Pin Credit
PinDebit	Varchar	1	Pin Debit
Hotlist	Varchar	1	Hotlist
HotlistedDate	Datetime	8	Hotlisted Date

- TABLE NAME: **TransactionDaily & TransactionMaster**

  PURPOSE: TransactionDaily keeps transactions pending

  the time it will be uploaded while TransactionMaster

  contains all transactions.

  PRIMARY KEY : ***TransactionID***

Column Name	Data type	Length	Description
TransactionID	Varchar	20	Transaction Identification
TransactionType	Varchar	11	Transaction Type
CardNumber	Varchar	13	Card Number
Catid	Varchar	15	Card Acceptor Terminal Identification
Caid	Varchar	15	Card Acceptor Identification
DateTime	Datetime	8	Date/Time of Transaction
Amount	Numeric	9	Amoount
UserId	varchar	12	User Identification

- TABLE NAME: **BranchMerchant**

  PURPOSE: Keeps the records of Branches and Merchants.

  PRIMARY KEY : ***Catid***

Column Name	Data type	Length	Description
BranchMerchantName	Varchar	60	Branch/Merchant Name
Address	Varchar	200	Address
Catid	Varchar	15	Card Acceptor Terminal Identification
Caid	Varchar	15	Card Acceptor Identification

- TABLE NAME: **BranchLiability**

  PURPOSE: Bank Liability entry is kept here.

Column Name	Data type	Length	Description
LiabilityAmount	Numeric	9	Liability Amount
AmendmentDate	datetime	8	Amendment Date

- TABLE NAME: **User_tab**

  PURPOSE: Keeps the records of users.

  PRIMARY KEY : ***Userid***

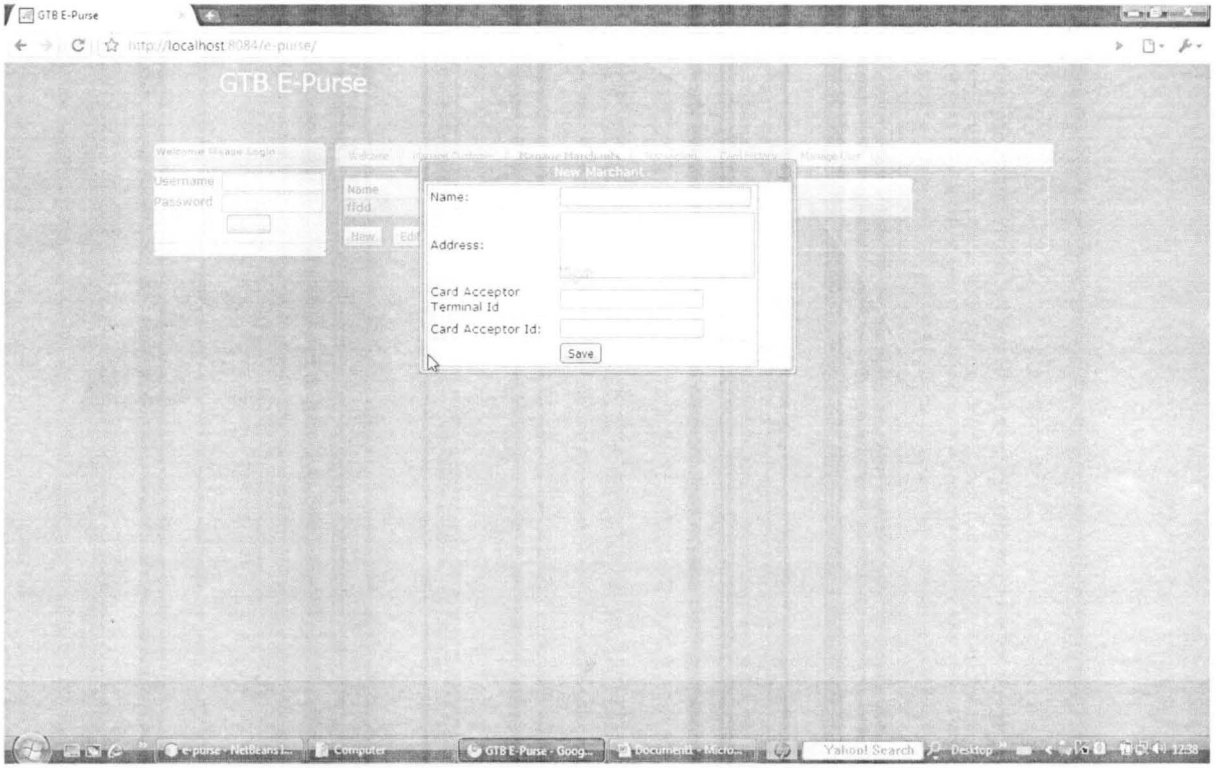
Column Name	Data type	Length	Description
User_name	Varchar	64	User Name
User_id	Varchar	12	User Identification
Staff_id	Varchar	5	Staff Identification No.
Start_date_profile	Datetime	8	Profile Start Date
End_date_profile	Datetime	8	Profile End Date
User_class	Varchar	20	User Class
User_branch	Varchar	30	User Branch
User_pass	Varchar	12	User Password
Remarks	Varchar	16	Remarks

3.5 System Specification

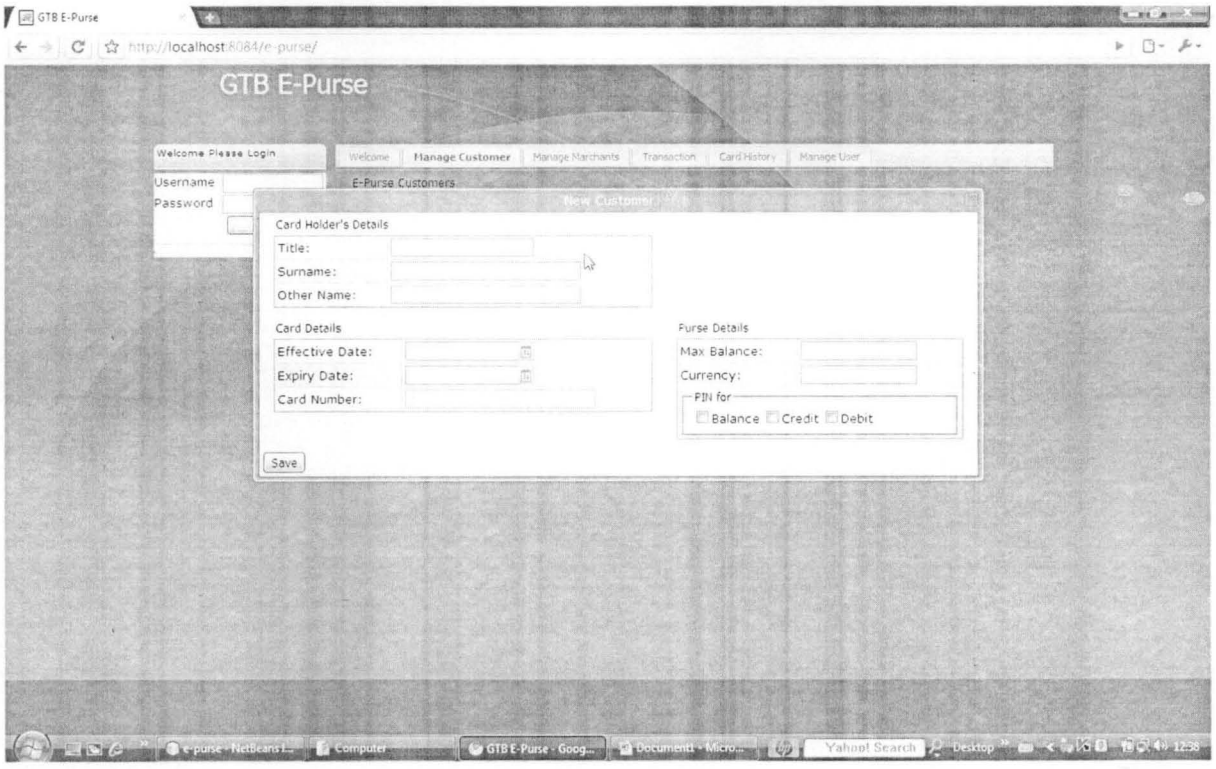
The system specification describes the new system. The software for the new is known as **GTB E-Purse Payment System**. This system is a windows based system using a menu and submenu technique.

- **REGISTRATION MENU**

This registration menu is used for the capturing of Cardholder and Branch/Merchant Data into the system.



Merchant Registration form



Customer Registration form

- **CARD OPERATIONS**

This menu handles the following operations:

  - Card Issue
  - Card Balance
  - Card Load
  - Cash Unload
  - Change PIN
  - Unlock PIN
  - Upload Transactions from BTT
  - Upload Transactions from Transport Card
  - Download Hotlist

- **REPORTS**

This produce reports for the following:

  - Daily Transaction Report
  - CardHolder Statement
  - Hotlisted Cards
  - Active Cards
  - Expired Cards

- **ADMINISTRATION**

This is used for setup of the following:

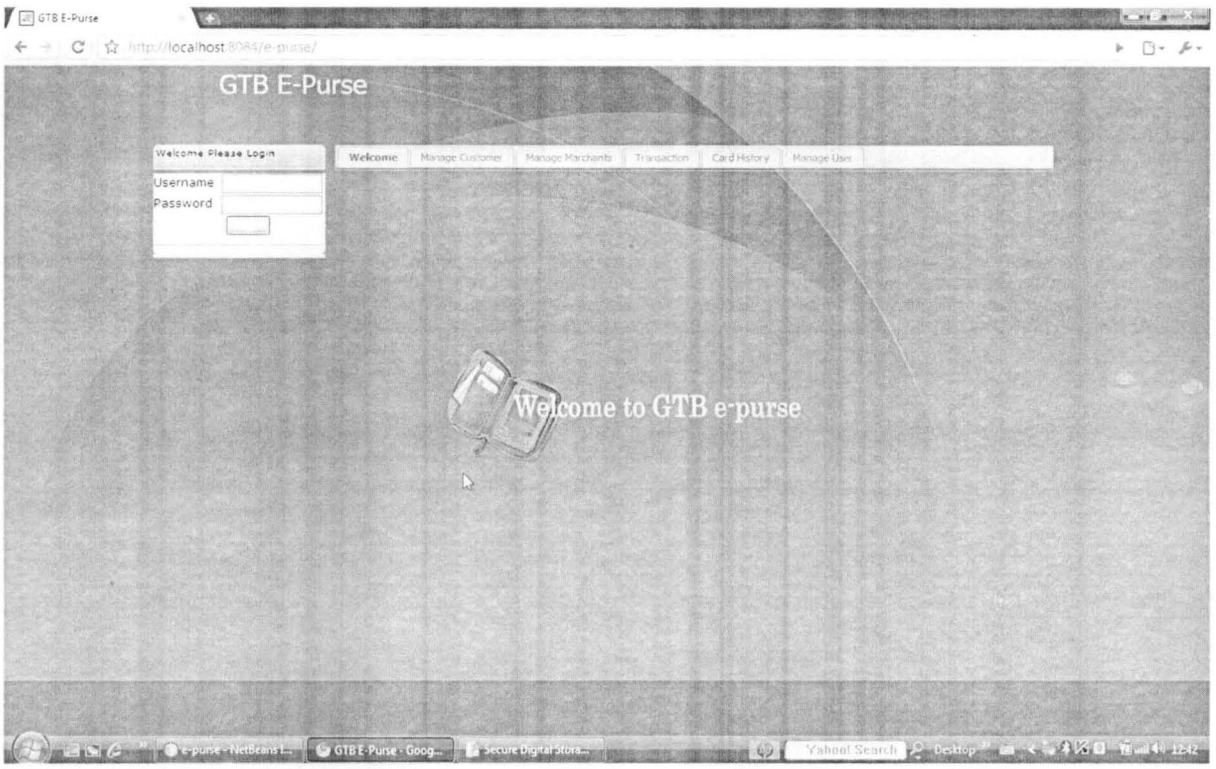
  - Users Configuration
  - Change Password
  - Bank's Liability Setup
  - Update Hotlist

3.6 System Security

Security is the protection of data against unauthorized access against accidental or intentional or destruction of data. While designing the system, the following essentials were put into mind:

- Users must be created by the System Administrator.
- Users must be positively identified by the system before they are given access to the data.
- User's action should be authorized once they have been allowed to access the system.
- Data should be auditable.
- All transactions carried users id.
- Data should be protected from fire, theft and other forms of destruction.
- Daily backup of data.

Login Screen



Users are expected to supply their Login name and Password before they can use the Cash plus card E-Purse Payment System. The system will check the data supplied against the stored data. If find it continues else it displays error message "User Unknown, Contact Administrator".

3.7 COSTS AND BENEFITS ANALYSIS

COSTS

The overall cost of producing smartcards depends on how long the cards last and how many cards are needed.

The project of smart cards production is comparable to any other stationary or security printing, in that, the higher the number of copies required, the cheaper the cost. This is so, because both the hardware and software that are installed to produce a single smart card is what is needed to produce a million copies. The only additional cost involved is the length of period of production, quantity of card and energy dissipation.

However, below is the approximate figurative analysis of the production of one single smart card. It is assumed here that, smart card facilities are just being introduced to a particular populace.

(i) Cost of carrying out feasibility study – ₦2.5m

(ii) Cost of carrying out public enlightenment – ₦1.5m

(iii) Cost of preparation of software – ₦0.5m

(iv) Cost of procurement of hardware – ₦2.0m

(v) Cost of other accessory equipment – ₦1.5m

(vi) Cost of rent/erection of stations – ₦15m

(vii) Cost of hiring personnel – ₦10m

(viii) Cost of procuring security cards – ₦5m

(ix) Cost of providing training for personnel– ₦3m

(x) Logistics - ₦2m

(xi) Miscellaneous expenses - ₦5m

Total ₦48m

Above is an approximate figure for the production of a single unit of smartcard. However, to produce 100 million smartcards may not exceed ₦55m.

From the analysis above, it will be observed that smartcard project is capital intensive. Meanwhile, the benefits accrued are enormous.

BENEFITS

Deploying Smart Card login to your network will provide the following benefits:

- It is an alternative to banknotes
- The Automated Currency solutions offers greater level of convenience to customers
- It provides increased sales opportunity and reduce operating cost of merchants and control fraud
- It carries holders health information for health care applications and provides secure vehicle for delivering government benefits such as social insurance and welfare programs
- It reduces the risks of carrying life banknotes around in exchange of services
- It protects the privacy of the cardholders
- It improves business transaction time.
- The card store information, money and or application that can be used for banking/payment, loyalty and promotion, access control, ticketing, store value, identification, parking and toll collection.
- Smart cards provide powerful authentication to prevent misuse of your resources.

Chapter 4
4.0 SYSTEM IMPLEMENTATION

4.1 Choice of Language

The choice of programming language used in this project work is Sun java system application server 9.1 (Glassfish v2)

This is because it is able to compile on any system that runs windows of any version. Since the work comprises the use of database management system and the data so stored, has the ability to grow by the day.

Sun java system application server 9.1 (Glassfish v2) is most suitable as it can handle large independent data conveniently. Independent, in that, it only acts as user interface, not having direct contact with the raw data, rather it exposes object that manipulate this data regardless of the database management system used.

4.2 Features of java programming language.

Java programming language has the following significant features namely:

1. Platform Independence: Java compilers do not produce native object code for a particular platform but rather byte code instructions for the Java Virtual Machine (JVM). Making Java code work on a particular platform is then simply a matter of writing a byte code interpreter to simulate a Java Virtual Machine (JVM), what this means is that the same compiled byte code will run unmodified on any platform that supports Java.

2. Applet Interface: In addition to being able to create stand alone applications, Java developers can create programs that can be downloaded from web page and run on a client browser.
3. Object Orientation: Java is a pure object oriented program. This means that everything in a Java program is an object and everything is descended from root object class.
4. Familiar C++ like syntax: One of the factors enabling the rapid adoption of Java is the similarity of the Java syntax to that of the popular C++ programming language.
5. Garbage Collection: Java does not require programmers to explicitly free dynamically allocated memory, this makes Java programs easier to write and less prone to memory errors.
6. Rich Standard Library: One of Java's attractive features is its standard library; the Java environment includes hundreds of classes and methods in six major functional areas.
7. Language support classes for advanced language features such as strings, arrays, threads and exception handling.
8. Networking classes to allow inter-computer communications over a local network or the Internet.
9. Input/Output classes to read and write data of many types to and from a variety of sources.
10. Applet is a class that makes it possible to create Java program that can be downloaded or run on client's servers.

11. Utility classes like a random number generator, date and time functions and container classes.
12. Abstract window Toolkit for creating platform-independent Graphical User Interface (GUI) applications.

4.3 Changeover/Conversion Procedure

This is simply moving from old system to a new system. There are various changeover methods, such as :- Direct changeover, parallel changeover and pilot / piecemeal changeover

Direct Changeover is discarding the old system completely and immediately using the new system, thus the old system is discontinued altogether and new system becomes operational immediately.

Parallel Changeover is using both old and new system until it is confirmed that, the new system performs correctly, thus the old and new systems are run concurrently using the same inputs. The outputs are compared and reasons for differences resolved. Outputs from old system continue to be distributed until the new system has proved satisfactorily. At this point the old system is discontinued and the new one takes its place.

Pilot Changeover is having only a small group of people over a period of time to establish its reliability and acceptance. Upon confirmation, the remaining section of

the populace is introduced into the new system. In this method the test period can either be run in parallel with the existing system or in direct changeover. This method is the same as the piecemeal method. It has an advantage of showing earlier, all potential implications in the new system to enable the proponent device appropriate solutions.

4.4 Software & Hardware Requirements

For successfull implementation of GTB E-purse payment system, the following equipments are required:

The following hardware resources need to be put in place in order to power the application:

1. A Pentium IV processor powered Computer
2. 80GB of hard disk space to meet up the space requirement of the system.
3. Intel processors
4. A high speed internet connection

Software requirement

1. Mysql database engine.
2. Sun java system application server 9.1 (Glassfish v2)
3. mysql connectorj (for database connectivity from the java program)
4. NetBeans Integrated development environment
5. Java development toolkit

- **Bank Teller Terminals (BTT)**
- **Point of Sales (POS) machine**
- **Printers (LaserJet)**
- **Uninterrupted Power Supply (UPS)**
- **Stabilizer**
- **`Wide Area Network`**

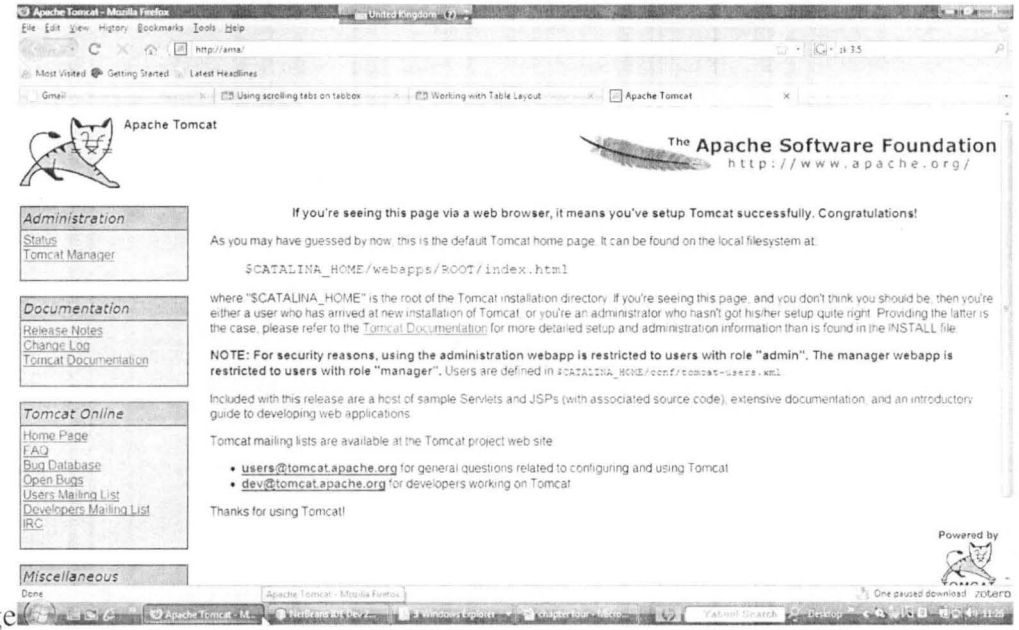
Installation procedure

1. Install tomcat web application server with an administrator password admin and user name admin.

2. Install mysql database and configure it with user name root and password admin

3. Start the tomcat web application server from the server control console

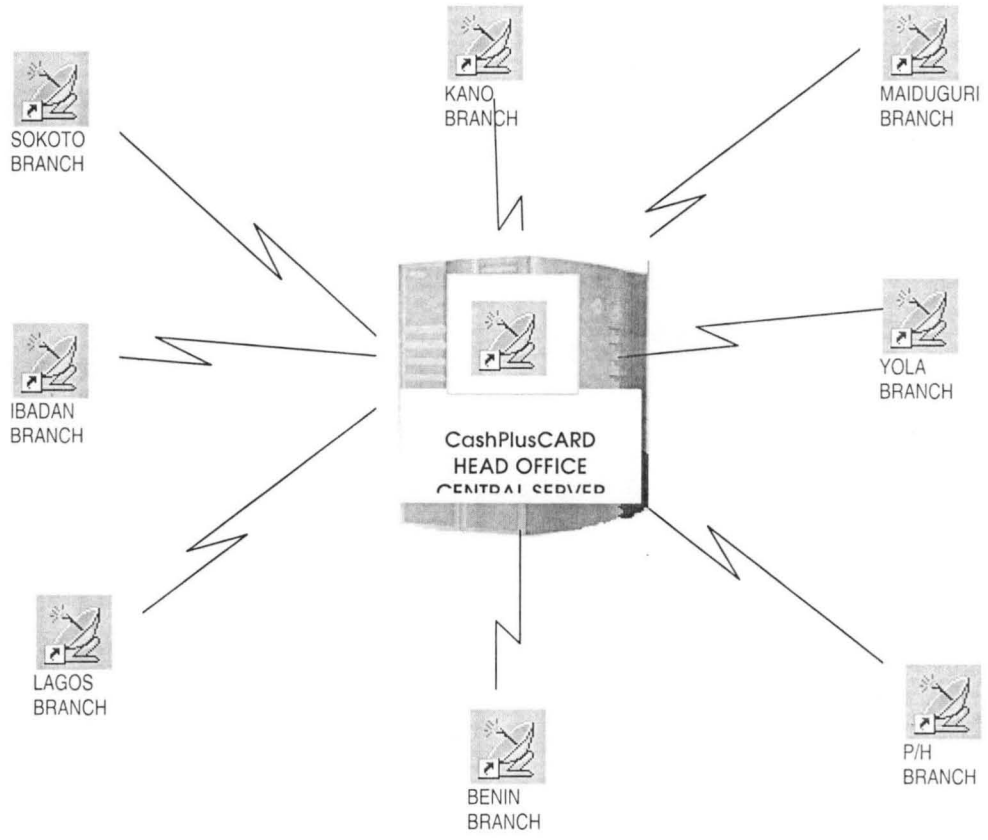
4. Open your web browser and type http://localhost to go to tomcat web server home page. Below is the tomcat home page



5. Click the tomcat manager link to logging to the tomcat web application manager

4.5 CashPluscard Communication Links

Designated branches for Guaranty Trust Bank card are linked up to the Head Office Server through VSAT.



A central server at the head office with branches connected through VSAT

4.6 Maintenance

For a smart card project to survive, a good maintenance framework must be put in place and sustained.

Foremost, trained personnel must be stationed at all the sub – stations of the project network to attend to errors on the supporting equipment.

All the supporting equipment such as encoder and decoder, servers and energy source must be placed on routine check, to ensure that they are on standby round the clock.

A smart card project, whether loaded or unloaded must be kept save in a manner that all financial transactional instruments are kept under lock and key. It must not be defaced in any form otherwise, the security print within it may be altered. Balance on a smart card must be confirmed periodically to avoid NILL balance.

Loss of smart card must be promptly reported to the issuing authority.

4.7 Training

There will be training on the use of Guaranty Trust Bank E-purse system, BTT, POS for the Bank's Tellers, POS for the Merchant, Mysql database engine and Sun java system application server 9.1 (Glassfish v2) for the Guaranty Trust Bank Administrator.

4.8 Starting up the system

When the system has been successfully installed, the system name Guaranty Trust Bank Electronic Purse appears on the start menu when clicked upon the main menu comes up after the logging has been successfully recognized by a user with a valid user identification and password. Once the main menu is on, other outputs are ready for functions.

Chapter 5

5.0 Summary

The worldwide growth in the popularity of smart card solutions is driven by the ability of smart cards (e-purse) to securely store information—for payment and other sensitive applications in a variety of environments. This security, provided by the microprocessor chip integrated into a smart card, is opening many application opportunities that were previously impractical or impossible to implement using conventional magnetic-stripe cards.

- Smart cards (e-purse) are platform for lucrative frequent-shopper loyalty programs used either independently (off line) or as integrated components of a retailer's computer system.
- Smart cards which serve as an electronic purse, acting as an electronic payment alternative to bank notes. Automated currency solutions offer greater levels of convenience to consumers, incremental sales opportunities and reduced operating costs to merchants and new service revenue streams to banks.
- Smart card applications reduce operating costs and control fraud in electronic benefits programs, carrying patient information for healthcare applications and providing a secure vehicle for delivering government benefits such as social insurance and welfare programs.
- Smart cards (e-purse) hold the promise of providing secure access to payment and other banking services from the home and other "virtual" environments, allowing consumers to securely initiate payment for goods and services over the Internet

and enabling them to reload their stored value electronic purse cards almost anywhere.

Smart cards (e-purse) are used in a variety of other applications that require greater processing power and/or more secure storage. College students use smart cards (e-purse) to pay for cafeteria and bookstore purchases and to access health, recreation and other services; commuters use smart cards (e-purse) to pay tolls and parking fees; and parents use smart cards (e-purse) to pay for child care.

As the number of smart card (e-purse) applications grows, consumer awareness of the convenience provided by smart card–based applications is driving the adoption of new ways to conduct business. This familiarity increases interest and further accelerates the expansion of opportunities to apply smart card technology.

The growth in the number and variety of smart card applications is also being accelerated by the current flurry of activity within business and industry as banks, credit card companies, equipment manufacturers, processors and others increase their commitment to the use of smart cards (e-purse). This commitment is perhaps best exemplified by the number of partnerships and industry forums devoted to the development and promotion of smart card technology.

5.1 CONCLUSION

In conclusion, all signs point to the rapidly expanded application of smart card technology. Industry leaders are positioning themselves now to profit from the application of this technology through various partnerships and alliances with organizations that are the innovators and principal drivers in the use of smart cards (e-purse).

REFERENCES

1. ALAN DANIELS & DON YEATES, (1989), **Basic Systems Analysis**, 3rd edition, Singapore, ELBS.

2. FERNANDO G. GUENENO & CARLOS DUARO ROJAS, (2001), **SQL Server 2000 Programming by Example**, 1st edition, New York, Que.

3. GERALD STUBER, **The Electronic Purse – An Overview of Recent Developments and Policy Issues**, http://www.bankofcanada.ca/

4. JOHN WENNINGER & DAVID LASTER, **The Electronic Purse**, http://www.ny.frb.org/

5. http://www..com

6. http://www.cenbank.org/

7. http://www.fsbint.com/

8. http://www.smartcardalliance.org/

9. http://www.smartcard.co.uk/

10. http://www.smartcardclub.co.uk/

11. http://www.valucardnigeria.com/

APPENDIX A

PROGRAM LISTING

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.naf.account;

import java.io.Serializable;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;

/**
 *
 * @author aliyu
 */
@Entity
public class Client implements Serializable {
    @OneToOne(mappedBy = "client",cascade=CascadeType.ALL)
    private Purse purse;
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String surname;
    private String othername;
    private String title;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof Client)) {
            return false;
        }
        Client other = (Client) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
```

```
}

@Override
public String toString() {
    return surname+" "+othername;
}

/**
 * @return the purse
 */
public Purse getPurse() {
    return purse;
}

/**
 * @param purse the purse to set
 */
public void setPurse(Purse purse) {
    this.purse = purse;
}

/**
 * @return the surname
 */
public String getSurname() {
    return surname;
}

/**
 * @param surname the surname to set
 */
public void setSurname(String surname) {
    this.surname = surname;
}

/**
 * @return the othername
 */
public String getOthername() {
    return othername;
}

/**
 * @param othername the othername to set
 */
public void setOthername(String othername) {
    this.othername = othername;
}

/**
 * @return the title
 */
public String getTitle() {
    return title;
}

/**
 * @param title the title to set
 */
public void setTitle(String title) {
```

```
        this.title = title;
    }

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.naf.account;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 *
 * @author aliyu
 */
@Entity
public class Marchant implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String address;
    private String cardAcceptorTerminalId;
    private String cardAcceptorId;
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof Marchant)) {
            return false;
        }
        Marchant other = (Marchant) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "org.naf.account.Marchant[id=" + id + "]";
```

```
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the address
     */
    public String getAddress() {
        return address;
    }

    /**
     * @param address the address to set
     */
    public void setAddress(String address) {
        this.address = address;
    }

    /**
     * @return the cardAcceptorTerminalId
     */
    public String getCardAcceptorTerminalId() {
        return cardAcceptorTerminalId;
    }

    /**
     * @param cardAcceptorTerminalId the cardAcceptorTerminalId to set
     */
    public void setCardAcceptorTerminalId(String cardAcceptorTerminalId)
{
        this.cardAcceptorTerminalId = cardAcceptorTerminalId;
    }

    /**
     * @return the cardAcceptorId
     */
    public String getCardAcceptorId() {
        return cardAcceptorId;
    }

    /**
     * @param cardAcceptorId the cardAcceptorId to set
     */
    public void setCardAcceptorId(String cardAcceptorId) {
        this.cardAcceptorId = cardAcceptorId;
    }

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.naf.account;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Temporal;

/**
 *
 * @author aliyu
 */
@Entity
public class Purse implements Serializable {
    private static final long serialVersionUID = 1L;
    @OneToOne
    private Client client;
    @Id
    private String pin;
    private String currency;
    private double maxBalance;
    @Temporal(value = javax.persistence.TemporalType.DATE)
    private Date effectiveDate;
    private boolean isPinForBalance;
    private boolean isPinForCredit;
    private boolean isPinForDebit;
    @Temporal(value = javax.persistence.TemporalType.DATE)
    private Date expiryDate;

    /**
     * @return the client
     */
    public Client getClient() {
        return client;
    }

    /**
     * @param client the client to set
     */
    public void setClient(Client client) {
        this.client = client;
    }

    /**
     * @return the pin
     */
    public String getPin() {
        return pin;
    }

    /**
     * @param pin the pin to set
```

```
     */
    public void setPin(String pin) {
        this.pin = pin;
    }

    /**
     * @return the currency
     */
    public String getCurrency() {
        return currency;
    }

    /**
     * @param currency the currency to set
     */
    public void setCurrency(String currency) {
        this.currency = currency;
    }

    /**
     * @return the maxBalance
     */
    public double getMaxBalance() {
        return maxBalance;
    }

    /**
     * @param maxBalance the maxBalance to set
     */
    public void setMaxBalance(double maxBalance) {
        this.maxBalance = maxBalance;
    }

    /**
     * @return the effectiveDate
     */
    public Date getEffectiveDate() {
        return effectiveDate;
    }

    /**
     * @param effectiveDate the effectiveDate to set
     */
    public void setEffectiveDate(Date effectiveDate) {
        this.effectiveDate = effectiveDate;
    }

    /**
     * @return the isPinForBalance
     */
    public boolean isIsPinForBalance() {
        return isPinForBalance;
    }

    /**
     * @param isPinForBalance the isPinForBalance to set
     */
    public void setIsPinForBalance(boolean isPinForBalance) {
        this.isPinForBalance = isPinForBalance;
    }
```

```
    /**
     * @return the isPinForCredit
     */
    public boolean isIsPinForCredit() {
        return isPinForCredit;
    }

    /**
     * @param isPinForCredit the isPinForCredit to set
     */
    public void setIsPinForCredit(boolean isPinForCredit) {
        this.isPinForCredit = isPinForCredit;
    }

    /**
     * @return the isPinForDebit
     */
    public boolean isIsPinForDebit() {
        return isPinForDebit;
    }

    /**
     * @param isPinForDebit the isPinForDebit to set
     */
    public void setIsPinForDebit(boolean isPinForDebit) {
        this.isPinForDebit = isPinForDebit;
    }

    /**
     * @return the expiryDate
     */
    public Date getExpiryDate() {
        return expiryDate;
    }

    /**
     * @param expiryDate the expiryDate to set
     */
    public void setExpiryDate(Date expiryDate) {
        this.expiryDate = expiryDate;
    }

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.naf.account;

import java.io.Serializable;
import java.util.Date;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.Temporal;
import org.naf.auth.User;

/**
 *
 * @author aliyu
 */
@Entity
public class Tx implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Temporal(value = javax.persistence.TemporalType.DATE)
    private Date txDate;
    private double amount;
    private String cardnumber;
    private String type;
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof Tx)) {
            return false;
        }
        Tx other = (Tx) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
```

```
}

@Override
public String toString() {
    return "org.naf.account.Transaction[id=" + id + "]";
}

/**
 * @return the txDate
 */
public Date getTxDate() {
    return txDate;
}

/**
 * @param txDate the txDate to set
 */
public void setTxDate(Date txDate) {
    this.txDate = txDate;
}

/**
 * @return the amount
 */
public double getAmount() {
    return amount;
}

/**
 * @param amount the amount to set
 */
public void setAmount(double amount) {
    this.amount = amount;
}

/**
 * @return the cardnumber
 */
public String getCardnumber() {
    return cardnumber;
}

/**
 * @param cardnumber the cardnumber to set
 */
public void setCardnumber(String cardnumber) {
    this.cardnumber = cardnumber;
}

/**
 * @return the type
 */
public String getType() {
    return type;
}

/**
 * @param type the type to set
 */
public void setType(String type) {
```

```
        this.type = type;
    }

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.naf.auth;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 *
 * @author aliyu
 */
@Entity
public class Department implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String title;
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof Department)) {
            return false;
        }
        Department other = (Department) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "org.naf.auth.Department[id=" + id + "]";
    }

    public String getTitle() {
```

```
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.naf.auth;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import org.naf.util.UserStatus;

/**
 *
 * @author aliyu
 */
@Entity(name="Users")
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String userName;
    private String password;
    private String firstName;
    private String lastName;
    private UserStatus status;
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id
fields are not set
        if (!(object instanceof User)) {
            return false;
        }
        User other = (User) object;
        if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
```

```
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "org.naf.auth.User[id=" + id + "]";
    }


    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    /**
     * @return the status
     */
    public UserStatus getStatus() {
        return status;
    }

    /**
     * @param status the status to set
     */
    public void setStatus(UserStatus status) {
        this.status = status;
    }

}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package org.naf.controllers;

import java.util.logging.Level;
import java.util.logging.Logger;
import org.naf.auth.User;
import org.naf.dao.facade.LoginService;
import org.naf.dao.facade.UserJpaController;
import org.naf.dao.facade.exceptions.NonexistentEntityException;
import org.naf.dao.interfaces.LoginManager;
import org.naf.exception.NafException;
import org.naf.util.PasswordManager;
import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.util.GenericForwardComposer;
import org.zkoss.zul.Button;
import org.zkoss.zul.Include;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

/**
 *
 * @author aliyu
 */
public class AppView extends GenericForwardComposer {

    private Window mainUI;
    private Window passUI;
    private User user;

    public void onCreate$appView(Event evt) {
        mainUI = (Window) evt.getTarget();
        showLogin();
    }
//
//    public void doAfterCompose(Component comp) throws Exception {
//        mainUI = (Window) comp;
//        mainUI.addEventListener("onLoginRequest", new EventListener()
{
//
//            public void onEvent(Event event) throws Exception {
//                doLogin();
//            }
//        });
//        mainUI.addEventListener("onLogoutRequest", new EventListener()
{
//
//            public void onEvent(Event event) throws Exception {
//                doLogout();
//            }
//        });
//        mainUI.addEventListener("onPasswordChangeRequest", new
EventListener() {
//
//            public void onEvent(Event event) throws Exception {
```

```
//                intPasswordChange();
//            }
//        });
//        showSplashScreen();
//    }

    private void doLogout() {
        try {
            int response = Messagebox.show("Are you sure You want to
logout", "Confirm Logout", Messagebox.YES | Messagebox.NO,
Messagebox.INFORMATION);
            if (response == Messagebox.NO) {
                return;
            }
            mainUI.getDesktop().getSession().invalidate();
            mainUI.getDesktop().getExecution().sendRedirect("/");
        } catch (InterruptedException ex) {
            Logger.getLogger(AppView.class.getName()).log(Level.SEVERE,
null, ex);
        }
    }

    private void doLogin() {
        try {
            Textbox username = (Textbox) passUI.getFellow("userid");
            Textbox password = (Textbox) passUI.getFellow("pass");
            LoginManager loginService = new LoginService();
            user = loginService.loginUser(username.getValue(),
password.getValue());
            mainUI.getDesktop().getSession().setAttribute("user", user);
            passUI.detach();
            return;
        } catch (NafException ex) {
            try {
                Logger.getLogger(AppView.class.getName()).log
(Level.SEVERE, null, ex);
                Messagebox.show(ex.getMessage(), "login error",
Messagebox.OK, Messagebox.ERROR);
                return;
            } catch (InterruptedException ex1) {
                Logger.getLogger(AppView.class.getName()).log
(Level.SEVERE, null, ex1);
                return;
            }
        }
    }

    private void showLogin() {
        passUI = (Window) mainUI.getDesktop().getExecution
().createComponents("pages/fragments/login.zul", null, null);
        Button btn = (Button) passUI.getFellow("loginbtn");
        btn.addEventListener("onClick", new EventListener() {

            public void onEvent(Event event) throws Exception {
                loginUser();
            }
        });
        passUI.doHighlighted();
    }
```

```
    private void loginUser() {
        doLogin();

    }

    private void showSplashScreen() {
        Include ic = (Include) mainUI.getFellow("nextframe");
        ic.setSrc(null);
        ic.setSrc("splash.zul");
    }

    private void showUserMenu(User user) {
    }

    private void show(String url) {
        Include ic = (Include) mainUI.getFellow("nextframe");
        ic.setSrc(null);
        ic.setSrc("pages/" + url + ".zul");
    //Messagebox.show("ok... " + id);

    }

    private void intPasswordChange() {
        passUI = (Window) mainUI.getDesktop().getExecution
().createComponents("WEB-INF/password.zul", null, null);
        Button btn = (Button) passUI.getFellow("passbtn");
        btn.addEventListener("onClick", new EventListener() {

            public void onEvent(Event event) throws Exception {
                changePassword();
            }
        });
        passUI.doHighlighted();
    }

    private void changePassword() {
//        user=(User) ntiUI.getDesktop().getSession().getAttribute
("user");
        Textbox oldpass = (Textbox) passUI.getFellow("oldpass");
        Textbox npass = (Textbox) passUI.getFellow("npass");
        Textbox cpass = (Textbox) passUI.getFellow("cpass");
        if (cpass.getValue().equalsIgnoreCase(npass.getValue())) {
            PasswordManager pm = new PasswordManager();
            if (pm.isPasswordValid(oldpass.getValue(), user.getPassword
())) {
                try {
                    user.setPassword(pm.encryptPassword(npass.getValue
()));
                    UserJpaController facade = new UserJpaController();
                    facade.edit(user);
                    Messagebox.show("your password change
successfully..", "password change", Messagebox.OK, Messagebox.ERROR);
                    passUI.detach();
                    return;
                } catch (NonexistentEntityException ex) {
                    Logger.getLogger(AppView.class.getName()).log
(Level.SEVERE, null, ex);
                } catch (Exception ex) {
                    Logger.getLogger(AppView.class.getName()).log
(Level.SEVERE, null, ex);
```

```
                }
            } else {
                try {
                    Messagebox.show("your old password does not match
the supplied old password try again..", "old password mismatch error",
Messagebox.OK, Messagebox.ERROR);
                    return;
                } catch (InterruptedException ex) {
                    Logger.getLogger(AppView.class.getName()).log
(Level.SEVERE, null, ex);
                    return;
                }
            }
        } else {
            try {
                Messagebox.show("you new password is not same as the
confirm password...", "password mismatch error", Messagebox.OK,
Messagebox.ERROR);
                return;
            } catch (InterruptedException ex) {
                Logger.getLogger(AppView.class.getName()).log
(Level.SEVERE, null, ex);
                return;
            }
        }
    }
}
```

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.naf.controllers;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.naf.account.Client;
import org.naf.account.Purse;
import org.naf.dao.facade.ClientFacade;
import org.naf.dao.interfaces.ClientManager;
import org.naf.renderers.ClientRenderer;
import org.naf.util.PinGenerator;
import org.zkoss.zk.ui.event.Event;
import org.zkoss.zk.ui.event.EventListener;
import org.zkoss.zk.ui.util.GenericForwardComposer;
import org.zkoss.zul.Button;
import org.zkoss.zul.Checkbox;
import org.zkoss.zul.Datebox;
import org.zkoss.zul.Doublebox;
import org.zkoss.zul.ListModel;
import org.zkoss.zul.ListModelList;
import org.zkoss.zul.Listbox;
import org.zkoss.zul.Listitem;
import org.zkoss.zul.ListitemRenderer;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;
import org.zkoss.zul.Window;

/**
 *
 * @author aliyu
 */
public class ClientsUI extends GenericForwardComposer{
    private Window clientsUI;
    private Window clientUI;
    private List<Client> clients=new ArrayList();

    public void onCreate$clientsUI(Event evt){
        clientsUI=(Window) evt.getTarget();
        fillClientList();
    }
    public void onClick$clientBTN(Event evt){
        showClientForm();
    }
    public void onClick$clientDeleteBTN(Event evt){
        Listbox list=(Listbox) clientsUI.getFellow("clientlist");
        Listitem item=list.getSelectedItem();
        if(item!=null){
            Client client=(Client) item.getValue();
            ClientManager facade=new ClientFacade();
            facade.deleteClient(client);
            fillClientList();
        }
    }
    private void deleteClient() {
```

```
        Listbox list = (Listbox) clientsUI.getFellow("clientlist");
        Listitem item = list.getSelectedItem();
        if (item != null) {
            Client s = (Client) item.getValue();
            ClientManager facade = new ClientFacade();
            clients.remove(s);
            facade.deleteClient(s);
            refreshClientList();
        } else {
            try {
                Messagebox.show("Please select supplier to delete..",
"supplier error", Messagebox.OK, Messagebox.ERROR);
            } catch (InterruptedException ex) {
                Logger.getLogger(ClientUI.class.getName()).log
(Level.SEVERE, null, ex);
            }
        }
    }


    private void fillClientList() {
        ClientManager facade = new ClientFacade();
        clients = facade.getClients();
        refreshClientList();
    }

    private void refreshClientList() {
        Listbox list = (Listbox) clientsUI.getFellow("clientlist");
        ListModel model = new ListModelList(clients);
        ListitemRenderer rnd = new ClientRenderer();
        list.setModel(model);
        list.setItemRenderer(rnd);
    }
    private void addClient() {
        try {
            Textbox title = (Textbox) clientUI.getFellow("title");
            Textbox name = (Textbox) clientUI.getFellow("surname");
            Textbox othername = (Textbox) clientUI.getFellow
("othername");
            Textbox cardnumber = (Textbox) clientUI.getFellow
("cardnumber");
            Datebox edate = (Datebox) clientUI.getFellow
("effectivedate");
            Datebox exdate = (Datebox) clientUI.getFellow("expirydate");
            Doublebox balance = (Doublebox) clientUI.getFellow
("balance");
            Textbox currency = (Textbox) clientUI.getFellow("currency");
            Checkbox pinbalance = (Checkbox) clientUI.getFellow
("pinbalance");
            Checkbox pindebit = (Checkbox) clientUI.getFellow
("pindebit");
            Checkbox pincredit = (Checkbox) clientUI.getFellow
("pincredit");
            Purse p = new Purse();
            p.setCurrency(currency.getValue());
            p.setEffectiveDate(edate.getValue());
            p.setExpiryDate(exdate.getValue());
            p.setMaxBalance(balance.getValue());
            p.setIsPinForBalance(pinbalance.isChecked());
            p.setIsPinForCredit(pincredit.isChecked());
```

```
            p.setIsPinForDebit(pindebit.isChecked());
            Client c = new Client();
            c.setOthername(othername.getValue());
            c.setSurname(name.getValue());
            c.setTitle(title.getValue());
            c.setPurse(p);
            p.setClient(c);
            p.setPin(cardnumber.getValue());
            ClientManager cm = new ClientFacade();
            c = cm.addClient(c);
            fillClientList();
            Messagebox.show("Customer registered successfully customer
        e is : " + p.getPin(), "Customer registration", Messagebox.OK,
       box.INFORMATION);
            clientUI.detach();
        } catch (InterruptedException ex) {
            Logger.getLogger(ClientUI.class.getName()).log(Level.SEVERE,
      ex);
        }
    }

    private void showClientForm() {
        if(clientUI!=null){
            clientUI.detach();
        }
        clientUI=(Window) clientsUI.getDesktop().getExecution
().createComponents("pages/fragments/client.zul",null,null);
        Button btn=(Button) clientUI.getFellow("nclientBTN");
        btn.addEventListener("onClick",new EventListener() {

            public void onEvent(Event event) throws Exception {
                addClient();
            }
        });
        Textbox cardid=(Textbox) clientUI.getFellow("cardnumber");
        PinGenerator pg=new PinGenerator();
        cardid.setValue(pg.getUniquePIN());
        cardid.setDisabled(true);
        clientUI.setPosition("center");
        clientUI.doOverlapped();
    }
}
```