UNIVERSITY OF ZIMBABWE

FACULTY OF ENGINEERING

ELECTRICAL ENGINEERING

A project report submitted in partial fulfilment of the requirements for the
BACHELOR OF SCIENCE HONOURS DEGREE IN ELECTRICAL ENGINEERING

ELECTRONIC COMMERCE ISO8583 PAYMENT GATEWAY (TELONE ZIMBABWE)

TERRENCE T MUNYUNGUMA R138279J
Year Of Study: 3rd Year **Email**: ttmunyunguma@gmail.com
Supervisor: ENG S CHARI

Abstract

The business to business or business to customer aspect of electronic commerce has become a very important part of running a business in this technology era. Most businesses with brick and mortar stores have seen the usefulness of migrating to online sales to take a bite of the large market base, among this companies is TelOne Zimbabwe.

The whole purpose of building an online business is to make money online, hence the need for a payment gateway to process those payments. TelOne wishes to implement their own payment gateway in their online commerce website, which is the sole purpose of this project.

The development of an online payment gateway is a hectic process which requires familiarization with complex technologies like JPOS for switch implementation and routing processes, and JAVA EEAPI's for client and server side business logic.

This is a project to develop a safe and secure payment system to handle credit card data for online shoppers on the TelOne commerce website.

Acknowledgments

I would like to give special thanks to my supervisor, Engineer Samuel Chari for the invaluable help and guidance throughout this project. Also it would be remise not to mention Alejandro P Revilla, the JPOS User's community and Stack Overflow Community for their help as well. And finally my Father (IN Munyunguma), thank you for the support.

Contents

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

1.1 Project Overview

On average, approximately 70% of online shoppers abandon their shopping cart during the checkout process. This means that after having gone through the trouble of finding the right product and adding it to their cart, a whopping 2 out of almost every 3 buyers will choose to abandon their purchase. This is that point where most online businesses lose a lot of money.

Electronic commerce is the process of buying and selling products or services on the net. Online shopping is increasingly becoming popular because of the ease of use and convenience for customers. In Zimbabwe, companies like Ownai, Foodworld Zimbabwe and 10ngah are already enjoying the sweet produce of online business. E-business activities can be directed at consumers or other businesses.
This project will focus on producing a robust, secure, user friendly payment system in direct competition to the existing companies. The gateway will be integrated into the TelOne Online commerce platform.

1.2 Background

Business to Consumer (B2C) involves the online sales of goods, services and provision of information directly to consumers. Business to Business (B2B) is the online exchange of products, facilities, or information between businesses. An e-commerce payment application ensures the acceptance of electronic payment for over the net transactions. Also referred to as a sample of Electronic Data Interchange (EDI), electronic commerce payment facilities have become increasingly famous due to the worldwide use of the internet-based shopping and banking. Over the years, methods It would be inconvenient for an online merchant to retail without supporting credit and debit cards as they are the industry's 'it' technology. Increased security concerns comprise of use of the card verification number (CVN) which alerts on fraud by comparing the verification number extracted from the signature strip on the back of the card with the data on file with the card-holder issuing bank. Also online retailers have to comply with tough rules set up by the credit and debit card issuers (Visa and MasterCard). This implies that merchants must have security procedures in place to make sure that transactions are more protected. This comprise of having a certificate from an authorized certification authority (CA) who provides PKI (Public-Key Infrastructure) for securing credit and debit card transactions, and implementing standards such as the ISO8583.

1.3 Problem statement

TelOne needs an ISO8583 payment gateway for an online goods store. This should handle VISA and MasterCard payments integrated with the bank servers.

1.4 Aim

To develop a safe and secure payment gateway system using existing API's for the TelOne e-commerce platform

1.5 Objectives

- To develop a general purpose e-commerce payment system where any TelOne products can be bought from the comfort of your home through the internet.
- Develop a system that handles payments initially via VISA and MasterCard.
- To develop a scalable system that can be developed to incorporate other payment gateways, bank RTGS, mobile money messaging, etc. as time goes on.
- To develop a robust and secure system that handle people's monies online without fear of fishing web-worms or fraud risks.

1.6 Justification

a. convenience for online sales – the online payment system allows convenience for selling goods online
b. Automatic – the online payments would be automatic, which is also a convenience for the company and the customer.
c. Fast transaction speed – the system would be quick, only depending on the speed of internet, thereby providing quick feedback to the customer and the company as well.
d. Low labour costs – since the payment system would be automatic, it would reduce labour costs eliminating the need of manual payment methods like cheques, debit orders, cash, etc.
e. Low risk of theft – after processing, payments would go straight into the company's bank account thereby reducing risk of theft

1.7 Methodology

In an attempt to solve this problem, the following procedure will be undertaken

Brainstorming
This stage involves the intensive familiarisation with the technologies required to build this system. It also involves gathering requirements on what methods, libraries, technologies TelOne prefers and preparing the right tools to tackle the problem. This will involve consulting the supervisor and other technical people in the development field.
Literature review
This stage involves research of some of the payment gateways already on the market like Paynow Zimbabwe and gateways used by Zimall. This also include extensive research on the internet and other projects done by other developers.
Design analysis and selection
This stage will involve analysing existing technologies in the form of API's and gateways and analysing how other gateway providers are doing it. Selecting the right API for development will be important for purposes of data security, encryption and also to meet the

standard requirements of the ISO8583. At this stage research will also be done to select the best checkout-form format and methods to make sure clients don't give up during the checkout process. Database selection is also done at that stage. The developer will use the Model-View-Controller (MVC) design pattern for this project. This is due to change though, as time goes on and under supervision and advisement. The designer will design for testability, maintainability and scalability.

Coding and simulation

This is the development stage. The researcher will sit down, having gathered the right tools and skills to use and actually attempt to solve the problem by developing the required software. This stage will be incremental, if one working technology is produced, the developer will try to improve and increment the development.

Testing

Since this payments system is going to be handling people's monies, it is critical for the software to be rigorously tested. Testing will be done throughout the development process and the supervisor will oversee the whole process to make sure a working platform is produced, with minimal bugs. In this project, both the alpha and beta testing will be necessary and will be implemented.

Deployment

This is when the supervisor is satisfied with the system and it's ready to be used by the company.

1.8 Tool

The tools and technology to be used in this project will be, but not limited to:

Java 8u162
Java EE 7 components (Java Saver Faces 2.2, Primefaces 6.2.RC)
JPOS 2.1.1
JPOS EE 2.1.1
MySQL (InnoDB)

1.9 Estimated budget

This project was carried out on zero budget.

1.10 Conclusion

Electronic commerce is now the It of most businesses. Not implementing this technology would be an unnecessary loss to the company. Handling payments online is becoming more and more popular and safer and everyone who wishes to make money should be doing it.

CHAPTER 2 - Literature Review

2.1 Introduction

In this chapter I am going to take a deep dive into the existing technologies I used in this project, and also the how the other payment gateways on the market are using these and other technologies available. Payment card processing is a very grey area to most developers because of the complexity of implementing the protocol of financial communication, ISO 8583.
A card originated transaction is routed from an e-commerce network hosted by a secure HTTPS server, through a series of networks, to the card issuer for authorization and routs back to the server for processing response to the client. The data would have the information derived from the card (e.g. PAN and CVV) or any other information that the client might have provided, like address, name, etc., as well as other data that might have been added for systematic purposes like the terminal ID and STAN. With regard to this information, the issuing organization may accept or reject authorization at their discretion.

2.2 ISO 8583

This is a standard for financial transactions originated by a payment card (credit or debit card). This standard is what describes the message orientation so that different institutions interpret these messages the same way. This standard describes a communication flow and message format over the wire when a cardholder makes a transaction using a payment card. In this project I will mainly focus on the two most popular payment cards, Visa and MasterCard. Both the MasterCard and Visa networks have their authorization communications based on the ISO 8583 standard, so does other institutions and networks.

An ISO message that is sent to a network will contain a message type indicator (MTI), primary (or optionally a secondary) bitmap and the fields containing the message itself, known as data elements. Below is the structure of an ASCII formatted ISO message, using iso8583 version 1983

```
0800202000000080000000000000001323931313030303031

  Message: 08002020 00000080 00000000 00000001
           32393131 30303031

      MTI: 0800
   Bitmap: 20200000 00800000
 Field 03: 000000
 Field 11: 000001
 field 41: 3239313130303031 (ASCII for "29110001")
```

Figure 1-2.1 ISO message format

The MTI is a 4 digit numeric field that indicates the version of iso8583 used, the message class, the function code and the message origin. In the above example, filed 0, the MTI is 0800 which means 0-version 1983, 8-network management message, 0-request message, and 0-the message was initiated by the acquirer.

A bitmap is a field or sub-field within a message, which indicates whether other data elements or data element sub-fields are present elsewhere in the message. It also shows exactly which they are. In the example above, 16 digit primary bitmap 20200000 00800000 is a hexadecimal equivalent of the binary

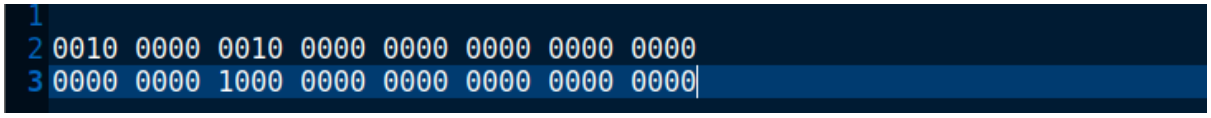


Figure 2 - 2.2 ISO8583v1987 full ASCII message

These binary values represent every single field from 1 through 64. The presence of an element in the message is represented by a set bit in the bitmap, otherwise a zero is placed on that place. For the above example, field 1 and 2 are all zeroes, field 3 is set, meaning that the message contains a data element field 3. If the message contains an element above 64, a secondary bitmap is suffice and the elements are listed accordingly.
The data elements are the individual fields which contain the data carried over the network. The above message contains 3 fields, field 3, field 11 and field 41. This briefly explains the whole ISO8583 message standard.

2.3 JPOS

2.3.1 ISO 8583 and JPOS

The java Point of Sale Framework created by Alejandro P Revilla, jPOS, handles all the iso8583 low level configurations letting the developer focus more on the business logic to be implemented. The most common class in jPOS is the ISOMsg class in the diagram below, which allows a developer to define elements and capture data from the user which they would then pack and send over the wire.

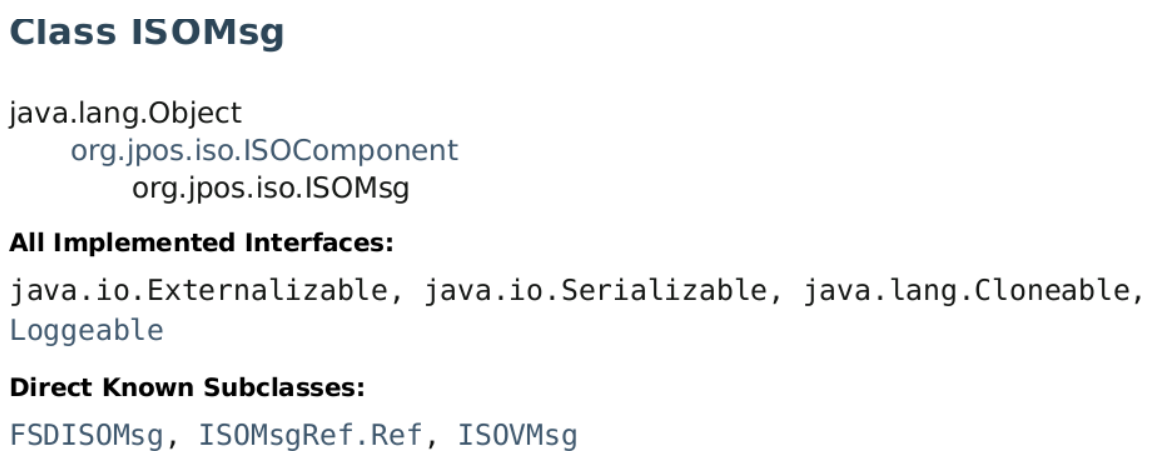


Figure 3 - 2.3 ISO Message API

Using this class, one can create clean ISOMsg components.
This library has helper classes which enables a programmer to pack messages and encode them so to speak, so that they can conform to the message standards set up by the ISO8583 standardisation, interchange specifications, and then be sent over the wire. Received messages can also be decoded accordingly.

pack

```
public byte[] pack()
            throws ISOException
```

pack the message with the current packager

Specified by:

pack in class ISOComponent

Returns:

the packed message

Throws:

ISOException

Figure 4 - 2.4 ISO Message API packager

unpack

```
public int unpack(byte[] b)
           throws ISOException
```

unpack a message

Specified by:

unpack in class ISOComponent

Parameters:

b - - raw message

Returns:

consumed bytes

Figure 5 - 2.5 ISO Message API unpack

In order to be able to send the messages over the wire using different protocols (TCP/IP and HTTPS), jPOS provides us with ISO Channels and Channel-Adaptor implementations which open a socket and connects to a remote host as defined by the configurations. The ISO Channel can open a remote connection to a payment gateway from inside a Java EE component (JSF bean) and the gateway can fully derive Q technology for all its resource manipulations. JPOS has also provided the ISO Server component to help with the server connections. These Technologies are pretty good for production use as they run on the fly for a mission critical project like a payment gateway. The channel multiplexer is used to multiplex a single socket connection than to open a single socket per transaction thread. This saves a lot of resources for the hosting servers and makes a faster transaction. This solves the problem of typical gateway systems which have to route thousands of transactions per second to an issuing organization using a channel.

2.3.2 JPOS' Logger and System Monitor

The Framework provides a logging system specifically because it is transaction cantered, rather than line cantered. This is useful for debug purposes and tracking the live objects handled by the Log Listener of the jPOS logger. This is good as per the PCI DSS requirements for transaction data integrity and security. The system monitor log by the jPOS logger looks like this:

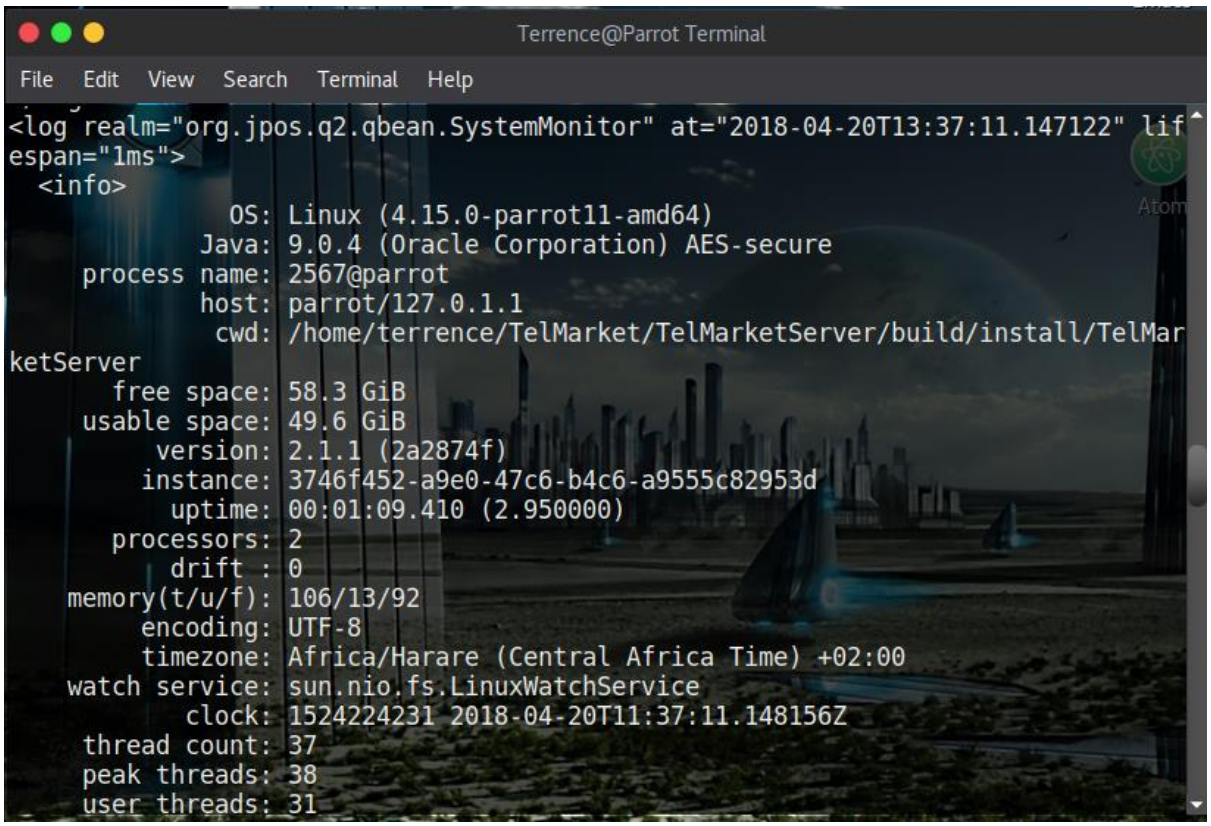


Figure 6 - 2.6 jPOS System Monitor

The System monitor uses the logger to periodically persist useful information about how the application is running.

2.3.3 Packagers

JPOS is bundled with a number of ISO-Packager and ISOFieldPackager classes that can be used as a centre point to encode (pack) and decode (unpack) messages that are written for the ISO-8583 standard. The packager are codded as a java class, or can be referenced from an XML file configuration that initialises an ISOFieldPackager right there and then. Since packagers are mostly instantiated once during the life time of an application, there's no performance impact between a packager implemented in pure Java or the Generic-Packager that reads an XML only at initialization time.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE isopackager PUBLIC
        "-//jPOS/jPOS Generic Packager DTD 1.0//EN"
        "http://jpos.org/dtd/generic-packager-1.0.dtd">

<!-- ISO 8583:1993 (BINARY) field descriptions for GenericPackager -->

<isopackager>
  <isofield
      id="0"
      length="4"
      name="Message Type Indicator"
      pad="false"
      class="org.jpos.iso.IFB_NUMERIC"/>
  <isofield
      id="1"
      length="16"
      name="Bitmap"
      class="org.jpos.iso.IFB_BITMAP"/>
  <isofield
      id="2"
      length="19"
```

Figure 7 - 2.7 *XML-formatted binary packager*

```
public class ISO93BPackager extends ISOBasePackager {
    private static final boolean pad = false;
    protected ISOFieldPackager fld[] = {
    /*000*/ new IFB_NUMERIC (  4, "Message Type Indicator", pad),
    /*001*/ new IFB_BITMAP  ( 16, "Bitmap"),
    /*002*/ new IFB_LLNUM   ( 19, "Primary Account number", pad),
    /*003*/ new IFB_NUMERIC (  6, "Processing Code", pad),
    /*004*/ new IFB_NUMERIC ( 12, "Amount, Transaction", pad),
    /*005*/ new IFB_NUMERIC ( 12, "Amount, Reconciliation", pad),
    /*006*/ new IFB_NUMERIC ( 12, "Amount, Cardholder billing", pad),
    /*007*/ new IFB_NUMERIC ( 10, "Date and time, transmission", pad),
    /*008*/ new IFB_NUMERIC (  8, "Amount, Cardholder billing fee", pad),
    /*009*/ new IFB_NUMERIC (  8, "Conversion rate, Reconciliation", pad),
    /*010*/ new IFB_NUMERIC (  8, "Conversion rate, Cardholder billing", pad),
    ...
    ...
    ...
```

Figure 8 - 2.8 java class for binary packing

2.3.4 Channels

The Framework comes with a bunch of channel implementations, most of which are TCP/IP based socket connection channels, but can be used for other protocols as well like HTTP, TPDU etc. Secure channel encryption is provided by the ISOClientSocketFactory and ISOServerSocketFactory packages that provides SSL encryption. This is one of the PCI requirements for online communicating transactions. Filters can be added to alter the message before transmission or after receiving the message, e.g. MD5 Filter which uses a Hashing algorithm for authentication.

2.3.5 JPOS Space

The jPOS Space is a coordination component derived from The Linda Coordination Language. You can consider jPOS Space segment as resembling a Map where its entrances are arrangements of articles and its activities are completely synchronized.

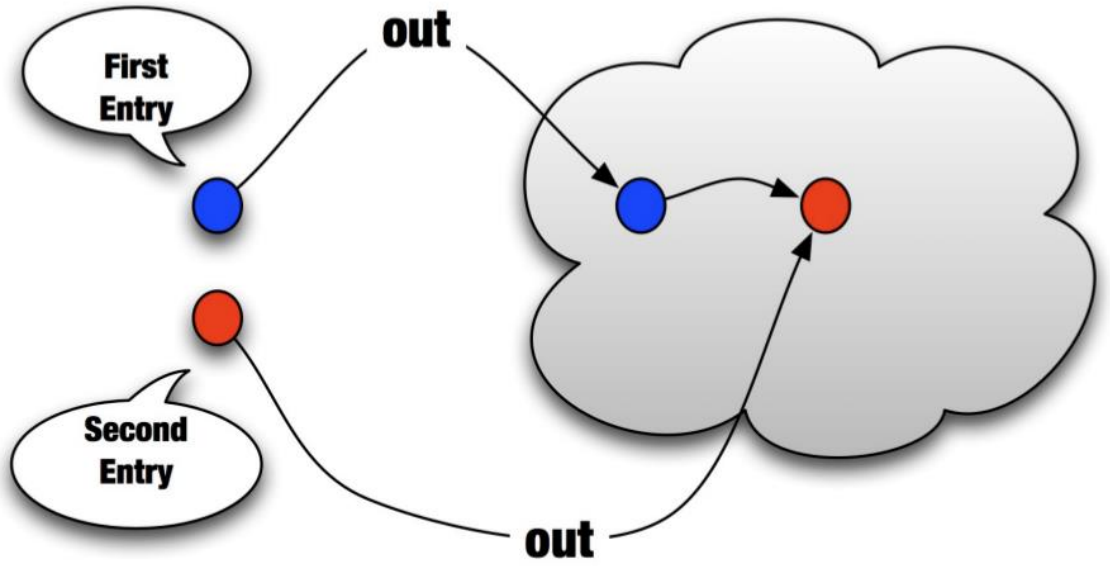


Figure 9 - 2.9 *jPOS Space*

Using this component, all the transactions to be processed at run-time a queued into space with unique names where the transaction manager takes them for processing one at a time. The Tspace is a default local space interface in jPOS and works similarly to the native java Tuples.

2.3.6 Q2

JPOS comes with the advance QSP helper which provides the helper classes at run time, Channel adaptor, QServer, QMUX and the transaction manager. At start up time, Q2 filters the convey registry searching for deployment descriptors. Those are minor XML records that are utilized to begin and design Q2's services. In many applications, the business rationale and packagers are accessible in the class-path, however there are circumstances where you have to apply a hot fix, for example, including another field packager, or an ISO channel, so there is this dynamic class stacking capacities. JPOS applications are normally mission critical and exceedingly touchy, so by and large, it is anything but a good plan to get the execution from remote locales.

The Channel-Adaptor utilizes the Space to speak with different jPOS segments, fundamentally through two Space lines, one for input and the other one for yield.

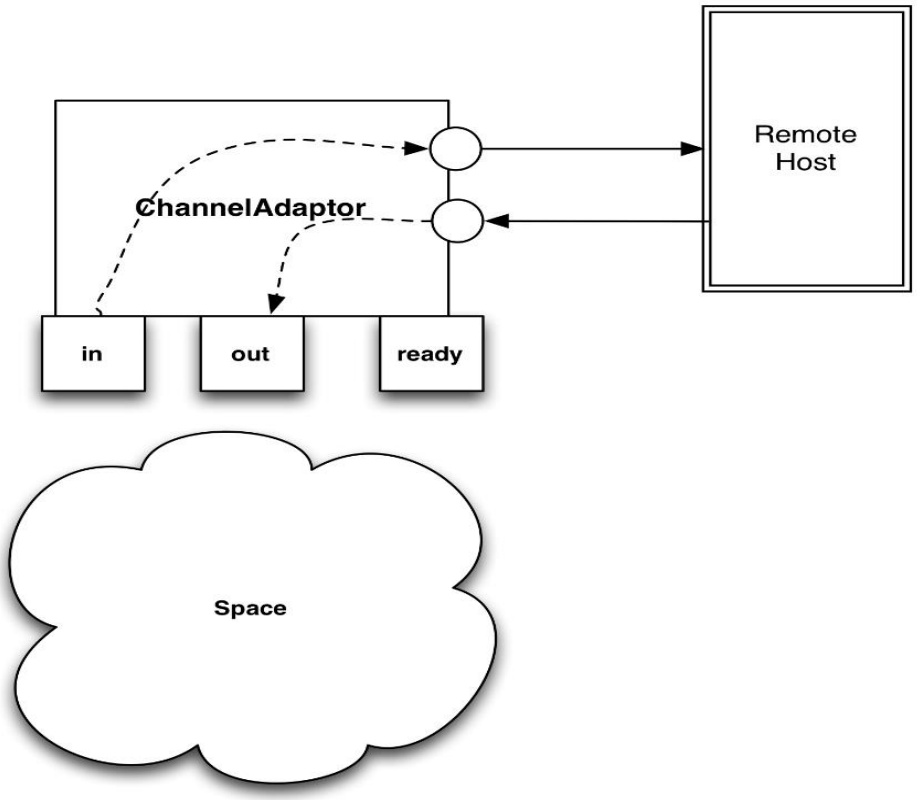


Figure 10 - 2.10 *Channel Adaptor*

QMUX utilizes the Space keeping in mind the end goal to speak with the basic channels; this procedure brings into play an entire new arrangement of organization choices, including the capacity to multiplex a few channels for excess/stack adjusting. These channels doesn't need to keep running on a similar machine. They could utilize appropriated/remote space usage. The new space-based code doesn't require an additional string, something exceptionally valuable in frameworks where countless are required.

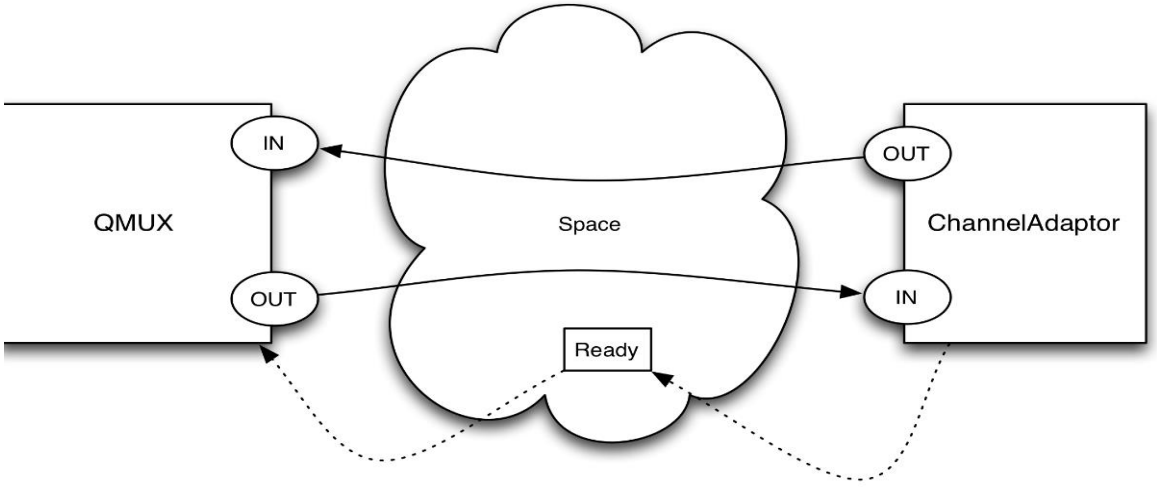


Figure 11 - 2.11 QMUX

QServer is a connector around ISOServer ("Tolerating associations with ISOServer") that associates with other Q2 segments, for example, QMUX, utilizing the Space by characterizing in and out lines, essentially like the Channel-Adaptor does.
Regardless of the way that QServer will go about as a server from a TCP/IP stance, and it will tune in to a configurable port, it can in any case be utilized to start exchanges to the remote endpoint. When going about as a server (from an exchange stand point), the QServer is ordinarily designed to forward exchanges to an arrangement of demand audience members, yet that is not required. It is conceivable to use in/out Space based lines also, interface QServer to different segments, for example, QMUX. The Channel definition utilized by QServer is the same as the one utilized by the Channel-Adaptor, where you can arrange SSL bolster, packager-level logging, and so forth.
JPOS is ordinarily used to execute mission-critical applications that need to precisely manage error conditions. When you get to a site page and a transient system blunder happens, you simply hit the reload catch on your program. By differentiate, a complex monetary exchange includes a considerable measure of exercises, for example, reaching remote hosts, telling chance administration frameworks, setting holds in card-holder's credit accounts, database logging, and so forth. Thus, if something turns out badly or your framework just kicks the bucket because of a power disappointment, it's more confounded than basically hitting the reload catch: you need to turn around the effect of whatever moves had been made up to the disappointment point.

The Transaction-Manager execution drives the exchange by calling the greater part of its members' prepare function. In the event that every one of them return PREPARED (showing that they are prepared to continue with the exchange), at that point the transaction moves to the COMMITTING stage, and soon thereafter the Transaction-Manager will call the majority of the members' submit technique. In the event that one of the members' plan strategy returns ABORTED , then the exchange moves into an ABORTING stage, and every one of the members beforehand called to get readied will get a call to their prematurely end strategy.
The Transaction-Manager encourages and allows developers to write reusable and configurable components called Participants. It is a jPOS Service that monitors a Space queue waiting for transactions to be processed. These transactions are expected to be any Serializable objects, but in most jPOS applications those are actually org.jPOS.transaction.Context objects.

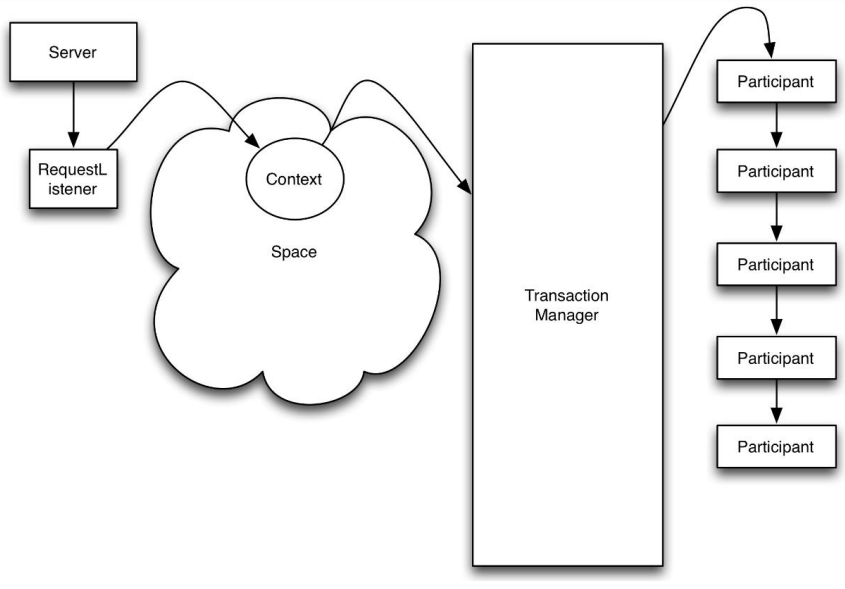


Figure 12 - 2.12 Transaction Manager

2.3.7 Result-Code

Most jPOS applications need to manage result codes going and reaching and from various endpoints. A "Card Expired" result code (DE 39) can be 14 for a given ISO-8583 v1987 endpoint, a 54 in another v87 one and 1001 of every a v2003 connect. JPOS and additional client applications need to characterize and outline claim result codes. Financial applications ordinarily need to play out a considerable measure of approvals, a commonplace jPOS application for example utilizing the Transaction Manager has members to check required and discretionary fields, check the terminal, the shipper, the card, PIN, and so on.
While guaranteeing these sort of uses we more often than not distinguish the principal mistake and prematurely end. Once the mistake gets rectified we locate there's another mistake in the following test, but another on a third one. So rather than early-coming up short, applications can "gather" result data in the Result question.

2.4 JAVA ENTERPRISE EDITION

Java EE provides component based application development for enterprise applications, mission critical applications with seamless business logic that works on the fly. Of interest is the Java Server faces API (JSF 2.2) and Primefaces which is a component based framework built on top of JSF. Developers today progressively perceive the requirement for distributed, transactional value-based, and versatile applications that use the speed, security, and dependability of server-side innovation. Undertaking applications give the business logic to an enterprise. They are halfway overseen and frequently associate with other venture programming. In the world of data innovation, undertaking applications must be planned, assembled, and delivered for less cash, with more noteworthy speed, and with less assets.

Containers are the interface between a part and the low-level, stage particular usefulness that backings the segment. Before it can be executed, a web, enterprise bean, or application customer segment must be amassed into a Java EE module and sent into its compartment.

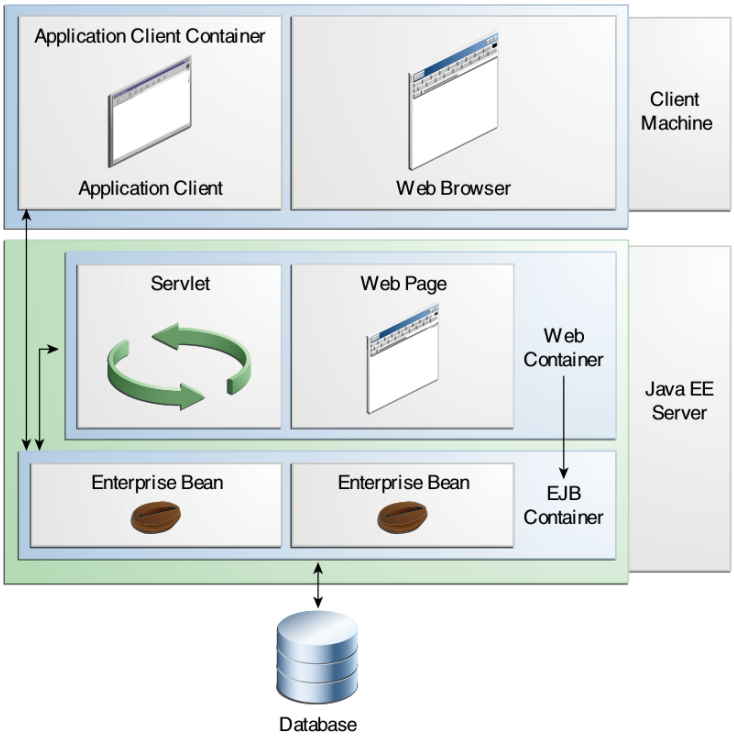


Figure 13 – 2.13 Java EE

2.4.1 Java Server Faces (JSF 2.2)

Java Server Faces technology is a user interface framework for building web applications. Web services are online based applications that utilization open, XML-based guidelines and transport conventions to trade information with calling customers. The Java EE stage gives the XML API's and apparatuses you have to rapidly configure, create, test, furthermore, convey web administrations and customers that completely inter-operate with other web administrations also, customers running on Java-based or non-Java-based stages. Customer asks for and web services responses are transmitted as Simple Object Access Protocol (SOAP) messages over HTTP to empower a totally inter-operable trade amongst customers and web benefits, all running on various stages and at different areas on the Internet. HTTP is a well-known demand and-reaction standard for sending messages over the Internet, and SOAP is a XML-based convention that takes after the HTTP request-and-response model
Managed Beans, lightweight containers managed Plain Old Java Objects (POJOs) with negligible prerequisites, bolster a little arrangement of fundamental administrations, for example, asset infusion, life-cycle call-backs, and interceptors. Managed Beans speak to a speculation of the managed beans determined by Java Server Faces innovation and can be utilized anyplace in a Java EE application, not simply in web modules.

2.4.2 Primefaces 6.0

PrimeFaces is a well-known open source library for Java Server Faces featuring over a hundred components, touch optimized mobile kit, client side validation, theme engine and more. Other than the considerable look and feel, PrimeFaces components support fine-grained AJAX abilities, are responsive qualified, and are good with any cutting edge program and gadget (for instance, from work area to cell phones). Likewise, PrimeFaces accompanies PrimeFaces PUSH (based on the Atmosphere framework), PrimeFaces Extensions, which is a lightweight open source segment library for JSF 2.2. Further, the Prime UI is a spin-off from PrimeFaces and is a gathering of rich JavaScript widgets in light of the jQuery UI. PrimeUI is accessible including PrimeElements, which is the Web-Components library to make UIs definitively with custom HTML components. For all intents and purposes, PrimeElements is an extra library for PrimeUI. What's more, we should not overlook the WIP project named Prime NG, which is a gathering of rich UI parts for AngularJS2. Every one of these treats were created by the PrimeFaces group (pioneer Cagatay Civici) and are characterized as "Ultimate UI Framework for Java EE."

2.5 MYSQL INNODB

InnoDB is a general-purpose database engine that scales high reliability and high performance. In MySQL 8.0, InnoDB is the main MySQL storage component. Its DML operations the ACID, with transactions, rollback, and crash-recovery capabilities to protect user data. Percona Server, and in addition adaptations of Maria-DB below 10.2, utilize a fork of InnoDB called XtraDB naturally. XtraDB is kept up by Percona. Oracle InnoDB's progressions are frequently transported in into XtraDB, and some bug fixes and additional highlights are included.

2.6 VISA

Visa dominates the credit card industry keeping almost a 70% marketplace percentage of all U.S. credit playing cards which are in stream. Visa provides a lot of the mandatory infrastructure to support monetary establishments in issuance and processing of debit and credit cards. Monetary establishments like Capital One and your native bank issue credit and debit cards as a result it makes them cash.

2.7 MASTERCARD

As a processor, MasterCard operates the world's quickest bills processing community, connecting purchasers, economic establishments, merchants, governments and organizations making their fee enjoy consistency, reliability, speed and value addition. While credit card strategies credit card branded transactions between monetary establishments via the credit card network, we authorize, clear and settle the transaction, making sure the transaction is valid, budget are to be had and bills are credited and debited as it should be. Switching with credit card opens the door to our wealth of value-introduced price processing offerings that help financial institutions and merchants correctly leverage fee generation to control their groups.

CHAPTER 3: DESIGN AND ANALYSIS

3.0 INTRODUCTION

This chapter aims to bring the possible concept that can be developed to meet the desired objectives through analysis. The chapter will explain the requirements of the systems and how they are going to be met. It also gives the advantages and disadvantages of major technologies that have been chosen for the design. The first step will be to design the flow diagrams and use case analysis of an electronic commerce checkout module and the payment gateway server.

3.1 SYSTEM DESCRIPTION

This section describes the operation of a payment gateway. A payment gateway is an online facility that routes transactional information between an electronic commerce website and the merchant's acquiring bank. It is the e-commerce equivalent of the physical point-of-sale (POS) terminal used by brick-and-mortar merchants in card-present transactions. So as to protect cardholder information, the data that the payment switch collects from the website is SSL-encrypted (Secure Socket Layer) before transmission.

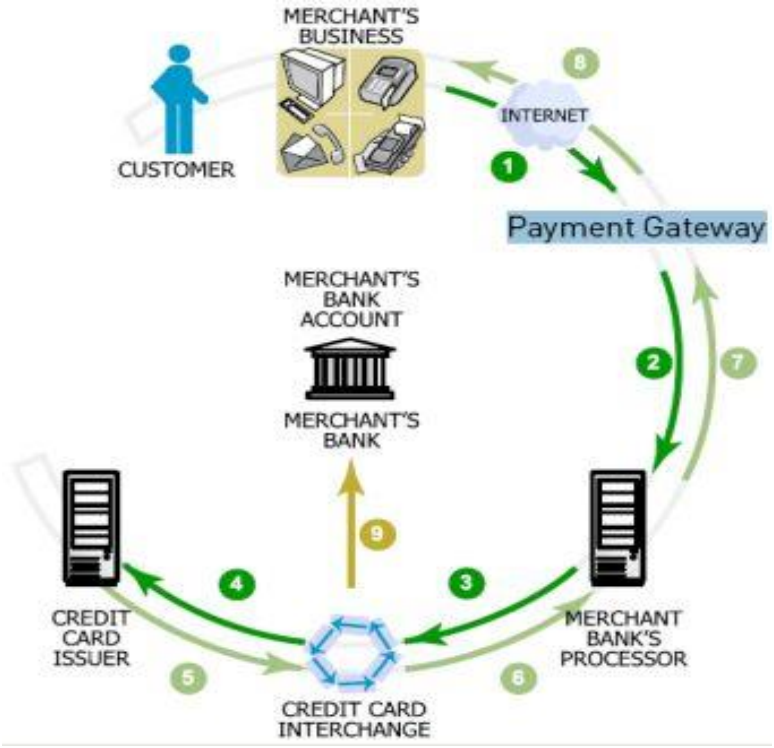


Figure 14 – 3.1 Payment Gateway Functionality

3.1.1 USE CASE

The Following use-case describe the system functionality The Best case scenario is the following use case:

1. A Customer (Let's Call Him John) Purchases an Item

John stops by our e-commerce website, finds the perfect gift to buy his girlfriend and enters his credit card information in the checkout module. He clicks the BUY button and waits approximately 3 to 7 seconds. As those seconds pass, the following steps are will be taking place in the background.

2. John's Transaction Details are sent to the Payment Gateway

John's personal and credit card information are routed over to the payment switch. This gateway is kind of like the guard, or middle man, between John's information and the banks. The transaction details are securely passed from the gateway through the payment processor, referencing the merchant's ID number and passing along the transaction details to the merchant account.

3. The Merchant's (our) Payment Processor Also Gets the Transaction Details

Besides sending the transaction details to the Merchant Account, the Payment Processors also has the job of contacting the issuing bank of the credit card used. So, John's bank would be sent a quick message to see if he hasn't gone over his limit and to see if it's a legitimate card. There is a Credit Card Network that serves as an intermediary between the Payment Processor and the issuing bank. The Credit Card Network is responsible for locating the issuing bank in question. This is usually either VISA or MASTERCARD Payment Processing Networks

4. John's Card Issuing Bank Accepts or Declines the Purchase

Regardless of whether the payment is accepted or denied, this information goes back through the Credit Card Network and to the Payment Processor.

5. The Merchant's Payment Processor Sends the Response Back to the Payment Gateway

In the event that John’s transaction was approved, then the payment processor switches that information to the payment gateway, which will then store the result so that the merchant's website can complete the transaction.

6. If John is approved, the Sale Can Move On

Once the merchant finds out about the approval, they can ship out products.

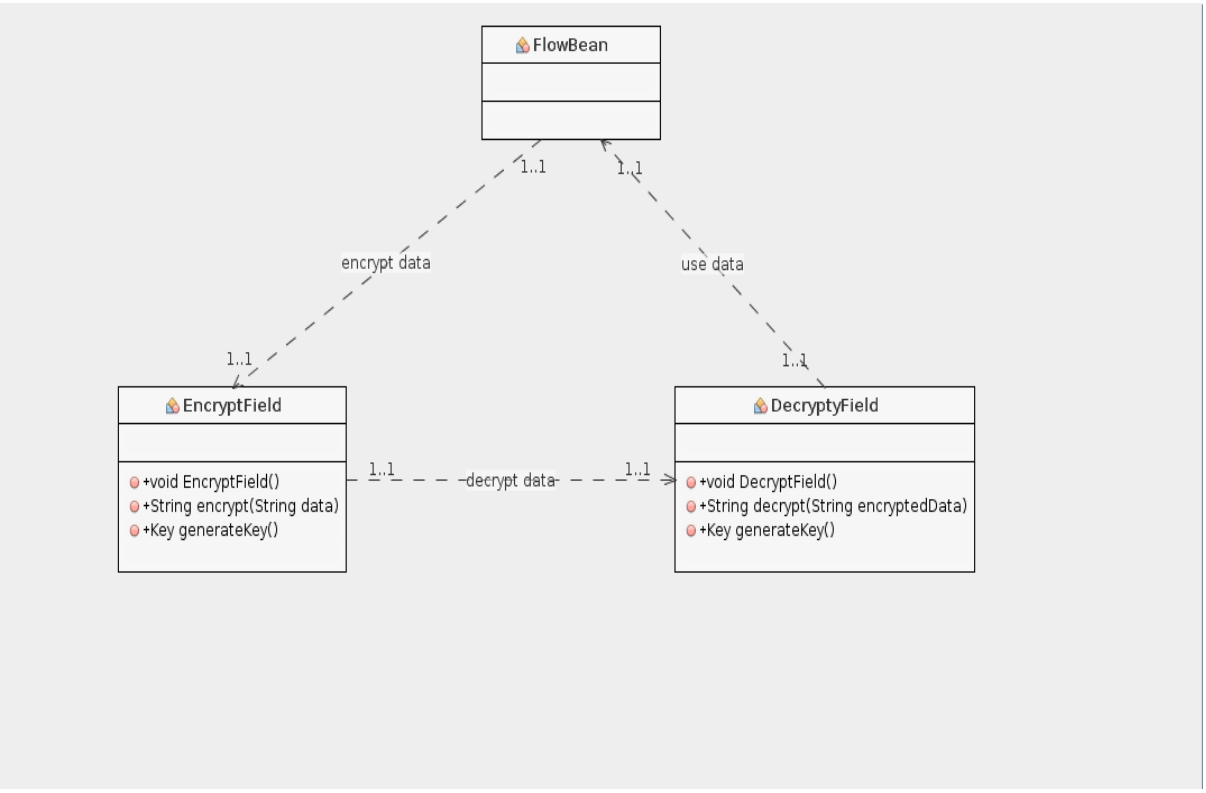
7. The Merchant Gets Paid

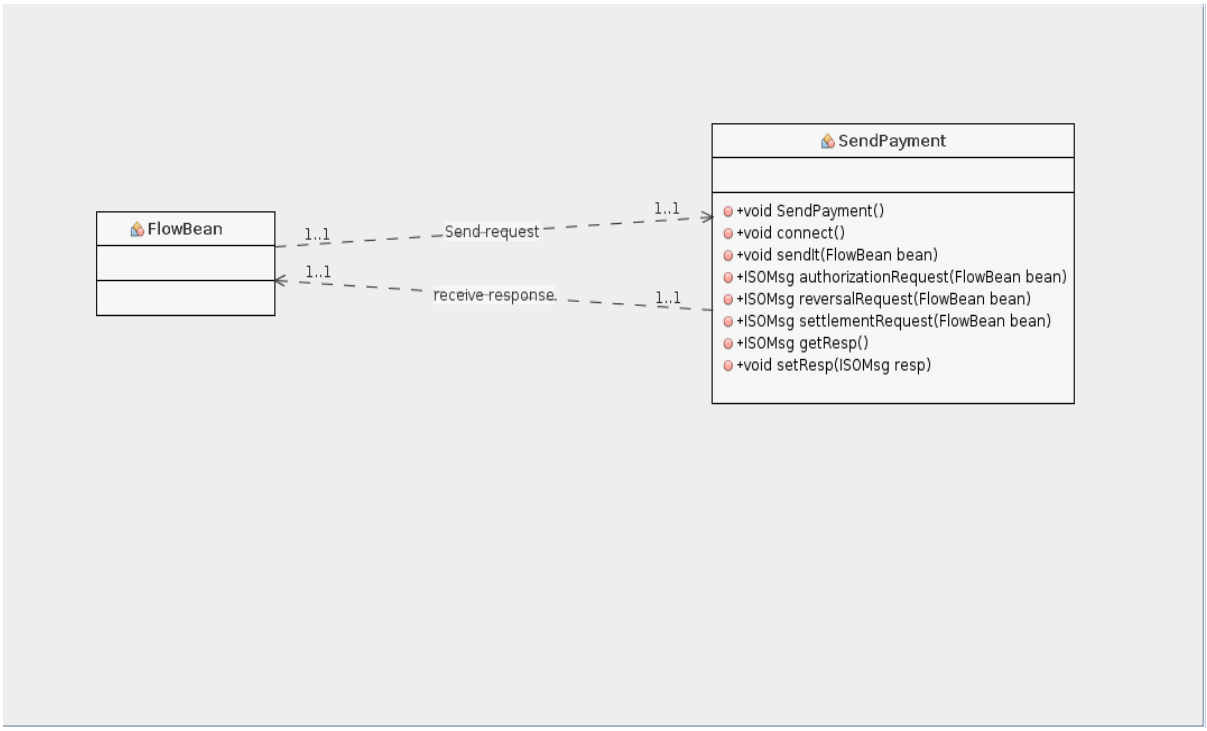
The bank that issued John’s card releases funds to the merchant's bank. After a short settlement period, the merchant's bank drops that money into the merchant account. At the end of the business day, all authorized transactions (also called a “batch”) are submitted to

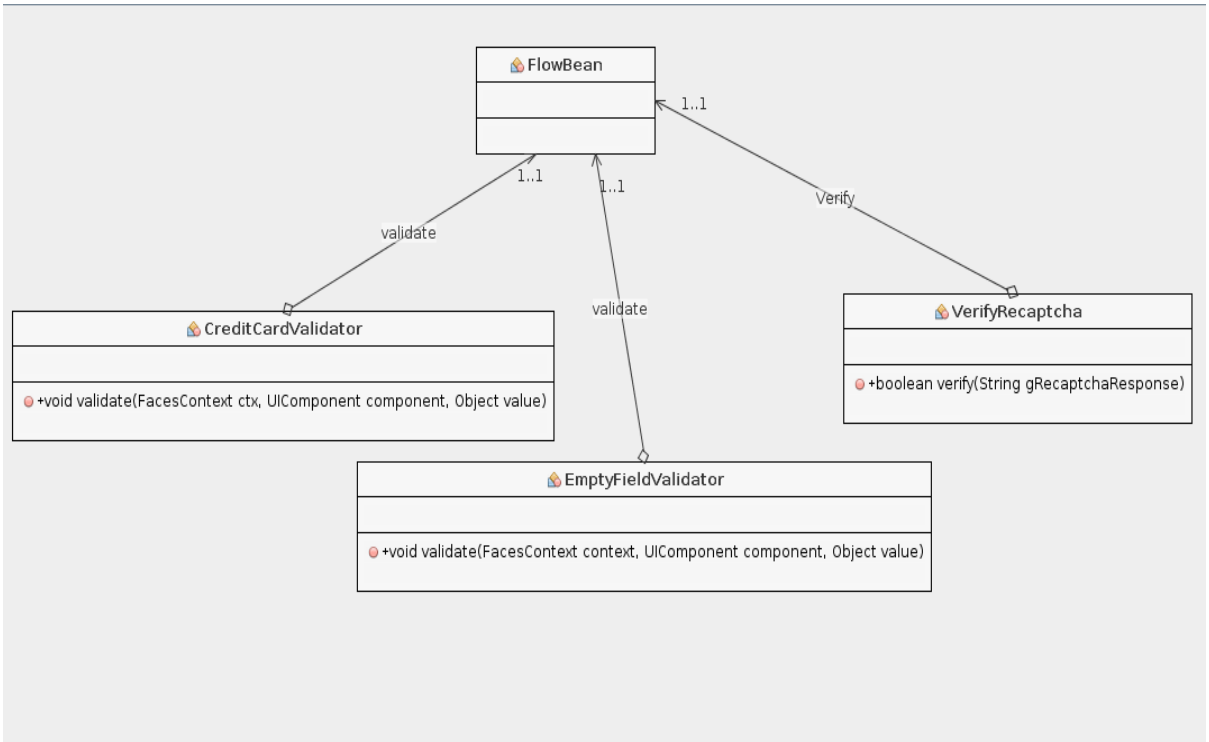
the processing bank for settlement.

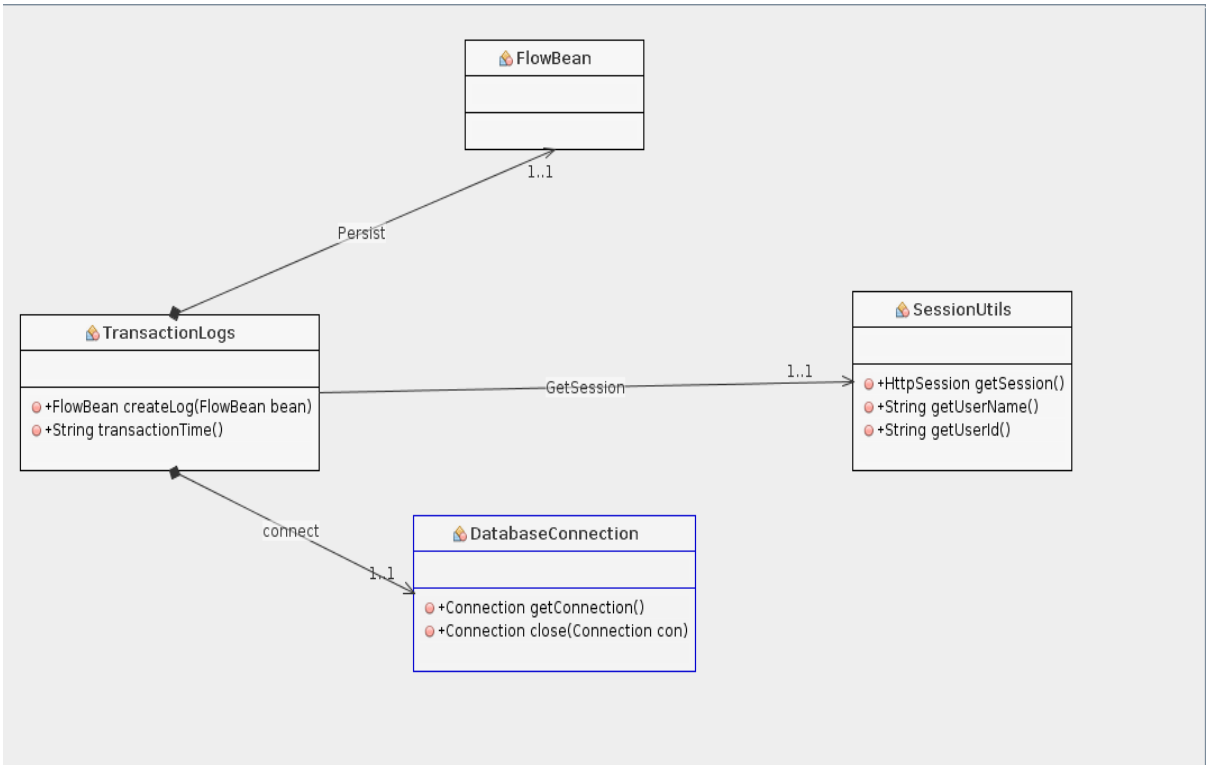
3.3.2 MODELING WITH CLASSES

The following is a description of the classes used in this project, and their inter-connectivity.


FlowBean
1..1
1..1
encrypt data
use data
1..1
1..1
EncryptField
DecryptyField
+void EncryptField()
+String encrypt(String data)
+Key generateKey()
1..1
decrypt data
1..1
+void DecryptField()
+String decrypt(String encryptedData)
+Key generateKey()


SendPayment
FlowBean
1..1
Send-request
1..1
1..1
receive-response.
1..1
+void SendPayment()
+void connect()
+void sendIt(FlowBean bean)
+ISOMsg authorizationRequest(FlowBean bean)
+ISOMsg reversalRequest(FlowBean bean)
+ISOMsg settlementRequest(FlowBean bean)
+ISOMsg getResp()
+void setResp(ISOMsg resp)


FlowBean
1..1
1..1
1..1
validate
validate
Verify
CreditCardValidator
+void validate(FacesContext ctx, UIComponent component, Object value)
VerifyRecaptcha
+boolean verify(String gRecaptchaResponse)
EmptyFieldValidator
+void validate(FacesContext context, UIComponent component, Object value)


FlowBean
1..1
Persist
TransactionLogs
+FlowBean createLog(FlowBean bean)
+String transactionTime()
GetSession
1..1
SessionUtils
+HttpSession getSession()
+String getUserName()
+String getUserId()
connect
1..1
DatabaseConnection
+Connection getConnection()
+Connection close(Connection con)

The following flow diagram describes the payment switch functionality

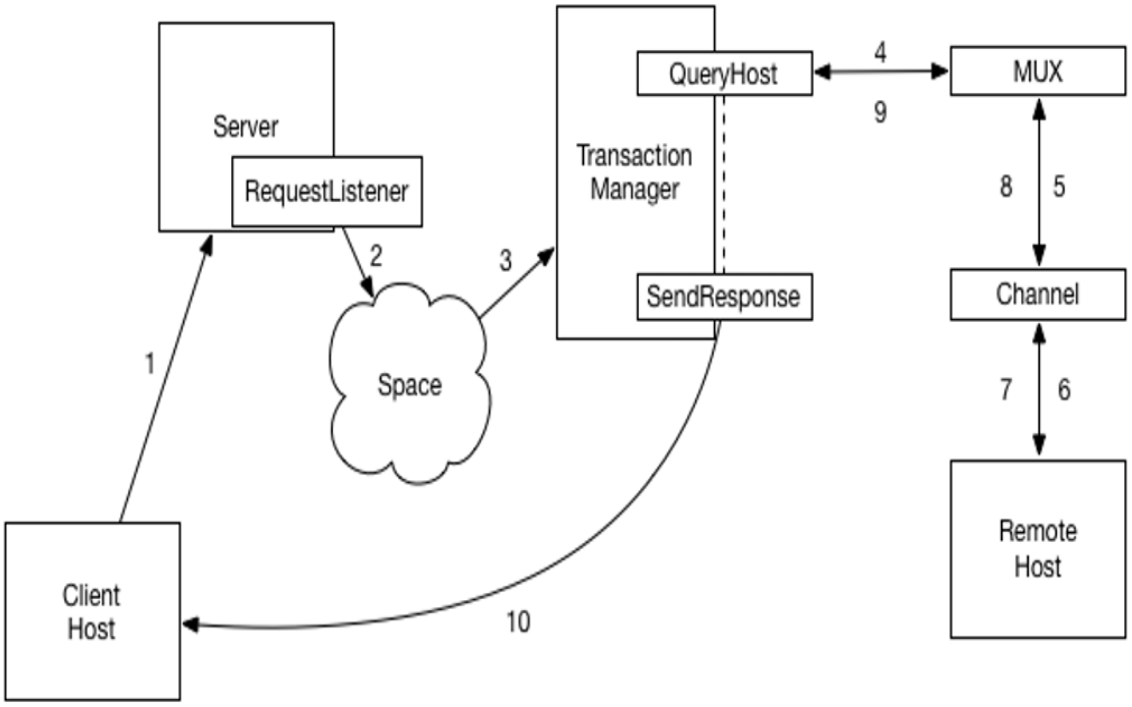


Figure 15 - 3.2 Payment Switch Block Diagram

Step 1
The client checkout module opens a channel socket to the targeted server, the TelMarket Server. All incoming messages are handled by the Incoming Listener ISORequestListener. The client host is the host machine the client trying to buy stuff from us starts a request. This process starts when the user presses the checkout button on our platform and is greeted by a wizard to capture data we need. Using a jPOS deployment packager, the data is encoded and packed into an ASCII data bundle which follows the iso8583 v1987A message format. The required data, (most importantly the primary account number and the CVV) are sent over the wire using a ASCII channel to a specific pre-configured I.P address which would already be listening at a specific port. This completes the first step when data is received by the QServer.

Step 2
The Listener places the message in queue space. The space implementation is the jPOS Tspace. The server configures the arbitrary variable DESTINATION in context with a value mux.DESTINATION so that the Transaction manager comes and picks up the correct destination context. The QServer has a request listener to listen to incoming data. As soon as data is successfully received, the Q server pushes the data into the Tspace configured using jPOS space, another jPOS component for cloud services. The data queued to space is stays there waiting to be retrieved by the transaction manager for further processing.

Step 3
The transaction manager picks up the message in context and creates an org.jPOS.transaction.Context object. The transaction manager takes the data received from space and does the business logic implementations. The TM is configured with many participants to implement different task, but most importantly the switching and routing the

data depending on the BIN (Bank Identification Number) to select a destination. The credit card BIN show the identity of the issuing organization which will have the sole responsibility of authorising a client request for purchase. Using the transaction manager, we can configure thousands of transaction destinations by simply adding a participant to take care of a certain issuer. The name of the destination, e.g. [STANBIC] would be registered in the Name Registrar, which is another component of jPos that makes it easy to locate transactions by name configuration. So after a certain destination has been configured and saved in context, the next thing to do is actually send the data to that particular destination in real time. Therefore the transaction manager will make use of a participant query host to push the data out of context.

Step 4
What the query host participant does is it locates a channel multiplexer with the same name in context registered in the Name Registrar as the destination participant. When this mux is located, all the data is pushed out from the transaction manager and it's up to the multiplexer to continue with the transaction forwarding. This is a great feature because it saves time and less load on the transaction manager, which in real life will be processing thousands of transactions per second on a busy and lucky day. We wouldn't want to lose money just because people can't buy because of an over load due to a poorly configured processing gateway.

Step 5
When the mux receives data, it does no further processing, but opens a channel to send the data. Our Tspace, the jPOS space helps us in achieving this goal. What jPos does is as soon as the transaction manager has sent all data to the mux, the out configuration of the mux pushes the data into space, specifically targeted for the jPos IN configuration, again using the name Registrar. The channel receives this data no problem and prepares for the next step.

Step 6
The channel opened sends the data over the wire to the remote host, which in our case is the bank server for authorisation. This opened channel remains open for a certain period of time (persistence channel). This open channel now waits for a response from the host to which it is connected in persistence.

Step 7
Should there be no problems the remote host sends the response data to the opened channel. This data is in the form of packed data, same as that was sent before, but this time with additional response fields in accordance with the iso8583 standards.

Step 8
The channel places the massage in space, in a reverse implementation of the step 5, in which the jPOS out configuration now sends data to the mux in configuration hence the tuple exchanges takes place smoothly.

Step 9
The query host participant takes data from the mux and automatically calls the send response method. Since the destination in this case is our I.P, there is no need to go through the destination selection process so this implementation will save us a lot of time and resources. This is great.

Step 10

The send response participant using an already open remote channel sends data into space, over the wire to be received in the client host machine. This is the final step of our payment process and this whole process, from the client request to the response received buy the client machine takes place in less than 10 seconds. On a good network, and a good day using this good technology that we have hacked together, thank you very much, the whole process could complete in under 3 seconds.

CHAPTER 4 RESULTS ANALYSIS

4.1 Test case

The following test case was used to test the applications integrity:

Check whether the respective card selection opens expiry date, card number and CVV options.
Check whether to enter the valid card no by using credit card option
Check whether the card no should accept more than 17 digit
Check whether the card no should accept less than 16 digit
Check whether to enter the invalid card no by using credit card option
Check whether the error message should be displayed as "Please enter valid Credit Card number"
Check whether to enter the existing card no by using credit card option
Check whether the error message should be displayed as "Please enter valid Credit Card number"
Check whether to enter the blocked card no by using debit card option
Check whether the error message should be displayed as "Please enter valid Credit Card number"
Check whether to enter the blank card no by using credit card option
Check whether the error message should be displayed as "Field cannot be left blank"
Check whether to enter the valid name by using credit card option
Check whether to enter the invalid name by using credit card option
Check whether to enter the blank name by using credit card option
Check whether the error message should be displayed as "Field cannot be left blank"
Check whether to enter the valid expiry date by using credit card option
Check whether the expiry date should allow month within 1-12 and date in between 1-31st
Check whether to enter the invalid expiry date by using credit card option
Check whether the error message should be displayed as "Please enter a valid card expiry date"
Check whether to enter the blank expiry date by using credit card option
Check whether the error message should be displayed as "Field cannot be left blank"
Check whether to enter the valid CVV by using debit card option
Check whether to enter the invalid CVV by using debit card option
Check whether the error message should be displayed as "Please enter a valid card expiry date CVV"
Check whether to enter the blank CVV by using credit card option
Check whether the error message should be displayed as "Field cannot be left blank"
Check whether the CVV field should not accept more than 3 digits
Check whether the copy and paste functionality should not be allowed
Check whether the tab sequence is working or not
Check whether to try out "cookie poisoning" technique in payment gateway
Check whether to test try to access the page with using back button of the browser to check that session is still active on successful confirmation of payments
Check whether to test try to access the page with using refresh of the browser to check that session is still active on successful confirmation of payments
Check whether the tab offers an option to save the card.
Check whether the tab offers an option to name the saved card.

Check whether the payment process success or not
Check whether the transaction processes immediately or processing is hand to your bank
Check whether the after successful transaction check if the payment gateway returns to default page
Check whether the proper message or alert message is being shown for the successful payment or for the payment issues, respectively.
Check whether the payment process gets failed
Check whether if session ends during the payment process
Check whether what happens if payment gateway stops responding during payment
Check whether the during payment process check error pages and security pages
Check if the card options allows card number, expiry date, CVV value text fields.
Check if the card options tab allows text to be added into the text fields.
Check if the number being added into the credit card number field detects the type of card (e.g. visa, MasterCard)
Check if the tab offers an option to name the saved card.

4.2 Test Results

This page will be replaced with an API that interacts with the shopping cart to get the total dollar value of transaction.

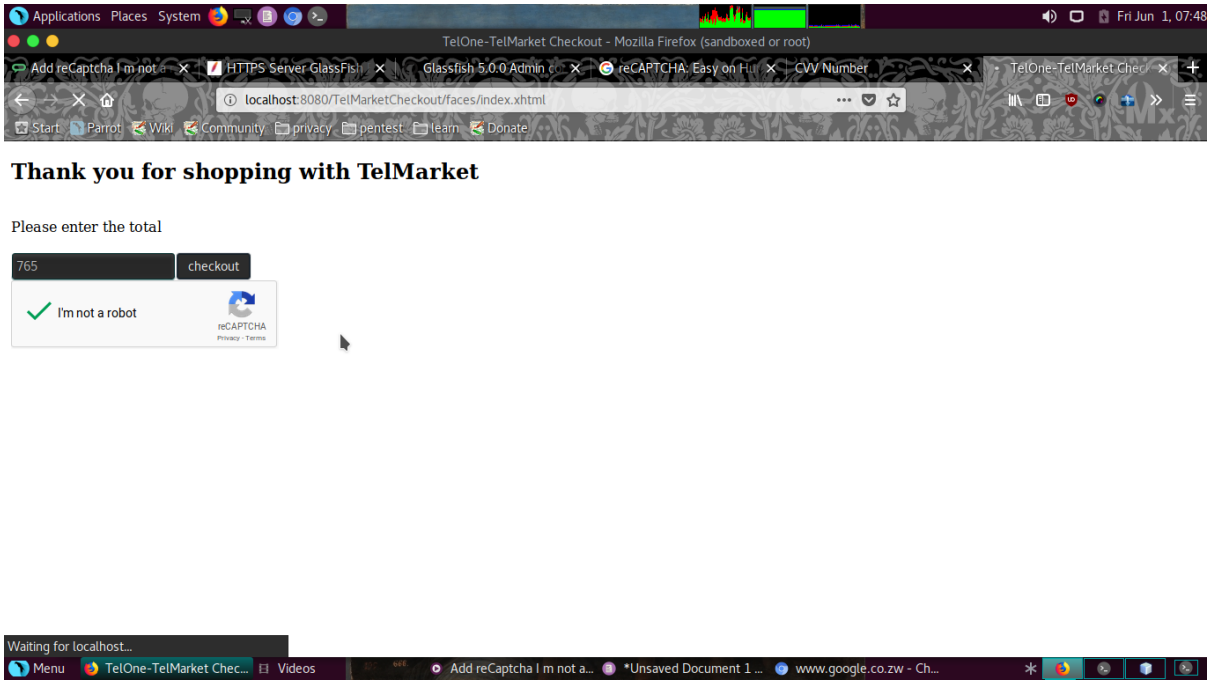


Figure 16 - 4.1 TelMarket page 1

After pressing the checkout button, the following page is forwarded to the following module

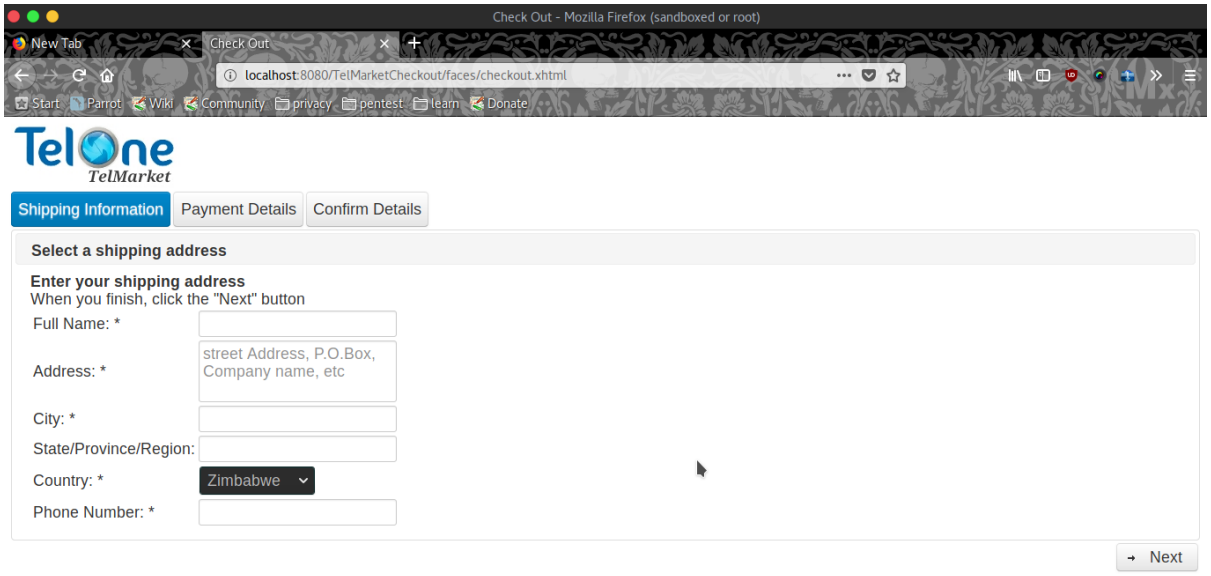


Figure 17 - 4.2 TelMarket page 2

All fields marked with an asterisk (*) are required fields and are validated on the server for data integrity. If any required fields are not provided, the following exceptions are thrown.

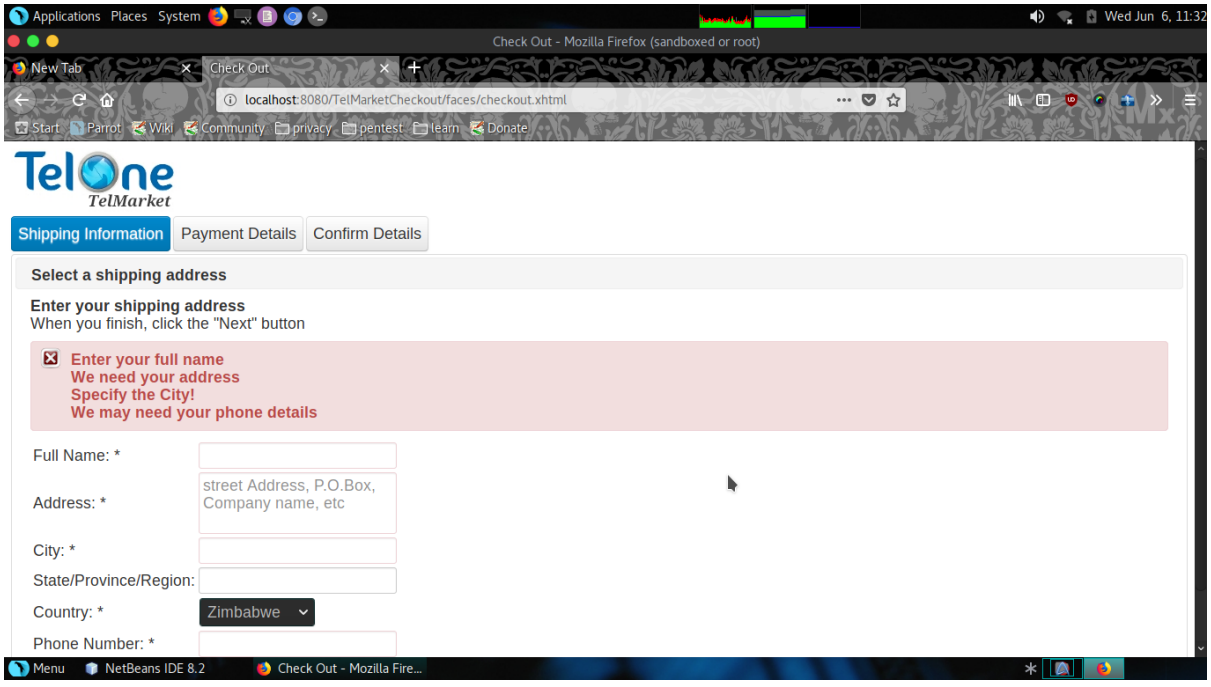


Figure 18 - 4.3 TelMarket page 3

If all the information is provided correctly, and the next button is clicked, the following is presented

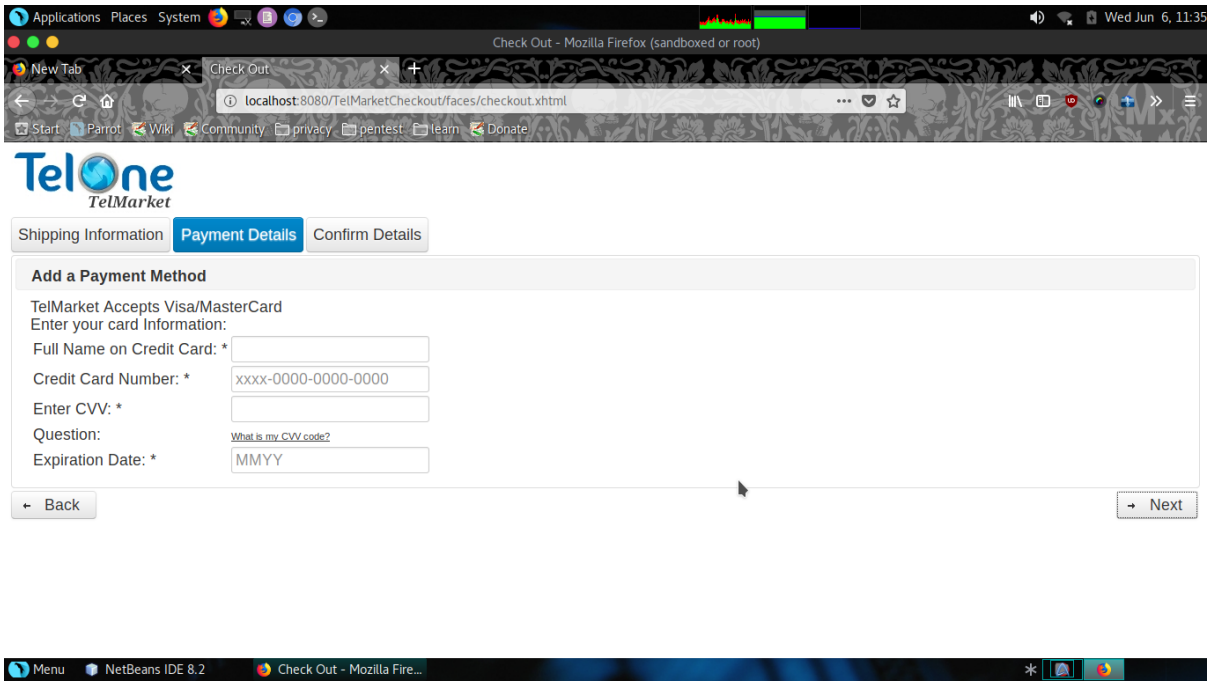


Figure 19 - 4.4 TelMarket page 4

All the data on this page is also mandatory. The Luhn algorithm is running in the background for credit card validation. The following are the exceptions that can be encountered.

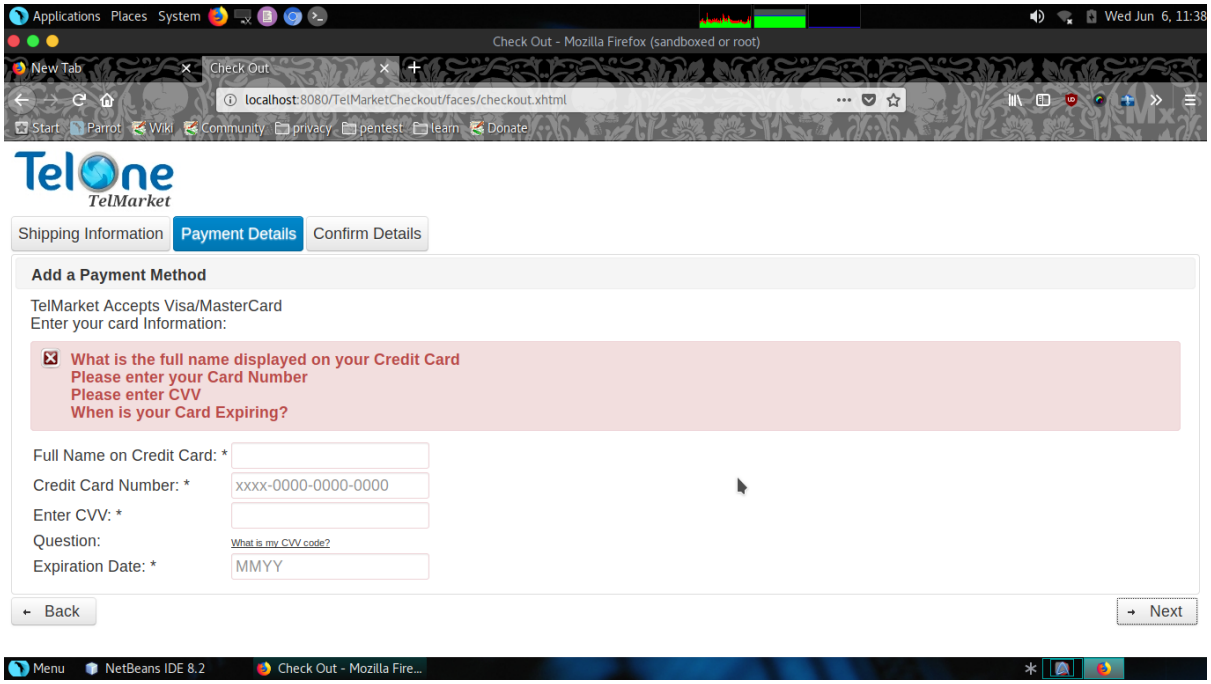


Figure 20 - 4.5 TelMarket page 5

On entering a bogus credit card number

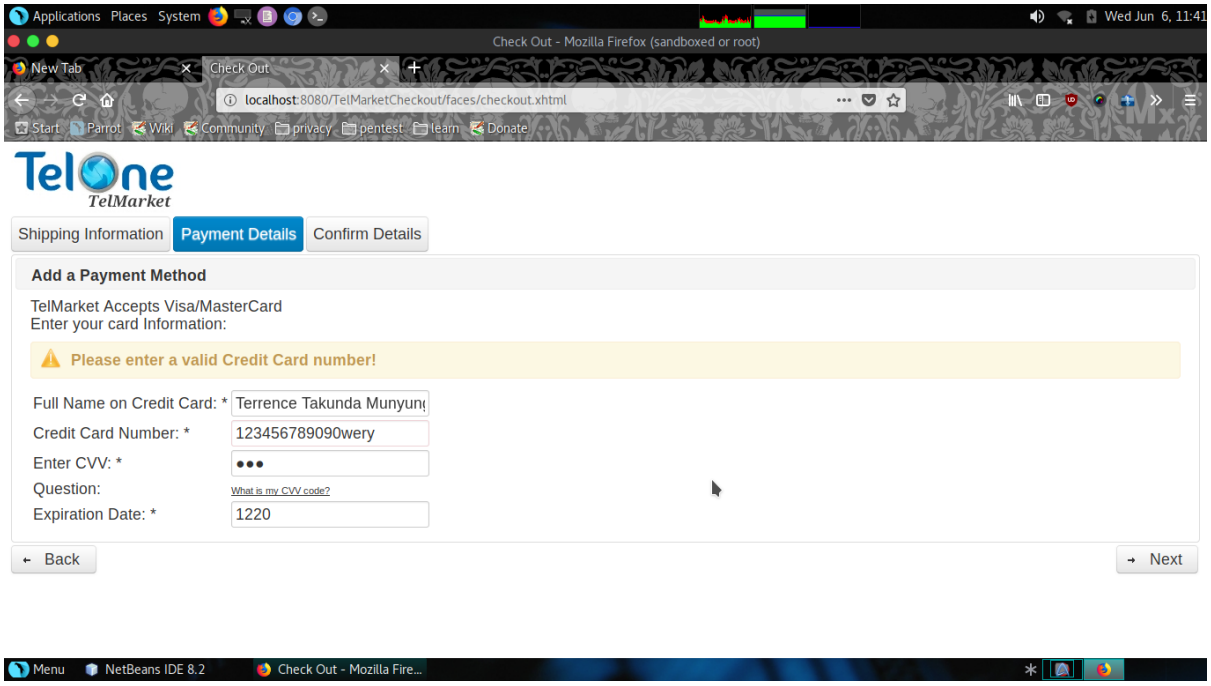


Figure 21 - 4.6 TelMarket page 6

When one clicks the question

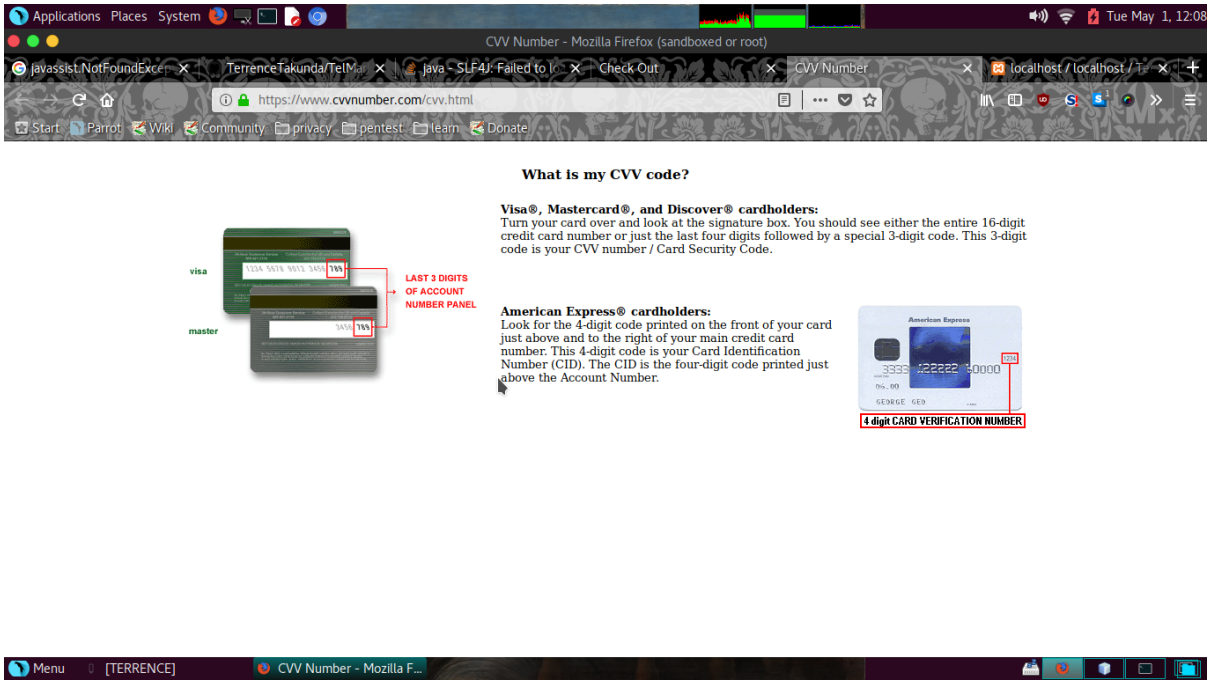


Figure 22 - 4.7 what is my CVV Number?

If all the information is entered and everything is in order, the confirmation stage will be the last step

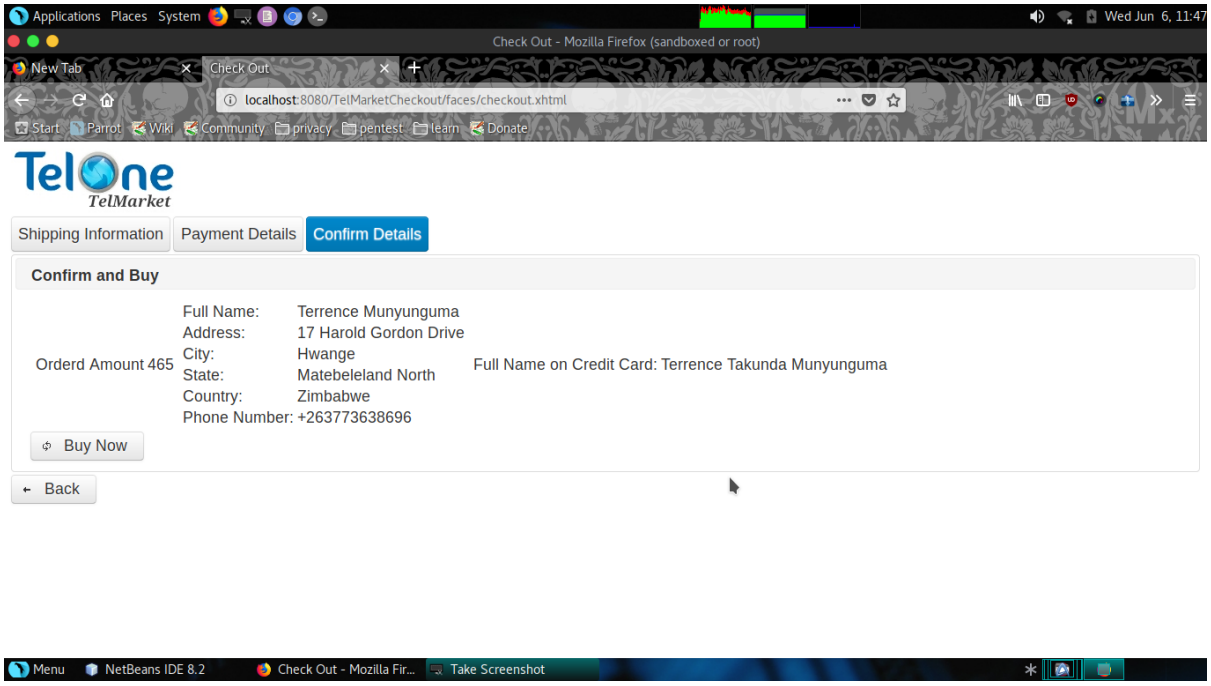


Figure 23 - 4.8 TelMarket page 7

If all the information is correct, the user presses the buy button to proceed with the transaction.

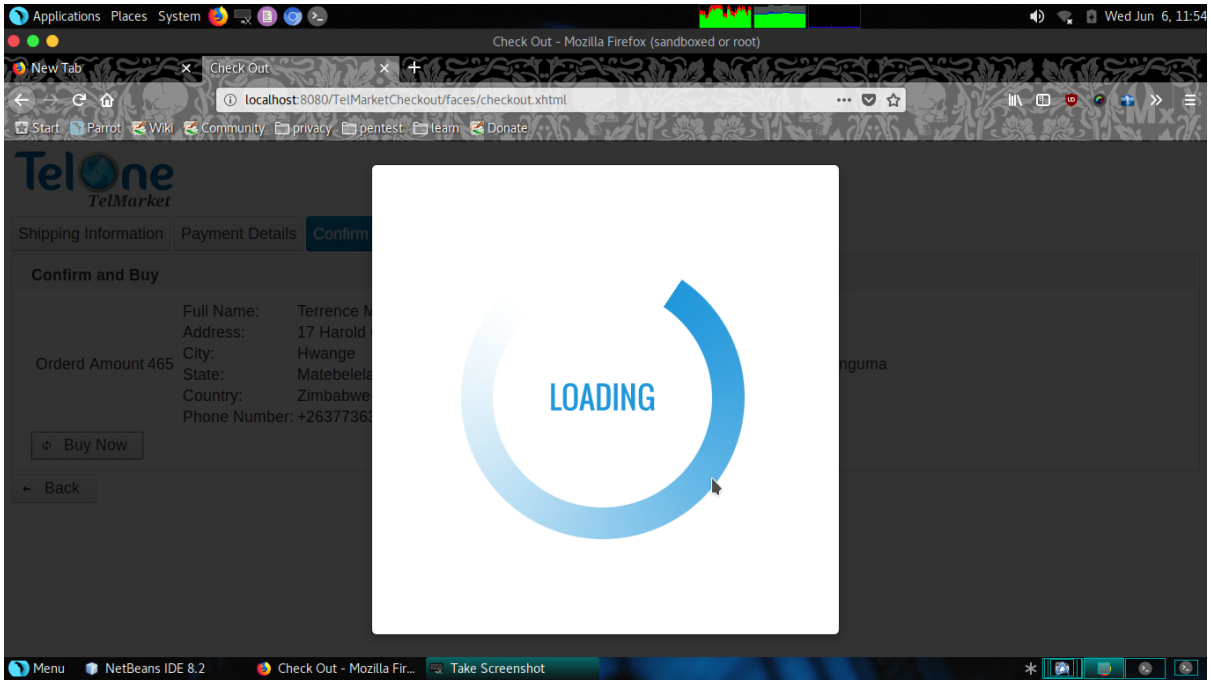


Figure 24 - 4.9 TelMarket loading screen

After about 5 seconds, the user gets the response from the sever

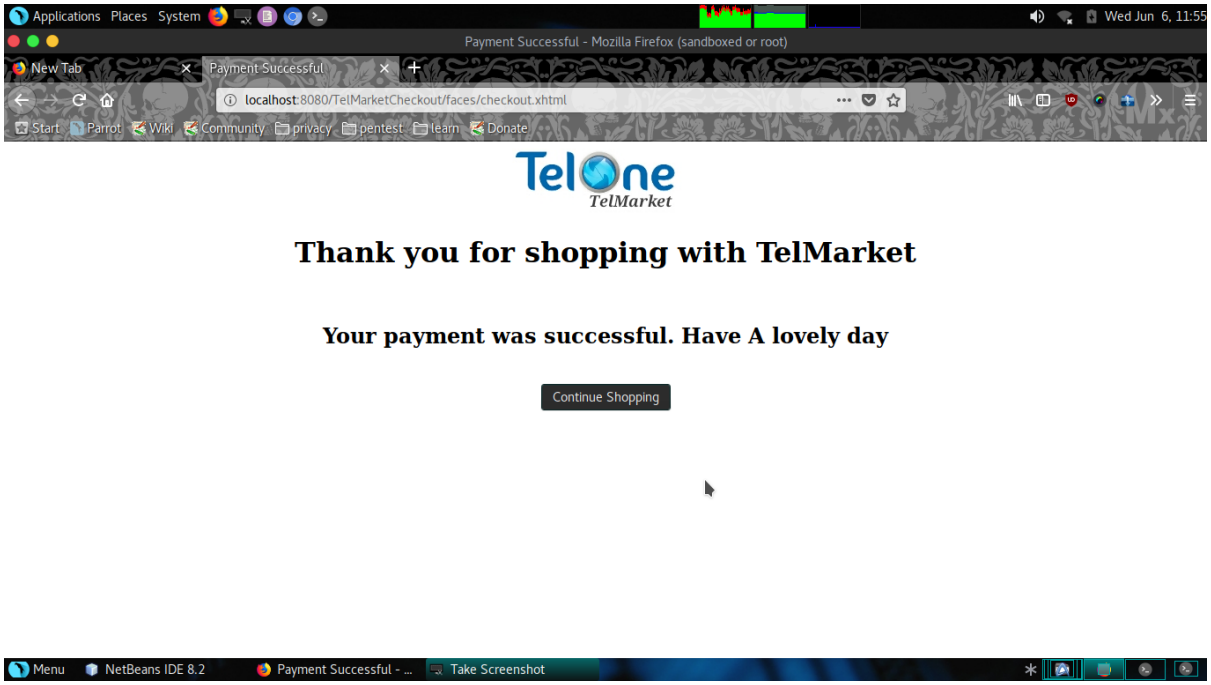


Figure 25 - 4.10 TelMarket Success page

The following is the message routing in the background from the server (glassfish)

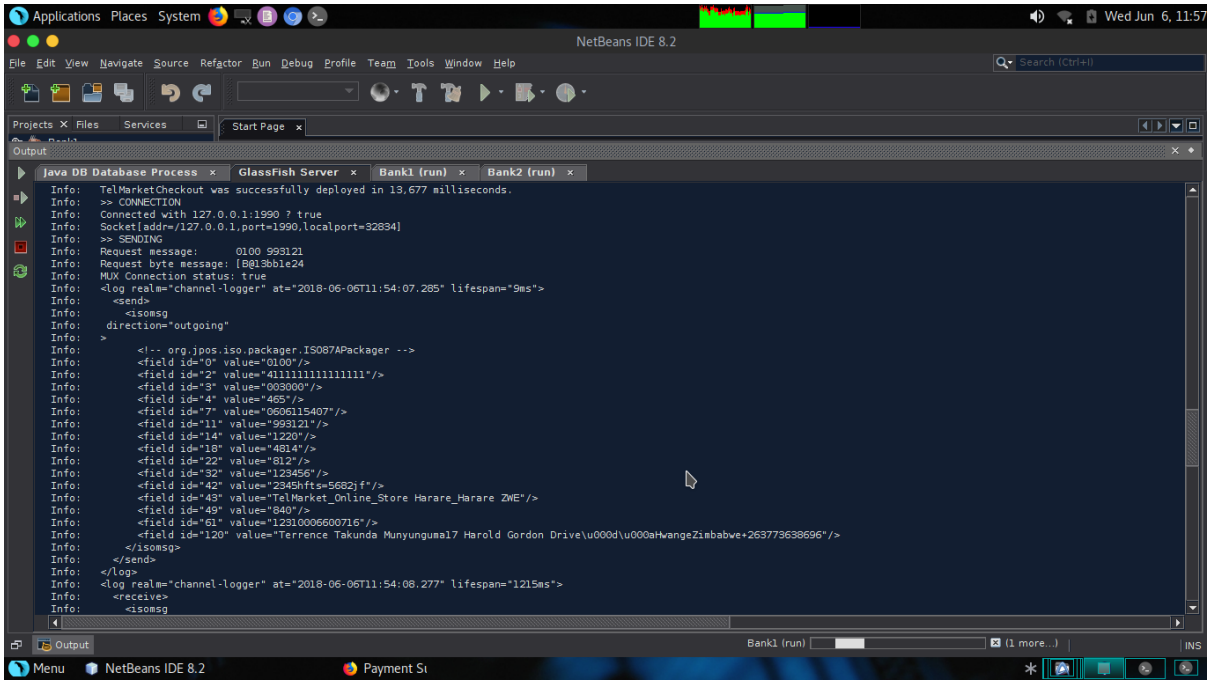


Figure 26 - 4.11 GlassFish5 Sever Log 1

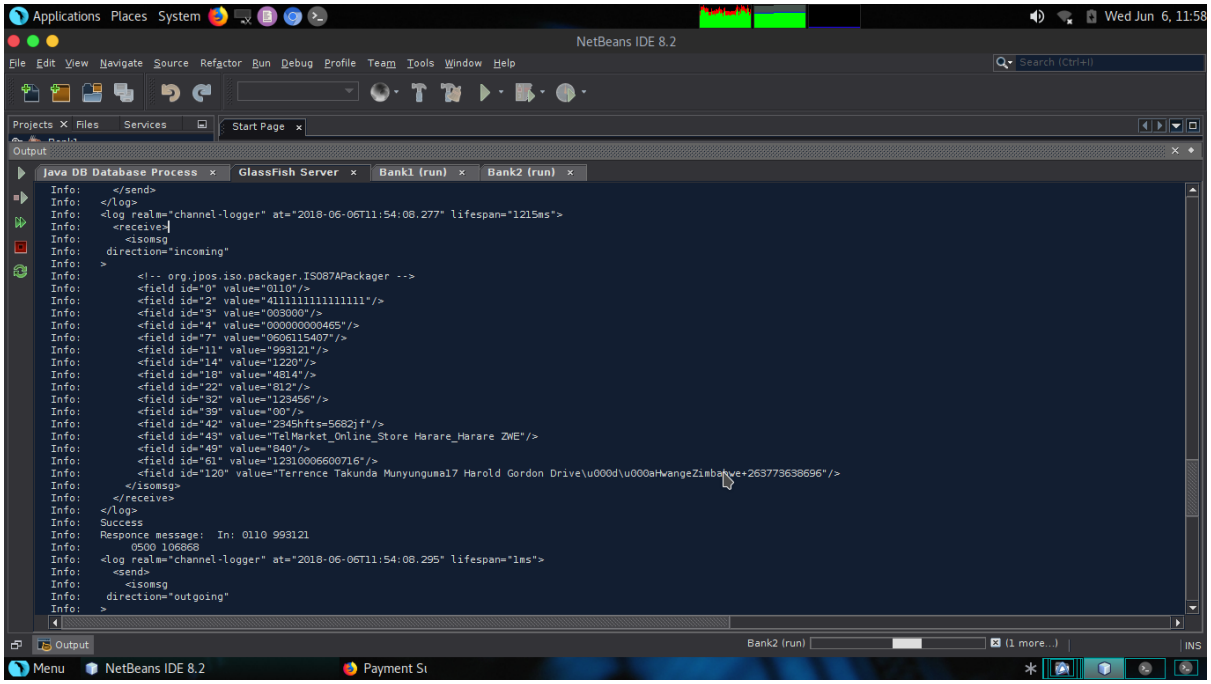


Figure 27 - 4.12 GlassFish5 Sever Log 2

The following is the server routing switch to the context destination

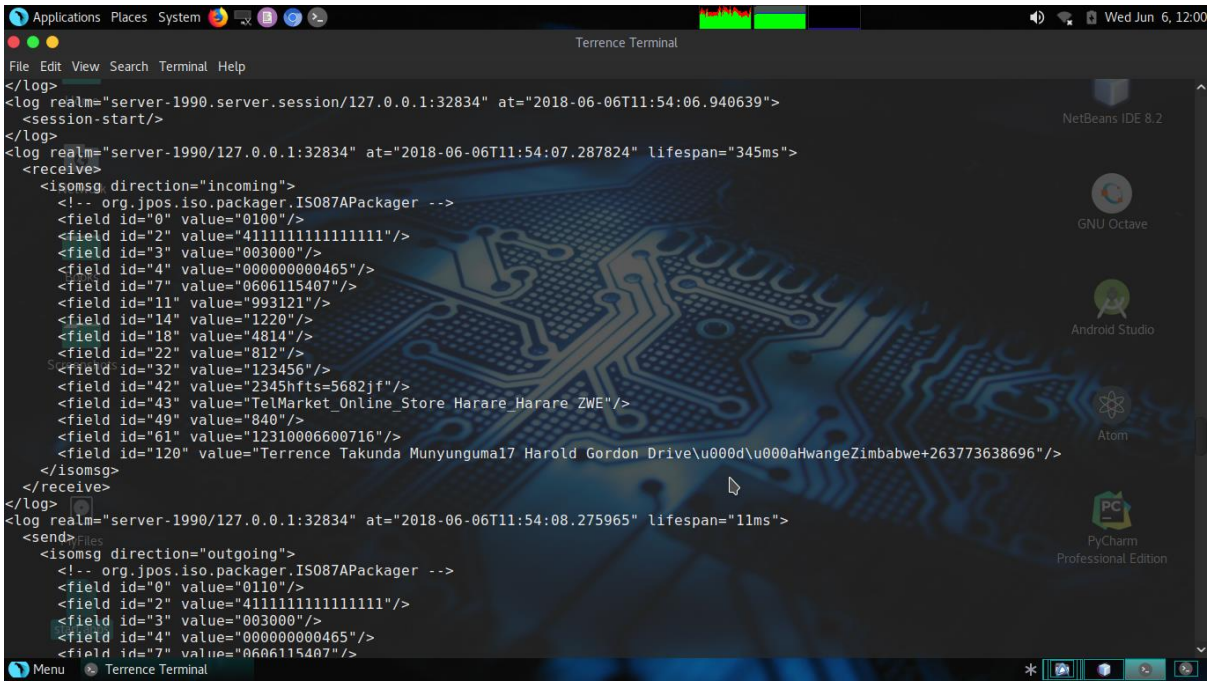


Figure 28 - 4.13 TelMarket Sever Log1

Figure 29 - 4.14 TelMarket Sever Log2

The Following shows a snippet of the payment processor simulation of authorisation response and settlement response

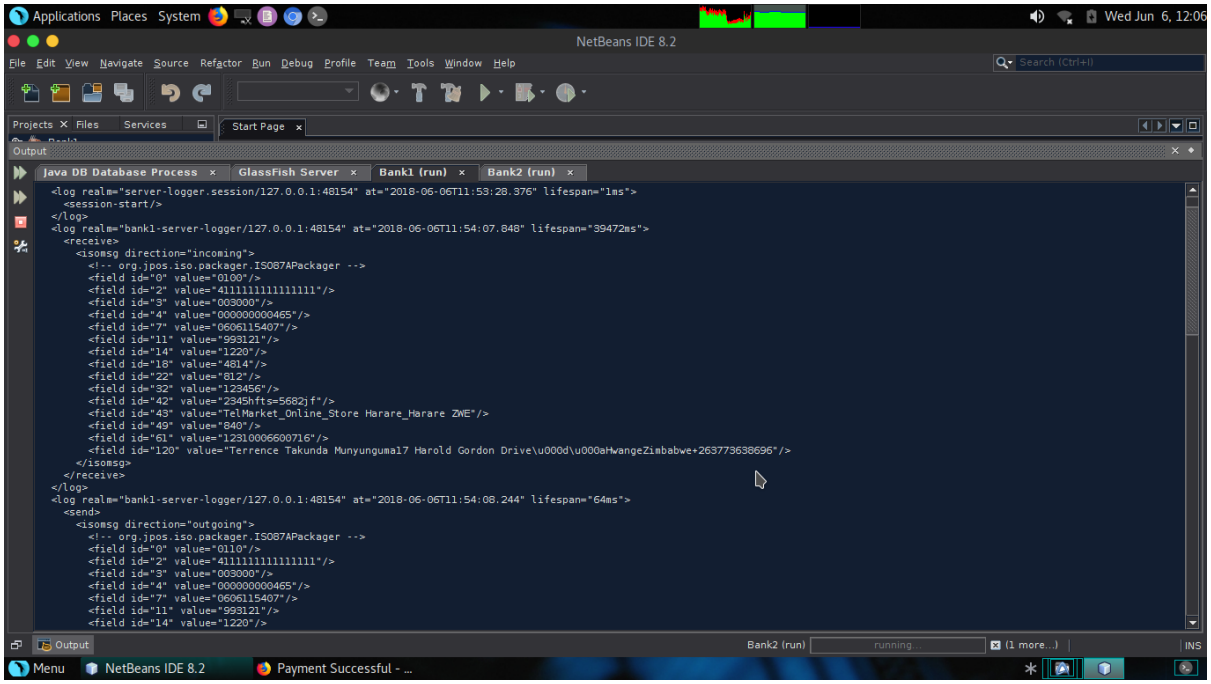


Figure 30 - 4.15 Bank Sever Log 1

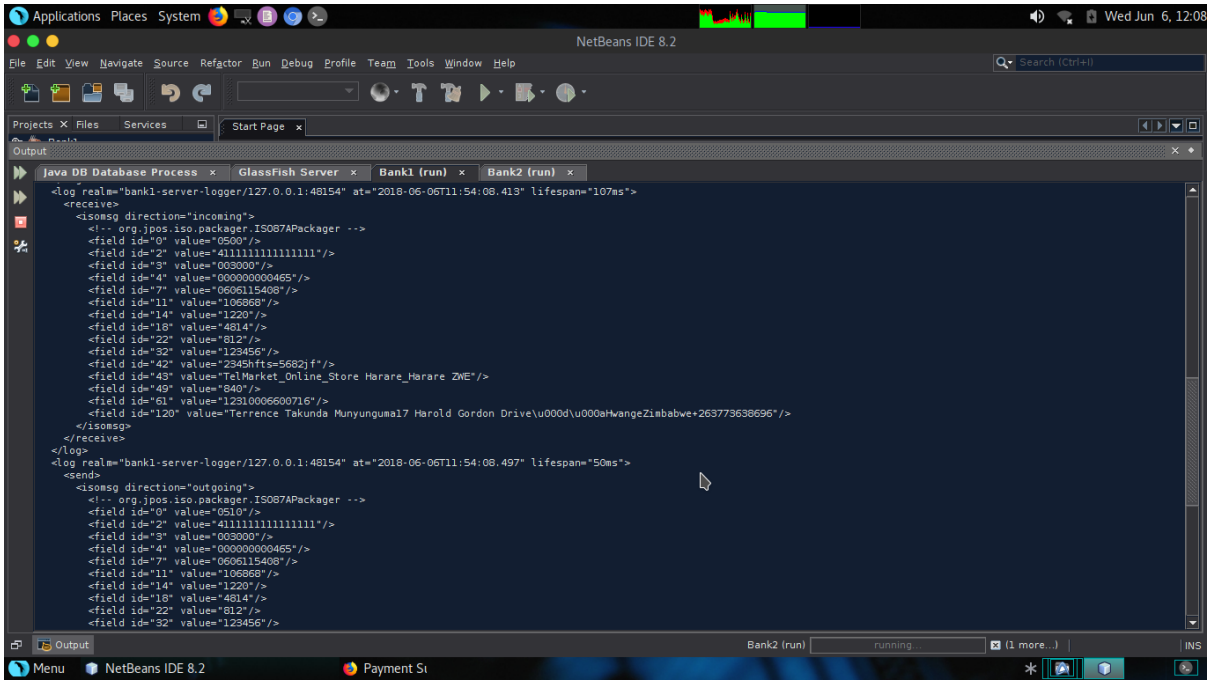


Figure 31 - 4.16 Bank Sever Log 2

The following shows the database logs for the entries persisted as the day progresses

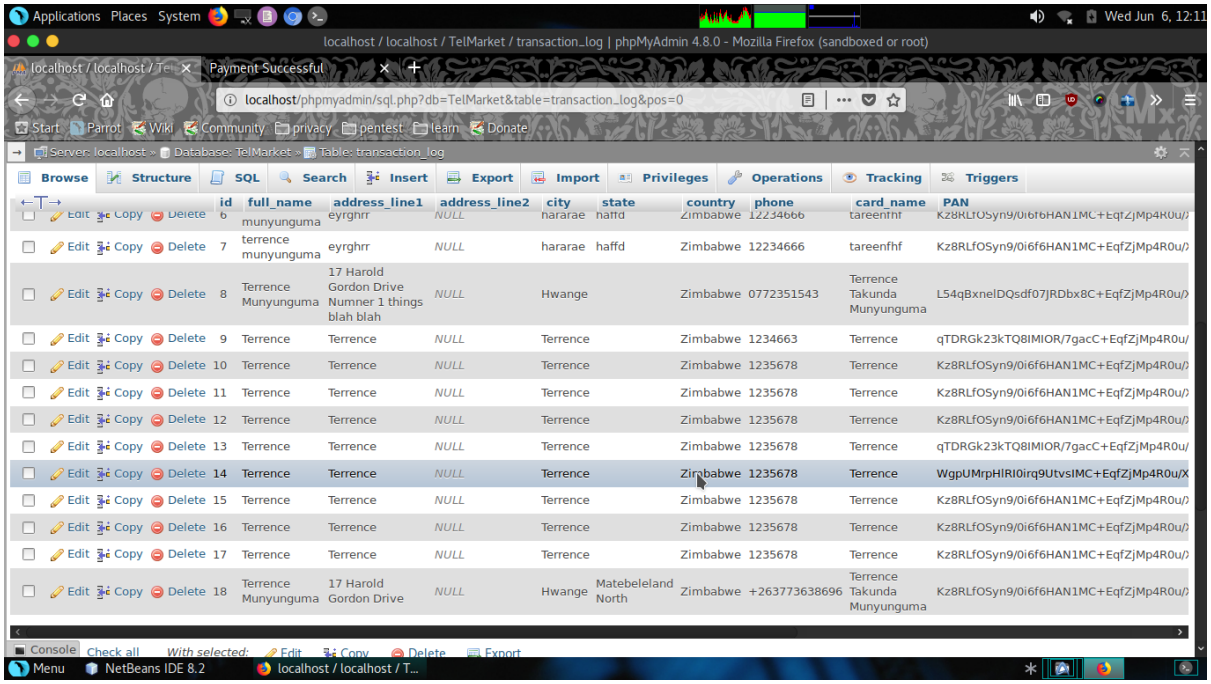


Figure 32 - 4.17 Database Log 1

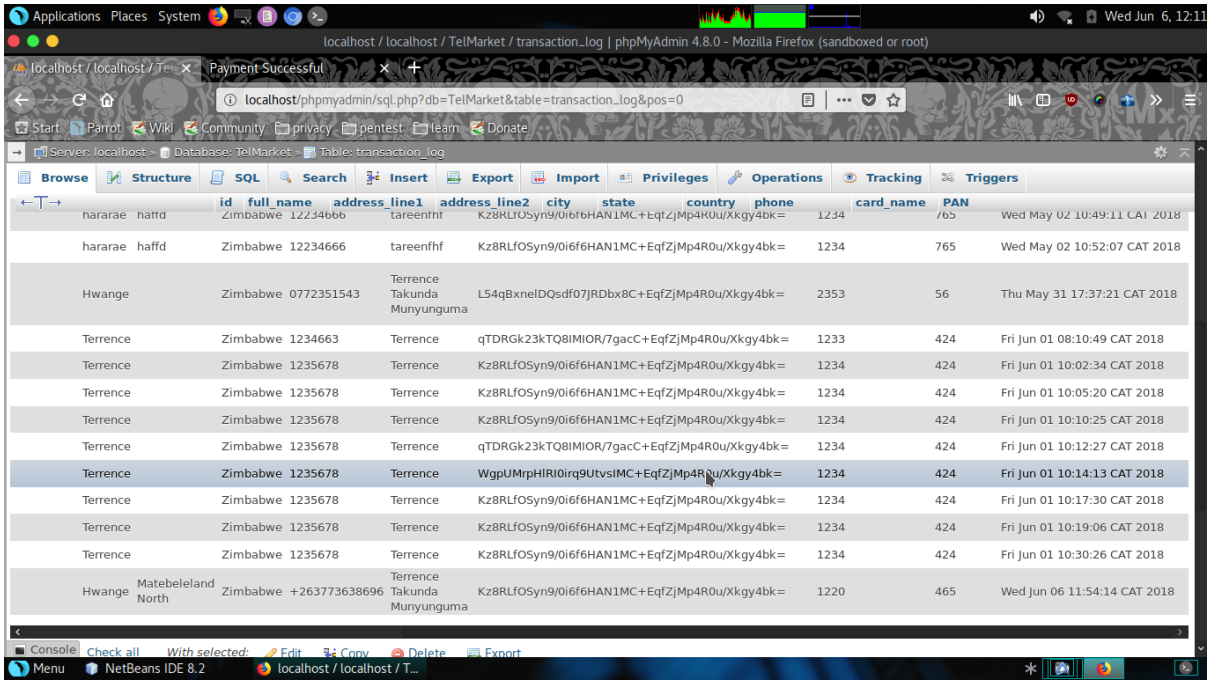


Figure 33 - 4.18 Database Log 2

CHAPTER 5 CONCLUSION AND RECOMENDATIONS

5.1 Conclusion

The payments switch/gateway in this project was built under the iso8583 standards. The goal was to build a safe and secure switch, which is usable and scalable, and competes in the current market.
The gateway handle Visa and MasterCard payment processors integrating with bank severs. The technology used was the current industry standard JPOS (A java implementation of ISO8583), as well as Java enterprise edition JSF and Primefaces.

5.1 Recommendations

Although the module is working to satisfy most of the project objectives, I would like to make the following recommendations before a consideration is made for deployment

1. Security
There is a major security concern so far, with the channels of communication not encrypted. There is need for professional systems architect to add security layers for secure channel interface to prevent man in the middle attacks and other malicious hacker exploitation's.

2. Application program Interface.
An API should be included for seamless connection to e-commerce sites. It should be easy to use to make an easier job for developers.

3. Batch Settlement
The current module uses real time settlement. I recommend the use of end of day batch settlement to minimise processing costs.

REFERENCES

JPOS Programmers Guide, Allejandro P Revilla, *jPOS 2.0.10 2017*

Java Platform, Enterprise Edition, The Java EE Tutorial Release 7 E39031-01
Copyright © 2014, Oracle Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito

MasterCard Customer Interface Specification, September 2010

Visa **BASE I Technical Specifications, Volume 2** V.I.P. SYSTEM
Effective: 1 June 2011

www.iso8583.info

https://smallbusiness.yahoo.com/advisor/shopping-via-smartphone-benefits-consumers-113000920.html

https://www.revolvy.com/topic/Consumer-to-business

https://www.scribd.com/document/362588798/E-comm-Project-by-SAHIL

http://kfo.ath.cx/windowmaker/2012_windowmaker.info/wiki/ISO_8583

http://www.fastcharge.com/woocommerce-plugin-for-fastcharge-payment-gateway/

https://emspayments.com/payment-processor-payment-gateway-merchant-account/

https://onecore.net/sample-test-cases-for-payment-gateway.html

APPENDIX A: SOURCE CODE

The source code for this project was hosted on GitHub on the author's repository.

The following link provides all the necessary material. Of Importance it the README.MD file with instructions for deployment.

https://www.github.com/TerrenceTakunda/TelMarket