ONLINE BILL PAYMENT SYSTEM

by

VENKATA SRI VATSAV REDDY KONREDDY

B. Tech., Jawaharlal Nehru Technological University, 2007

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2009

Approved by:

Major Professor
Dr. Daniel Andresen

Abstract

Keeping track of paper bills is always difficult and there is always a chance of missing bill payment dates. Online Bill Payment application is an interactive, effective and secure website designed for customers to manage all their bills. The main objective of this application is to help customers to receive, view and pay all the bills from one personalized, secure website there by eliminating the need of paper bills.

Once customers register in the website, they can add various company accounts. The information is verified with the company and the accounts are added. After the customers add the company accounts they can receive notifications about new bills, payments and payment reminders. All the information dealing with sensitive data is passed through a Secure Socket Layer for the sake of security.

This website follows MVC architecture. Struts is used to develop the application. Well established and well proven design patterns like Business Delegate, Data Access Object, and Transfer Object are used to simplify the maintenance of the application. For the communication between the website and companies, web services are used. Apache Axis2 serves as the web services container and Apache Rampart is used to secure the information flow between the web services. Tiles, JSP, HTML, CSS and JavaScript are used to provide a rich user interface. A part from these, Java Mail is used to send emails and concepts like one way hashing, certificates, key store's, and encryption are implemented for the sake of security.

The overall system is tested using unit testing, manual testing and performance testing techniques. Automated test cases are written whenever possible to ensure correctness of the functions. Manual testing further ensures that the application is working as expected. The system is subjected to different loads and the corresponding behavior is observed at different loads. The unit and manual testing revealed that the functionality of each module in the system is behaving as expected for both valid and invalid inputs. Performance testing revealed that the website works fine even when the server is subjected to huge loads.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my Major Professor Dr. Daniel Andresen for his constant help, encouragement and guidance throughout the project.

I would also like to thank Dr. Torben Amtoft and Dr. Mitchell Neilsen for serving in my committee and for their valuable cooperation during the project.

Finally, I wish to thank my family and friends for all their support and encouragement.

CHAPTER 1 - Introduction

Online Bill Payment website provides an effective, interactive and secure solution for customers to manage all their bills. Customers need not keep track of all the paper bills once they register on the website and add the company accounts. Customers will receive emails once a new bill is posted to the account or when a payment is due or when the payment is made.

In order to keep the sensitive data safely, the website uses SSL for the data transmission. The website provides a good User Interface to encourage the customers to use the website. The website is built using the J2EE technologies and also enough care is taken to support scalability and low maintenance.

Web services are used to communicate between bill payment website and the companies. Security is also implemented in the web services so that the data is transmitted securely.

1.1 Motivation

To implement the online bill payment website lot of technologies must be used. In order to differentiate this website from others in the market, the website must be built effectively. While designing an application of this type, it gives a lot of scope to learn various new technologies and new concepts. The software development life cycle can be put into practice while developing the application. The main motivation for designing this application comes from the idea that I could learn a lot of new technologies and concepts during development phases.

1.2 Objective

The main objective of the project is to build an effective and secure web site. The website must be implemented using J2EE technologies and struts frame work. Design patterns must be used to simplify the maintenance of the application.

Another objective is to secure the data by providing SSL layer for data transmission. The interaction between the website and the companies is through web services. It is also required to provide the security to the web services.

Another objective is to send notifications to customer via emails whenever a new bill is posted to the account or a payment is processed.

Another objective is to store the passwords in encrypted format.

The last objective is to provide a good user interface to the customer to encourage using the web site.

Various technologies like JSP's, JDBC, Java Mail, Struts Framework, Struts Tiles, web services, design patterns must be used and security concepts like SSL, certificates, one way hashing must be implemented.

1.3 Salient Features of the Website

The website provides the following features:

- Users can register in the website and login to their account
- Users can browse the various companies to configure the account with the company
- Users can configure the company accounts
- Users can view the current pending bills and make payments
- Users can view the payment history
- Reset password feature which allows to reset the password if a user forgets their password and an email will be sent to them
- Users can receive emails when a new bill is posted and when a payment is processed
- Passwords are encrypted

1.4 Document Overview

The first part of the document describes the features, motivation and objectives of website. Chapter 2 explains the technologies and tools used in the project. Chapter 3 gives the general system architecture and implementation details of the website. The screen shots of the project are shown in the next section. Chapter 4 gives the testing results and chapter 5 gives the conclusion along with problems faced and future work. The last section has various references that are used while designing this application.

CHAPTER 2 - Related Work

2.1 Tools and Technologies

2.1.1 J2EE

The J2EE platform is a collection of related technology specifications that describe required API's and policies. It aims to simplify the design and implementation of enterprise applications. J2EE provides vast API's which can be re used while developing an application.

I have used the following technologies in J2EE.

- JSP: Java Server Pages technology is used to create dynamic web content. It is a simple, yet powerful technology for creating and maintaining dynamic-content web pages. JSP technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.
- JDBC: The Java Database Connectivity API provides a standard way to access relational databases from the application.
- JavaMail: JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications. JavaMail supports different protocols like SMTP, POP3 and IMAP.

2.1.2 Struts

Apache Struts is an open source framework for building J2EE web applications. Struts provide the basic infrastructure for implementing MVC (Model View Controller) thereby allowing the separation of business logic and data of the application from the presentation of data to the user.

2.1.3 Struts Tiles

Struts Tiles is a template mechanism which allows the presentation layer to be composed from various independent web page components.

2.1.4 Web Services

Web services are web based applications that use open, XML-based standards and transport protocols to exchange data with calling clients [1]. Web services are based on a set of standards like SOAP, WSDL and UDDI.

- SOAP: SOAP is a protocol by which a remote client can invoke the functionality implemented by the web service. SOAP usually uses XML as its message format. SOAP is platform independent and also language independent.
- WSDL: WSDL is used to define metadata describing the web service. WSDL is used both by developers and clients. Developers use WSDL to define the web service and clients use WSDL to invoke the web service by learning about the arguments and the way to call the web service.
- UDDI: Web services are published over the internet using UDDI. Clients can use UDDI to find information about the web services.

2.1.5 Design Patterns

A Design pattern is a reusable solution to a commonly occurring problem in software design. Design patterns can speed up the development process by providing tested, proven development paradigms. I have used Model View Controller pattern, Delegate Pattern, Data Access Object pattern and Transfer Object pattern.

- MVC Pattern: The application is designed based on the MVC pattern. The Model represents enterprise data and the business rules that govern access to and updates if this data. The View renders the enterprise data through the model and specifies how the data should be presented to the client. The controller selects the appropriate view based on the user interactions and the outcome of the model actions. A change in any one of the layer doesn’t affect the other layers.
- Delegate Pattern: Delegate pattern helps to reduce coupling between the presentation tier and business services. Delegate objects are used for the sake of accessing the business logic which resides in another layer, typically a DAO [6].
- DAO Pattern: DAO pattern separates a data resource’s client interface from its data access mechanisms. It allows data access mechanisms to change independently of the code that uses the data [7].

- Transfer Object Pattern: Transfer object is used to reduce the number of network calls and network overhead. When a client requests some attribute values, instead of making many remote calls to get the values, a single remote method invocation can be made using a transfer object [8].

2.1.6 One Way Hashing

It is not a good idea to store the passwords or other sensitive information in the database in plain text format. Instead the passwords and other sensitive information can be encrypted and then stored in the database. Passwords are encrypted using one way hash algorithms because it would be impossible to guess the password by seeing the encrypted data and there is no way the password can be decrypted. I have used SHA algorithm to encrypt the password.

2.1.7 HTTPS Protocol

HTTPS is used to secure the sensitive data from hackers during transmission. HTTPS is a combination of the HTTP along with SSL/TLS protocol to provide encryption and secure information [12]. HTTPS creates a secure channel over a network. A server certificate is used when HTTPS is implemented. Usually this certificate must be verified by a certificate authority. For testing purposes a key store is generated using the java key tool. When a client requests the page, a certificate would be presented to the client. If the client accepts the certificate then the secure channel would be created and all the information would be passed over the secure channel.

2.1.8 Web Services Security

By default, the data communication between the web services is not encrypted. It is always a good idea to encrypt the data. In other words the SOAP data needs to be secured. I have used Apache Rampart to implement the security for the web services. After implementing the web services security, when two web services communicate they exchange certificates. A web service WS1 encrypts the data that needs to be sent to other web service WS2 using the WS2 public key and WS2 would decrypt using its private key and vice versa. So the data is always sent in an encrypted format. Rampart provides the implementation details of the public and private keys, certificates needed for the security.

In addition to the above technologies HTML, CSS and JavaScript are used to design the application.

CHAPTER 3 - Design and Implementation

3.1 System Architecture

3.1.1 Struts MVC Architecture

The System is built based on MVC architecture. Struts provides the infrastructure for implementing the application using MVC. The Model, View and Controller components are separated and each component is independent of the other. It encourages the reusability of the code. In struts, the ActionServlet class acts as the controller. The jsp files are present in the view layer and the business logic related to the application is present in the model layer.

The struts-config.xml contains all the information about the Action Classes, Action Forms, Action Mappings and Action Forwards. When a client makes a request the Action Servlet calls the corresponding action class by using the struts-config.xml file. After the control reaches the Action Class then the necessary operations that reside in the model layer will be called and after getting back the response the action class will notify the container to forward the client to a particular view.

Since the layers are independent to each other, a change in one layer doesn't affect the other layers. So even if some logic needs to be changed at some point it can be done very easily without affecting the other parts of the application.

The overall performance and ease of use would be increased if MVC architecture is used. But in order to further improve the maintenance of the application, business logic needs to be refined. The way the business logic is written and the way to access the business logic needs to be enhanced. This is the reason to use design patterns. Design patterns provide an elegant and well proved solution approach.

The general flow of the control in the application is as follows:

The ActionServlet class forwards the control to the appropriate action class. From the action class the business logic will be accessed by using a Delegate Class. The Delegate class will call the DAOFactory class and get the appropriate DAOImpl object. The DatabaseDAOFactory class contains the method to create a database connection and methods to get the DAOImpl objects. The DAOImpl class contains the business logic. In the future if some other developer wants a different method implementation then it would be enough to implement the DAO Interface and provide a method implementation. Also if the underlying relational database needs to be changed then it would be enough to change it in the DatabaseDAOFactory class. So the code is loosely coupled which improves reusability and efficiency.

3.1.2 Web Services

Some functions call Web services. By using Axis-2, the Stub and CallBackHandler classes are generated automatically. So when a web service needs to be called then the Stub is initialized and called to get a response object from the web service.

3.1.3 Securing the data using Rampart

By using Rampart the data is encrypted and sent to the web service. All the incoming messages are decrypted. Rampart support is added to both server and client.

The system architecture is shown in the following figure.

3.2 Use Case Diagrams

The Use case Diagram describes the system functionality from an external observer's view. It shows who can do what. The following are the use case diagrams for my application. The first one shows the customer and the second use case shows the company activities.

The customer can perform various activities like Register or setup an account with the website, Login, update the profile, browse the list of companies, configure accounts with the company, view the current bills and payment history and logout.

The company can perform activities like verifying the user accounts, sending bills to customers and processing customer payments.

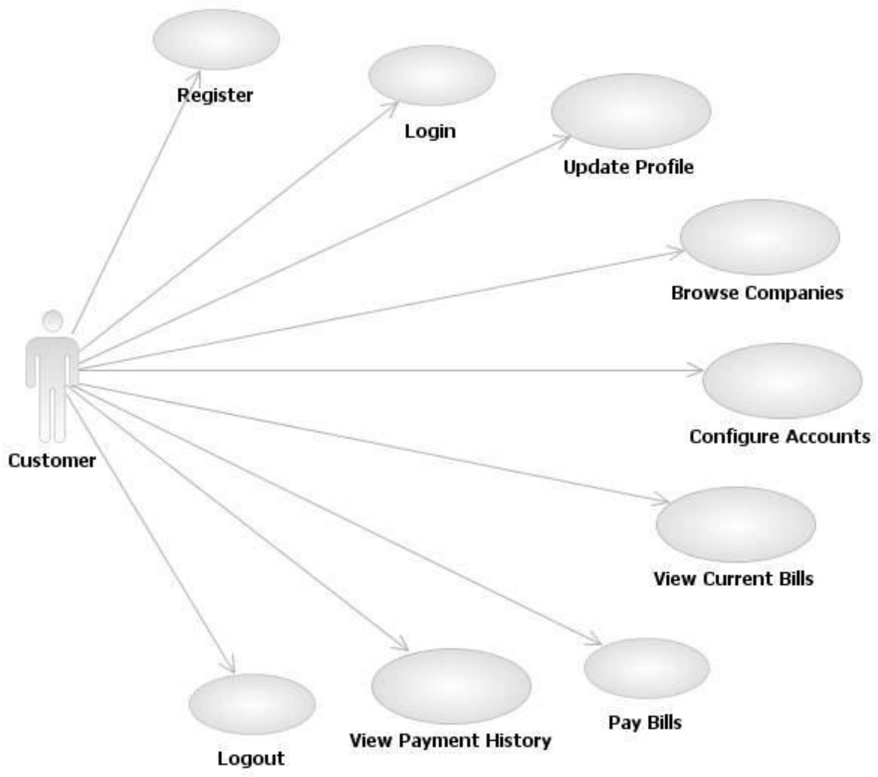


Figure 3-1 Customer Use Case Diagram

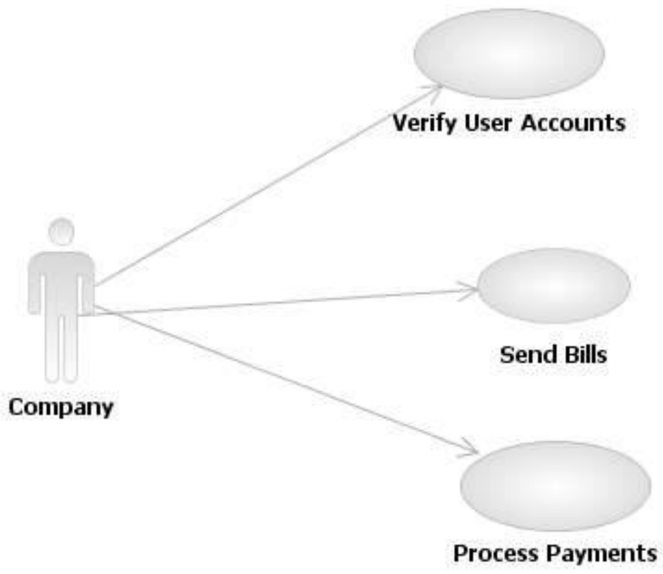


Figure 3-2 Company Use Case Diagram

3.3 Class Diagrams

Below are the class diagrams that show the various classes that I have used. The general design that I have used for designing the classes is that for each module, there will be

i. A Form class which extends from ActionForm class provided by Struts framework
ii. An Action class which extends DispatchAction class provided by Struts framework
iii. A DAO interface which has the methods related to the module
iv. A DAOImpl class which implements DAO
v. A VO class which has the attributes related to the module
vi. A Delegate class which has the required method and which returns the same method implementation present in DAOImpl class

There are some modules for which one of the above classes is not present. But overall I tried to follow a similar structure whenever possible so as to promote reuse and flexibility.

3.3.1 Class Diagram for Login Module

I will explain the flow by using one of the modules as an example. Consider the login module.

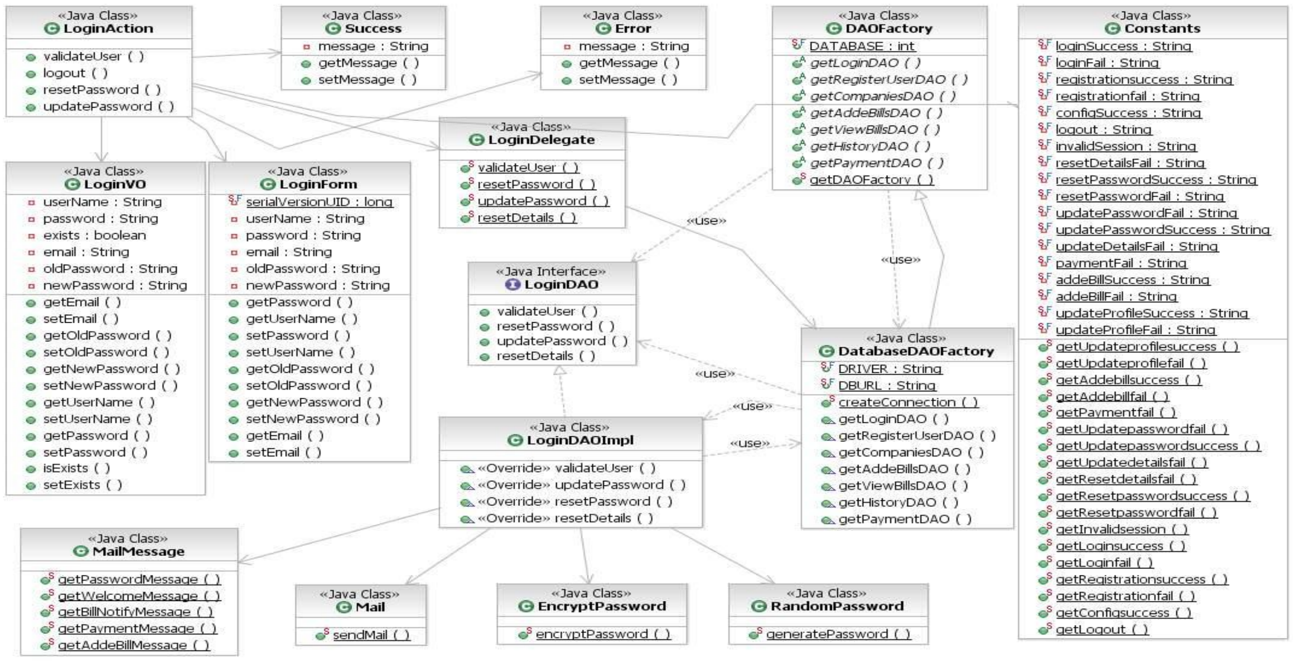


Figure 3-3 Class Diagram for Login Module

The control first reaches the LoginAction class and then the validateUser method of the LoginAction class is called. The LoginForm is sent along with the method call. The attributes of login form are retrieved and set to a LoginVO object. Then the validateUser method in the LoginDelegate class is called by passing the LoginVO object. The validateUser method in LoginDelegate class in turn calls the getLoginDAOImpl method of DatabaseDAOFactory class. This method returns an implementation of the LoginDAO interface. The logic related to validating the user is present in the LoginDAOImpl class which is the implementation of LoginDAO interface. After the validateUser method in LoginDelegate class gets the implementation it returns the result to the LoginAction class. Depending on the result the action class will send a ForwardAction object to the container which matches the action forwards defined in the struts-config.xml file and redirects to an appropriate view.

3.3.2 Configuring an account with company module

The overall flow is similar to the above described flow but in addition the business logic deals with interacting with the web services. The classes VerifyAccountStub, VerifyAccountResponse, VerifyAccountCallBackHandler are auto generated by the axis2 plug-in.

The next class diagrams show the user registration module and also the payment in the company module.

I have used some more classes like Success, Error, and Constants. These classes contain the message strings that need to be displayed to the user depending on an action. The Mail class contains the logic to send an email to the customer at the specified email address during registration process. The MailMessage class contains the mail body strings that need to be sent to the customer depending on the situation. The Encryptpassword class contains the method to encrypt the password using SHA algorithm. The GeneratePassword class contains the method to auto generate a password.

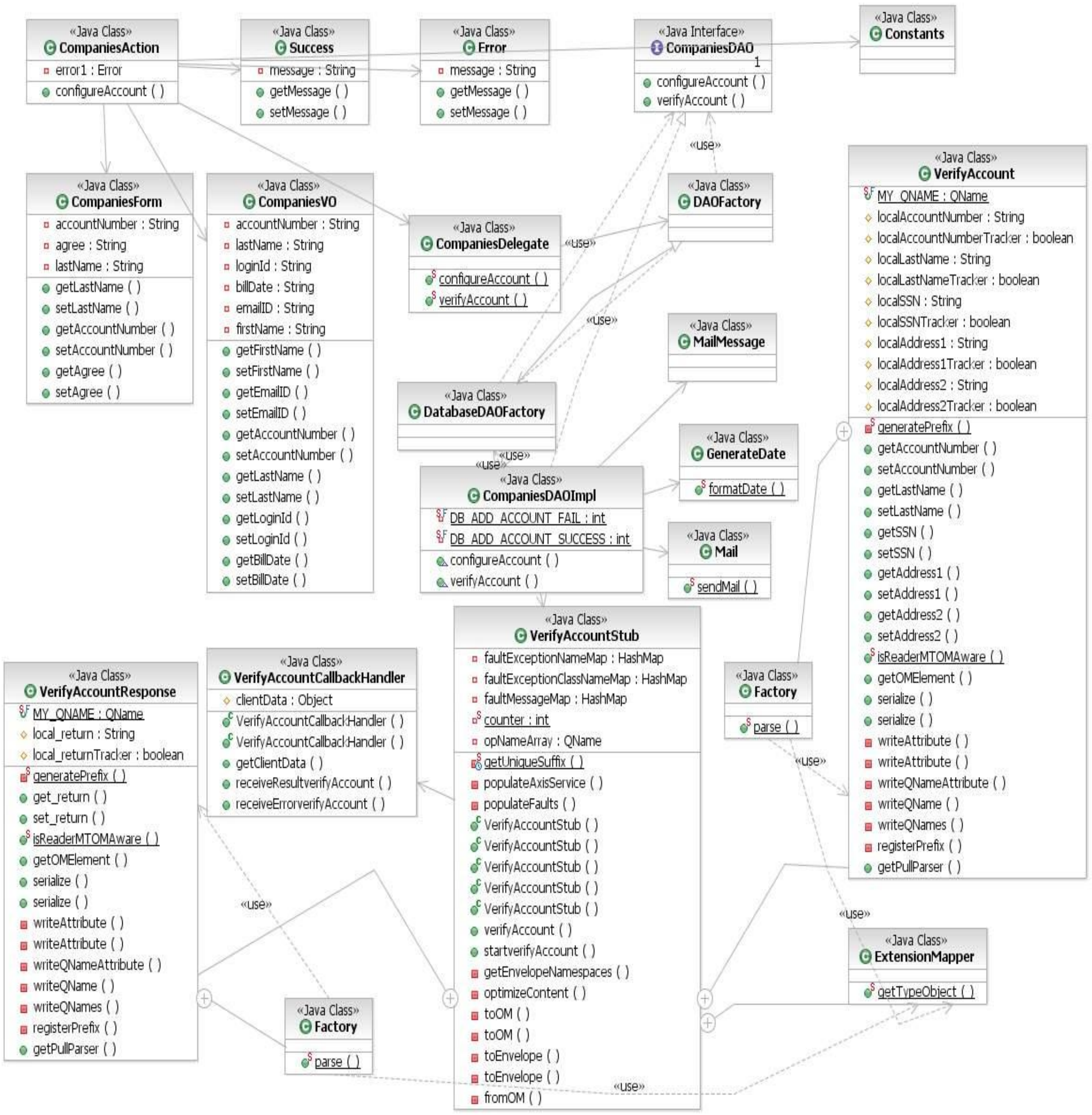


Figure 3-4 Class Diagram for Configuring an Account Module

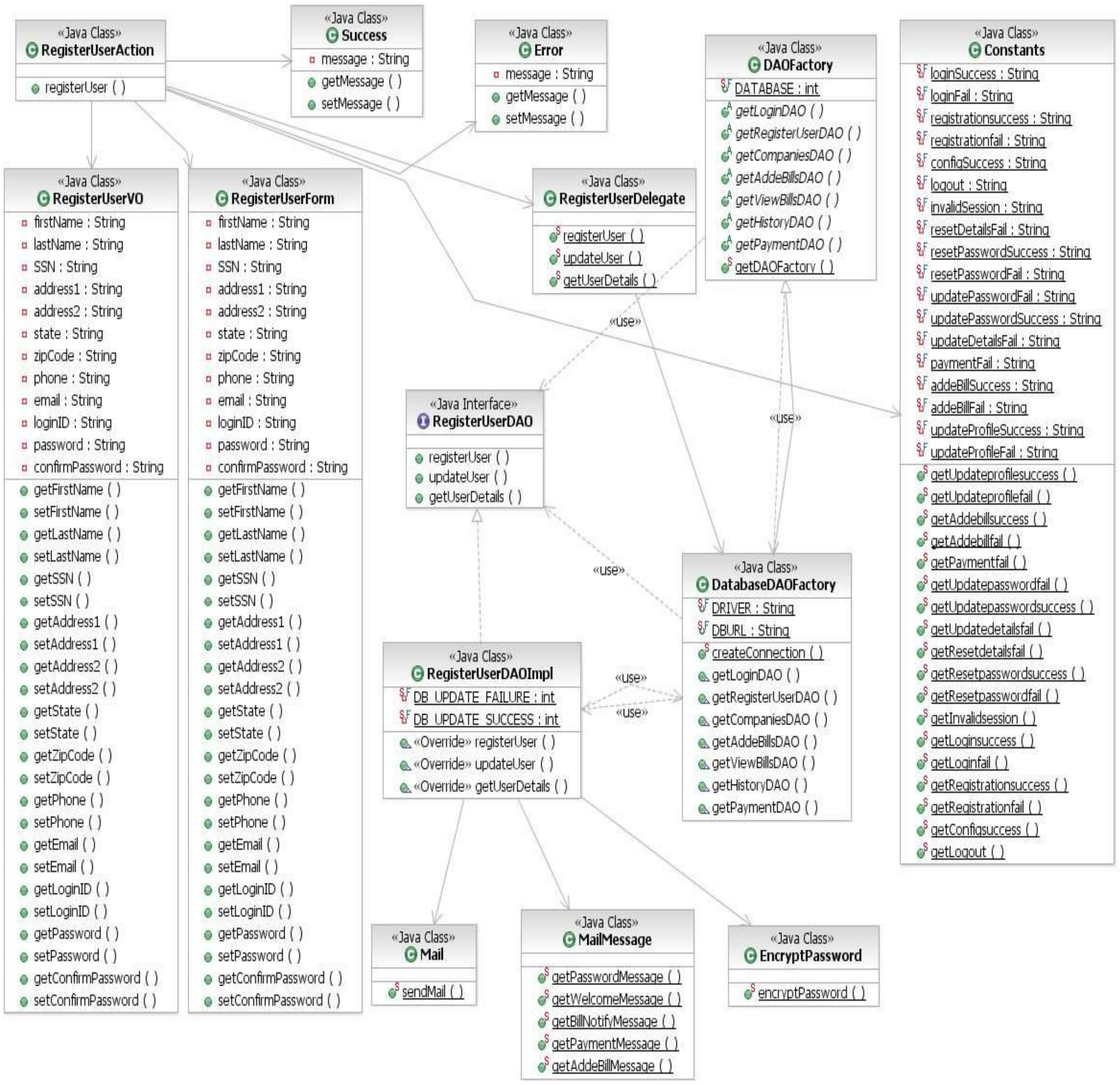


Figure 3-5 Class Diagram for Registration Module

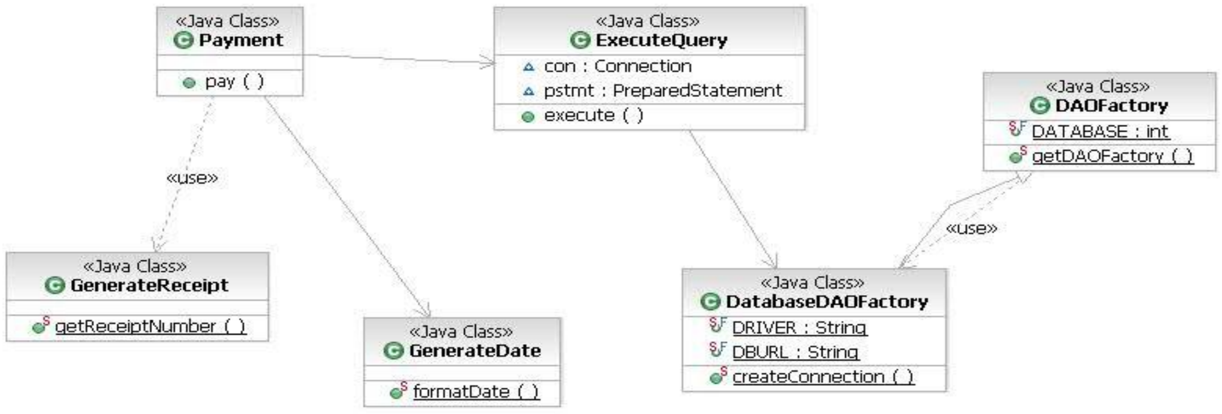
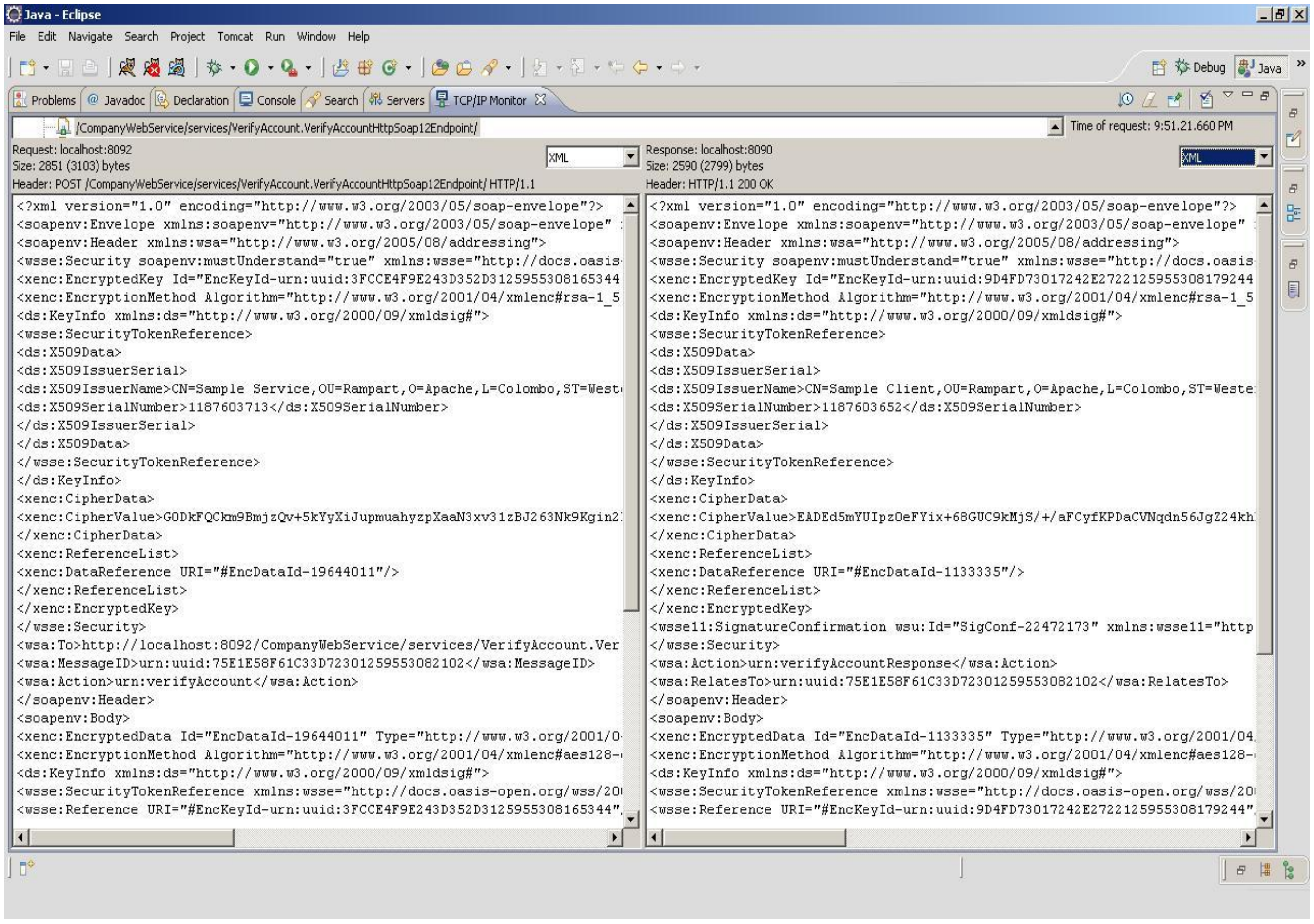


Figure 3-6 Class Diagram for Payment Module

The following screenshot shows the encrypted XML request and XML response in web services


Java - Eclipse
File Edit Navigate Search Project Tomcat Run Window Help
Debug Java
Problems Javadoc Declaration Console Search Servers TCP/IP Monitor

/CompanyWebService/services/VerifyAccount.VerifyAccountHttpSoap12Endpoint/

Time of request: 9:51.21.660 PM

Request: localhost:8092
Size: 2851 (3103) bytes
Header: POST /CompanyWebService/services/VerifyAccount.VerifyAccountHttpSoap12Endpoint/ HTTP/1.1

XML

```
<?xml version="1.0" encoding="http://www.w3.org/2003/05/soap-envelope"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
<soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsse:Security soapenv:mustUnderstand="true" xmlns:wsse="http://docs.oasis
<xenc:EncryptedKey Id="EncKeyId-urn:uuid:3FCCE4F9E243D352D3125955308165344
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference>
<ds:X509Data>
<ds:X509IssuerSerial>
<ds:X509IssuerName>CN=Sample Service,OU=Rampart,O=Apache,L=Colombo,ST=West
<ds:X509SerialNumber>1187603713</ds:X509SerialNumber>
</ds:X509IssuerSerial>
</ds:X509Data>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>GODkFQCkm9BmjzQv+5kYyXiJupmuahyzpXaaN3xv31zBJ263Nk9Kgin2
</xenc:CipherData>
<xenc:ReferenceList>
<xenc:DataReference URI="#EncDataId-19644011"/>
</xenc:ReferenceList>
</xenc:EncryptedKey>
</wsse:Security>
<wsa:To>http://localhost:8092/CompanyWebService/services/VerifyAccount.Ver
<wsa:MessageID>urn:uuid:75E1E58F61C33D72301259553082102</wsa:MessageID>
<wsa:Action>urn:verifyAccount</wsa:Action>
</soapenv:Header>
<soapenv:Body>
<xenc:EncryptedData Id="EncDataId-19644011" Type="http://www.w3.org/2001/0
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-open.org/wss/20
<wsse:Reference URI="#EncKeyId-urn:uuid:3FCCE4F9E243D352D3125955308165344".
```

Response: localhost:8090
Size: 2590 (2799) bytes
Header: HTTP/1.1 200 OK

XML

```
<?xml version="1.0" encoding="http://www.w3.org/2003/05/soap-envelope"?>
<soapenv:Envelope xmlns:soapenv="http://www.w3.org/2003/05/soap-envelope"
<soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
<wsse:Security soapenv:mustUnderstand="true" xmlns:wsse="http://docs.oasis
<xenc:EncryptedKey Id="EncKeyId-urn:uuid:9D4FD73017242E2722125955308179244
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference>
<ds:X509Data>
<ds:X509IssuerSerial>
<ds:X509IssuerName>CN=Sample Client,OU=Rampart,O=Apache,L=Colombo,ST=Weste
<ds:X509SerialNumber>1187603652</ds:X509SerialNumber>
</ds:X509IssuerSerial>
</ds:X509Data>
</wsse:SecurityTokenReference>
</ds:KeyInfo>
<xenc:CipherData>
<xenc:CipherValue>EADEd5mYUIpzOeFYix+68GUC9kMjS/+/aFCyfKPDaCVNqdn56JgZ24kh
</xenc:CipherData>
<xenc:ReferenceList>
<xenc:DataReference URI="#EncDataId-1133335"/>
</xenc:ReferenceList>
</xenc:EncryptedKey>
<wsse11:SignatureConfirmation wsu:Id="SigConf-22472173" xmlns:wsse11="http
</wsse:Security>
<wsa:Action>urn:verifyAccountResponse</wsa:Action>
<wsa:RelatesTo>urn:uuid:75E1E58F61C33D72301259553082102</wsa:RelatesTo>
</soapenv:Header>
<soapenv:Body>
<xenc:EncryptedData Id="EncDataId-1133335" Type="http://www.w3.org/2001/04
<xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-open.org/wss/20
<wsse:Reference URI="#EncKeyId-urn:uuid:9D4FD73017242E2722125955308179244".
```

Figure 3-7 Encrypted XML Data

3.4 Database Diagram

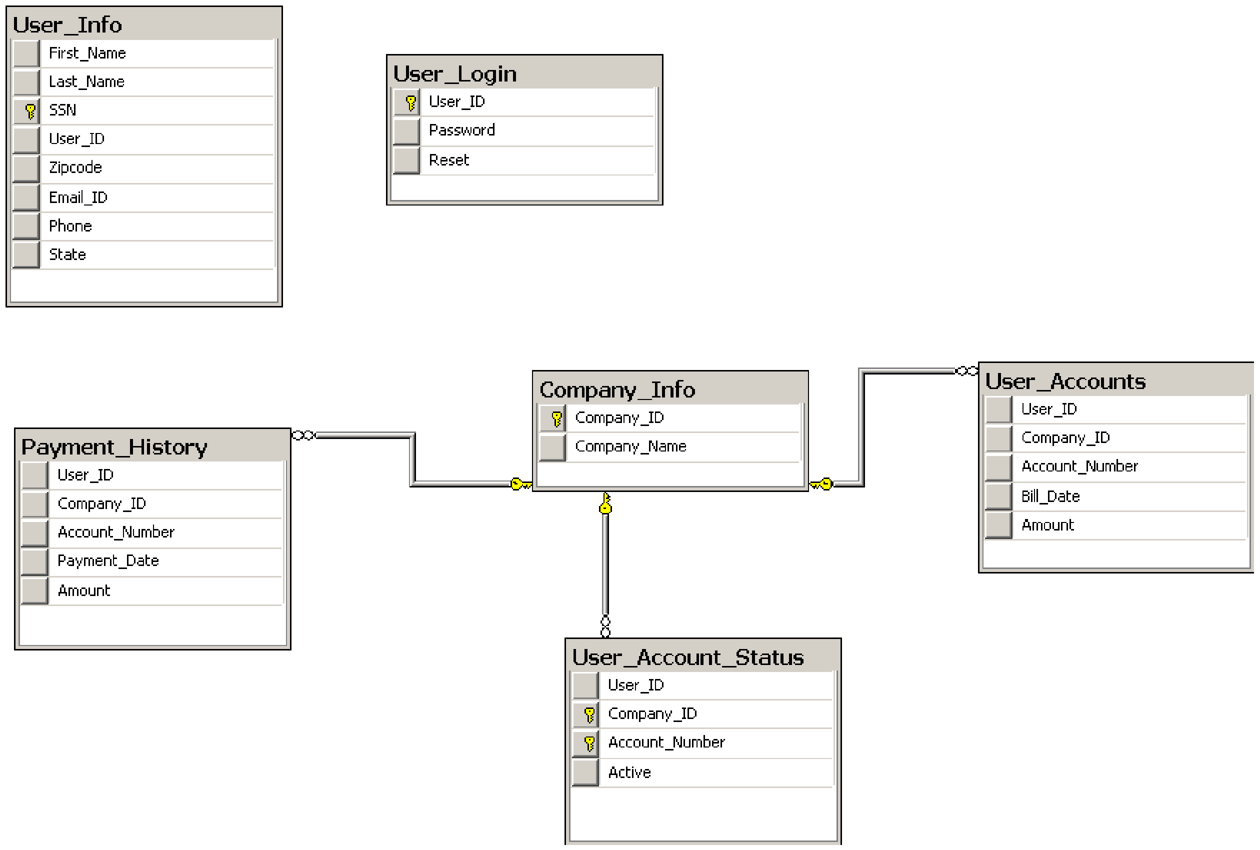
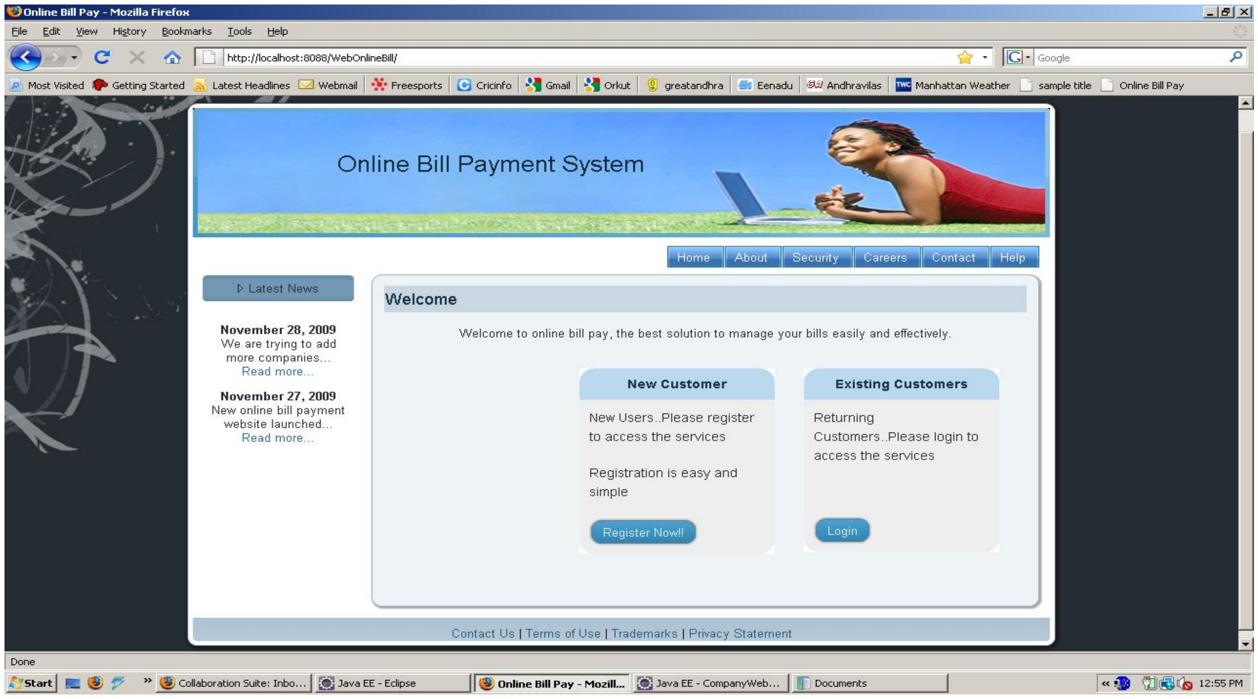


Figure 3-8 Database Diagram

- User_Info table contains all the information about the users.
- User_Login contains all the information about the user login details.
- Payment_History contains all the information about the payments of users.
- Company_Info contains the information about the company.
- User_Accounts contains all the information about the bills of the user.
- User_Account_Status contains the information the user accounts and whether the user is a verified user or not.

3.5 Screen Shots


Online Bill Pay - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8088/WebOnlineBill/
Online Bill Payment System
Home About Security Careers Contact Help
Latest News

November 28, 2009
We are trying to add more companies...
Read more...

November 27, 2009
New online bill payment website launched...
Read more...

Welcome

Welcome to online bill pay, the best solution to manage your bills easily and effectively.

New Customer

New Users..Please register to access the services

Registration is easy and simple

Register Now!!

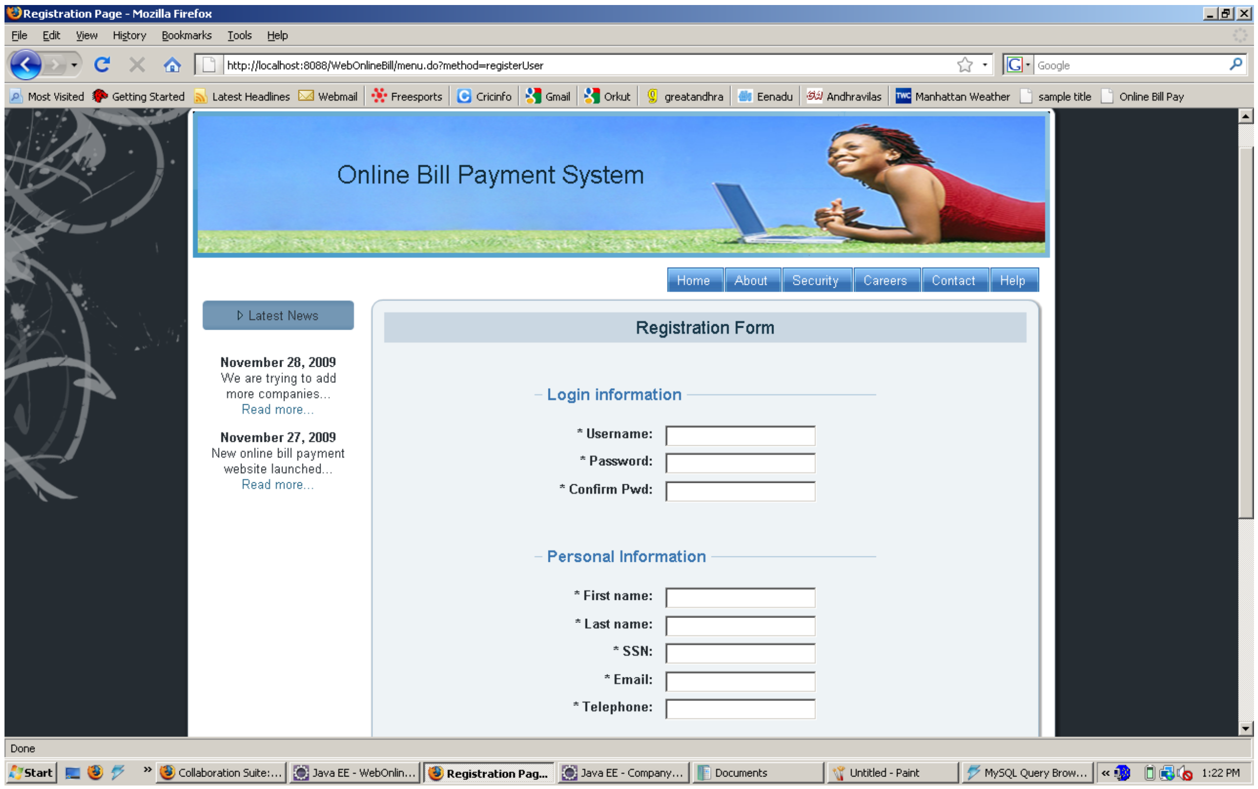
Existing Customers

Returning Customers..Please login to access the services

Login

Contact Us | Terms of Use | Trademarks | Privacy Statement

Figure 3-9 Home Page


Registration Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8088/WebOnlineBill/menu.do?method=registerUser
Online Bill Payment System
Home About Security Careers Contact Help
Latest News

November 28, 2009
We are trying to add more companies...
Read more...

November 27, 2009
New online bill payment website launched...
Read more...

Registration Form

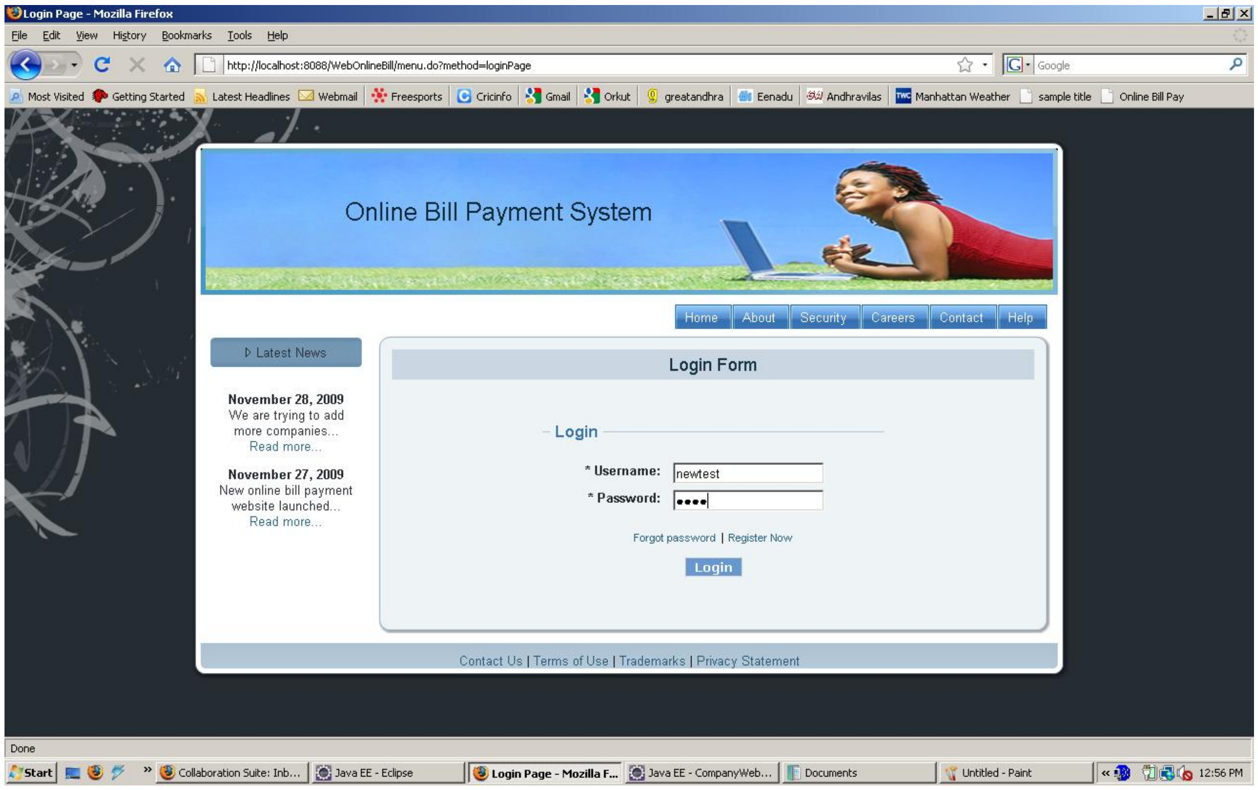
Login information

* Username:	
* Password:	
* Confirm Pwd:	

Personal Information

* First name:	
* Last name:	
* SSN:	
* Email:	
* Telephone:	

Figure 3-10 Registration Page


Login Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Online Bill Payment System
Home About Security Careers Contact Help
Latest News
Login Form
Login
Forgot password | Register Now
Login
Contact Us | Terms of Use | Trademarks | Privacy Statement
Done

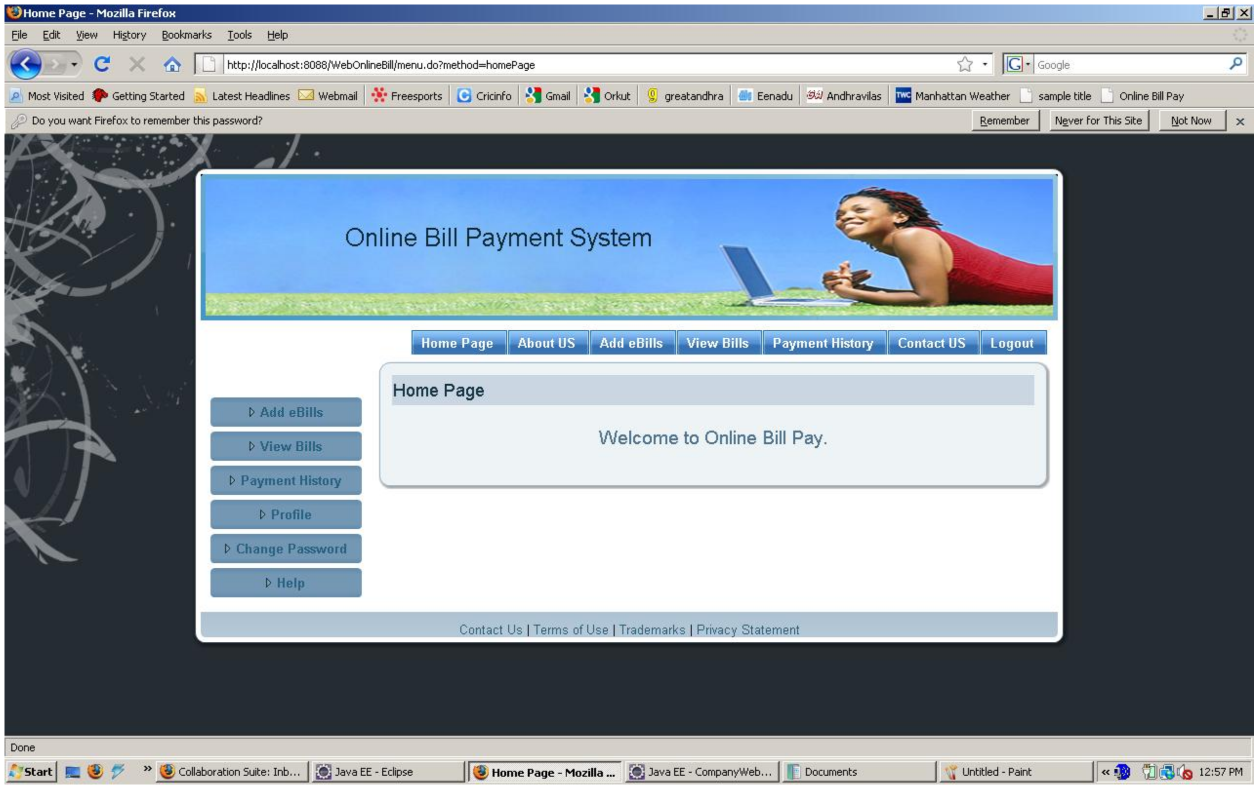
http://localhost:8088/WebOnlineBill/menu.do?method=loginPage

November 28, 2009
We are trying to add more companies...
Read more...

November 27, 2009
New online bill payment website launched...
Read more...

* Username: newtest
* Password: ••••

Figure 3-11 Login Page

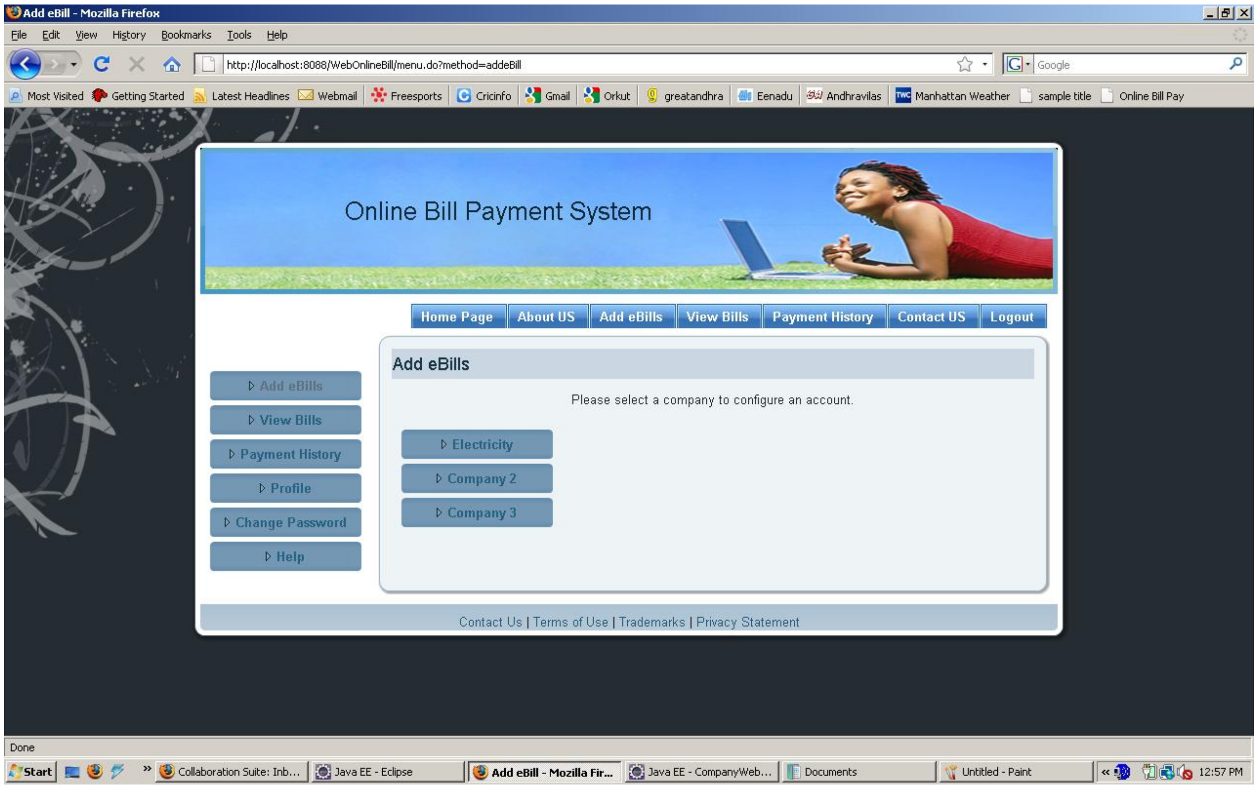

Home Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Remember Never for This Site Not Now
Online Bill Payment System
Home Page About US Add eBills View Bills Payment History Contact US Logout
Add eBills
View Bills
Payment History
Profile
Change Password
Help
Home Page
Contact Us | Terms of Use | Trademarks | Privacy Statement
Done

http://localhost:8088/WebOnlineBill/menu.do?method=homePage

Do you want Firefox to remember this password?

Welcome to Online Bill Pay.

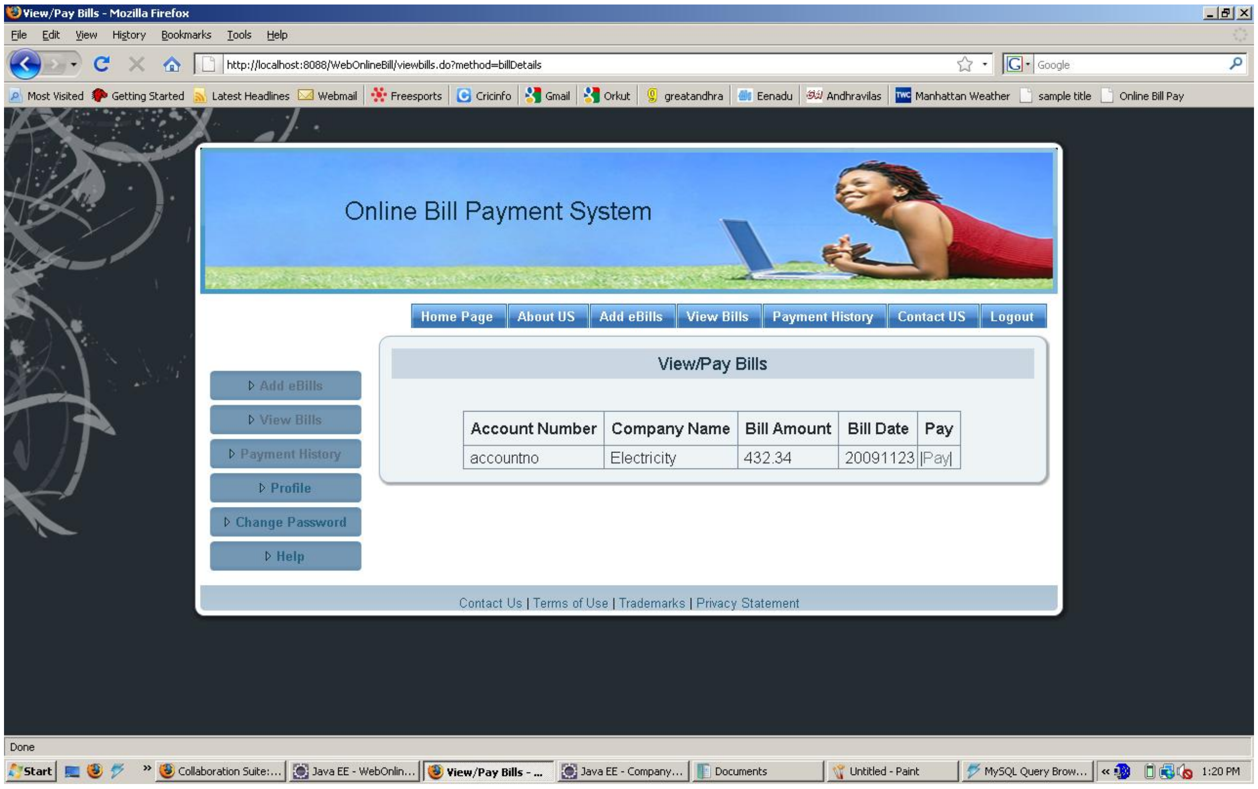
Figure 3-12 Home page after logging in


Online Bill Payment System
Home Page | About US | Add eBills | View Bills | Payment History | Contact US | Logout
Add eBills
View Bills
Payment History
Profile
Change Password
Help
Electricity
Company 2
Company 3
Contact Us | Terms of Use | Trademarks | Privacy Statement

Add eBills

Please select a company to configure an account.

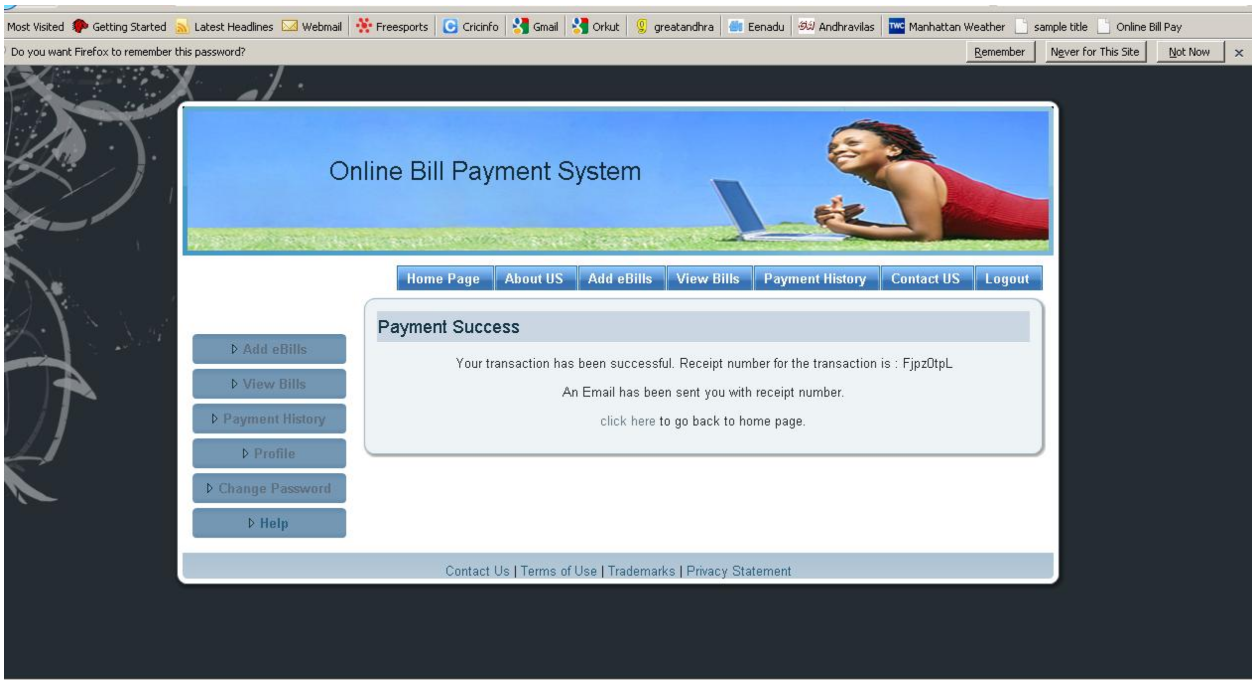
Figure 3-13 Company Selection Page


Online Bill Payment System
Home Page | About US | Add eBills | View Bills | Payment History | Contact US | Logout
Add eBills
View Bills
Payment History
Profile
Change Password
Help
Contact Us | Terms of Use | Trademarks | Privacy Statement

View/Pay Bills

Account Number	Company Name	Bill Amount	Bill Date	Pay
accountno	Electricity	432.34	20091123	Pay

Figure 3-14 View Bills Page



Payment Success

Your transaction has been successful. Receipt number for the transaction is : FjpzOtpL

An Email has been sent you with receipt number.

click here to go back to home page.

Figure 3-15 Payment Success Page

CHAPTER 4 - Testing

4.1 Unit Testing

Unit Testing is a method where each individual part of the software like a class, a method or a module is tested with various possible inputs to ensure the correctness of the software. Unit testing is very important because it gives the developers an opportunity to verify the functionality of all the individual functions before integrating them.

I have used JUnit framework to perform unit testing. JUnit is a unit testing framework for the Java Programming language. Test cases need to be created in order to test the application. To create a test case in JUnit the testing class must extend the TestCase class. JUnit is integrated in eclipse. All the test cases can be included in a test suite so that they can be executed at a time. I have tested all the possible methods using JUnit. For the rest of the methods and functionality, I have performed manual testing. Following are the methods which were tested using JUnit framework.

All the methods are called from the action class using the corresponding delegate class. So I wrote a test for each delegate class to observe the outcome.

i. AddeBillsDelegateTest: AddeBillsDelegate class has a method to verify whether a customer has an account configured with some company. I used three test cases for the method, one with valid input and two with invalid inputs.

ii. HistoryDelegateTest: HistoryDelegate class has a method to get the payment history details of the user. I used two test cases one with a valid input and another with an invalid input.

iii. LoginDelegateTest: LoginDelegate class has methods to validate a user, update password and reset password. I used three test cases for each of the methods, one with valid input and two with invalid inputs.

iv. PaymentDelegateTest: PaymentDelegate class has method that allows the customer to pay the current bill. This method calls the web service corresponding to the company associated with the bill. I used three test cases for the method one with valid input and two with invalid inputs like a negative bill amount and an invalid user.

v. RegisterUserDelegateTest: RegisterUserDelegate class has methods to register a user, update the user profile and get the user details. I used three test cases for all the methods, one with valid input and two with invalid inputs.

vi. ViewBillsDelegateTest: ViewBillsDelegate has a method to get the current bills of the user. I have used the same inputs that were used in the HistoryDelegateTest.

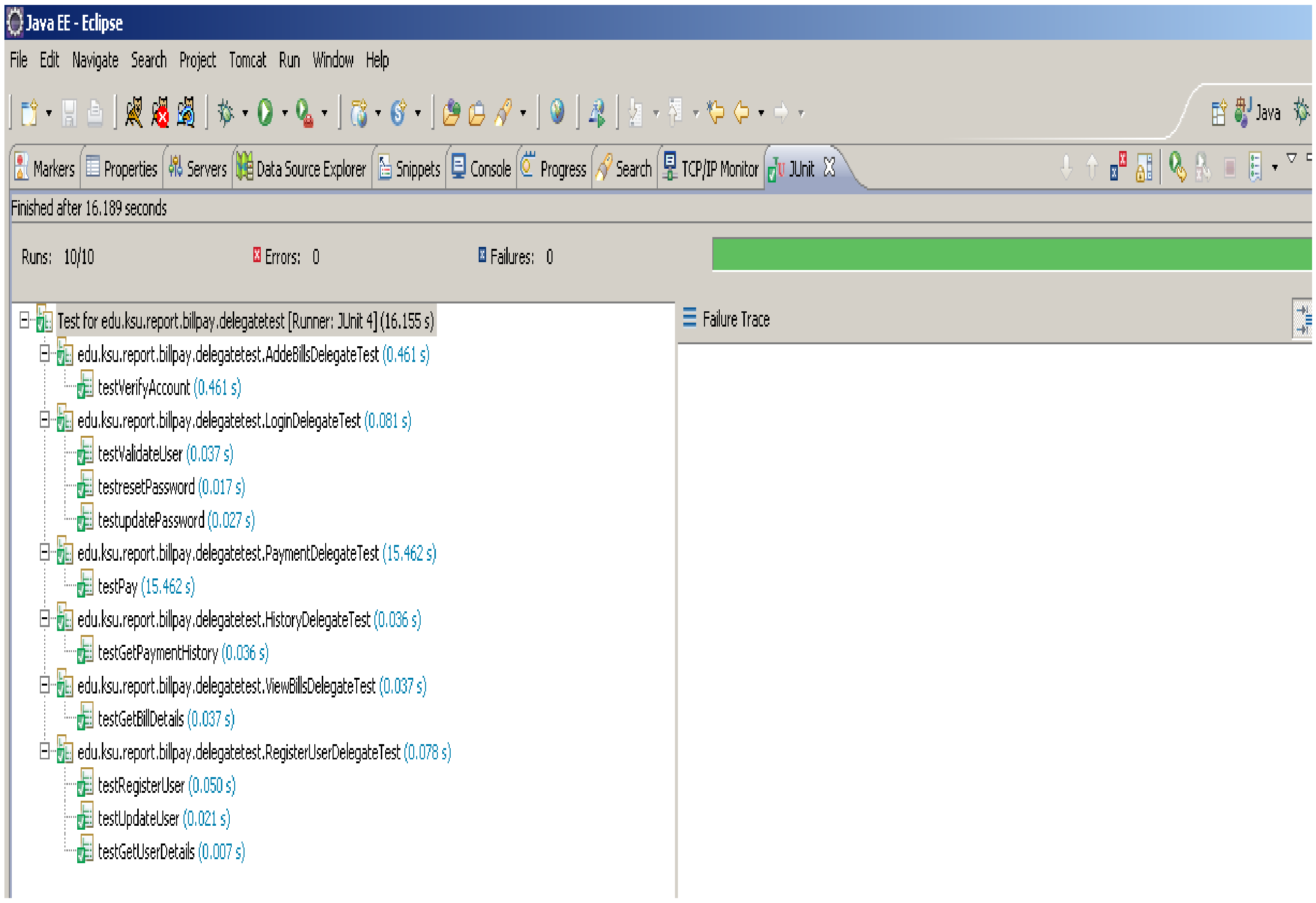
Apart from these methods there were some things that needed some manual testing. The passwords are encrypted using one way hash algorithm and then stored in the database. After a registration task or after updating/reset the password, I manually checked the database to confirm that the passwords in the database are in an encrypted form. Also a mail will be sent to the user in the following scenarios:

i. When a user registers in the website
ii. When a user resets the password
iii. When a user configures an account with a company
iv. When a new bill is generated
v. When a payment is made

In all the above cases I made sure that the email is sent to the users at the address associated with their account.

Also I have manually verified all the action forwards in the Struts action classes. I made sure that the action forwards are behaving in the expected manner. Along with the action forwards I made sure that the tiles corresponding to the action forward are being called.

I also made sure that the data communication between web services is always in an encrypted form.


Java EE - Eclipse
File Edit Navigate Search Project Tomcat Run Window Help
Java
Markers Properties Servers Data Source Explorer Snippets Console Progress Search TCP/IP Monitor JUnit
Failure Trace

```
Finished after 16.189 seconds

Runs: 10/10          Errors: 0          Failures: 0

Test for edu.ksu.report.billpay.delegatetest [Runner: JUnit 4] (16.155 s)
  edu.ksu.report.billpay.delegatetest.AddeBillsDelegateTest (0.461 s)
    testVerifyAccount (0.461 s)
  edu.ksu.report.billpay.delegatetest.LoginDelegateTest (0.081 s)
    testValidateUser (0.037 s)
    testresetPassword (0.017 s)
    testupdatePassword (0.027 s)
  edu.ksu.report.billpay.delegatetest.PaymentDelegateTest (15.462 s)
    testPay (15.462 s)
  edu.ksu.report.billpay.delegatetest.HistoryDelegateTest (0.036 s)
    testGetPaymentHistory (0.036 s)
  edu.ksu.report.billpay.delegatetest.ViewBillsDelegateTest (0.037 s)
    testGetBillDetails (0.037 s)
  edu.ksu.report.billpay.delegatetest.RegisterUserDelegateTest (0.078 s)
    testRegisterUser (0.050 s)
    testUpdateUser (0.021 s)
    testGetUserDetails (0.007 s)
```

Figure 4-1 JUnit tests in eclipse

4.2 Performance Testing

Performance testing is defined as the technical investigation done to determine or validate the speed, scalability, and/or stability characteristics of the web application under test [18]. A scenario is simulated where in many clients access the web application at a time. The number of clients trying to access the web site can be increased gradually and the effects are observed. By performance testing we can find the maximum number of users that can access the web application at some point of time. Also the efficiency of the server can be found out. With the help of these observations the application can be further tuned to increase the performance.

I have used Apache JMeter to test the performance of the website.

4.2.1 System Configuration:

I have used a system with following configuration for testing the application

System Configuration	
Number of Processors	**1**
RAM	**1 GB**
Processor Type	**586**
Processor Speed	**2 GHZ**
OS Name	**Windows Vista SP1**

Table 4-1 System Configuration

4.2.2 Test Plan

I have tested three pages in my application.

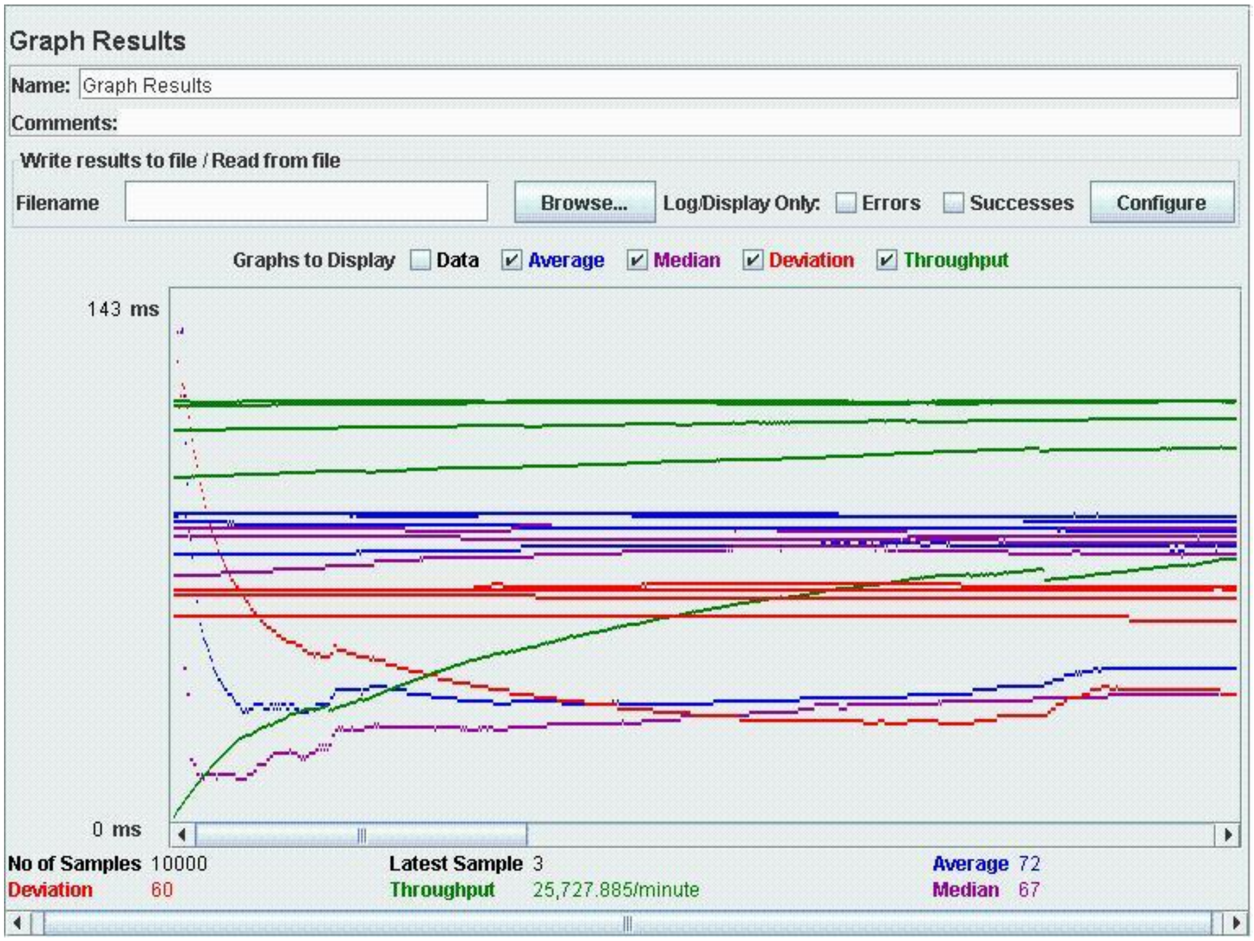
- The Home page using HTTP protocol
- Login page, by passing login credentials to the login page using HTTPS protocol
- Configuring an account with a company using HTTPS protocol, this involves communicating with the web services.

Each test is run using 50 users sending requests 200 times on a local machine. That was the maximum load that my system could handle. Whenever I try to increase the number of users or the loop count I am getting a "Java.lang.OutOfMemoryError-dumping heap space" exception. So I have limited the number of users to a maximum of 50 and the loop count to 200. When I tried to run the tests on a network using K-State wireless network, I have observed that the number of users that can access are 50 and the loop count is about 100.

4.2.3 Test Results & Evaluation

4.2.3.1 Home Page

The following are the test results:


Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: ☐ Errors ☐ Successes

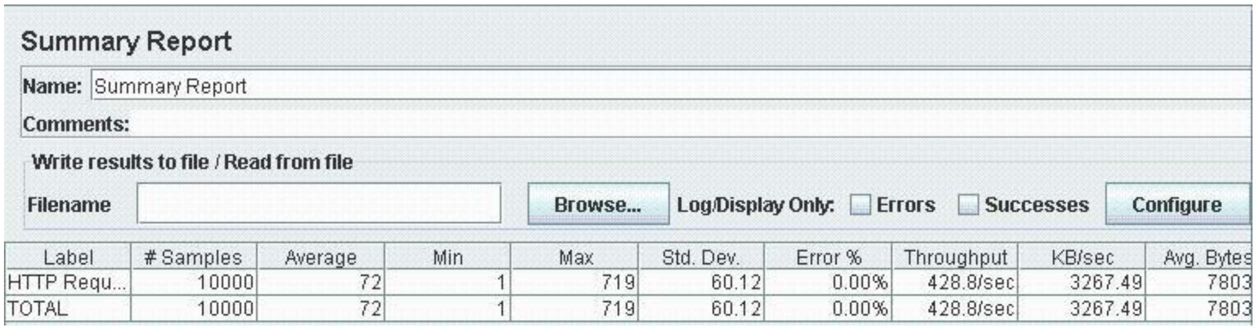
Graphs to Display ☐ Data ☑ Average ☑ Median ☑ Deviation ☑ Throughput

No of Samples 10000 Latest Sample 3 Average 72

Deviation 60 Throughput 25,727.885/minute Median 67

Figure 4-2 Summary graph for the home page

It can be observed that the throughput is very high and response time is very low. HTTP protocol is used while calling this web page. It doesn't involve any encryption. Also the page doesn't involve any communication with the database. So there is no additional overhead other than retrieving the page. The performance may decrease if the number of samples is increased but very often my system crashes by giving a Java.lang.OutOfMemoryError. The same results on a summary report can be observed below.


Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

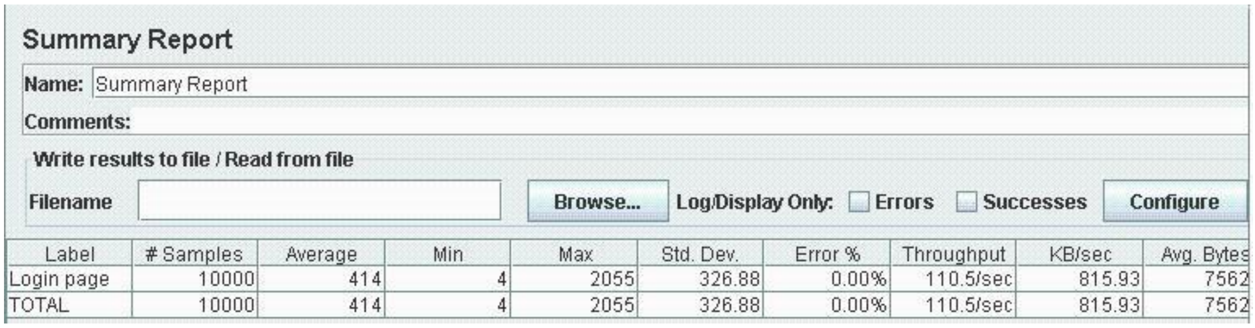
Filename | Browse... | Log/Display Only: Errors | Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Requ...	10000	72	1	719	60.12	0.00%	428.8/sec	3267.49	7803
TOTAL	10000	72	1	719	60.12	0.00%	428.8/sec	3267.49	7803

Table 4-2 Summary table for home page

4.2.3.2 Login Method

The following are the test results:


Summary Report

Name: Summary Report

Comments:

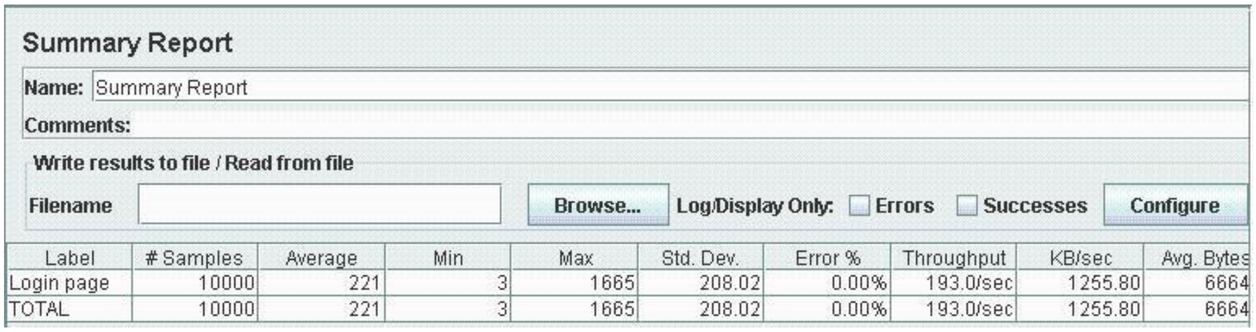
Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors | Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Login page	10000	414	4	2055	326.88	0.00%	110.5/sec	815.93	7562
TOTAL	10000	414	4	2055	326.88	0.00%	110.5/sec	815.93	7562

Table 4-3 Summary table for login page over HTTPS

Verifying the user details involves connecting to the database and also applying a one way hash algorithm to the password. This affects the response and throughput time. Since the data is sent using HTTPS protocol, the response time would be a bit high. This can be reflected in the below screenshots. The below screenshot shows the same test using HTTP protocol


Summary Report

Name: Summary Report

Comments:

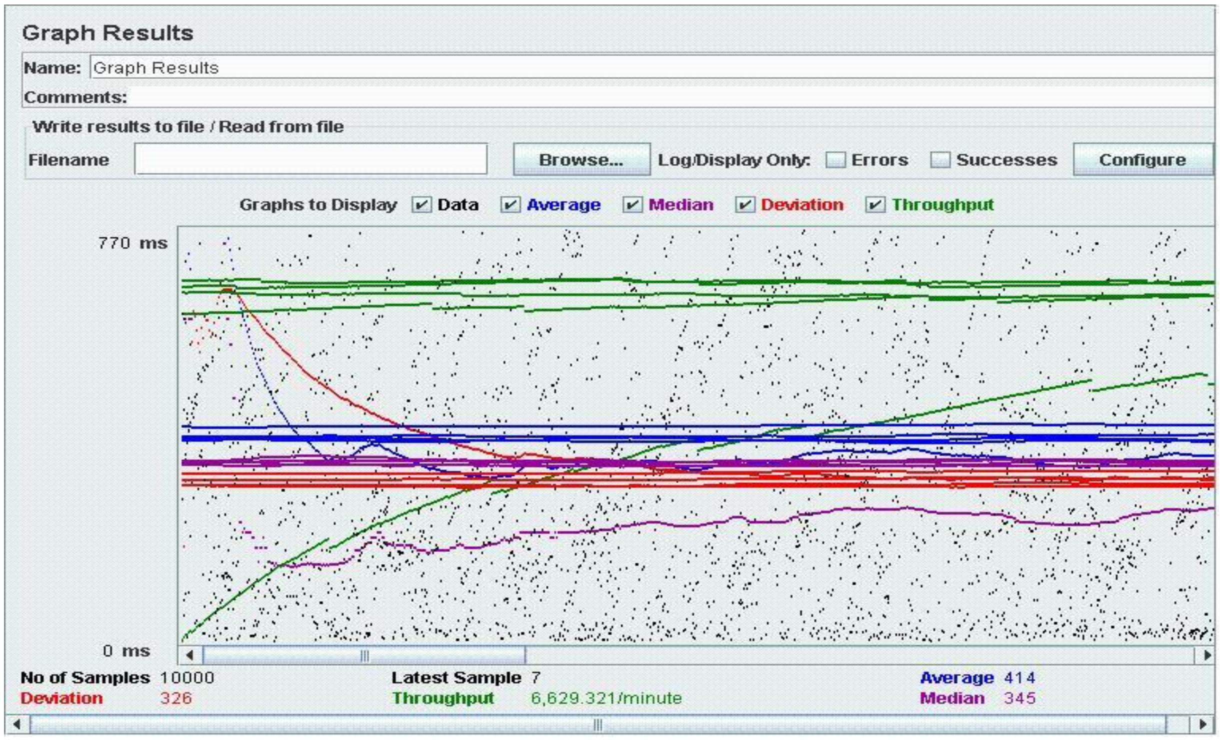
Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors | Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Login page	10000	221	3	1665	208.02	0.00%	193.0/sec	1255.80	6664
TOTAL	10000	221	3	1665	208.02	0.00%	193.0/sec	1255.80	6664

Table 4-4 Summary table for login page over HTTP

The same results can be observed in the following graphs:


Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

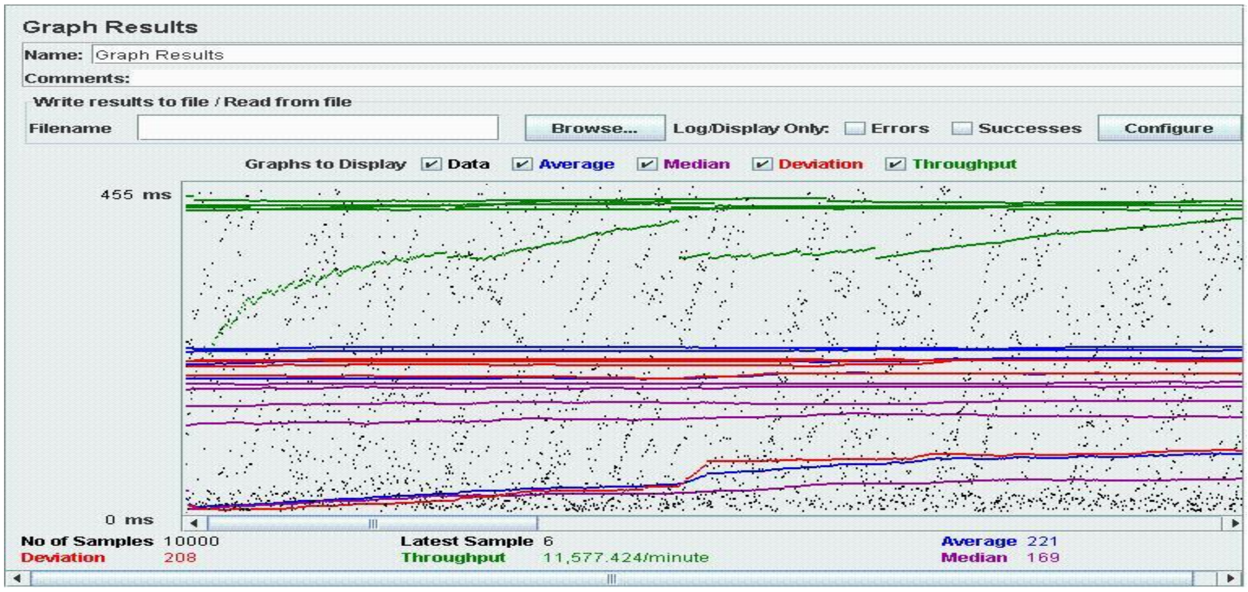
Filename

Log/Display Only: Errors Successes

Graphs to Display ✓ Data ✓ Average ✓ Median ✓ Deviation ✓ Throughput

No of Samples 10000 Latest Sample 7 Average 414
Deviation 326 Throughput 6,629.321/minute Median 345

Figure 4-3 Summary graph for login page over HTTPS


Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename

Log/Display Only: Errors Successes

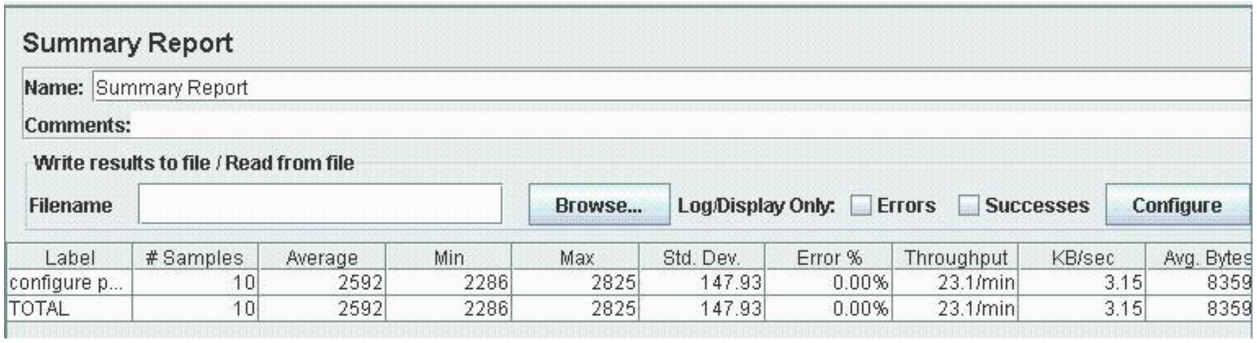
Graphs to Display ✓ Data ✓ Average ✓ Median ✓ Deviation ✓ Throughput

No of Samples 10000 Latest Sample 6 Average 221
Deviation 208 Throughput 11,577.424/minute Median 169

Figure 4-4 Summary graph for login page over HTTP

4.2.3.3 Configuring an account:

The test results are as follows



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

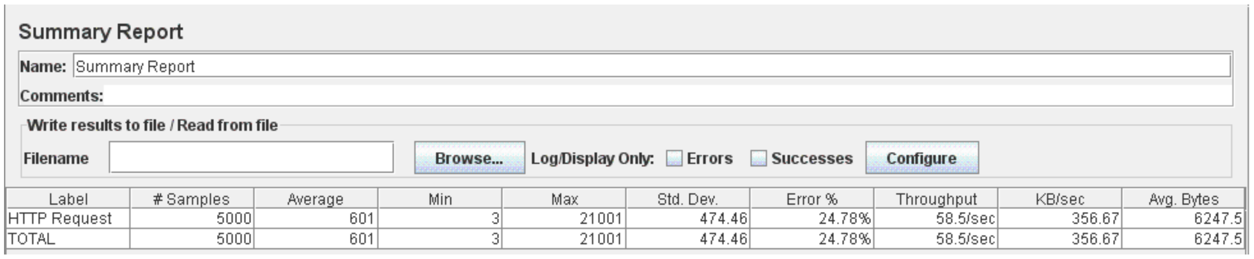
Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
configure p...	10	2592	2286	2825	147.93	0.00%	23.1/min	3.15	8359
TOTAL	10	2592	2286	2825	147.93	0.00%	23.1/min	3.15	8359

Table 4-5 Summary table for configuring an account page

The main reason for the low throughput and high response time for the above test case is that the xml data used to communicate has to be marshaled and un-marshaled before using. Also the data is being encrypted and decrypted. So along with the data transfer between two servers there is also an additional overhead associated with marshalling/un-marshalling and encrypting and decrypting the data. I have used HTTPS protocol which induces an additional over head along with the factors above.

The following are the test results when these tests are performed over a network.



Summary Report

Name: Summary Report

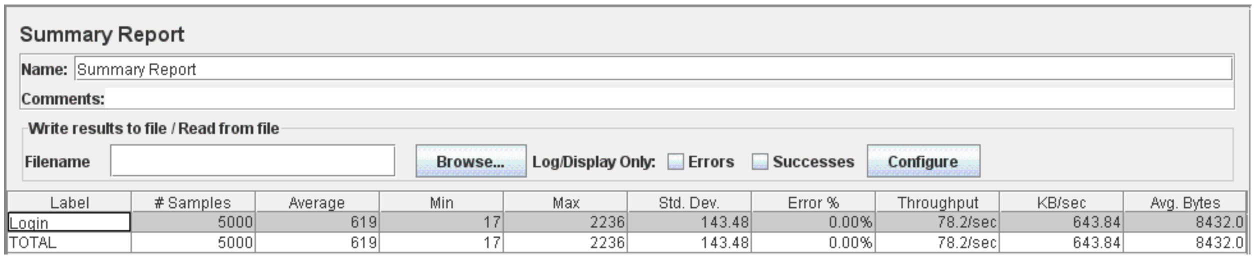
Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	5000	601	3	21001	474.46	24.78%	58.5/sec	356.67	6247.5
TOTAL	5000	601	3	21001	474.46	24.78%	58.5/sec	356.67	6247.5

Table 4-6 Summary table Homepage over wireless network



Summary Report

Name: Summary Report

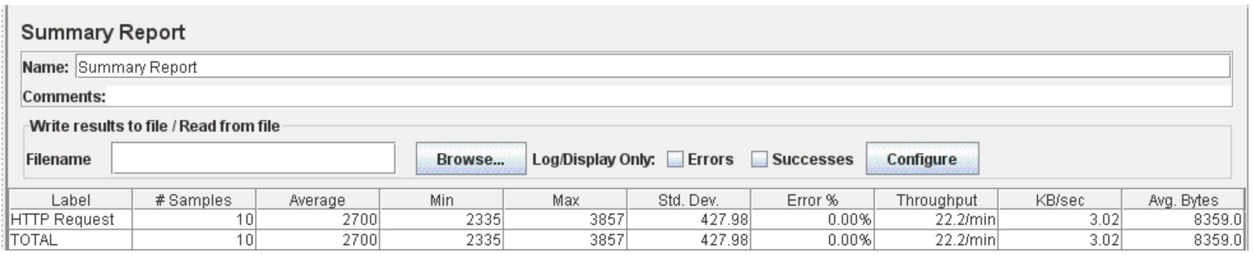
Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Login	5000	619	17	2236	143.48	0.00%	78.2/sec	643.84	8432.0
TOTAL	5000	619	17	2236	143.48	0.00%	78.2/sec	643.84	8432.0

Table 4-7 Summary table for Login page over wireless network



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	10	2700	2335	3857	427.98	0.00%	22.2/min	3.02	8359.0
TOTAL	10	2700	2335	3857	427.98	0.00%	22.2/min	3.02	8359.0

Table 4-8 Summary table for configuring an account over wireless network

Over a wireless network, the performance might have been decreased due to the factors like network usage at the testing time and also my system configuration. It can be observed that the login page has more throughput than the home page. Ideally the login page should take more time because it uses HTTPS protocol and involves hashing the password and communicating with the database but due to the factors like network usage at the testing time the home page has a less throughput. I think that the overall performance can be increased by improving the system configuration or using multiple servers instead of one.

CHAPTER 5 - Conclusions and Future Work

5.1 Conclusion

By implementing the Online Bill Payment website, I have gained a lot of experience on various technologies like Struts, Tiles, Servlets, Web services, Design Patterns, Java Mail API, XML, HTML, CSS, and Java Script. In addition I also gained experience on security concepts like one way hashing algorithms, SSL encryption, certificates, web services security.

5.2 Problems Faced

1. The main problem was learning the security concepts and learning how to implement them using java.
2. While implementing web services security using rampart module, I faced a lot of problems with some jar file dependencies.

5.3 Future Work

The present application is scalable and can be easily improved without much hassle. The use of struts and design patterns makes it easier to add more functionality in the future. The present application has all the basic features including security that a bill payment website needs. I think the following features can be added in the future:

1. Communication between the web services using HTTPS protocol instead of HTTP.
2. Add more companies to the bill payment website.
3. Implementing the payment methods. Giving an option to the user to select from various payment methods.
4. Provide auto payment options.
5. Use Maven to build the application.

References

[1] The Java EE 5 Tutorial,
http://java.sun.com/javaee/5/docs/tutorial/doc

[2] Apache Struts,
http://struts.apache.org/1.x/index.html

[3] Struts Tiles,
http://struts.apache.org/1.x/struts-tiles/index.html

[4] Struts Taglib,
http://struts.apache.org/1.x/struts-taglib/index.html

[5] Web Services Tutorial
http://www.w3schools.com/webservices/default.asp

[6] Core J2EE Patterns – Business Delegate
http://java.sun.com/blueprints/corej2eepatterns/Patterns/BusinessDelegate.html

[7] Core J2EE Patterns – Data Access Object
http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

[8] Core J2EE Patterns – Transfer Object
http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html

[9] Core J2EE Patterns: Best Practices and Design Strategies, Second Edition by Deepak Alur, John Crupi, Dan Malks

[10] Cryptographic hash function

http://en.wikipedia.org/wiki/Cryptographic_hash_function

[11] Hashing Concepts and the Java Programming Language

http://www.serve.net/buz/hash.adt/java.000.html

[12] HTTP Secure

http://en.wikipedia.org/wiki/HTTP_Secure

[13] Web Services Security

http://www.ibm.com/developerworks/library/specification/ws-secure/

[14] Tomcat and SSL

http://tomcat.apache.org/tomcat-3.3-doc/tomcat-ssl-howto.html

[15] Implementing Web Services Security with Apache Rampart

http://sacrosanctblood.blogspot.com/2007/03/step-by-step-tutorial-to-use-rampart.html

[16] Testing with JUnit

http://java.sun.com/developer/Books/javaprogramming/ant/ant_chap04.pdf

[17] JUnit Testing in Eclipse

http://www.tutorialized.com/tutorial/Junit-testing-in-Eclipse/10643

[18] Performance Testing Guidance for Web Applications

HTTP://msdn.microsoft.com/en-us/library/bb924375.aspx

[19] Performance Testing with JMeter

http://blogs.atlassian.com/developer/2008/10/performance_testing_with_jmete.html