



Jaypee University of Information Technology
Solan (H.P.)

LEARNING RESOURCE CENTER

Acc. Num. *SP07041* Call Num:

General Guidelines:

- ◆ Library books should be used with great care.
- ◆ Tearing, folding, cutting of library books or making any marks on them is not permitted and shall lead to disciplinary action.
- ◆ Any defect noticed at the time of borrowing books must be brought to the library staff immediately. Otherwise the borrower may be required to replace the book by a new copy.
- ◆ The loss of LRC book(s) must be immediately brought to the notice of the Librarian in writing.

Learning Resource Centre-JUIT



SP07041

DEVELOPMENT OF MOBILE APPLICATIONS USING WEB SERVICES (JSR172)

Pooja Singh 071263

Ketan Gulati 071275

Aditya Chandel 071294

Under the Supervision of

Mr. Pradeep Kumar Gupta

Senior Lecturer, CSE& IT



May – 2011

Submitted in partial fulfillment of the degree of

BACHELOR OF TECHNOLOGY

**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND
INFORMATION TECHNOLOGY**

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT**

Table of Contents

<i>CERTIFICATE</i>	4
<i>ACKNOWLEDGEMENTS</i>	5
<i>SUMMARY</i>	6
<i>LIST OF FIGURES</i>	7
<i>LIST OF TABLES</i>	8
CHAPTER 1	
Introduction	9
1.1 Objective	9
1.2 Scope	9
CHAPTER 2	
Methodology & Summary	11
CHAPTER 3	
Resources & Limitations	12
3.1 Resources	12
3.1.1 Java 2 MicroEdition	12
3.1.2 J2ME Configurations	13
3.1.2.1 CLDC	13
3.1.2.2 CDC	15
3.1.3 J2ME Profiles	15
3.1.3.1 KJava	16
3.1.3.2 MIDP	16
3.2 Limitations	18
CHAPTER 4	
Process Description	19
4.1 Description	19
4.1.1 Functional Requirements	19
4.1.2 Non Functional Requirements	23
CHAPTER 5	
Module Structure & Explanation	24
5.1 Module Structure	24
5.2 Module Explanation	24
5.2.1 Access SHAR – EX Home Page	24
5.2.2 Admin/F_User/P_user Login & Home Page	25
5.2.3 Checking / Updating Information	26
5.2.4 Detailed Nonfunctional requirements	28

CHAPTER 6	
Sample Code	29
CHAPTER 7	
Testing Techniques	31
7.1 Software Testing	31
7.2 Hardware Testing	31
CHAPTER 8	
RESULTS & CONCLUSION	32
Appendices	
<i>Appendix A Project Contribution</i>	33
References	33
<i>Appendix B Additional System Information</i>	34
<i>Appendix C Sample Code</i>	38

Signature of Supervisor:

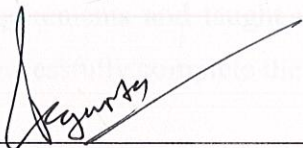
Name of Supervisor:

Praveen Kumar Gupta

CERTIFICATE

This is to certify that the work titled "*Development of Mobile Application using Web Services (JSR 172)*" submitted by "*Pooja Singh (071263), Ketan Gulati (071275) and Aditya Chandel (071294)*" in partial fulfillment for the award of degree of B.Tech in Computer Science and Engineering of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor:



Name of Supervisor:

PRADEEP KUMAR GUPTA

Designation:

Sr. Lecturer

Date:

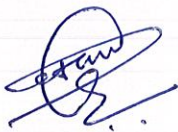
23/05/11

Acknowledgement

We would like to take this opportunity to express our sincere indebtedness and sense of gratitude to all those who have contributed greatly towards the successful partial completion of our project "DEVELOPMENT OF MOBILE APPLICATIONS USING WEB SERVICES".

It would not have been possible to see through the undertaken project without the guidance and constant support of our local guide Mr. Pradeep Kr. Gupta. For his coherent guidance we feel fortunate to be taught by him, who gave us his unwavering support. We owe our heartiest thanks to Brig. (Retd.) S.P.Ghrera(H.O.D.-CSE/IT Department) who've always inspired confidence in us to take initiative.

As a final note, we are grateful to CSE Department of Jaypee University of Information and Technology ,who inspired us to undertake difficult tasks by their strength of understanding our calibre and our requirements and taught us to work with patience and provided constant encouragement to successfully complete the project.



KETAN GULATI
071275



ADITYA CHANDEL
071294



POOJA SINGH
071263

Summary

While mentioning '**developing mobile applications using web services (JSR172)**', the first and foremost concept that evolved was the implementation of one such product which was previously undeveloped and could be of much use to general public. **SHAR – EX**, making use of various network technologies, databases, complex security measures, OLAP and online transactional abilities for banking enabled environment, has been built and coded on J2ME and allows the user to connect to the stock market and do '**every possible thing**' one can do in the real stock market.

We have developed this application which will run on mobile phones due to its hardware or processing constraints. It will take the Company's name as an input (taken on mobile, referred to as client) to laptop connected via a Bluetooth (referred to as a work station) for processing, will search the name of the company in the database present in the laptop and will finally give back the Company's price as output on the same mobile through which input was sent.

List of Figures

<u>Figures</u>	<u>Description</u>	<u>Page no.</u>
Fig.1	J2ME Architecture	14
Fig.2	Configuration Libraries	16
Fig.3	Mobile Information Device Architecture	17
Fig.4	Enterprise Diagram	19
Fig.5	Level 0 DFD	20
Fig.6	Level 1 DFD	20
Fig.7	Level 2 DFD	21
Fig.8	Use Case Diagram	22
Fig.9	Activity Diagram	24
Fig.10	Sample Run – application initialization	29
Fig.11	Sample Run – adding new entries	30
Fig.12	Sample Run – sorting entries	30
Fig.13	Entity Relationship Diagram	34
Fig.14	State Chart Diagram	34
Fig.15	Class Diagram	35
Fig.16	Deployment Diagram	35
Fig.17	Functional Dependency Diagram	36
Fig.18	Sequence Diagram	36

List of Tables

<u>Table No.</u>	<u>Description</u>	<u>Page No.</u>
Table 1.	Functional Req. – 1: Access SHAR – EX Home Page	25
Table 2.	Functional Req. – 2: Admin/F_user/P_user Login & HomePage	26
Table 3.	Functional Req. – 3: Checking/Updating Information	27
Table 4.	Detailed Non Functional Requirements	28

1.1 Objective

'Shar-ex' has been designed with a special purpose and goal kept in mind. The design has been made such that the very nature of the stock markets is easily adapted and presented at the root level of the system to every user in almost no time. The Current systems that are deployed on i-phones or on some websites present delayed information – but the nature of stock market doesn't give enough room for delayed information, as the indices are very sensitive and dynamic.

Hence, to make the information speedier and more readily available, **Shar-ex** provides for effective business with a high degree of mobility. Free users can also enquire and keep updated about the stock indices, whereas for transactions, a premium login has been designed.

'Shar-ex', in all therefore becomes a good remedy for the overall stock exchange trading and its entire spectrum of solutions offered. From enquiry to listing to portfolio or trading shares or company forecasting 'Shar-ex' is an ideal solution which can match any user's A to Z requirements. It is just not easy and speedy but even more safe and secured.

1.2 Scope

'Shar-ex' provides a varied number of solutions as so many options to the users of this system. A complete package in itself, the core services that it holds to provide include:

1. Login as a manager/admin

1.1 Admin panel

- 1.2 Database and query management
- 1.3 Updating and deletion of company records and profiling
- 1.4 Verification of all the transactions
- 2. *Login as a customer*
 - 2.1 Login details and hence classification of customer type
 - 2.2 Customer panel, as based on the customer type
- 3. *A dedicated database holding the customer details, based on the customer type.*
 - 3.1 Personal details of all the customers
 - 3.2 Details of only premium customers
 - 3.3 Personal portfolio
 - 3.4 Banking details
 - 3.5 Transaction Records
- 4. *A different database holding the transaction logs and details*
 - 4.1 Transaction ID
 - 4.2 Transaction History
 - 4.3 Amount Balance
- 5. *A database that holds all the companies details*
 - 5.1 Brief profiling and updates of the company
 - 5.2 Company's listing
 - 5.3 Brief history of company's indices and related forecasts
 - 5.4 Personal Portfolio
 - 5.5 Graphs
 - 5.6 Reports
 - 5.7 Alarm / Watch

The first phase of the project opens up with the coding of the application itself – something required at the most elementary level. The application coding for various modules pertaining to the different scopes and user segments is being done in J2ME and the entire coding is done on NetBeans 6.5. The interactive platform has enabled the automation of codes quite easy and also very much visible as the application runs. The second phase of the project will take into action the integration of all these modules designed previously. The third phase will then bring into picture the databases and their integration. It is evident that the individual databases will have been created in the first and second phases itself but integrating them and enabling them for OLAPs is again a task. This phase will also involve the white box testing of all the modules and integration testing of the entire application at the software end. The next phase will bring into picture the networking associated with it – the GPRS connectivity of the mobile phone with the central server. Before this stage, a local database will only work for all such database purposes. Once the networking has been resolved successfully, appropriate security protocols will be aligned to the application. The security in itself will be a separate and pre concluding phase followed by the concluding phases of alpha and beta testing.

3.1 Resources

Software Specification:

- ▶ *NetBeans 6.5*
- ▶ *mySQL*
- ▶ *J2ME and JSR172 specifications*

Hardware Specifications:

- ▶ *S60 Ed. 2 Sp. 3+ handset*
- ▶ *A GPRS enabled connection*

3.1.1 Java 2 Micro Edition

J2ME application allows you to perform various tasks on your mobile phones that were not earlier possible like cross functionality in mobile phones.

Definition: Sun Microsystems define J2ME as "a highly optimized Java run-time environment targeting a wide range of consumer products, including pagers, cellular phones, screen-phones, digital settop boxes and car navigation systems."

J2ME is a highly optimized Java runtime environment. J2ME is aimed at the consumer and embedded devices market. This includes devices such as cellular telephones, Personal Digital Assistants (PDAs) and other small devices.

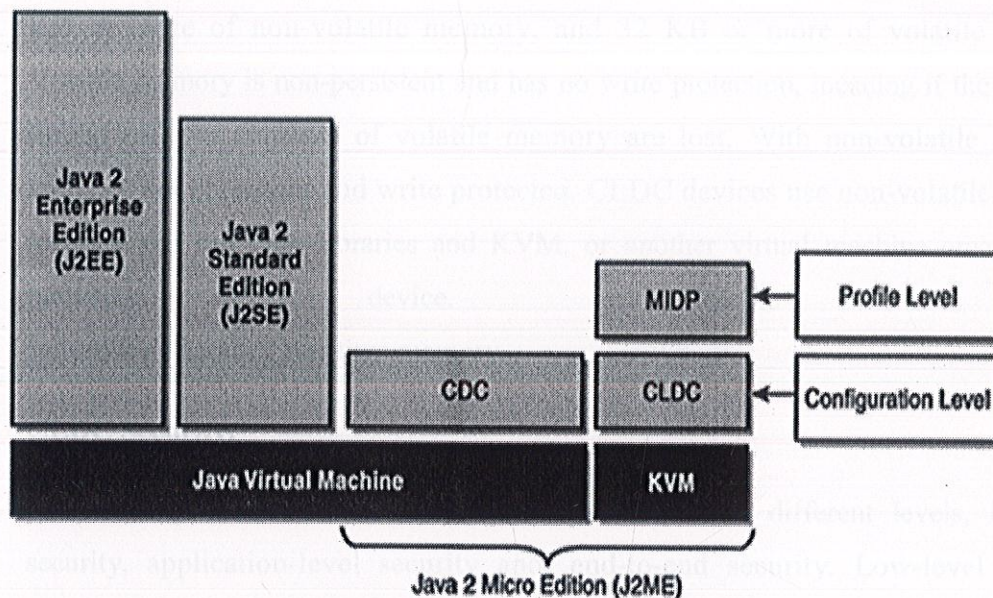


Figure 1 J2ME Architecture

3.1.2 J2ME Configurations

The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. There are two types of configurations for J2ME, CLDC and CDC - CLDC for small devices and CDC for larger devices. A J2ME environment can be configured dynamically to provide the environment needed to run an application, regardless of whether or not all Java technology-based libraries necessary to run the application are present on the device. The core platform receives both application code and libraries. Configuration is performed by server software running on the network.

3.1.2.1 CLDC

The J2ME CLDC configuration provides for a virtual machine and set of core libraries to be used within an industry-defined profile. The K virtual machine (KVM), CLDC's

reference implementation of a virtual machine, and its KJava profile run on top of CLDC. CLDC outlines the most basic set of libraries and Java virtual machine features required for each implementation of J2ME on highly constrained devices. CLDC targets devices with slow network connections, limited power (often battery operated), 128 KB or more of non-volatile memory, and 32 KB or more of volatile memory. Volatile memory is non-persistent and has no write protection, meaning if the device is turned off, the contents of volatile memory are lost. With non-volatile memory, contents are persistent and write protected. CLDC devices use non-volatile memory to store the run-time libraries and KVM, or another virtual machine created for a particular device. Volatile memory is used for allocating run-time memory.

CLDC Security:

The security model of the CLDC is defined at three different levels, low-level security, application-level security and, end-to-end security. Low-level security ensures that the application follows the semantics of the Java programming language. It also ensures that an ill-formed or maliciously encoded class file does not crash or in any other way harm the target device. In a standard Java virtual machine implementation this is guaranteed by a class file verifier.

Application-level security means that the application will run in the CLDC sandbox model. The application should only have access the resources and libraries permitted by the Java application environment. This means that the application programmer must not be able to modify or bypass the standard class loading mechanisms of the virtual machine. The CLDC sandbox model also requires that a closed, predefined set of Java APIs is available to the application programmer, defined by the CLDC, profiles (e.g. MIDP) and manufacturer-specific classes. The application programmer must not be able to override, modify, or add any classes to the protected java.*, javax.microedition.*, profile-specific or manufacturer-specific packages.

End-to-end security usually requires a number of advanced security solutions (e.g.

encryption and authentication). Therefore, all end-to-end security solutions are assumed to be implementation dependent and outside the scope of the CLDC specification.

3.1.2.2 CDC

Connected Device Configuration (CDC) has been defined as a stripped-down version of Java 2 Standard Edition (J2SE) with the CLDC classes added to it. Therefore, applications developed for CLDC devices also run on CDC devices. CDC, also developed by the Java Community Process, provides a standardized, portable, full-featured Java 2 virtual machine building block for consumer electronic and embedded devices, such as smart phones, two-way pagers, PDAs, home appliances, point-of-sale terminals, and car navigation systems. These devices run a 32-bit microprocessor and have more than 2 MB of memory, which is needed to store the C virtual machine and libraries. While the K virtual machine supports CLDC, the C virtual machine (CVM) supports CDC.

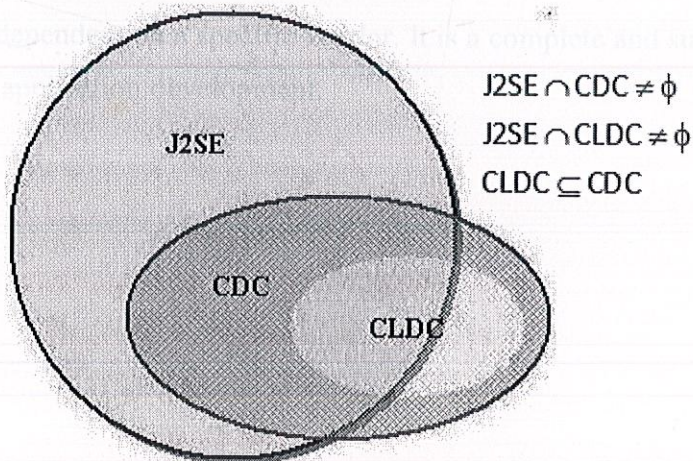


Figure 2 Configuration Libraries

3.1.3 J2ME Profiles

Two profiles have been defined for J2ME and are built upon CLDC: KJava and MIDP. Both KJava and MIDP are associated with CLDC and smaller devices. Profiles are built on

top of configurations. Because profiles are specific to the size of the device (amount of memory) on which an application runs, certain profiles are associated with certain configurations.

3.1.3.1 KJava

The KJava profile is built on top of the CLDC configuration. The KJava virtual machine, KVM, accepts the same byte codes and class file format as the classic J2SE virtual machine. KJava contains a Sun-specific API that runs on the Palm OS. However, because it is not a standard J2ME package, its main package is `com.sun.kjava`.

3.1.3.2 MIDP

MIDP is geared toward mobile devices such as cellular phones and pagers. The MIDP, like KJava, is built upon CLDC and provides a standard run-time environment that allows new applications and services to be deployed dynamically on end-user devices. MIDP is a common, industry-standard profile for mobile devices that is not dependent on a specific vendor. It is a complete and supported foundation for mobile application development.

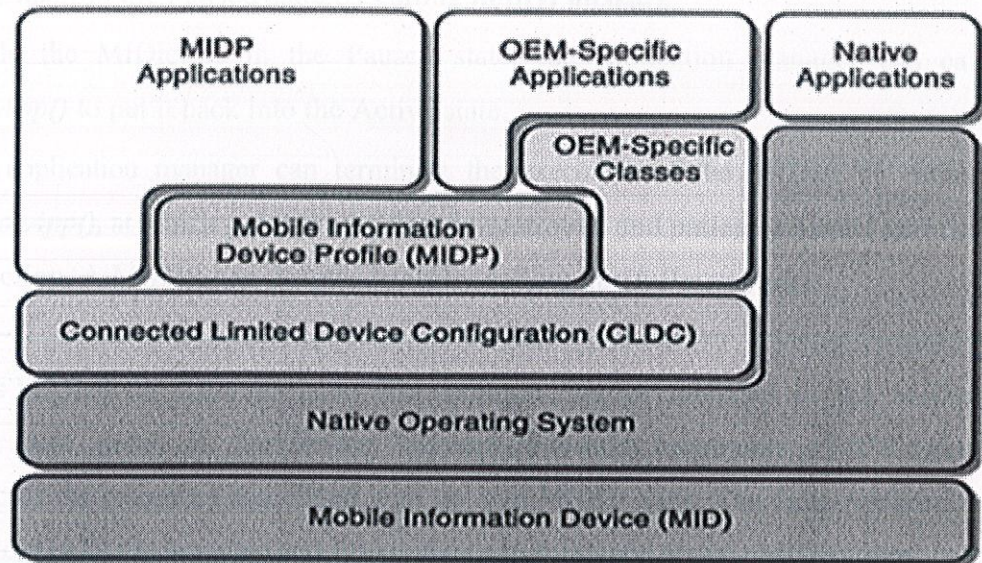


Figure 3 Mobile Information Device Architecture

MIDP applications are represented by instances of the `javax.microedition.midlet.MIDlet` class. A piece of device-specific software, the application manager, controls the installation, execution, and life cycle of MIDlets. MIDlets have no access to the application manager. A MIDlet is installed by moving its class files to a device. The class files will be packaged in a Java Archive (JAR), while an accompanying descriptor file (with a `.jad` extension) describes the contents of the JAR.

A MIDlet goes through the following states:

1. When the MIDlet is about to be run, an instance is created. The MIDlet's constructor is run, and the MIDlet is in the Paused state.
2. Next, the MIDlet enters the Active state after the application manager calls *startApp()*.
3. While the MIDlet is Active, the application manager can suspend its execution by calling *pauseApp()*. This puts the MIDlet back in the Paused state. A MIDlet can

place itself in the Paused state by calling *notifyPaused()*.

4. While the MIDlet is in the Paused state, the application manager can call *startApp()* to put it back into the Active state.
5. The application manager can terminate the execution of the MIDlet by calling *destroyApp()*, at which point the MIDlet is destroyed and patiently awaits garbage collection. A MIDlet can destroy itself by calling *notifyDestroyed()*.

Multiple MIDlets can share resources, like common libraries included in the MIDlet suite or data stored on the device. Because of security constraints, a MIDlet may only access the resources associated with its own MIDlet suite. The Java Application Descriptor (JAD) file is a plain text file containing information about a MIDlet suite.

3.2 Limitations

- The entire application is based upon the quality of internet connection being used. Hence, the dependency can result in lossy transactions, owing to problems with the network.
- Databases may turn oversized result in possible delays for producing required queries.
- Delay can occur also because of the sensitivity of the stock market.
- Dependent on the mobile handset also, and hence mobile phones with higher configurations are expected to perform better.

4.1 Description

4.1.1 Functional Requirements

The **SHAR-EX** encompasses numerous files and information from the Central Server, as well as files on the central database. This system will be completely web-based, linking to **SHAR-EX** and the remote web server from a standard mobile web browser. An Internet connection is necessary to access the system.

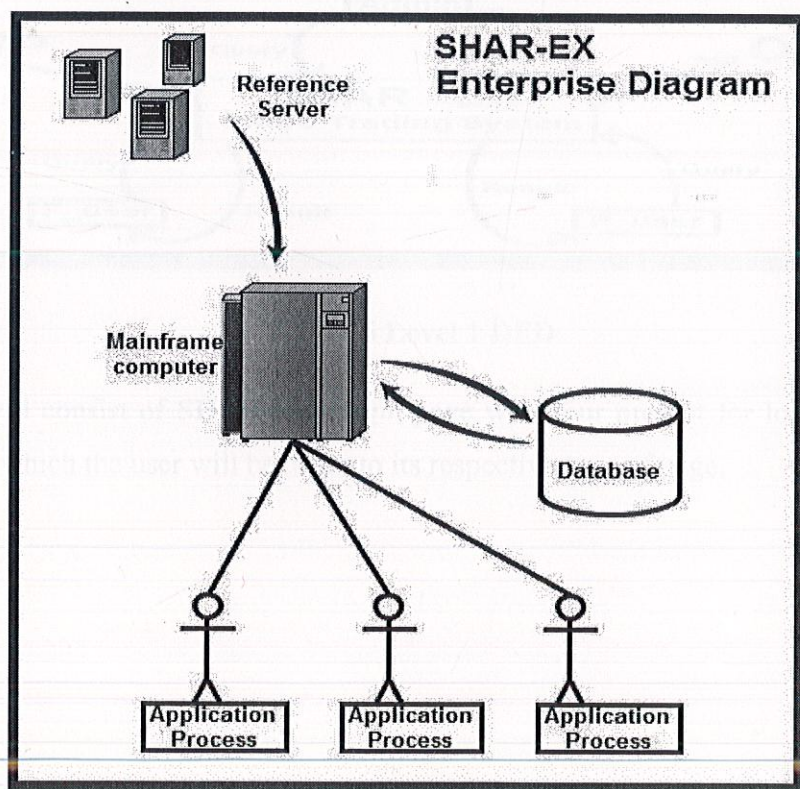


Figure 4 Enterprise Diagram

The **SHAR-EX** web site will be operated from the application web server. When a user connects to the Bank Web Server, the Web Server will pass the user to the application Server. The application Server will then interact with the Central Database through DBC, which allows the mentioned OS type program to transfer data to and from a database.

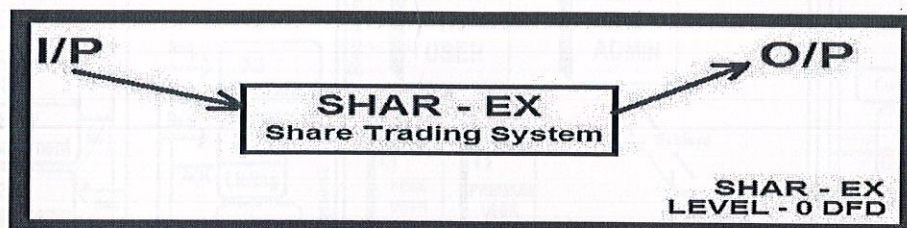


Figure 5 Level 0 DFD

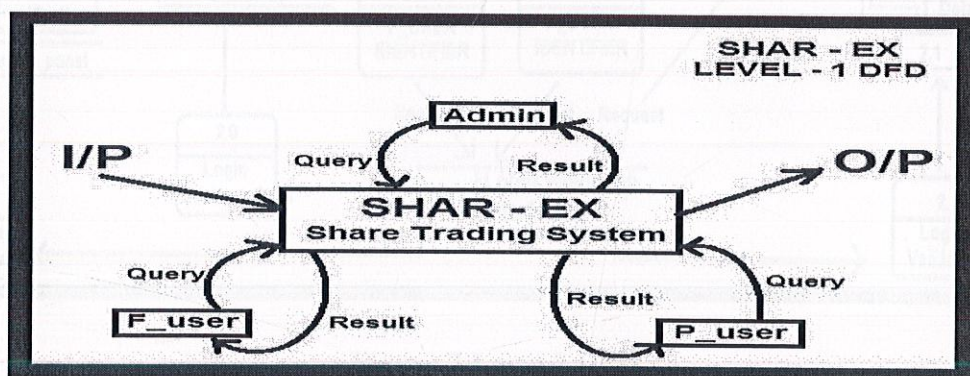


Figure 6 Level 1 DFD

The system will consist of **SHAR-EX** Login page with four prompt for login information, depending on which the user will be taken to its respective type of page.

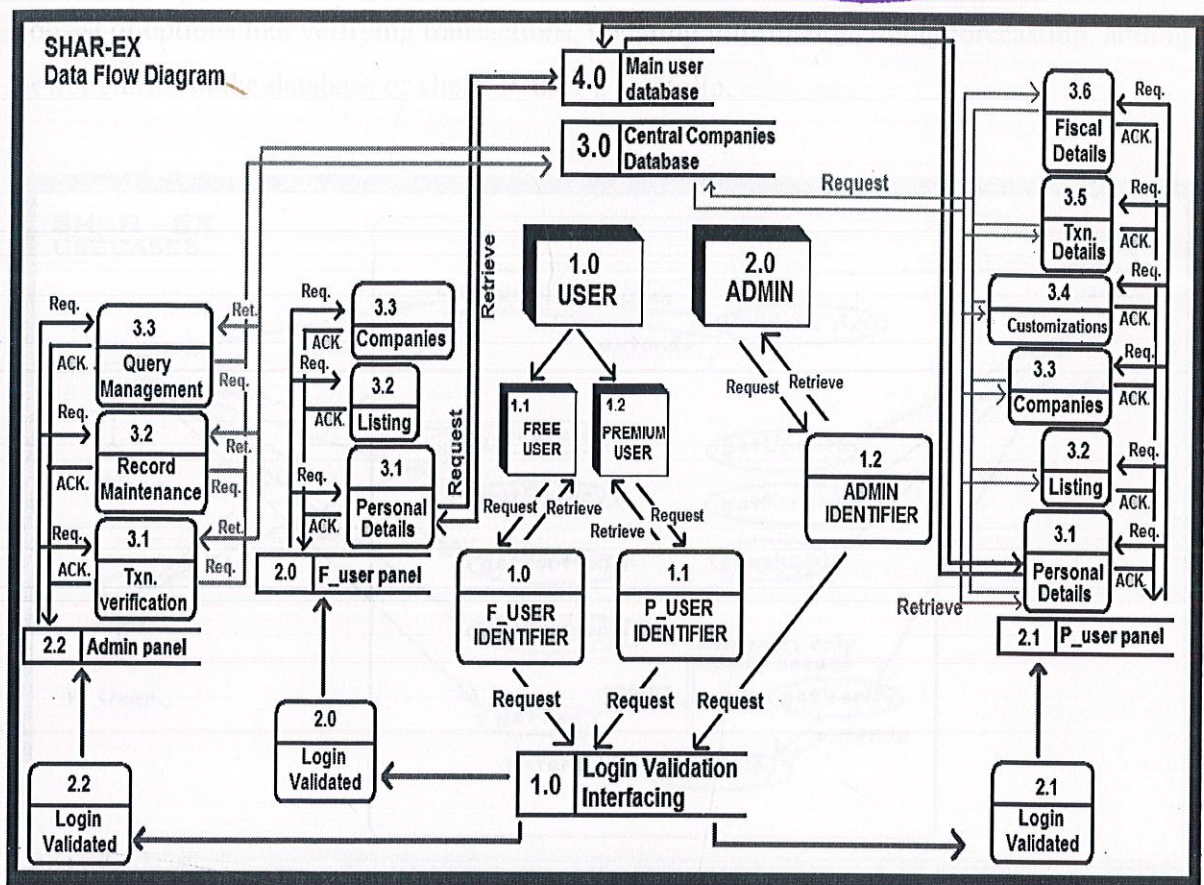


Figure 7 Level 2 DFD

The first selection is to login as an admin, a free user(used as **F_user**, from now on in this document) or a premium user(used as **P_user**, from now on in this document). Only after proper verification of the correct match of UserID and password, the user would be authenticated to enter the core system, that too with restricted rights and user options, as per the class of the user viz., admin, F_user or P_user. This information will be retained on the application server and an e-mail regarding the login time and session information will be sent to the designated network and server engineer.

The second selection will entirely depend upon the first selection.

If the first selection was an admin, the user will be prompted for additional Login securities for a secondary password or a pin. It will then take the admin to the admin panel which will

consist of options like verifying transactions, updating information, doing forecasting, adding newer entries in the database or check P_user's portfolio.

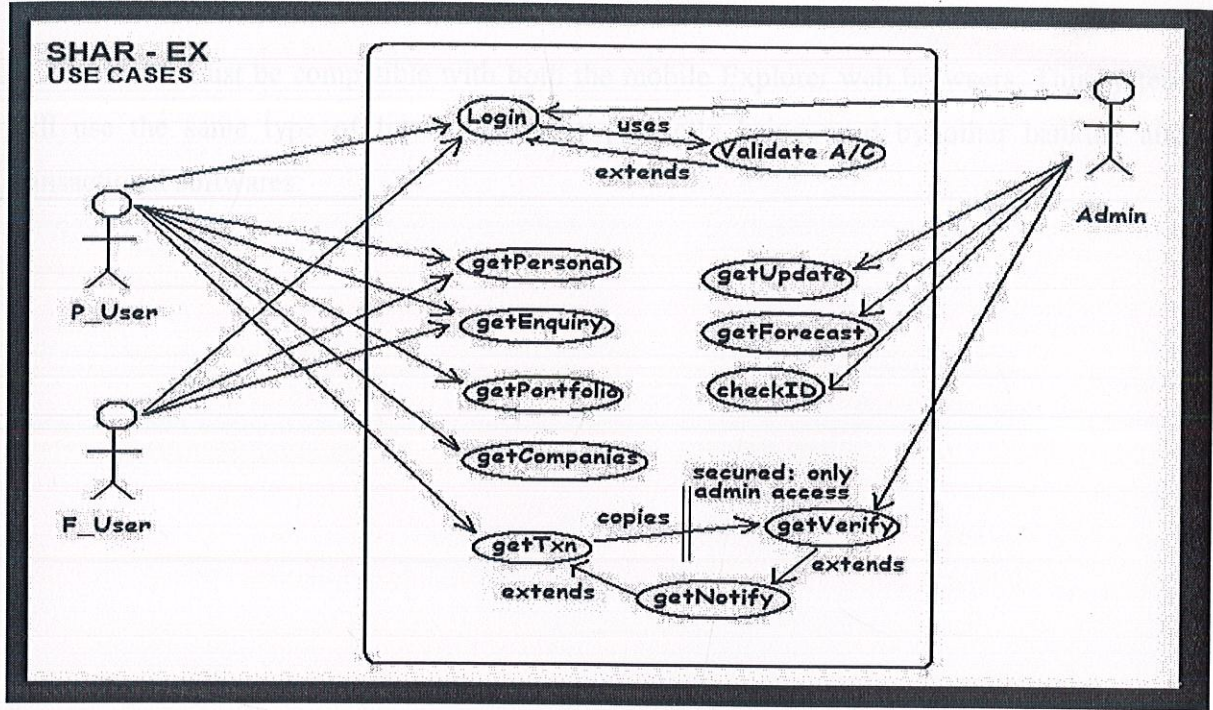


Figure 8 Use Case Diagram

The second possibility is that the first selection be an F_user. If it was a F_user, after proper login details validation, the user will be taken to F_user's home page where the available options would be checking and editing of personal details, going to the listing's page, where the user will be able to see the current index values and status of various listed companies.

The third selection on the first page was the P_user which meant for the premium users. This provides for a complex premium user panel that incorporates a number of options, such as personal profile, banking profile, companies listing, viewing and editing personal portfolios, producing various customer oriented services such as forecasting, reports and graphs.

All pages will return the user to the **SHAR-EX** Home Page.

4.1.2 Non Functional Requirements

There are requirements that are not functional in nature. Specifically, these are the constraints the system must work within.

The web site must be compatible with both the mobile Explorer web browsers. This system will use the same type of Internet security presently being used by other banking and transactional softwares.

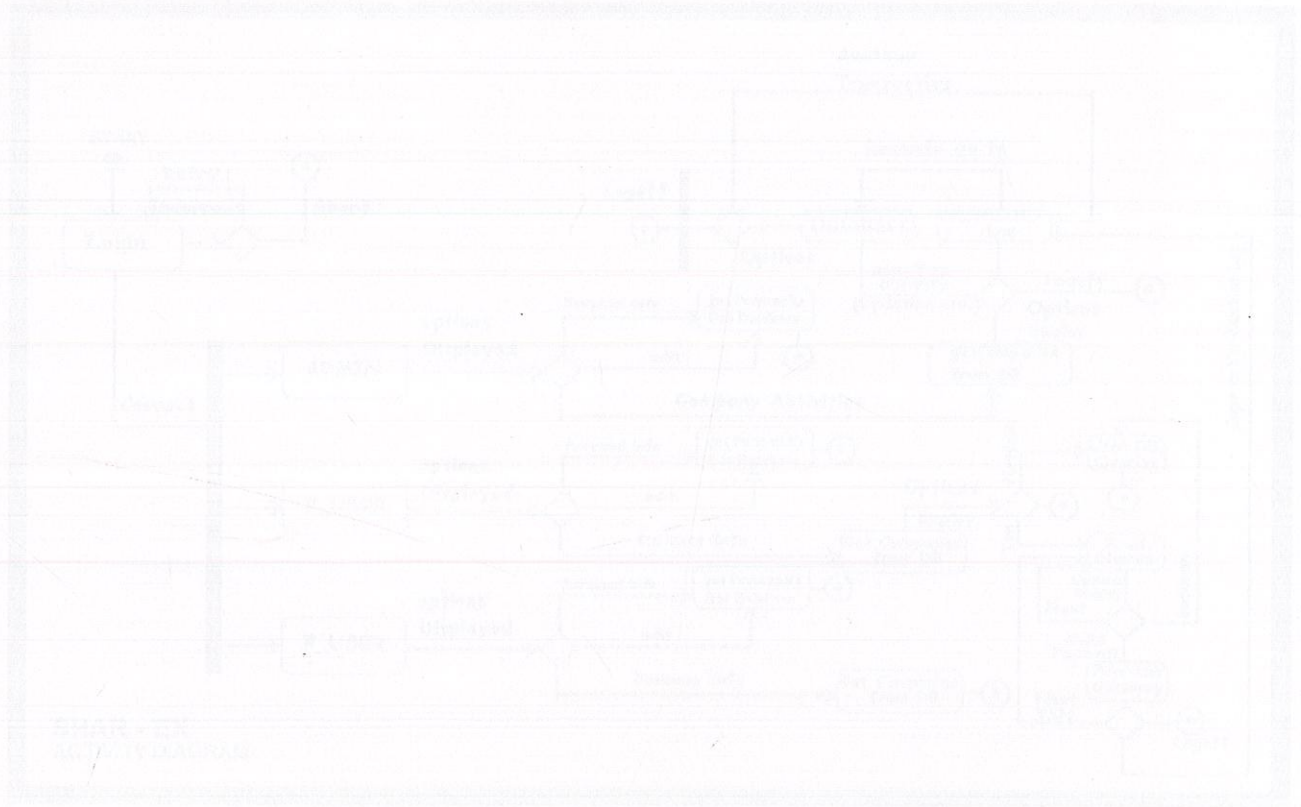


Figure 7 Activity Diagram

5.2 Module Explanation

5.2.1 Access Management Module

The first module out of the system will be the access management module, in the form of a login page.

It is given this page, because when different users have access to the system, they will be prompted for their respective

MODULE STRUCTURE & EXPLANATION

5

5.1 Module Structure

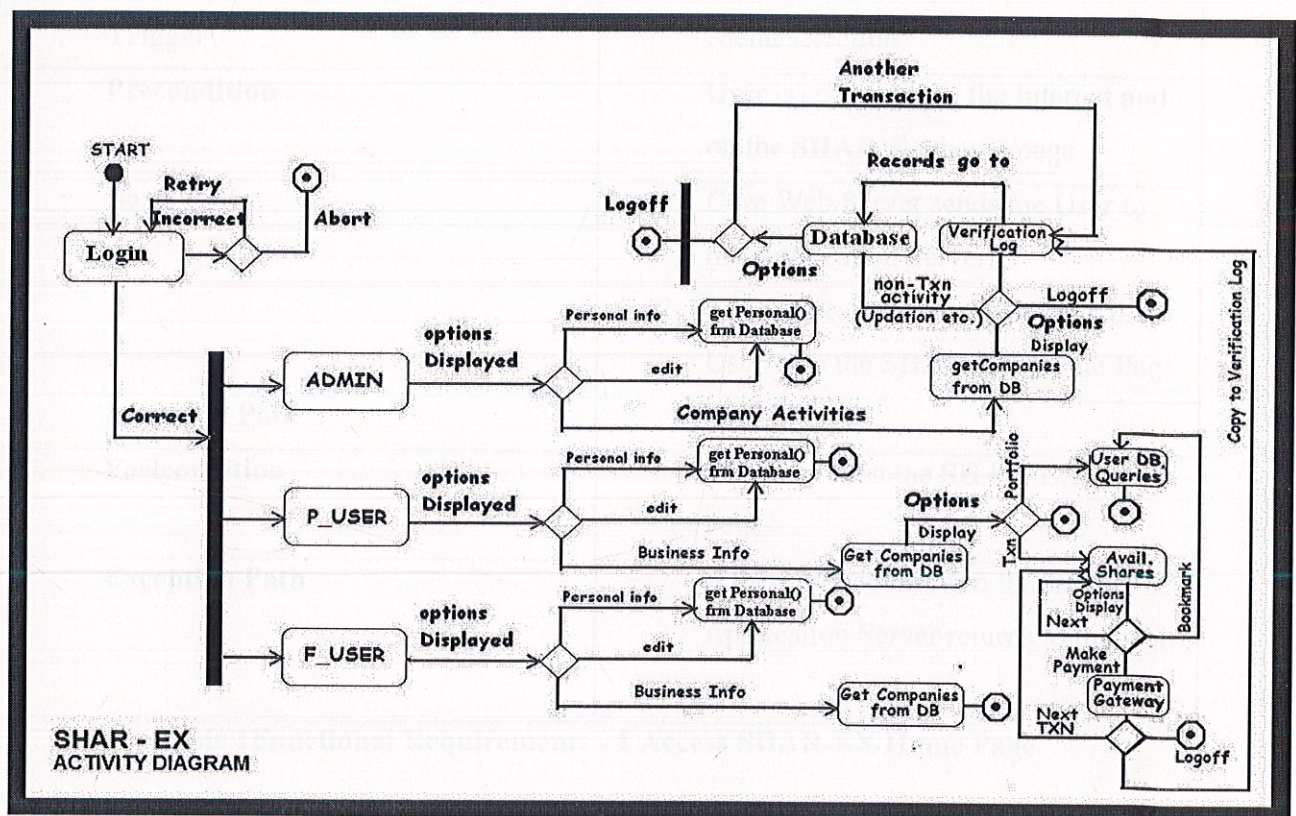


Figure 9 Activity Diagram

5.2 Module Explanation

5.2.1 Access SHAR-EX Home Page

The first module any of the three kind of users will get to see – in the form of a Home Page. It is from this page, based upon different users type after prompting for their respective

authentication, they would be provided with options they are allowed to do and various sections of the applications they are allowed to use.

Table 1 summarizes what all kind of details this particular state of the system may hold in the pre and post conditions of authentication.

Use Case Name:	Access Shar-ex Home Page
Priority	Essential
Trigger	Menu selection
Precondition	User is connected to the Internet and on the SHAR-EX home page
Basic Path	<ol style="list-style-type: none"> 1. Core Web Server sends the User to the Application Server. 2. The application Server presents the User with the SHAR-EX Home Page.
Alternate Path	N/A
Postcondition	The User is on the SHAR-EX Home Page
Exception Path	If there is a connection failure the Application Server returns to the wait state

Table 1 Functional Requirements - 1 Access SHAR-EX Home Page

5.2.2 Admin/F_user/P_user Login & HomePage

Use Case Name:	Admin/F_user/P_user Login & HomePage
Priority	Essential
Trigger	Selects
Precondition	The User is connected to the Internet

	and on the Shar-ex Home Page
Basic Path	<ol style="list-style-type: none"> 1. The Application Server presents the user with different options. 2. The User clicks the required option 3. The Application Server checks to see if the login details have been keyed in properly and authenticate or not. 4. If the verification done is valid, the Application Server creates a new log of the user in the login Database. 5. If the login is not authenticated, the user is given a prompt. 6. The Application Server returns the User to the SHAR-EX Home Page
Alternate Path	N/A
Postcondition	The login record is created in the Login Table of the Users' Database.
Exception Path	<ol style="list-style-type: none"> 1. If the connection is terminated before login verification, the fields are all cleared and the Application Server is returned to the wait state.

Table 2 Functional Requirements - 2 Admin/F_user/P_user Login & HomePage

5.2.3 Checking/Updating Information

Use Case Name:	Checking/Updating Personal Information
Priority	Essential
Trigger	Menu selection
Precondition	The User must be connected to the Internet and on the SHAR-EX Entries page.

Basic Path	<ol style="list-style-type: none"> 1. The User clicks on check/update personal information. 2/ The Application Server returns two switches – view info, edit info. 3. The User views info and returns OR edits info 4. The Application Server checks to see if any required field is empty. 5. If any required field is empty the Application Server will send a message and return the User to the new entry form page. 6. If no required field is empty the Application Server will create a new record in the corresponding (Employee/Customer/Manager) Table in the Bank Database, and return the User to the SHAR-EX Home Page. 7. The User may select Cancel. 8. If the User selects Cancel, the form is cleared and the User is returned to the SHAR-EX Home page.
Postcondition	A record is created in the corresponding Table of the Bank Database.
Exception Path	<ol style="list-style-type: none"> 1. If the connection is terminated before the update is clicked, the fields are cleared and the Application Server is returned to the wait state. 2. If the connection is terminated after the update is clicked, but before the User is returned to the SHAR-EX Home Page, the record is created in the corresponding Table of the Bank Database.

Table 3 Functional Requirement - 3 Checking/Updating Information

5.2.4 Detailed non-functional requirements

Attribute name	Attribute Type	Attribute Size
A_id	String	30
A_Name	String	30
A_Branch	String	50
U_id	String	30
Password	String	6 to 12
Tel#	Int	10
Name	String	30
C_id	String	20
C_name	String	50
C_Price	Int	5
C_Log	Memo	32000
C_Note	Memo	32000
T_id	String	10
S#	Int	10
Ac#	String	20
Bank	String	50
NEFT#	String	10

Table 4 Detailed non-functional requirements

The programming and its entire evaluation is being done in NetBeans 6.5. So far the company listing module has been coded, which is mentioned below in section 6.1

6.1 Sample Code for Company Listing

This code runs on the admin end to add new companies, delete previously listed companies and keep on updating the records of all the listed companies. For referring to the code, kindly see appendix at the end of this report. Following are the diagrams showing a few sample runs.

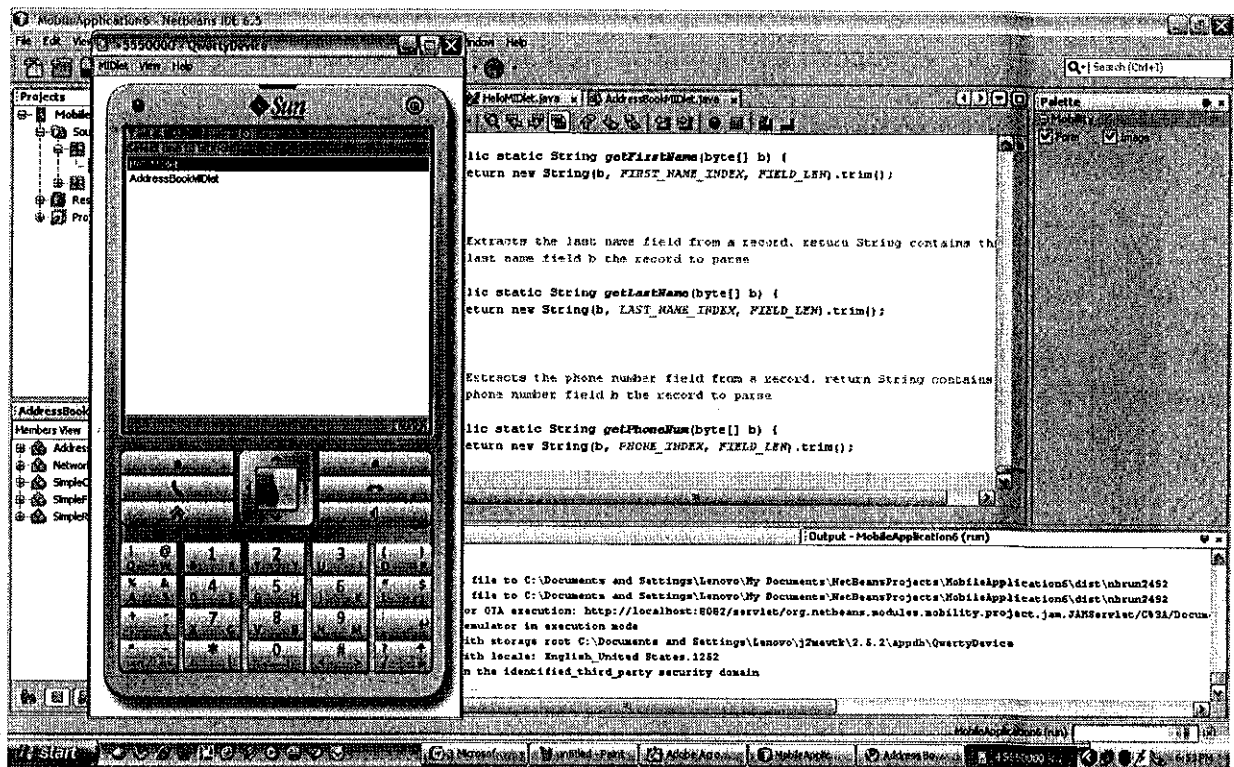


Figure 10 Sample Run – application initialization

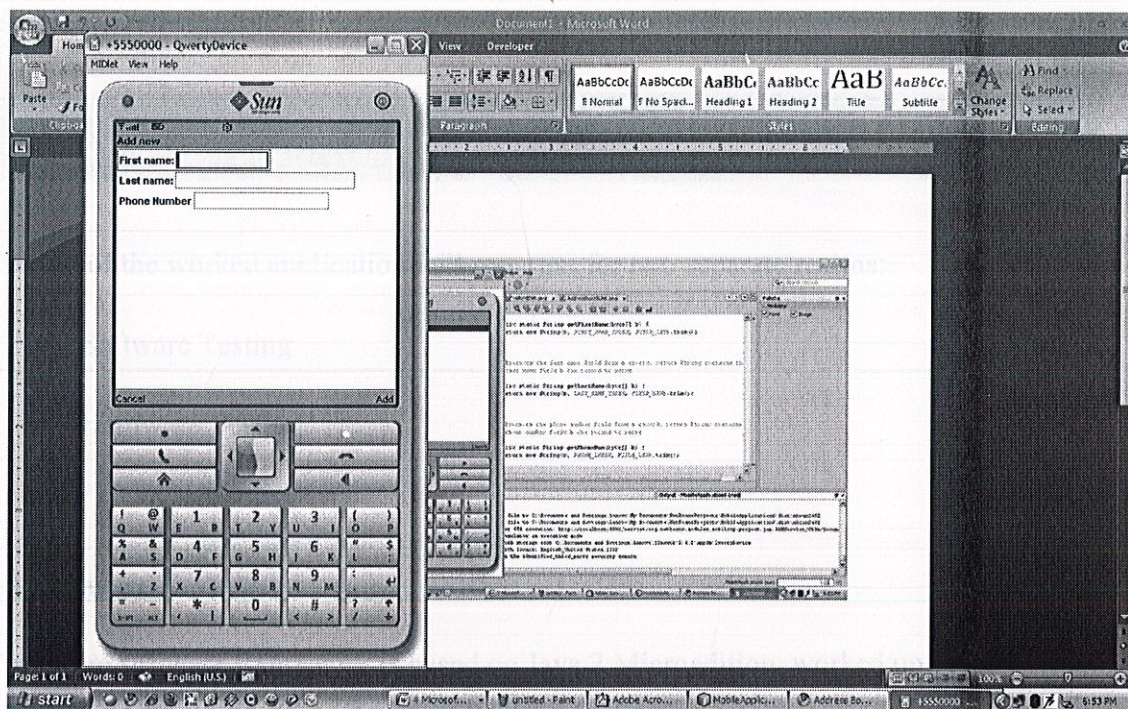


Figure 11 Sample Run – Adding new entries

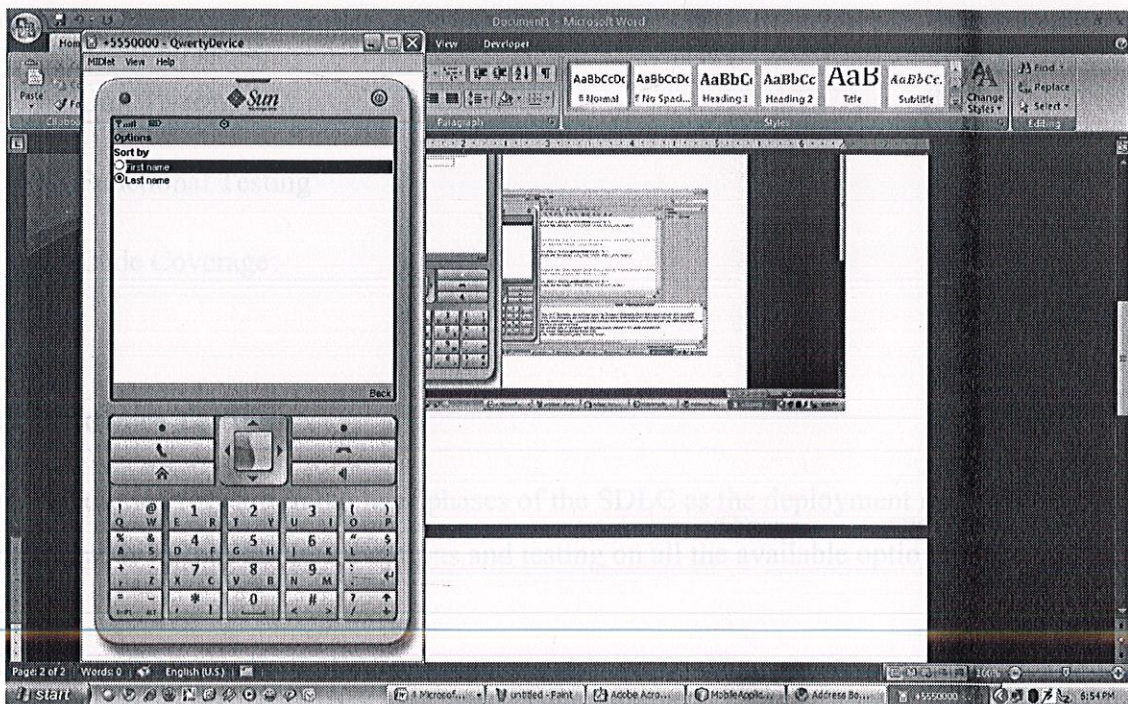


Figure 12 Sample Run – sorting entries

Testing of the worked application will progress for two separate realms:

- ▶ Software Testing
- ▶ Hardware Testing

7.1 Software Testing

The entire software framework is based on Java 2 Microedition, worked upon Netbeans, which in itself provide all testing solutions out of the box.

Using **NB JUnit**, which is a NetBeans Platform Extension to the JUnit Testing Framework.

Using this framework, complete care can be taken for:

- ▶ Unit Testing
- ▶ Functional Testing
- ▶ Code Coverage

7.2 Hardware Testing

Will be carried forward in the later phases of the SDLC as the deployment needs to be started, with various hardware options and testing on all the available options

RESULTS & CONCLUSIONS

8

A better platform focusing on reduction of delay and lossless transactions, providing the best of the results related to stock market and giving the term mobility its true meaning is what we tried to achieve. Also, as commercial application is involved, money and banking is involved we need good security solutions.

A few points we have been able to ponder upon so far, related to the issues mentioned above, that would be incorporated at various stages through the process of coding and application development can be summed upon as follows: -

1. Maximized customizations at the premium user login end.
2. Setting up a buffer to reduce lossy transactions by always ensuring either high grade network availability or none at all.
3. Ensuring maximum OLAP as mySQL doesn't create problems but hardware devices do, when the data size increases exponentially.
4. Ensuring security measures that are least prone to brute force attacks or other outside attacks over the secured payment gateways. SSL certificates, MD5 Hashing could be a few suggested solutions.

APPENDIX A

PROJECT CONTRIBUTION

- ▶ The final simulation developed can find its implementation in integrating the mobile industry to the stock exchange, opening a new gateway for those who deal in the same, at the comfort of their mobility.
- ▶ Based on its successful completion, it can contribute to the increased reliability of mobile applications for such commercial and investment purposes.

References

- [1] J2ME: Step by Step, developerWorks, ibm.com/developerWorks
- [2] Enterprise J2ME: Developing Mobile Java Applications by Michael Juntao Yuan
- [3] Writing portable J2ME applications, Anurag Gupta, Mindfire Solutions
- [4] J2ME in a Nutshell, Kim Topley, O'Reilly, 2002
- [5] Sun's J2ME Wireless Toolkit Documentation
<http://java.sun.com/j2me/docs/index.html>
- [6] Documentation in J2ME Wireless Toolkit download
<http://www.eli.sdsu.edu/courses/fall04/cs683/j2me/index.html>
- [7] Wireless J2ME™ Platform Programming by Vartan Piroumian, Prentice Hall PTR
- [8] Core J2ME™ Technology & MIDP by John W. Muchow, Prentice Hall PTR

APPENDIX B

ADDITIONAL SYSTEM INFORMATION

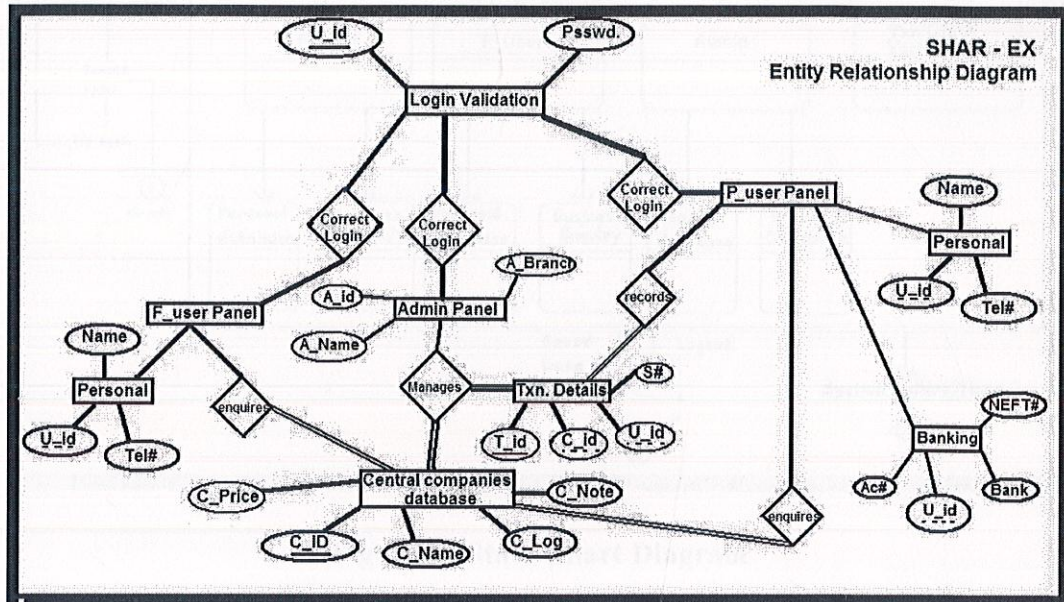


Figure 13 Entity Relationship Diagram

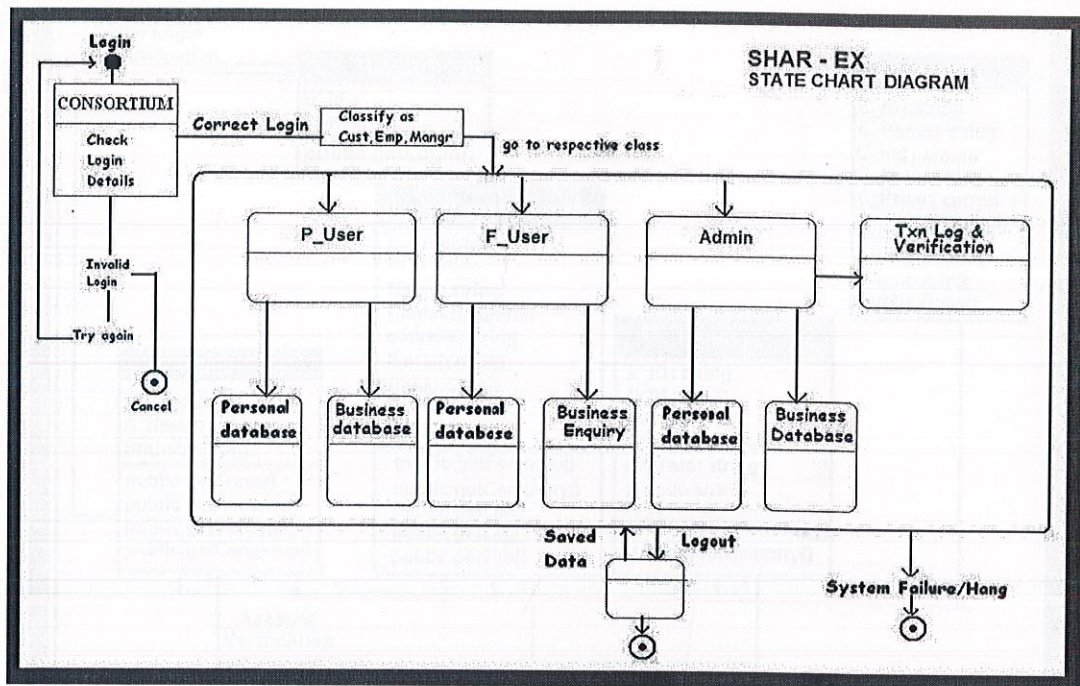


Figure 14 State Chart Diagram

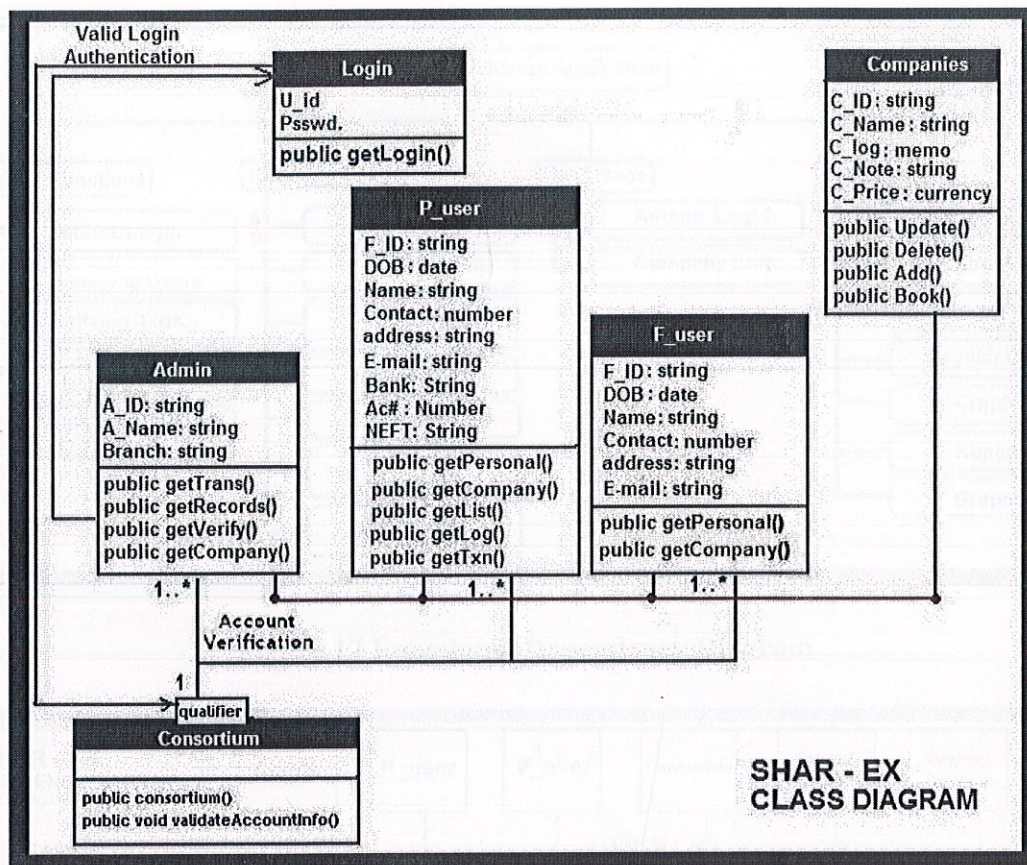


Figure 15 Class Diagram

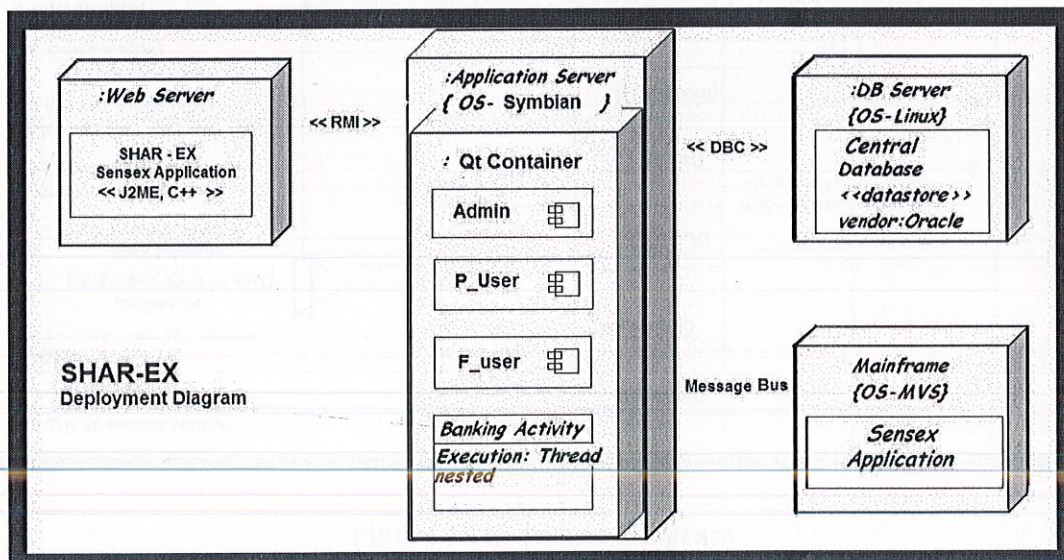


Figure 16 Deployment Diagram

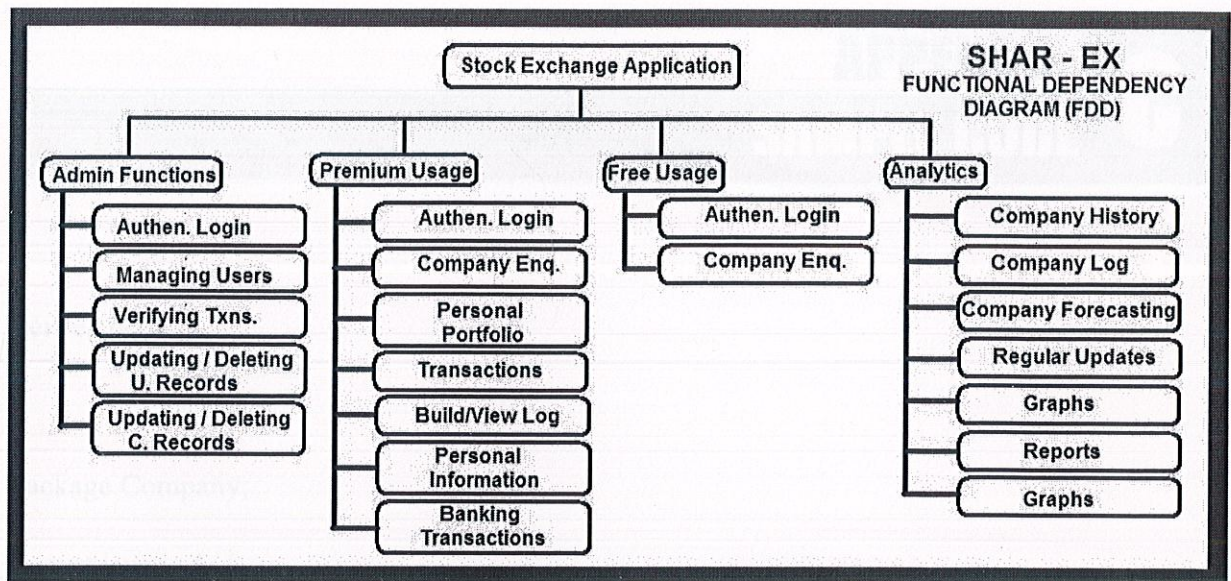


Figure 17 Functional Dependency Diagram

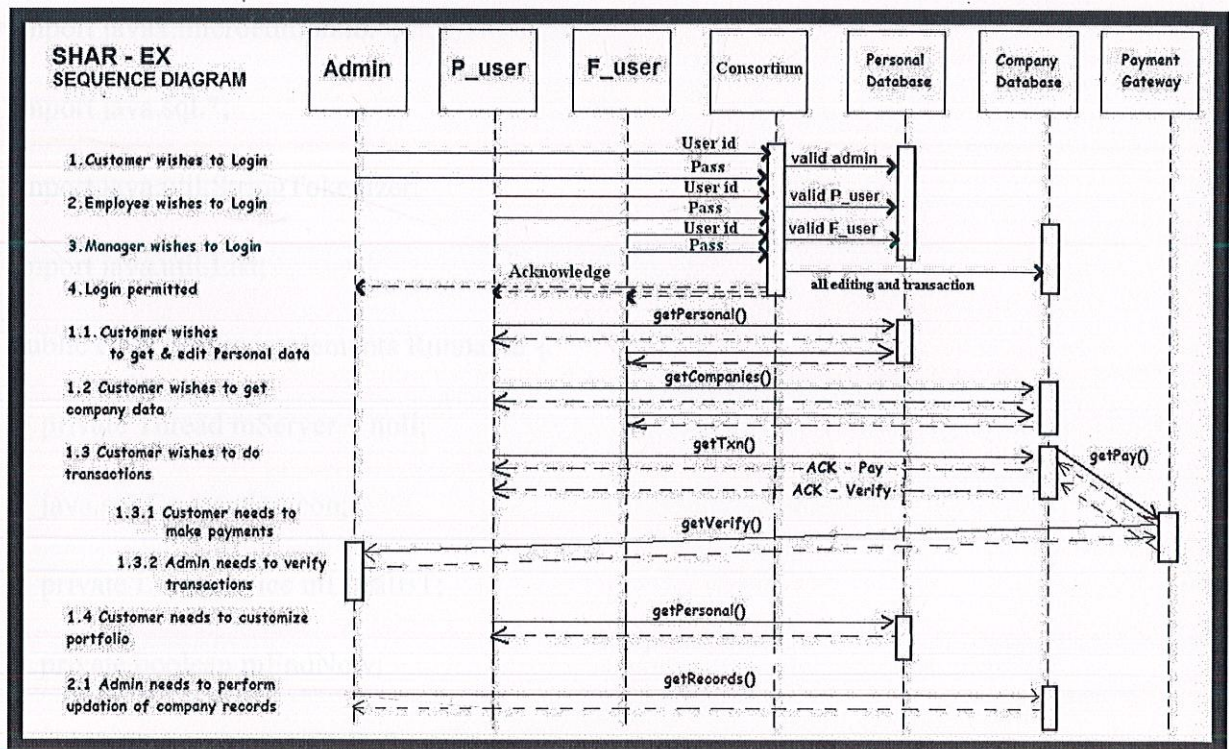


Figure 18 Sequence Diagram

Server.java

```
package Company;

import java.io.*;

import javax.bluetooth.*;

import javax.microedition.io.*;

import java.sql.*;

import java.util.StringTokenizer;

import java.util.List;

public class Server implements Runnable {

    private Thread mServer = null;

    java.sql.Connection con;

    private LocalDevice mLocalBT;

    private boolean mEndNow;

    public String messageToBeSent="";

    private StreamConnectionNotifier mServerNotifier;
```



```

private static final UUID MY_SERVICE_ID = new
UUID("BAE0D0C0B0A000955570605040302010", false);

public void run() {

    try {

        try{

            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            con = DriverManager.getConnection("jdbc:odbc:BlueServer");

        }

        catch(Exception ex){

            System.out.println(ex.getMessage());

        }

        // get local BT manager

        mLocalBT = LocalDevice.getLocalDevice();

        // set we are discoverable

        mLocalBT.setDiscoverable(DiscoveryAgent.GIAC);

        String url = "btspp://localhost:" + MY_SERVICE_ID.toString() + ";name=Hacker
Service;authorize=false";

        // create notifier now

        mServerNotifier = (StreamConnectionNotifier) Connector.open( url.toString());

        //System.out.println(" got notifier ");

    } catch (Exception e) {

```



```

        System.err.println("Can't initialize bluetooth: " + e);
    }

    StreamConnection conn = null;

    while (!mEndNow) {

        conn = null;

        try {

            conn = mServerNotifier.acceptAndOpen();

        } catch (IOException e) {

            continue;

        }

        if (conn != null)

            processRequest(conn);

    }

}

public void startServer() {

    if (mServer != null)

        return;

    // start receive thread

    mServer = new Thread(this);

    mServer.start();

}

```



```

private void processRequest(StreamConnection conn) {

    DataInputStream dis = null;

    DataOutputStream dos=null;

    try {

        dis=conn.openDataInputStream();

        dos=conn.openDataOutputStream();

        String read=dis.readUTF();

        if(read.toUpperCase().startsWith("STUDENT")){

            read=read.substring(read.indexOf(":")+1);

            Statement st=con.createStatement();

            ResultSet rs=st.executeQuery("select * from students where admno='" + read +
""");

            if(rs.next()){

                String info="";

                info="Name: " + rs.getString("sname") + ", Class: "+ rs.getString("Class") + ",
Roll: " + rs.getString("Roll");

                dos.writeUTF(info);

            }

            else{

                dos.writeUTF("Sorry, no result found!");
            }
        }
    }
}

```



```

        }

    }

    dos.flush();

    dos.close();

    dis.close();

    //conn.close();

}

catch (Exception e) {

    System.out.println(e.getMessage());

}

}

}

```

SearchDevice.java

```

package Company;

import java.io.IOException;

import java.util.Vector;

import javax.bluetooth.*;

/**

```

* Minimal Device Discovery example.

*/

```
public class SearchDevice {

    public static final Vector/*<RemoteDevice>*/ devicesDiscovered = new Vector();

    public static void main(String[] args) throws IOException, InterruptedException {

        final Object inquiryCompletedEvent = new Object();

        devicesDiscovered.removeAllElements();

        DiscoveryListener listener = new DiscoveryListener() {

            public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {

                System.out.println("Device " + btDevice.getBluetoothAddress() + " found");

                devicesDiscovered.addElement(btDevice);

                try {

                    System.out.println(" name " + btDevice.getFriendlyName(false));

                } catch (IOException cantGetDeviceName) {

                }

            }

        }

        synchronized(inquiryCompletedEvent){
```



```

        inquiryCompletedEvent.notifyAll();
    }
}

public void serviceSearchCompleted(int transID, int respCode) {

}

public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {

}

};

synchronized(inquiryCompletedEvent) {

    boolean started =
LocalDevice.getLocalDevice().getDiscoveryAgent().startInquiry(DiscoveryAgent.GIAC,
listener);

    if (started) {

        System.out.println("wait for device inquiry to complete...");

        inquiryCompletedEvent.wait();

        System.out.println(devicesDiscovered.size() + " device(s) found");

    }
}

```



```
}  
  
}  
  
}
```

Record.java

```
package Company;  
  
public class Record {  
  
    static void debug(String message) {  
  
        System.out.println(message);  
  
    }  
  
    static void debug(String message, Object o) {  
  
        System.out.println(message + " " + o);  
  
    }  
  
    static void debug(String message, Throwable e) {  
  
        System.out.println(message + " " + e.getMessage());  
  
        e.printStackTrace();  
  
    }  
  
}
```

```

static void debug(Throwable e) {

    System.out.println(e.getMessage());

    e.printStackTrace();

}

static void error(String message) {

    System.out.println(message);

}

static void error(String message, Throwable e) {

    System.out.println(message + " " + e.getMessage());

    e.printStackTrace();

}
}

```

SearchService.java

```

package Company;

import java.io.IOException;

import java.util.Enumeration;

import java.util.Vector;

```



```

import javax.bluetooth.*;

public class SearchService {

    static final UUID OBEX_FILE_TRANSFER = new UUID(0x1106);

    public static final Vector/*<String>*/ serviceFound = new Vector();

    public static void main(String[] args) throws IOException, InterruptedException {

        // First run SearchDevice and use discovered device

        SearchDevice.main(null);

        serviceFound.removeAllElements();

        String deviceName="";

        UUID serviceUUID = OBEX_FILE_TRANSFER;

        if ((args != null) && (args.length > 0)) {

            //serviceUUID = new UUID(args[0], false);

            deviceName=args[0];

        }

        final Object SearchServiceCompletedEvent = new Object();

        DiscoveryListener listener = new DiscoveryListener() {

            public void deviceDiscovered(RemoteDevice btDevice, DeviceClass cod) {

```

```
}
```

```
public void inquiryCompleted(int discType) {
```

```
}
```

```
public void servicesDiscovered(int transID, ServiceRecord[] servRecord) {
```

```
    //for (int i = 0; i < servRecord.length; i++) {
```

```
        for (int i = 0; i < 1; i++) {
```

```
            String
```

```
url=servRecord[i].getConnectionURL(ServiceRecord.NOAUTHENTICATE_NOENCRYPT  
, false);
```

```
        if (url == null) {
```

```
            continue;
```

```
        }
```

```
        serviceFound.addElement(url);
```

```
        DataElement serviceName = servRecord[i].getAttributeValue(0x0100);
```

```
        if (serviceName != null) {
```

```
            System.out.println("service " + serviceName.getValue() + " found " + url);
```



```

        } else {

            System.out.println("service found " + url);

        }

    }

}

public void SearchServiceCompleted(int transID, int respCode) {

    System.out.println("service search completed!");

    synchronized(SearchServiceCompletedEvent){

        SearchServiceCompletedEvent.notifyAll();

    }

}

};

UUID[] searchUuidSet = new UUID[] { serviceUUID };

int[] attrIDs = new int[] {

    0x0100 // Service name

};

for(Enumeration en = SearchDevice.devicesDiscovered.elements();
en.hasMoreElements(); ) {

    RemoteDevice btDevice = (RemoteDevice)en.nextElement();

```

```

        synchronized(SearchServiceCompletedEvent) {

            if(btDevice.getFriendlyName(false).equals(deviceName)){

                System.out.println("search services on " + btDevice.getBluetoothAddress() + " "
+ btDevice.getFriendlyName(false));

                LocalDevice.getLocalDevice().getDiscoveryAgent().ServiceSearch (attrIDs,
searchUuidSet, btDevice, listener);

                SearchServiceCompletedEvent.wait();

            }

        }

    }

}

```

SearchInfo.java

```

package Company;

import java.io.*;

import javax.microedition.midlet.*;

import javax.microedition.lcdui.*;

```



```
import javax.bluetooth.*;
```

```
import javax.microedition.io.*;
```

```
import javax.microedition.lcdui.Display;
```

```
import javax.microedition.lcdui.Displayable;
```

```
public class SearchInfo extends MIDlet implements CommandListener {
```

```
    private boolean midletPaused = false;
```

```
    private Display display=Display.getDisplay(this);;
```

```
    private Thread mClientThread = null;
```

```
    private boolean mEndNow = false;
```

```
    private DiscoveryAgent mDiscoveryAgent = null;
```

```
    private String mConnect = null;
```

```
    private String lastMessage="";
```

```
    StreamConnection conn = null;
```

```
    DataInputStream dis=null;
```

```
    DataOutputStream dos=null;
```

```
    //display = Display.getDisplay(this);
```

```
    private static final UUID MY_SERVICE_ID = new  
    UUID("BAE0D0C0B0A000955570605040302010", false);
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Fields ">
```

```
private Form form;
```

```
private TextField txtSearch;
```

```
private StringItem lblResult;
```

```
private Command okCommand;
```

```
private Command exitCommand;
```

```
//</editor-fold>
```

```
/**
```

```
 * The SearchInfo constructor.
```

```
 */
```

```
public SearchInfo() {
```

```
    while(mDiscoveryAgent==null){
```

```
        try {
```

```
            mDiscoveryAgent = LocalDevice.getLocalDevice().getDiscoveryAgent();
```

```
        } catch (Exception ex) {
```

```
            System.out.println(ex.getMessage());
```

```
        }
```

```
    }
```



```
}
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Methods ">
```

```
//</editor-fold>
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Method: initialize ">
```

```
/**
```

```
 * Initilizes the application.
```

```
 * It is called only once when the MIDlet is started. The method is called before the  
<code>startMIDlet</code> method.
```

```
*/
```

```
private void initialize() {
```

```
    // write pre-initialize user code here
```

```
    // write post-initialize user code here
```

```
}
```

```
//</editor-fold>
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Method: startMIDlet ">
```

```
/**
```

* Performs an action assigned to the Mobile Device - MIDlet Started point.

*/

```
public void startMIDlet() {
```

```
    // write pre-action user code here
```

```
    switchDisplayable(null, getForm());
```

```
    // write post-action user code here
```

```
}
```

```
//</editor-fold>
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Method: resumeMIDlet ">
```

```
/**
```

* Performs an action assigned to the Mobile Device - MIDlet Resumed point.

*/

```
public void resumeMIDlet() {
```

```
    // write pre-action user code here
```

```
    // write post-action user code here
```

```
}
```

```
//</editor-fold>
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Method: switchDisplayable ">
```


/**

* Switches a current displayable in a display. The `display` instance is taken from `getDisplay` method. This method is used by all actions in the design for switching displayable.

* @param alert the Alert which is temporarily set to the display; if `null`, then `nextDisplayable` is set immediately

* @param nextDisplayable the Displayable to be set

*/

```
public void switchDisplayable(Alert alert, Displayable nextDisplayable) {
```

```
    // write pre-switch user code here Display display = getDisplay();
```

```
    if (alert == null) {
```

```
        display.setCurrent(nextDisplayable);
```

```
    } else {
```

```
        display.setCurrent(alert, nextDisplayable);
```

```
    }
```

```
    // write post-switch user code here
```

```
}
```

```
//</editor-fold>
```

```
//<editor-fold defaultstate="collapsed" desc=" Generated Method: commandAction for  
Displayables ">
```

```
/**
```

* Called by a system to indicated that a command has been invoked on a particular displayable.

* @param command the Command that was invoked

* @param displayable the Displayable where the command was invoked

*/

```
public void commandAction(Command command, Displayable displayable) {
```

```
    // write pre-action user code here
```

```
    if (displayable == form) {
```

```
        if (command == exitCommand) {
```

```
            // write pre-action user code here
```

```
            // write post-action user code here
```

```
            this.exitMIDlet();
```

```
        } else if (command == okCommand) {
```

```
            // write pre-action user code here
```

```
            // write post-action user code here
```

```
            searchDict();
```

```
        }
```

```
    }
```

```
    // write post-action user code here
```

```
}
```



```

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc=" Generated Getter: form ">

/**
 * Returns an initiliazed instance of form component.
 * @return the initialized component instance
 */

public Form getForm() {
    if (form == null) {
        // write pre-init user code here

        form = new Form("Search Student Info.", new Item[] { getTxtSearch(),
getLblResult() });

        form.addCommand(getOkCommand());

        form.addCommand(getExitCommand());

        form.setCommandListener(this);

        // write post-init user code here
    }

    return form;
}

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc=" Generated Getter: txtSearch ">

```

```

/**
 * Returns an initiliazed instance of txtSearch component.
 * @return the initialized component instance
 */

public TextField getTxtSearch() {
    if (txtSearch == null) {
        // write pre-init user code here

        txtSearch = new TextField("Admsn No:", null, 32, TextField.ANY);

        // write post-init user code here
    }

    return txtSearch;
}

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc=" Generated Getter: okCommand ">

/**
 * Returns an initiliazed instance of okCommand component.
 * @return the initialized component instance
 */

public Command getOkCommand() {

```



```

    if (okCommand == null) {

        // write pre-init user code here

        okCommand = new Command("Ok", Command.OK, 0);

        // write post-init user code here

    }

    return okCommand;

}

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc=" Generated Getter: lblResult ">

/**

 * Returns an initiliazed instance of lblResult component.

 * @return the initialized component instance

 */

public StringItem getLblResult() {

    if (lblResult == null) {

        // write pre-init user code here

        lblResult = new StringItem("Result:", "");

        // write post-init user code here

    }

    return lblResult;

```

```

}

//</editor-fold>

//<editor-fold defaultstate="collapsed" desc=" Generated Getter: exitCommand ">

/**

 * Returns an initiliazed instance of exitCommand component.

 * @return the initialized component instance

 */

public Command getExitCommand() {

    if (exitCommand == null) {

        // write pre-init user code here

        exitCommand = new Command("Exit", Command.EXIT, 0);

        // write post-init user code here

    }

    return exitCommand;

}

//</editor-fold>

/**

 * Returns a display instance.

 * @return the display instance.

 */

```



```

public Display getDisplay () {

    return Display.getDisplay(this);

}

class ShowMessage implements Runnable {

    Display disp = null;

    String message = null;

    public ShowMessage(String mess) {

        try{

            message = mess;

        } catch(Exception ex){

            //ignore

        }

    }

    public void run() {

        if(message!=null)

            lblResult.setText(message);

    }

}

void searchDict(){

```

```

try {

    mConnect = mDiscoveryAgent.selectService(MY_SERVICE_ID,
ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);

    String readMessage="";

    if (mConnect != null) {

        conn = (StreamConnection) Connector.open(mConnect);

        dis=conn.openDataInputStream();

        dos=conn.openDataOutputStream();

        dos.writeUTF("STUDENT:" + txtSearch.getString());

        dos.close();

        readMessage=dis.readUTF();

        dis.close();

    } else{

        lblResult.setText("Unable to connect!");

    }

    conn.close();

    lastMessage=readMessage;

    getDisplay().callSerially(new ShowMessage(lastMessage));

} catch (Exception ex) {

    System.err.println(ex.getMessage());

}

```



```

    }

    /**

    * Exits MIDlet.

    */

    public void exitMIDlet() {

        switchDisplayable (null, null);

        destroyApp(true);

        notifyDestroyed();

    }

    /**

    * Called when MIDlet is started.

    * Checks whether the MIDlet have been already started and initialize/starts or resumes the
    MIDlet.

    */

    public void startApp() {

        if (midletPaused) {

            resumeMIDlet ();

        } else {

            initialize ();

```

```

        startMIDlet ();

    }

    midletPaused = false;

}

/**
 * Called when MIDlet is paused.
 */

public void pauseApp() {

    midletPaused = true;

}

/**
 * Called to signal the MIDlet to terminate.
 *
 * @param unconditional if true, then the MIDlet has to be unconditionally terminated and
all resources has to be released.
 */

public void destroyApp(boolean unconditional) {}

}

```