School of Computer Science & IT

Jayanagar 9th Block Campus, Bangalore

Department of BCA

A Report on

Mobile Database Management

Submitted By

SABARINATHAN S S

HARSHITHA K

DIVYA N

GANESH PRASAD R

PRADEEP KUMAR N

Under the guidance of

Asst. Prof. Saswathi Mahapatra

School of CS & IT

Department of BCA

Jain University.

Bangalore.

Asst. Prof. Akshatha VS

School of CS & IT

Department of BCA

Jain University.

Bangalore.

INDEX

ABSTRACTION

This 'Mobile Database Management Project' has been designed taking into account the practical needs to manage the details of the mobile devices launched in the market. It has been developed to override the problems of customers and mobile store managers. A customer can find the list of phones available in the matket through the smooth and easy to operate graphical interface provided in the application. This project is designed to ease the work of store managers, who can maintain an accurate record of the available mobile phones that are available in their store.

This mini project is developed to give a basic idea of how to work with DBMS and Java Interface so as to understand the basics of Database System and storing and generating large amount of data easily and efficiently.

This project is designed to take advantage of today's technology and reduce or avoid the burden of storing data on paper or in files. Our database management project aims to provide computerized interface to all the data stored and manipulated. The application provides for retrieving, storing, modifying, maintaining and easy use of data.

CHAPTER 1: INTRODUCTION

The mobile database has been designed taking into account the practical needs to manage the details of the mobile devices launched in the market. It has been designed for desktop systems. This software will provide a simple and easy to operate graphical interface which can be utilized by any user without the depth knowledge of technology. It provides security at product level as well as user level.

Through the Java Interface, the administrator can view and manipulate the records in the table.

- The table consisting of different mobile records will be provided to the user and therefore they will have various options to choose from.
- Once a new mobile record is added, the database will store the information of the mobile into the relation **'mobile'**.

1.1 DATA STORAGE

- **Mobile Entity:** name, brand, model, price, pcamera, scamera, ram, processor, storage, battery.
- **Account Entity:** username, password, isadmin

1.2 MANIPULATION OF DATABASE:

- Addition of mobiles
- Modification of mobiles
- Deletion of mobiles

CHAPTER 2: SYSTEM SPECIFICATIONS

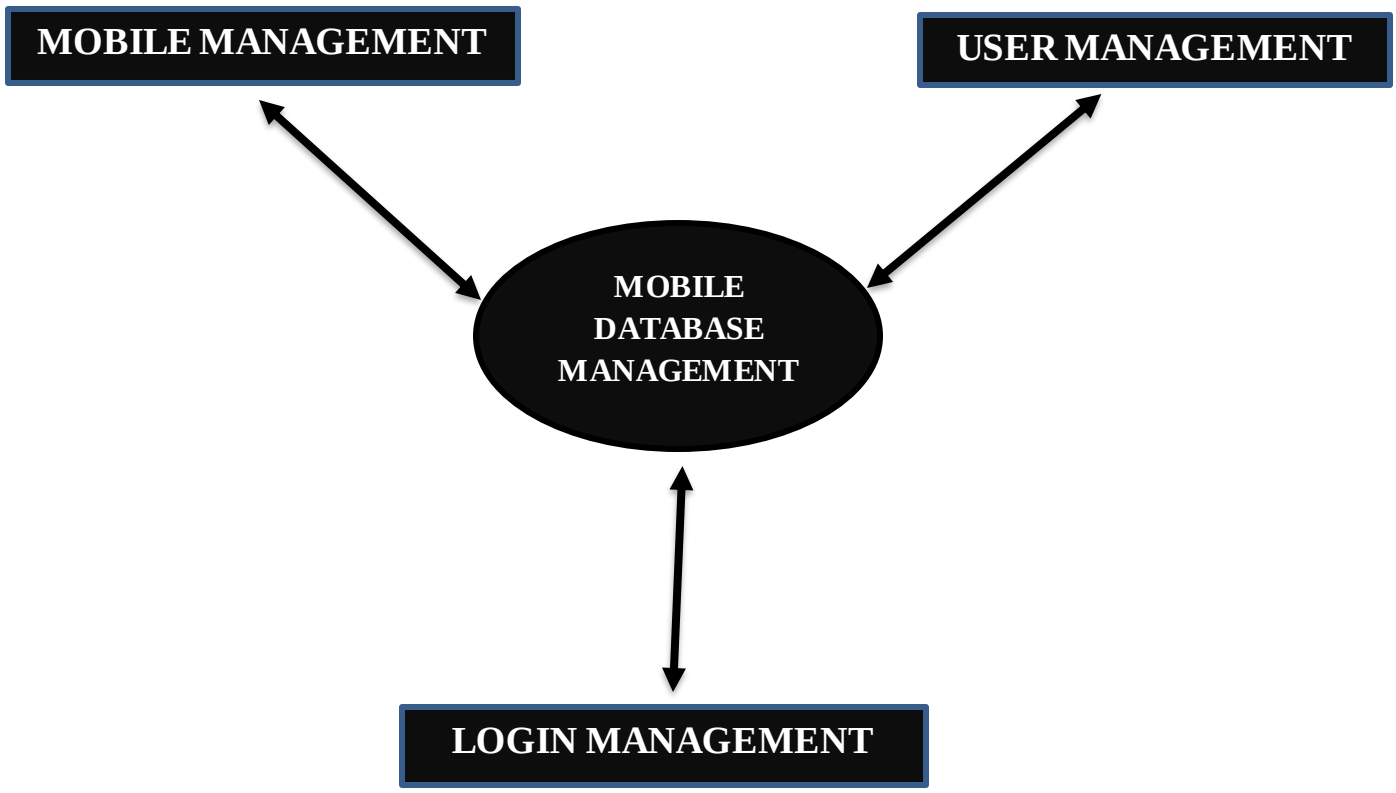
2.1 HARDWARE REQUIREMENTS

- Intel or AMD processor
- 512 MB RAM
- 250 MB HDD

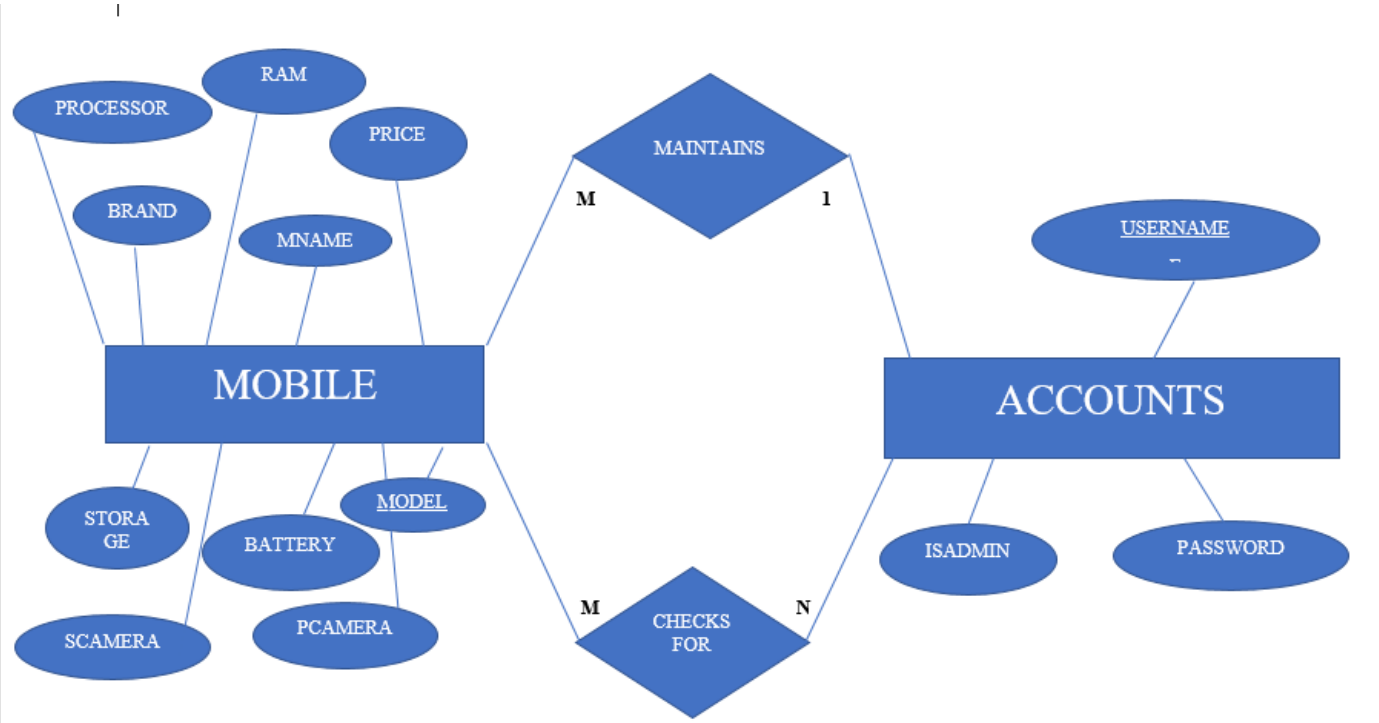
2.2 SOFTWARE REQUIREMENTS

- MySQL
- Intellij for JAVA
- Local Server (Xampp)
- Windows operating system

CHAPTER 3: DATAFLOW DIAGRAM



CHAPTER 4: ER DIAGRAM



CHAPTER 5: DATABASE DESIGN

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems.

It is the organization of data according to a database model. The designer determines what data must be stored and how the data elements interrelate. With this information, they can begin to fit the data to the database model.

Database design involves classifying data and identifying interrelationships. In a majority of cases, a person who is doing the design of a database is a person with expertise in the area of database design, rather than expertise in the domain from which the data to be stored is drawn e.g. financial information, biological information etc. Therefore, the data to be stored in the database must be determined in cooperation with a person who does have expertise in that domain, and who is aware of what data must be stored within the system.

This process is one which is generally considered part of requirements analysis, and requires skill on the part of the database designer to elicit the needed information from those with the domain knowledge. This is because those with the necessary domain knowledge frequently cannot express clearly what their system requirements for the database are as they are unaccustomed to thinking in terms of the discrete data elements which must be stored. Data to be stored can be determined by Requirement Specification.
Once a database designer is aware of the data which is to be stored within the database, they must then determine where dependency is within the data. Sometimes when data is changed you can be changing other data that is not visible.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

The logical model concentrates on the data requirements and the data to be stored independent of physical considerations. It does not concern itself with how the data will be stored or where it will be stored physically.

The physical data design model involves translating the logical design of the database onto physical media using hardware resources and software systems such as database management systems (DBMS).
Database development life cycle
the database development life cycle has a number of stages that are followed when developing database systems.
The steps in the development life cycle do not necessary have to be followed religiously in a sequential manner.
On small database systems, the database system development life cycle is usually very simple and does not involve a lot of steps.

In order to fully appreciate the above diagram, let's look at the individual components listed in each step.

Requirements analysis

Planning - This stages concerns with planning of entire Database Development Life Cycle it takes into consideration the Information Systems strategy of the organization.

System definition - This stage defines the scope and boundaries of the proposed database system.

Database designing

Logical model - This stage is concerned with developing a database model based on requirements. The entire design is on paper without any physical implementations or specific DBMS considerations.

Physical model - This stage implements the logical model of the database taking into account the DBMS and physical implementation factors

TABLES:

Accounts

username	password	isadmin
admin	admin	1
divya	div123	0
ganesh	gani123	0
harshitha	harshi123	0
pradeep	deep123	0
sabari	sabari123	0

Mobile

name	model	brand	price	pcamera	scamera	ram	processor	storage	battery
OnePlus 6	1+6	OnePlus	38999	16MP+20MP	16MP	8GB	Snapdragon 845	128GB	3300mAh
OnePlus 7	1+7	OnePlus	32999	48MP+5MP	16MP	8GB	Snapdragon 845	128GB	3700mAh
Samsung M20	M20	Samsung	13999	48MP	16MP	4GB	Exynos 7904	64GB	5000mAh
Samsung M30	M30	Samsung	16999	48MP+16MP+5MP	16MP	6GB	Exynos 7924	128GB	5000mAh
Samsung M40	M40	Samsung	24000	48MP+32MP+16MP	16MP+8MP	8GB	Exynos 9820	256GB	3750mAh
ASUS ROG	ROG	ASUS	37990	16MP+8MP	8MP	8GB	Snapdragon 845	128GB	4000mAh
Samsung S9	S9	Samsung	49999	16MP	8MP	8GB	Exynos 9810	256GB	4750mAh
vivi v17	v17	vivo	30000	16MP	8MP	6GB	snapdragon 855	64GB	3500Mah
Vivo V9	V9	Vivo	18000	32MP	16MP	4GB	Snapdragon 870	64GB	4000Mah

CHAPTER 6: SOURCE CODE

1. Main.java

```
package sample;

    import javafx.application.Application;
    import javafx.fxml.FXMLLoader;
    import javafx.scene.Parent;
    import javafx.scene.Scene;
    import javafx.stage.Stage;

public class Main extends Application {

  @Override
  public void start(Stage primaryStage) throws Exception{
    Parent root = FXMLLoader.load(getClass().getResource("login.fxml"));
    primaryStage.setTitle("User Login");
    primaryStage.setScene(new Scene(root, 311, 448));
    primaryStage.show();
  }


  public static void main(String[] args) {
    launch(args);
  }
}
```

2. Controller.java

```
package sample;

import javafx.collections.ObservableList;
import javafx.scene.control.*;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import java.sql.SQLException;

public class Controller {

  @FXML private Label status;
```

```
    @FXML TextField mobilename_field;
    @FXML TextField model_field;
    @FXML TextField brand_field;
    @FXML TextField price_field;
    @FXML TextField primary_camera_field;
    @FXML TextField secondary_camera_field;
    @FXML TextField ram_field;
    @FXML TextField processor_field;
    @FXML TextField storage_field;
    @FXML TextField battery_field;

    @FXML TextArea console;
    @FXML TextField searchfield;

    @FXML private TableView phonetable;

    @FXML private TableColumn<PopulateTable,String> col_name;
    @FXML private TableColumn<PopulateTable,String> col_model;
    @FXML private TableColumn<PopulateTable,String> col_brand;
    @FXML private TableColumn<PopulateTable,Integer> col_price;
    @FXML private TableColumn<PopulateTable,String> col_pcamera;
    @FXML private TableColumn<PopulateTable,String> col_scamera;
    @FXML private TableColumn<PopulateTable,String> col_ram;
    @FXML private TableColumn<PopulateTable,String> col_processor;
    @FXML private TableColumn<PopulateTable,String> col_storage;
    @FXML private TableColumn<PopulateTable,String> col_battery;

    @FXML
    private void addDataBtn(ActionEvent event) throws SQLException,ClassNotFoundException
      {
      try {
         Phones.insertphones(mobilename_field.getText(), model_field.getText(),
brand_field.getText(), Integer.parseInt(price_field.getText()), primary_camera_field.getText(),
secondary_camera_field.getText(), ram_field.getText(), processor_field.getText(),
storage_field.getText(),battery_field.getText());
         console.setText("One Row Added");
         ObservableList<PopulateTable> phonelist = Phones.getAllRecords(); //To refresh the Table
         populateTable(phonelist);                                  //To refresh the Table
         }
      catch (SQLException e)
         {console.setText("Error Adding Data: "+e);}
```

```
    }

  @FXML
  public void deleteDataBtn(ActionEvent event) throws SQLException,ClassNotFoundException
    {
     try {
        Phones.deletephones(searchfield.getText());
        console.setText("Row Deleted");
        ObservableList<PopulateTable> phonelist = Phones.getAllRecords();    //To refresh the
Table
        populateTable(phonelist);                              //To refresh the Table
       }
     catch (SQLException e)
       {console.setText("Error Deleting the Data: "+e);}
    }


  public void initialize() throws Exception{

       col_name.setCellValueFactory(cellData ->cellData.getValue().getNameProp());
       col_model.setCellValueFactory(cellData ->cellData.getValue().getModelProp());
       col_brand.setCellValueFactory(cellData ->cellData.getValue().getBrandProp());
       col_price.setCellValueFactory(cellData ->cellData.getValue().getPriceProp().asObject());
       col_pcamera.setCellValueFactory(cellData ->cellData.getValue().getPcameraProp());
       col_scamera.setCellValueFactory(cellData ->cellData.getValue().getScameraProp());
       col_ram.setCellValueFactory(cellData ->cellData.getValue().getRamProp());
       col_processor.setCellValueFactory(cellData ->cellData.getValue().getProcessorProp());
       col_storage.setCellValueFactory(cellData ->cellData.getValue().getStorageProp());
       col_battery.setCellValueFactory(cellData ->cellData.getValue().getBatteryProp());
       ObservableList<PopulateTable> phonelist = Phones.getAllRecords();          //To refresh
the Table
       populateTable(phonelist);                                  //To refresh the Table

  }

  private void populateTable(ObservableList<PopulateTable> phonelist)
  {
  phonetable.setItems(phonelist);
  }

  @FXML
```

```
    private void searchPhone(ActionEvent event) throws ClassNotFoundException,SQLException
    {
    ObservableList<PopulateTable> list = Phones.searchForPhone(searchfield.getText());      //Taking
 model from the textfield
        populateTable(list);
    }

    @FXML
    private void refreshTable(ActionEvent event) throws ClassNotFoundException,SQLException
    {
     ObservableList<PopulateTable> phonelist = Phones.getAllRecords();
     populateTable(phonelist);
    }
 }
```

3. **Phones.java**

```
package sample;




import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import java.sql.ResultSet;

import java.sql.SQLException;


public class Phones

{

   public static void insertphones(String m_name,String m_model, String m_brand,int
m_price,String m_pcamera,String m_scamera,String m_ram,String m_processor,String
m_storage,String m_battery) throws ClassNotFoundException,SQLException

   {

      String sql = "Insert into
mobile(name,model,brand,price,pcamera,scamera,ram,processor,storage,battery)
values('"+m_name+"','"+m_model+"','"+m_brand+"','"+m_price+"','"+m_pcamera+"','"+m_sca
mera+"','"+m_ram+"','"+m_processor+"','"+m_storage+"','"+m_battery+"');";
```

```
    try {                                                    //SingleInvertedComma  so that the
items shld be in Invertedcommas
        MySqlConnector.dbExecuteQuery(sql);
        System.out.println("Data  Inserted");
        }

    catch(SQLException  e)
        {System.out.println("Exception  occurred  while  Inserting Data: "+e);}
  }

  public  static void deletephones(String  id) throws  SQLException,ClassNotFoundException
  {
    String  sql = "delete from mobile where  model = '"+id+"'";
    try {
        MySqlConnector.dbExecuteQuery(sql);
        System.out.println("Data  Deleted");
    }
    catch (SQLException  e)
        {System.out.println("Error  While  Deleting the data: "+e);}
  }

  //Search Query
  public  static ObservableList<PopulateTable>  searchForPhone(String  model) throws
ClassNotFoundException,SQLException
  {
    String  sqlQuery = "select * from mobile where  model='"+model+"';";
```

```
        try {
            ResultSet resultSet = MySqlConnector.dbRetreiveQuery(sqlQuery);
            ObservableList<PopulateTable>  list = getPhonesObjects(resultSet);
            return list;
        }
        catch (SQLException e)
        {System.out.println("Error Searching Database: "+e);}

        return null;
    }

    //Retreive Data Function
    public static ObservableList<PopulateTable> getAllRecords() throws
SQLException,ClassNotFoundException
    {
        String sql = "select * from mobile;"  ;
         try{
             ResultSet resultset = MySqlConnector.dbRetreiveQuery(sql);
             ObservableList<PopulateTable>  phonelist = getPhonesObjects(resultset);
             return phonelist;
         }
         catch (SQLException e)
           {System.out.println("Error Fetching Records from DB: "+e);}
        return null;
    }
```

```
private static ObservableList<PopulateTable> getPhonesObjects(ResultSet resultset) throws
ClassNotFoundException,SQLException
    {
      try {
        ObservableList<PopulateTable> phonelist = FXCollections.observableArrayList();
        while (resultset.next())
          {
          PopulateTable phone = new PopulateTable();
          phone.setName(resultset.getString("name"));
          phone.setBrand(resultset.getString("brand"));
          phone.setModel(resultset.getString("model"));
          phone.setPrice(resultset.getInt("price"));
          phone.setRam(resultset.getString("ram"));
          phone.setProcessor(resultset.getString("processor"));
          phone.setStorage(resultset.getString("storage"));
          phone.setScamera(resultset.getString("scamera"));
          phone.setPcamera(resultset.getString("pcamera"));
          phone.setBattery(resultset.getString("battery"));
          phonelist.add(phone);
          }
          return phonelist;
        }
      catch (SQLException e)
        {System.out.println("Error Occurred Fetching Data"+e);}

    return null;
```

```
    }
  //Retrieve Data Function END



  //For Login
  public static int accountsearch(String user,String pass) throws
ClassNotFoundException,SQLException
  {
    String sqlQuery = "select * from accounts where username='"+user+"' and
password='"+pass+"';";
    try {
      ResultSet resultSet = MySqlConnector.dbRetreiveQuery(sqlQuery);
      while (resultSet.next())
      {
        String uname = resultSet.getString(1);
        String upass = resultSet.getString(2);
        int isadmin = resultSet.getInt(3);

        if(uname.equals(user) && upass.equals(pass) && isadmin==1) return 2;
        else if(uname.equals(user) && upass.equals(pass)) return 1;
        else return 0;
      }
    }
    catch (SQLException e)
    {System.out.println("Error Searching Database: "+e);}
```

```
        return 0;
    }
} //Retrieve Data Function END
}
```

4. **PopulateTable.java**

```
package sample;

import javafx.beans.property.IntegerProperty;
import javafx.beans.property.SimpleIntegerProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.beans.property.StringProperty;

public class PopulateTable {

    private StringProperty
nameProperty,modelProperty,brandProperty,pcameraProperty,scameraProperty,ramProperty,pro
cessorProperty,storageProperty,batteryProperty;
    private IntegerProperty priceProperty;



    public PopulateTable() {
        this.nameProperty = new SimpleStringProperty();
        this.modelProperty = new SimpleStringProperty();
        this.brandProperty = new SimpleStringProperty();
        this.priceProperty = new SimpleIntegerProperty();
```

```
        this.pcameraProperty  = new SimpleStringProperty();
        this.scameraProperty  = new SimpleStringProperty();
        this.ramProperty  = new SimpleStringProperty();
        this.processorProperty  = new SimpleStringProperty();
        this.storageProperty  = new SimpleStringProperty();
        this.batteryProperty  = new SimpleStringProperty();
    }

    public String getName()         { return nameProperty.get(); }
    public void setName(String name) { this.nameProperty.set(name);}
    public StringProperty getNameProp() {return nameProperty;}

    public String getModel()        { return modelProperty.get(); }
    public void setModel(String model) { this.modelProperty.set(model);}
    public StringProperty getModelProp() {return modelProperty;}

    public String getBrand()        { return brandProperty.get(); }
    public void setBrand(String brand) { this.brandProperty.set(brand);}
    public StringProperty getBrandProp() {return brandProperty;}

    public Integer getPrice()       { return priceProperty.get(); }
    public void setPrice(Integer price) { this.priceProperty.set(price);}
    public IntegerProperty getPriceProp() {return priceProperty;}

    public String getPcamera()        { return pcameraProperty.get(); }
    public void setPcamera(String pcamera) { this.pcameraProperty.set(pcamera);}
```

```
    public StringProperty getPcameraProp()  {return pcameraProperty;}

    public String getScamera()        { return scameraProperty.get(); }
    public void setScamera(String scamera) { this.scameraProperty.set(scamera);}
    public StringProperty getScameraProp()  {return scameraProperty;}

    public String getRam()        { return ramProperty.get(); }
    public void setRam(String ram) { this.ramProperty.set(ram);}
    public StringProperty getRamProp()  {return ramProperty;}

    public String getProcessor()        { return processorProperty.get(); }
    public void setProcessor(String processor) { this.processorProperty.set(processor);}
    public StringProperty getProcessorProp()  {return processorProperty;}

    public String getStorage()        { return storageProperty.get(); }
    public void setStorage(String storage) { this.storageProperty.set(storage);}
    public StringProperty getStorageProp()  {return storageProperty;}

    public String getBattery()        { return batteryProperty.get(); }
    public void setBattery(String battery) { this.batteryProperty.set(battery);}
    public StringProperty getBatteryProp()  {return batteryProperty;}

}
```

5. **MySQLConnector.java**

```
package sample;

import com.sun.rowset.CachedRowSetImpl;
import java.sql.*;

class MySqlConnector {
   static Connection conn = null;
   public static final String url ="jdbc:mysql://localhost:3306/";
   public static final String db =
"mobiledb?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=f
alse&serverTimezone=UTC";
   public static final String user="root";
   public static final String pass="";


   public static void getconnection(){

      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         System.out.println("Driver Connected");
         conn = DriverManager.getConnection(url+db,user,pass);
         System.out.println("Connection Established");
      }

      catch (ClassNotFoundException e)
      { System.out.println("Driver not Loaded. ERROR: "+e); }
      catch (SQLException e)
      { System.out.println("Link Not Established. ERROR: "+e); }

   }

   public static void closeconnection()
   {
      try {
      conn.close();
      System.out.println("Connection Closed");
   }
   catch (SQLException e)
   {System.out.println("Connection Cannot be Closed. ERROR: "+e);}}

   //Function to Insert,Delete and Update
```

```
    public static String dbExecuteQuery(String sqlStmt) throws
SQLException,ClassNotFoundException
    {
        Statement stmt = null;

        try {
            getconnection();
            stmt = conn.createStatement();
            stmt.executeUpdate(sqlStmt);

            }
        catch (SQLException e)
            {return "Error occurred";
               // System.out.println("Error in Executing Query: " +e);
            }

        finally {
            if (stmt != null)
               stmt.close();
            closeconnection();
               }
        return null;
    }

    //Funtion to Retrive Data

    public static ResultSet dbRetreiveQuery(String sqlQuery) throws
ClassNotFoundException,SQLException
    {
        Statement stmt = null;
        ResultSet resultset = null;
        CachedRowSetImpl crs = null;

        try {
            getconnection();
            stmt = conn.createStatement();
            resultset = stmt.executeQuery(sqlQuery);
            crs = new CachedRowSetImpl();
            crs.populate(resultset);
            }
```

```
        catch (SQLException e)
            { System.out.println("Error in Retrieving Database: "+e); }

        finally {
            if(stmt!=null) stmt.close();
            if (resultset!=null) resultset.close();
            closeconnection();
            }
        return crs;
    }
}
```

6. adminUserMobile.fxml

```
<?xml version="1.0" encoding="UTF-8"?>


<?import javafx.geometry.Insets?>

<?import javafx.scene.control.Button?>

<?import javafx.scene.control.Label?>

<?import javafx.scene.control.TableColumn?>

<?import javafx.scene.control.TableView?>

<?import javafx.scene.control.TextArea?>

<?import javafx.scene.control.TextField?>

<?import javafx.scene.layout.AnchorPane?>

<?import javafx.scene.text.Font?>


<AnchorPane prefHeight="595.0" prefWidth="996.0" xmlns="http://javafx.com/javafx/11.0.1"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="sample.Controller">

  <children>

    <Button fx:id="addbtn" layoutX="38.0" layoutY="488.0" mnemonicParsing="false"
onAction="#addDataBtn" prefHeight="25.0" prefWidth="149.0" text="ADD" />
```

```
<AnchorPane layoutX="208.0" layoutY="-5.0" minHeight="-Infinity" minWidth="-Infinity" prefHeight="550.0" prefWidth="650.0">
   <children>
      <TableView fx:id="phonetable" layoutY="55.0" prefHeight="464.0" prefWidth="770.0">
        <columns>
          <TableColumn fx:id="col_name" prefWidth="92.0" text="Name" />
          <TableColumn fx:id="col_model" prefWidth="85.0" text="Model" />
           <TableColumn fx:id="col_brand" prefWidth="75.0" text="Brand" />
           <TableColumn fx:id="col_price" prefWidth="70.0" style="alignment: center;" text="Price" />
           <TableColumn fx:id="col_ram" prefWidth="67.0" text="RAM" />
           <TableColumn fx:id="col_processor" prefWidth="101.0" text="Processor" />
           <TableColumn fx:id="col_storage" prefWidth="75.0" text="Storage" />
           <TableColumn fx:id="col_pcamera" minWidth="0.0" prefWidth="74.0" text="PCamera" />
           <TableColumn fx:id="col_scamera" prefWidth="66.0" text="Scamera" />
           <TableColumn fx:id="col_battery" minWidth="7.0" prefWidth="64.0" text="Battery" />
        </columns>
      </TableView>
      <Label layoutX="41.0" layoutY="540.0" text="CONSOLE:">
        <font>
          <Font name="System Bold" size="14.0" />
        </font>
      </Label>
   </children>
   <padding>
```

```
        <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
      </padding>
    </AnchorPane>
    <TextField fx:id="mobilename_field" layoutX="38.0" layoutY="81.0" promptText="Mobile
Name" />
    <TextField fx:id="model_field" layoutX="38.0" layoutY="123.0" promptText="Model" />
    <TextField fx:id="brand_field" layoutX="38.0" layoutY="164.0" promptText="Brand" />
    <TextField fx:id="price_field" layoutX="38.0" layoutY="205.0" promptText="Price" />
    <TextField fx:id="primary_camera_field" layoutX="38.0" layoutY="246.0"
promptText="Primary camera" />
    <TextField fx:id="secondary_camera_field" layoutX="38.0" layoutY="286.0"
promptText="Secondary Camera" />
    <TextField fx:id="ram_field" layoutX="38.0" layoutY="326.0" promptText="RAM" />
    <TextField fx:id="processor_field" layoutX="38.0" layoutY="367.0" promptText="Processor"
/>
    <TextField fx:id="storage_field" layoutX="38.0" layoutY="409.0" promptText="Storage" />
    <Label fx:id="status" layoutX="25.0" layoutY="14.0" text="MOBILES">
      <font>
        <Font name="System Bold Italic" size="21.0" />
      </font>
    </Label>
    <TextField fx:id="battery_field" layoutX="38.0" layoutY="446.0" promptText="Battery" />
    <TextArea fx:id="console" editable="false" layoutX="327.0" layoutY="527.0"
prefHeight="37.0" prefWidth="587.0">
      <font>
        <Font name="System Bold" size="12.0" />
      </font>
    </TextArea>
```

```
    <Button layoutX="757.0" layoutY="17.0" mnemonicParsing="false"
onAction="#deleteDataBtn" prefHeight="25.0" prefWidth="67.0" text="DELETE" />

    <Button layoutX="832.0" layoutY="17.0" mnemonicParsing="false" onAction="#searchPhone"
prefHeight="25.0" prefWidth="67.0" text="SEARCH" />

    <TextField fx:id="searchfield" layoutX="588.0" layoutY="17.0" promptText="ModelNo" />

    <Button layoutX="908.0" layoutY="17.0" mnemonicParsing="false"
onAction="#refreshTable" prefHeight="25.0" prefWidth="67.0" text="REFRESH" />

  </children>

</AnchorPane>
```

7. **Standardusermobile**

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.TableColumn?>
<?import javafx.scene.control.TableView?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.text.Font?>

<AnchorPane prefHeight="549.0" prefWidth="833.0"
xmlns="http://javafx.com/javafx/11.0.1" xmlns:fx="http://javafx.com/fxml/1"
fx:controller="sample.Controller">
   <children>
      <AnchorPane layoutX="18.0" layoutY="3.0" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="512.0" prefWidth="697.0">
         <children>
            <TableView fx:id="phonetable" layoutX="4.0" layoutY="50.0" prefHeight="471.0"
prefWidth="788.0">
               <columns>
                  <TableColumn fx:id="col_name" prefWidth="99.0" text="Name" />
                  <TableColumn fx:id="col_model" prefWidth="83.0" text="Model" />
                  <TableColumn fx:id="col_brand" prefWidth="68.0" text="Brand" />
```

```
                <TableColumn fx:id="col_price" prefWidth="67.0" style="alignment: center;"
text="Price" />
                <TableColumn fx:id="col_ram" prefWidth="66.0" text="RAM" />
                <TableColumn fx:id="col_processor" prefWidth="116.0" text="Processor" />
                <TableColumn fx:id="col_storage" minWidth="1.0" prefWidth="58.0"
text="Storage" />
                <TableColumn fx:id="col_pcamera" prefWidth="93.0" text="PCamera" />
                <TableColumn fx:id="col_scamera" minWidth="7.0" prefWidth="59.0"
text="Scamera" />
                <TableColumn fx:id="col_battery" prefWidth="77.0" text="Battery" />
              </columns>
            </TableView>
            <TextField fx:id="searchfield" layoutX="466.0" layoutY="20.0" prefHeight="25.0"
prefWidth="162.0" promptText="ModelNo" />
          <Label fx:id="status" layoutX="14.0" layoutY="14.0" text="MOBILES
DATABASE">
              <font>
                <Font name="System Bold Italic" size="21.0" />
              </font>
            </Label>
            <Button layoutX="642.0" layoutY="20.0" mnemonicParsing="false"
onAction="#searchPhone" prefHeight="25.0" prefWidth="67.0" text="SEARCH" />
          </children>
          <padding>
            <Insets bottom="10.0" left="10.0" right="10.0" top="10.0" />
          </padding>
        </AnchorPane>
        <Button layoutX="739.0" layoutY="23.0" mnemonicParsing="false"
onAction="#refreshTable" prefHeight="25.0" prefWidth="67.0" text="REFRESH" />
    </children>
</AnchorPane>
```

8. LoginController.java

```
package sample;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
import java.io.IOException;

public class Logincontroller {

    int status;

    @FXML    private PasswordField passfield;
    @FXML    private TextField unamefield;
    @FXML    private Label lblstatus;
    @FXML    private Label likelbl;

    public void loginbuttonaction(ActionEvent event) throws IOException, Exception {
        status=Phones.accountsearch(unamefield.getText(),passfield.getText());
```

```
        if (status==2)
        {
            lblstatus.setText("Login Successful");                    //Show status as successful
            Stage primaryStage  = new Stage();                        //5 Lines To Open a new Window
            Parent root = FXMLLoader.load(getClass().getResource("mobile.fxml"));
            primaryStage.setTitle("Mobile  Database Management");
            primaryStage.setScene(new Scene(root, 996, 595));
            primaryStage.show();

        }
        else if (status==1)
        {
            lblstatus.setText("Login Successful");                    //Show status as successful
            Stage primaryStage  = new Stage();                        //5 Lines To Open a new Window
            Parent root = FXMLLoader.load(getClass().getResource("standardusermobile.fxml"));
            primaryStage.setTitle("Mobile  Database Management");
            primaryStage.setScene(new Scene(root, 833, 549));
            primaryStage.show();
        }else
            lblstatus.setText("Login Failed");                        //Wrong  input
    }
}
```

CHAPTER 7: SCREENSHOTS

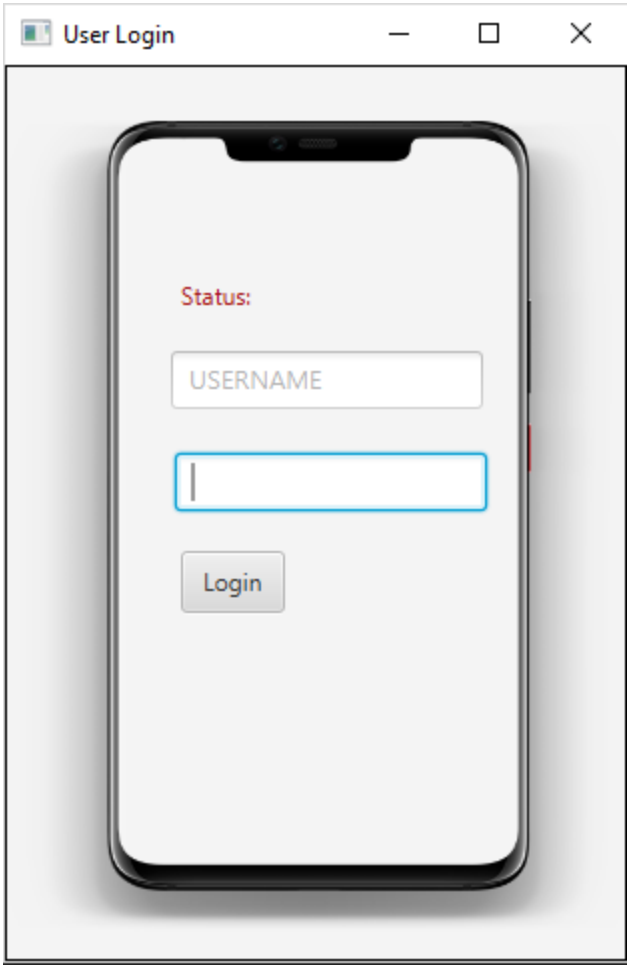
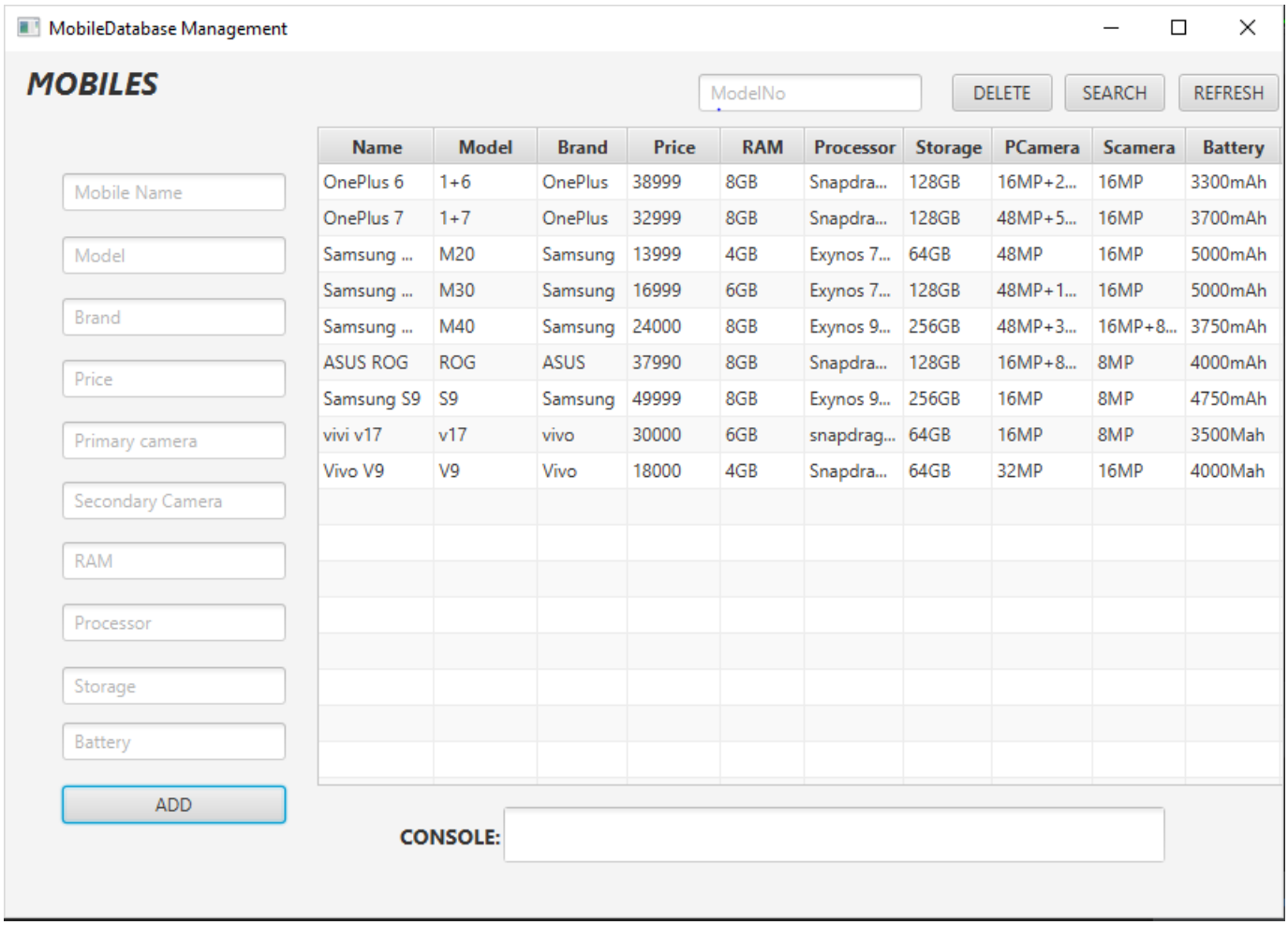
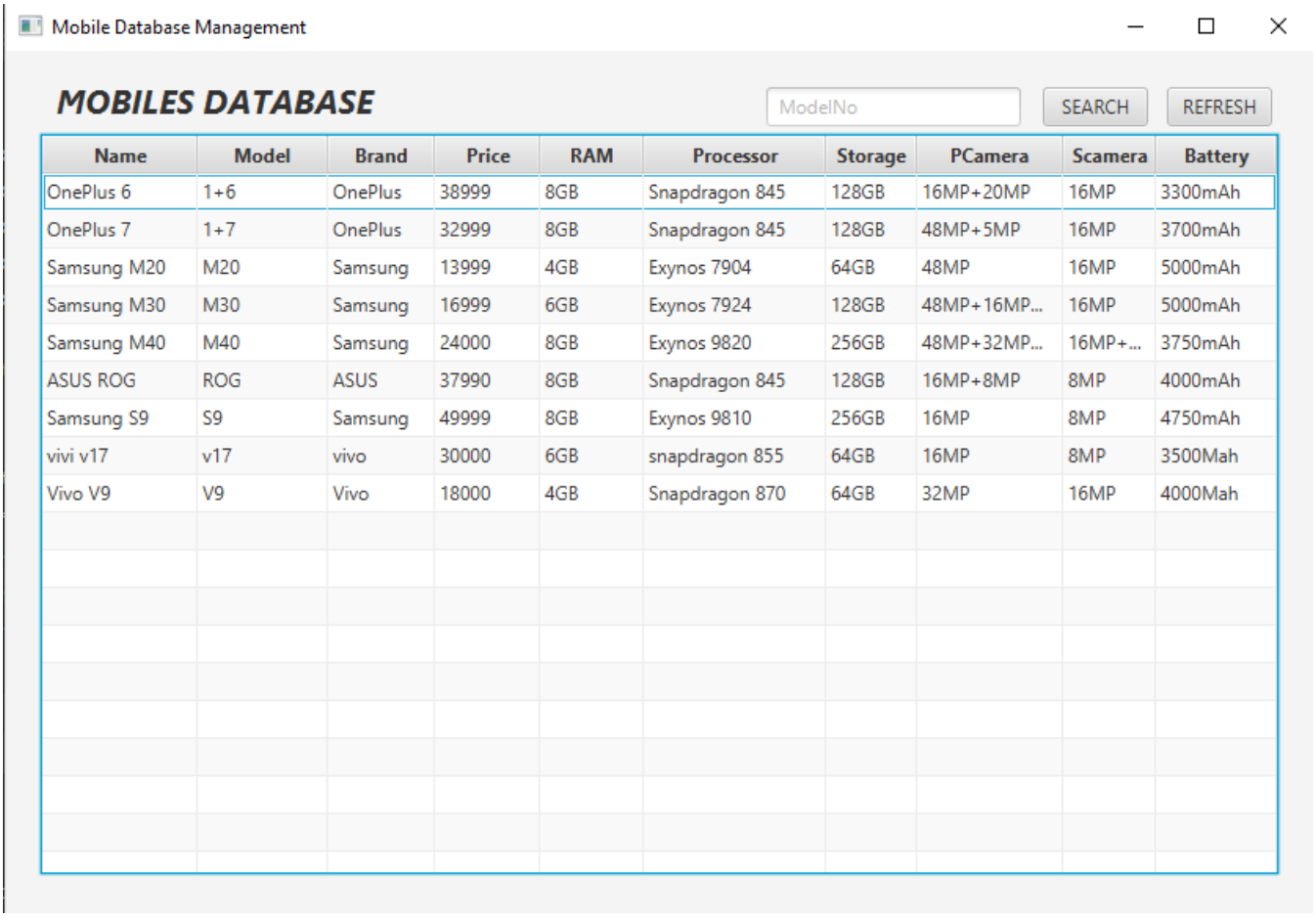


Fig 7.1: Login Interface



Name	Model	Brand	Price	RAM	Processor	Storage	PCamera	Scamera	Battery
OnePlus 6	1+6	OnePlus	38999	8GB	Snapdra...	128GB	16MP+2...	16MP	3300mAh
OnePlus 7	1+7	OnePlus	32999	8GB	Snapdra...	128GB	48MP+5...	16MP	3700mAh
Samsung ...	M20	Samsung	13999	4GB	Exynos 7...	64GB	48MP	16MP	5000mAh
Samsung ...	M30	Samsung	16999	6GB	Exynos 7...	128GB	48MP+1...	16MP	5000mAh
Samsung ...	M40	Samsung	24000	8GB	Exynos 9...	256GB	48MP+3...	16MP+8...	3750mAh
ASUS ROG	ROG	ASUS	37990	8GB	Snapdra...	128GB	16MP+8...	8MP	4000mAh
Samsung S9	S9	Samsung	49999	8GB	Exynos 9...	256GB	16MP	8MP	4750mAh
vivi v17	v17	vivo	30000	6GB	snapdrag...	64GB	16MP	8MP	3500Mah
Vivo V9	V9	Vivo	18000	4GB	Snapdra...	64GB	32MP	16MP	4000Mah

Fig 7.2: Admin User Interface



Name	Model	Brand	Price	RAM	Processor	Storage	PCamera	Scamera	Battery
OnePlus 6	1+6	OnePlus	38999	8GB	Snapdragon 845	128GB	16MP+20MP	16MP	3300mAh
OnePlus 7	1+7	OnePlus	32999	8GB	Snapdragon 845	128GB	48MP+5MP	16MP	3700mAh
Samsung M20	M20	Samsung	13999	4GB	Exynos 7904	64GB	48MP	16MP	5000mAh
Samsung M30	M30	Samsung	16999	6GB	Exynos 7924	128GB	48MP+16MP...	16MP	5000mAh
Samsung M40	M40	Samsung	24000	8GB	Exynos 9820	256GB	48MP+32MP...	16MP+...	3750mAh
ASUS ROG	ROG	ASUS	37990	8GB	Snapdragon 845	128GB	16MP+8MP	8MP	4000mAh
Samsung S9	S9	Samsung	49999	8GB	Exynos 9810	256GB	16MP	8MP	4750mAh
vivi v17	v17	vivo	30000	6GB	snapdragon 855	64GB	16MP	8MP	3500Mah
Vivo V9	V9	Vivo	18000	4GB	Snapdragon 870	64GB	32MP	16MP	4000Mah

Fig 7.3: StandardUser Interface

CHAPTER 8: CONCLUSION

This mini project is to give a basic idea of how to work with DBMS and Java Interface so as to understand the basics of Database System and storing and generating large amount of data easily and efficiently.

This project is design to take advantage of today's technology and reduce or avoid the burden of storing data on paper or in files. Our database management project aims to provide computerized interface to all the data stored and manipulated. The application provides for retrieving, stroing, modifying, maintaining and easy use of data.

CHAPTER 9: FUTURE SCOPE

The project can be hosted to a global server where each and every person can access the software. The program can be updated later by adding few more functionalities like pictures and other detailed information about the mobile phones. Other gadget devices can also be included in the software. The program is versatile and can be updated with the exiting code.

REFERENCE

- http://www.javapoint.com
- http://www.geekforgeeks.com
- http://www.wikipedia.com
- http://www.tutorialspoint.com