Testing Report

For

Road Repair and Tracking Software

* **R.Sri Charan Reddy (14CS10037)**
* **G.Sai Bharath Chandra (14CS10020)**

1. **INTRODUCTION**

This document sketches out the test plan for Road Repair and Tracking Software (RRTS). This prescribes the nature, purpose and methodology of all testing activities. The test plan is aimed at verifying the functionality and correct working of every aspect and part of the Road Repair and Tracking Software (RRTS) which aims to automate the various book keeping activities of road repair of the Public Works Department of the Corporation of a large city.

**1.1 OBJECTIVE**

The software is tested using appropriate testing strategies which exhaustively test the entire program and any bugs, if detected are reported and ways to correct them are suggested.

Software Testing is an important stage in Software Development Life Cycle (SDLC).The definition of testing is the process of analyzing a software item, to detect the differences between existing and required conditions i.e. defects/errors/bugs and to evaluate the features of the software item.

* 1. **Testing Strategy**

The strategy is to detect the differences between existing and required conditions and to evaluate the features of the software item.

Specific test plan components include:

* Purpose for testing
* Items/Features to be tested
* Pass/Fail criteria
* Bugs if any
  1. **Scope**

The testing will be performed at several points in the life cycle as the product is constructed. Testing is a very ‘dependent’ activity. As a result, test planning is a continuing activity performed throughout the system development life cycle. Test plans must be developed for each level of product testing.

* 1. **Purpose**

The purpose of testing is to verify and validate a software to check if it fits the requirements specified and to find the bugs present in a software. The bugs found are then fixed.

 **Bug** is a variance between the expected and actual result. The bug’s ultimate source may be traced to a fault introduced in the specification, design, or development (coding) phases.

**2. Features to be tested**

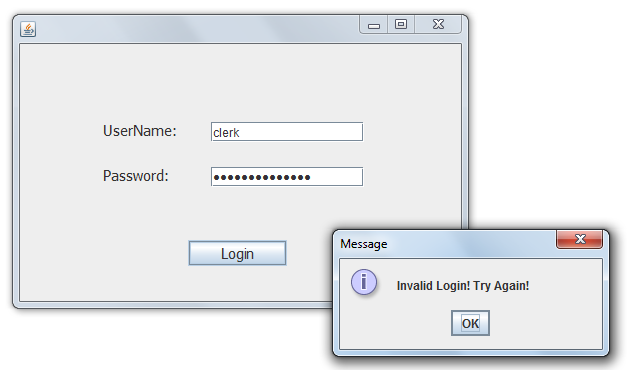
Some areas/functions in the software are to be examined and handled carefully as they are error prone.

Majority of the users of RRTS include:

* Clerks
* Supervisors
* City Corporation Administrator
* Mayor

1. If any person (Clerk/Supervisor/City Admin/Mayor) try to access their accounts of Clerk with unregistered usernames or with wrong passwords then the software raises an alert.
2. If a Supervisor doesn’t prioritize the work then the repair for that particular complaint should not be scheduled and the software gives a warning message in these scenarios.
3. Whenever administrator updates an existing calendar element an alert is generated to all the supervisors when they login to their respective accounts showing them to reschedule the work.
4. When supervisor clicks schedule button ,if all complaints are already scheduled ,then software raises an alert showing "scheduling already done" else it schedules only those works which are previously unscheduled.

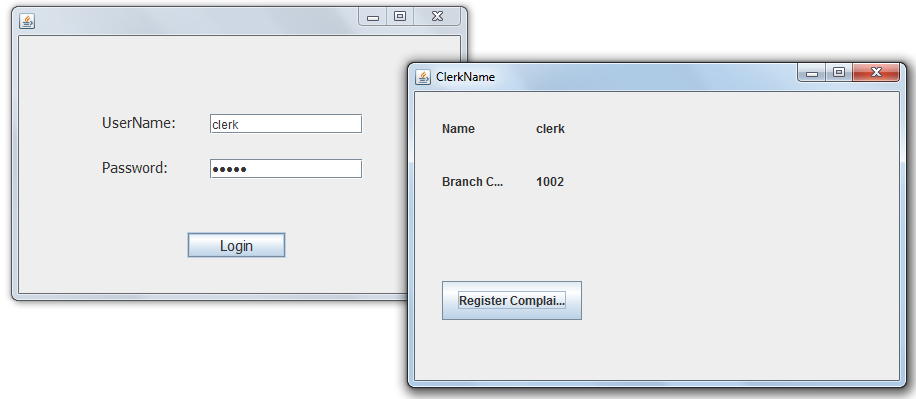
User Interface Testing:

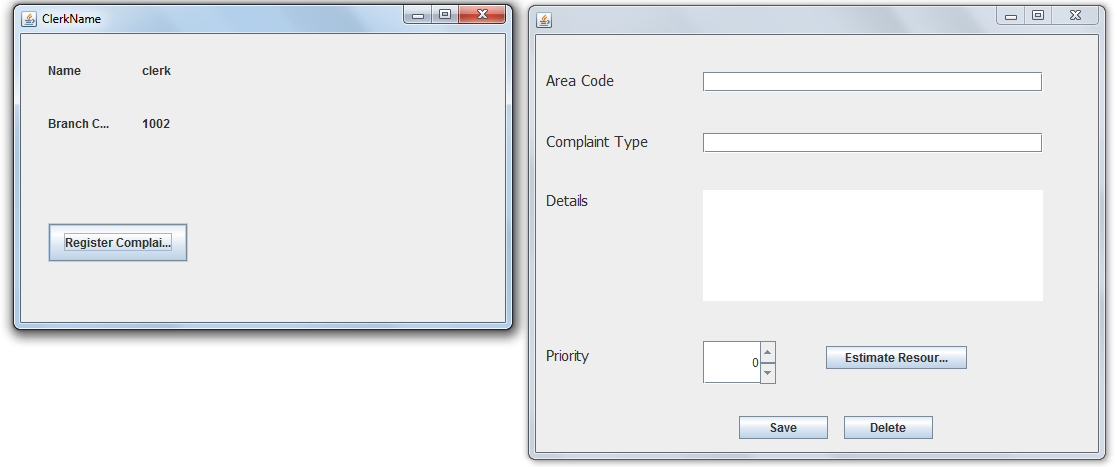


Login Window of Clerk:

**Input :** Wrong Username or Password

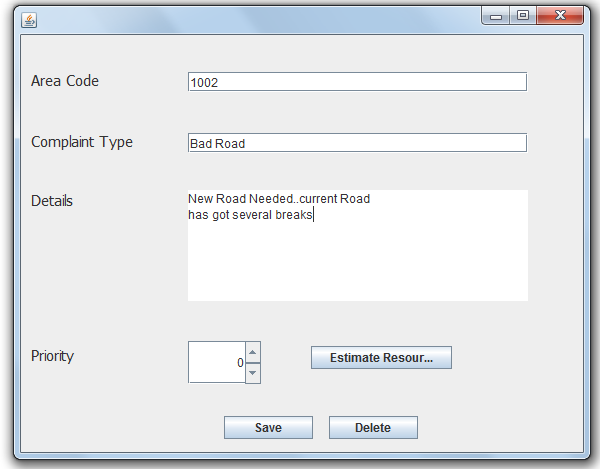
**Output:** Error Message

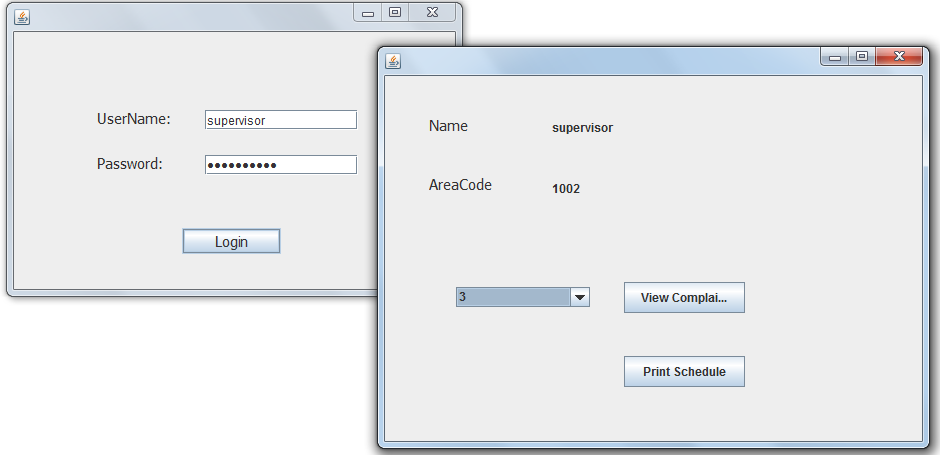




Complaint Window of Clerk:

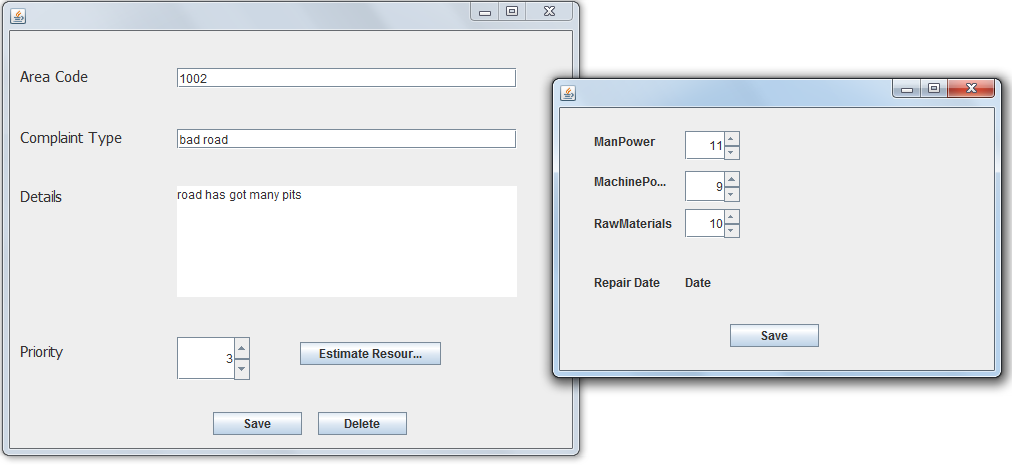
* Clerk after clicking Register Complaint Button will be directed to new Complaint window, where he can enter his complaint.
* Priority, Estimate Resources, Delete buttons are inactive for Clerk.



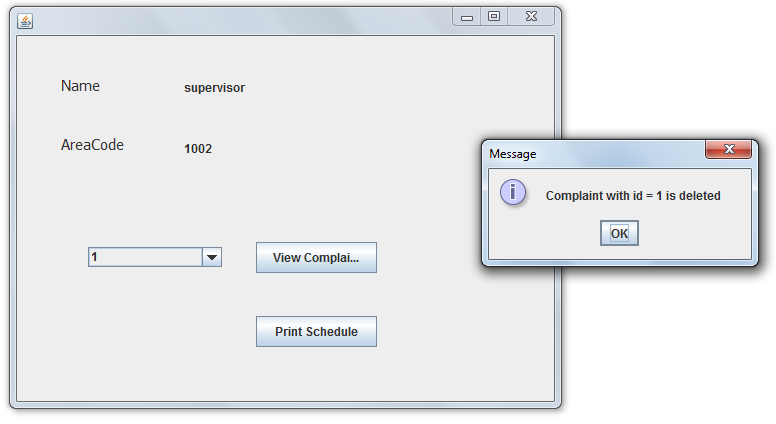
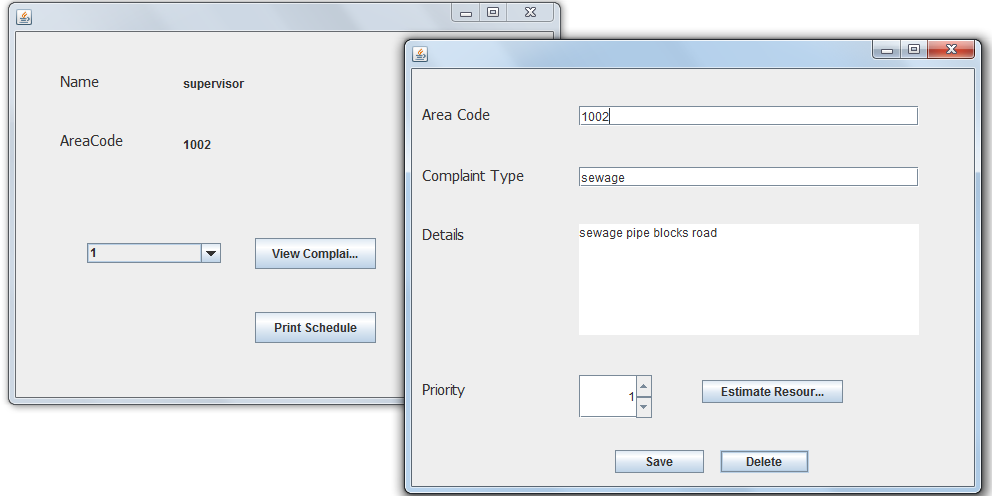


Login Window of Supervisor:

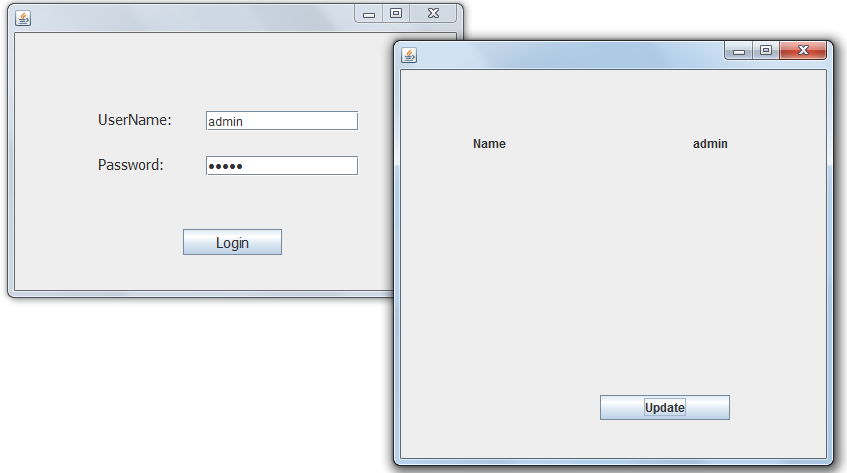
* On logging in supervisor window is opened where he can open Complaint window and also can Schedule based on Priorities,Man Power,Machine Power,Raw Materials.



* Supervisor has access to Priority JSpinner and Estimate Resources Window,where he can enter them and save them.

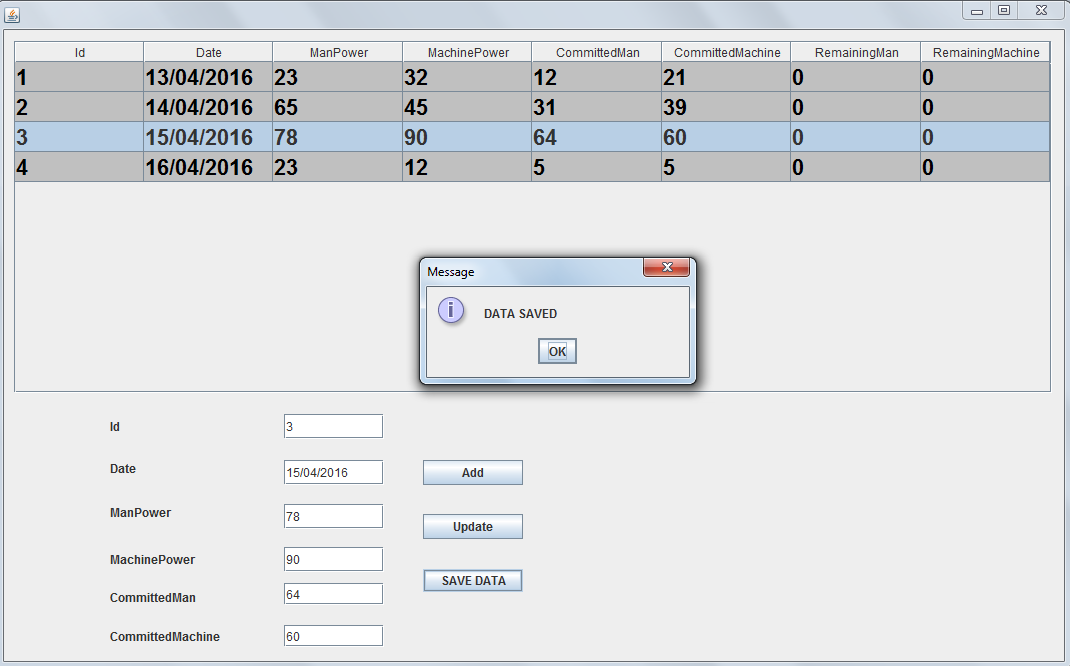
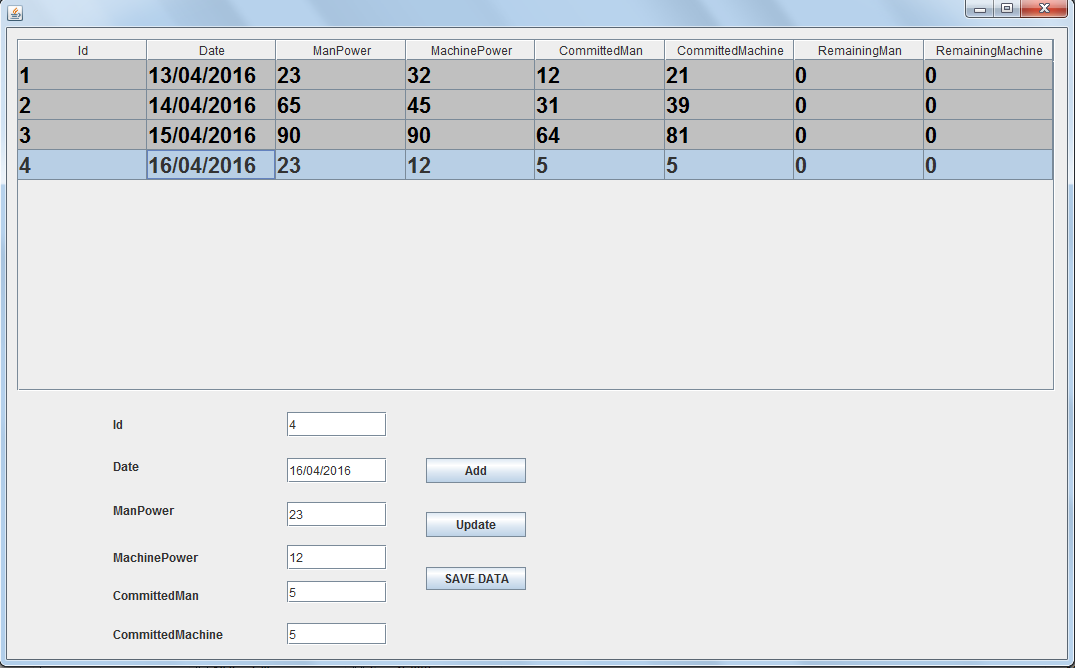
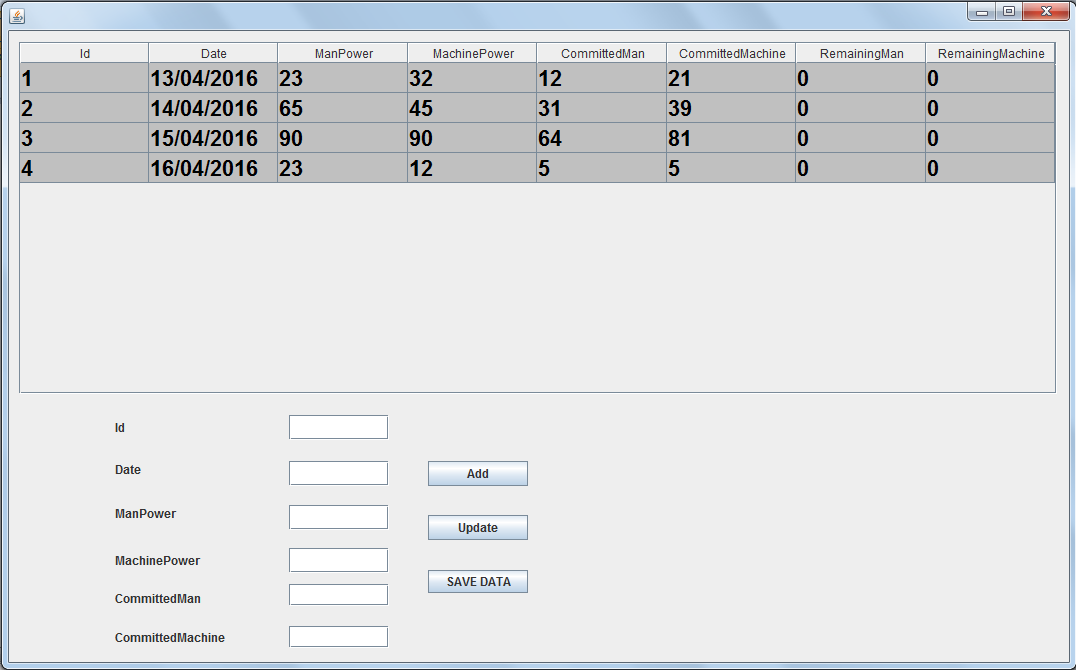


* Supervisor can also delete a Complaint.

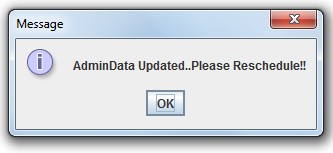
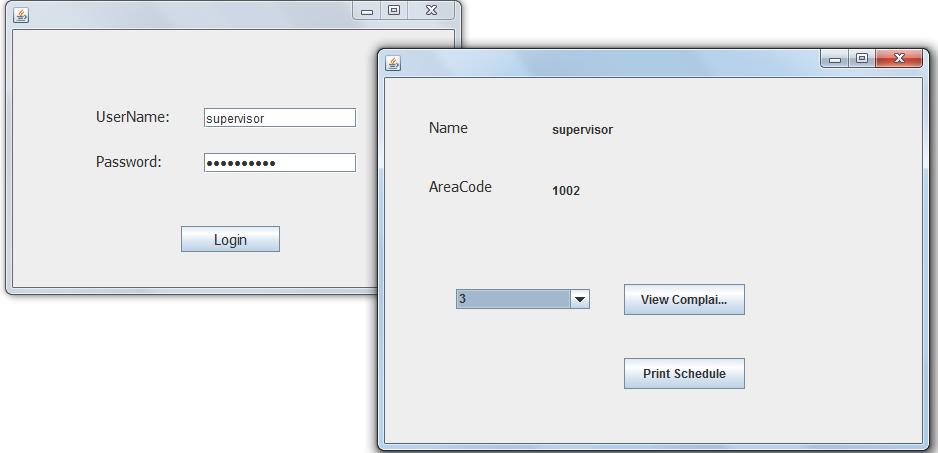


City Administrator Login Window:

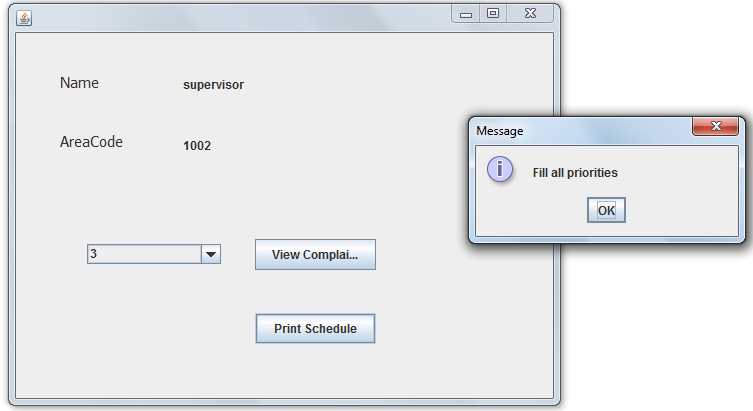
* Administrator on logging in will have access to Update button,where he can update Resources Data at any time.



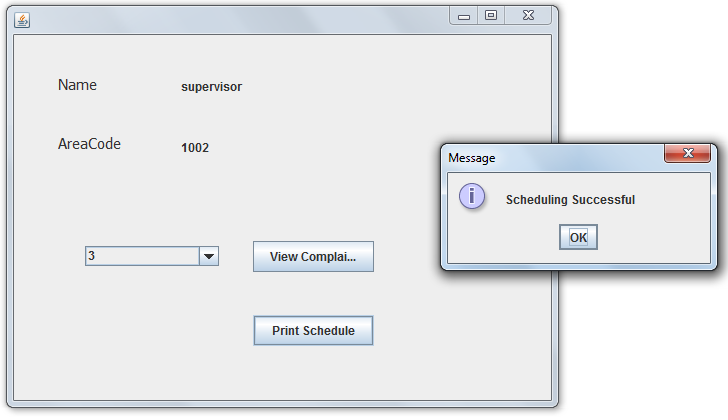
* Administrator can add/update the Data and the Data will be saved to Database by clicking on SAVE DATA Button.



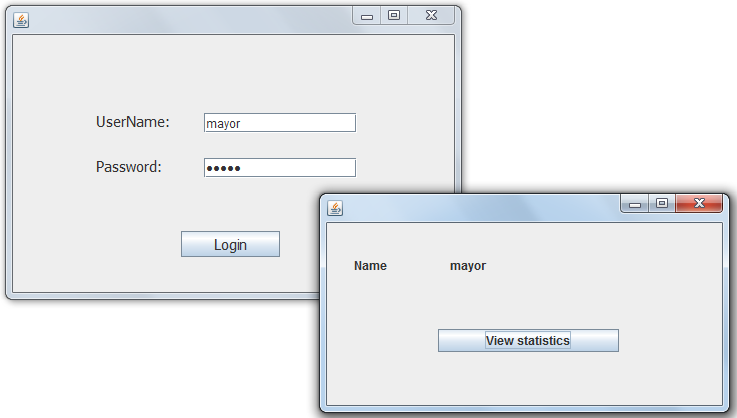
* After the Updating of Details in Database by administrator, if again supervisor logins he will get a message to again schedule as the data is updated.



* For the supervisor to schedule he needs to give priorities to all complaint requests as the scheduling is done based on priorities and various materials available, Otherwise a Error message will be displayed.

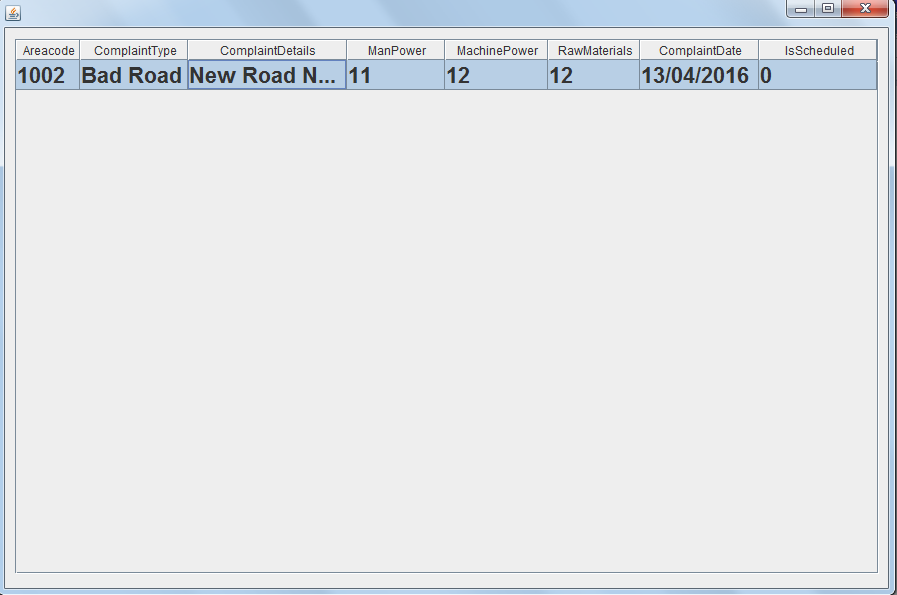


* Message is displayed on successful Scheduling of the Complaints.



Login Window of Mayor:

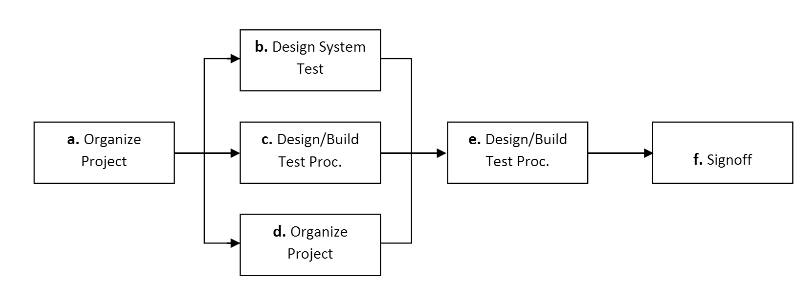
* On Logging on, mayor can View Statistics.

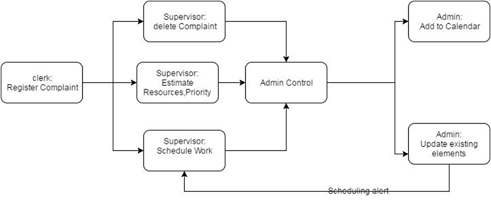


View Statistics Window of Mayor:

* On Clicking View Statistics button, he can view only the copy of Complaint Database in which he cannot update any of the values.
* It is only for viewing purpose.

Testing Process:





Black Box Testing:

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end user would. We have decided to perform Error guessing and Boundary Value Analysis testing on our application.

**BBT for Login:**

|  |  |  |
| --- | --- | --- |
| Test Case: | Input<Username,password> | Output corresponding user window > else <Error Message> |
| #1 | <clerk,clerk> | < clerk window> |
| #2 | <clerk,qwerty> | <Invalid Login Message> |
| #3 | <supervisor,supervisor> | <supervisor window> |
| #4 | <admin, admin > | < admin window> |
| #5 | <mayor, mayor > | < mayor window> |

**BBT for Register Complaint:**

|  |  |  |
| --- | --- | --- |
| Testcase: | input<Valid clerk login,Enter Complaint info ,Save> | output<Saved> |
| #1 | <{clerk ,clerk},{Complaint info},{Save}> | <Saved> |

**BBT for Delete Complaint:**

|  |  |  |
| --- | --- | --- |
| Testcase: | input<Valid supervisor login, View Complaints, Delete> | output<Deleted> |
| #1 | <{ supervisor , supervisor },{View Complaint},{Delete}> | <Deleted> |

**BBT for Scheduling:**

|  |  |  |
| --- | --- | --- |
| Testcase: | input<Valid supervisor login, Priority, Resources info, Schedule> | output <Scheduled> |
| #1 | <{ supervisor , supervisor },{priority},{resources info},{schedule}> | <Scheduled> |
| Testcase: | input<Valid supervisor login, No Priority, Schedule> | output <Error Message> |
| #1 | <{ supervisor , supervisor },{no priority},{schedule}> | <Error message> |

**BBT for Updating Calendar admin:**

|  |  |  |
| --- | --- | --- |
| Testcase: | input<Valid admin login,Enter info,Add button,save> | output <Added to Table,Added to Database> |
| #1 | <{admin,admin},{info},{Add},{Save}> | <Added to Table,Added to Database> |
| Testcase: | input<Valid admin login,Enter info,Update button,Save,Login Supervisor> | output <Updated in Table, Updated database, Reschedule Message{supervisor login}> |
| #1 | <{admin, admin},{info},{Add},{Save},{ supervisor , supervisor }> | < Updated in Table, Updated database, Reschedule Message{supervisor login}> |

**BBT of Mayor:**

|  |  |  |
| --- | --- | --- |
| Testcase: | input<Valid Mayor login,View> | output<Display table> |
| #1 | <{mayor,mayor},{View}> | <Dislplay table> |

White Box Testing:

In white box testing, the UI is bypassed. Inputs and outputs are tested directly at the code level and the results are compared against specifications. This form of testing ignores the function of the program under test and will focus only on its code and the structure of that code. Test cases are generated that not only cause each condition to take on all possible values at least once, but that cause each such condition to be executed at least once.

**WBT for Login:**

String name=textField.getText()

String pass=passwordField.getText();

try {

PreparedStatement ps=con.prepareStatement("select \*from Logindetails where Username=? and Password=?");

ps.setString(1,name);

ps.setString(2,pass);

ResultSet rs=ps.executeQuery();

if(rs.next()){

int i=rs.getInt("Employee Type");

int e=0,y=0;

if(i==0){

Sclrk=new SClerk();

Sclrk.setName(name);

Sclrk.getnamelbl().setText(name);

String p=Integer.toString(rs.getInt("AreaCode"));

Sclrk.getcodelbl().setText(p);

Sclrk.setBranch(rs.getInt("AreaCode"));

Sclrk.setVisible(true);

SComplaint.whocalled=0;

}

if(i==1){

Ssup=new SSupervisor();

Ssup.setname(name);

Ssup.getnamelbl().setText(name);

String p=Integer.toString(rs.getInt("AreaCode"));

Ssup.getAreaCodelbl().setText(p);

Ssup.setcode(rs.getInt("AreaCode"));

Ssup.setVisible(true);

SComplaint.whocalled=1;

}

if(i==2){

Sadm.setname(name);

Sadm.getnamelbl().setText(name);

Sadm.setVisible(true);

}

if(i==3){

Smay.setname(name);

Smay.getnamelbl().setText(name);

Smay.setVisible(true);

}

if(i!=0&&i!=1&&i!=2&&i!=3){

JOptionPane.showMessageDialog(null, "Invalid Login! Try Again!");

}

}

rs.close();

ps.close();

} catch (SQLException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

CFG for Login:

**WBT for Scheduling:**

PreparedStatement pstisadmin;

**try** {

pstisadmin = con.prepareStatement("select \* from AdminData"); ResultSet rsisadmin=pstisadmin.executeQuery();

**if**(!rsisadmin.next()){JOptionPane.*showMessageDialog*(**null**, "AdminData is empty");}

**else**{

**boolean** com=**true**;

**try**{

String sq=lblCode.getText();

**int** it=Integer.*parseInt*(sq) PreparedStatement pst=con.prepareStatement("select \* from ComplaintData where AreaCode='"+it+"' and boolschedule='"+0+"' ");

ResultSet rs=pst.executeQuery();

**if**(!rs.next()){com=**false**;JOptionPane.*showMessageDialog*(**null**, "Scheduling done already");}

**else**{

**do**

{ **boolean** isf=rs.getBoolean("Filled");

**if**(!isf)

{ JOptionPane.*showMessageDialog*(**null**,"Fill all priorities"); com=**false**;

**break**;

}

}**while**(rs.next());

}

rs.close();

pst.close();

} **catch** (SQLException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

**if**(com){

**int** i;

**int** a=comboBox.getItemCount();

ArrayList<SComplaint> cpl=**new** ArrayList<SComplaint>(a);

**int** arr[]=**new** **int**[a];

**try** {

**for**(i=0;i<a;i++)

{ **int** poi=(**int**)comboBox.getItemAt(i);

PreparedStatement pst=con.prepareStatement("select \* from ComplaintData where Complaintid='"+poi+"'");

ResultSet rs=pst.executeQuery();

cpl.add(**new** SComplaint());

**while**(rs.next())

{

cpl.get(i).setmanp(rs.getInt("ManPower"));

cpl.get(i).setmachp(rs.getInt("MachinePower"));

cpl.get(i).setpriority(rs.getInt("Priority"));

cpl.get(i).setid(rs.getInt("Complaintid"));

**break**;

//arr[i]=cpl[i].getpriority();

}

rs.close();

pst.close();

}

cpl.sort(**new** Comparator<SComplaint>() {

**public** **int** compare(SComplaint o1, SComplaint o2) {

**if**(o1.getpriority()>o2.getpriority())

**return** o1.getpriority();

**else**

**return** o2.getpriority();

}

});

PreparedStatement pst1=**null**;

ResultSet rs1=**null**;

**for**(**int** k=0;k<a;k++)

{

pst1=con.prepareStatement("select \* from AdminData");

rs1=pst1.executeQuery();

**int** manp=cpl.get(k).getmanpower();

**int** macp=cpl.get(k).getmachinepower(); **int** prip=cpl.get(k).getpriority();

**int** cid=cpl.get(k).getid(); **while**(rs1.next()) {

**if**(manp<(rs1.getInt("ManPower")-rs1.getInt("CommittedMan")) && macp<(rs1.getInt("MachinePower")-rs1.getInt("CommittedMachine")))

{

PreparedStatement pst2=con.prepareStatement("update ComplaintData set IsScheduled='"+rs1.getInt("Id")+"', boolschedule='"+1+"' where Complaintid='"+cid+"'");

pst2.execute();

pst2.close();

**int** j=rs1.getInt("CommittedMan")+manp;

**int** jq=rs1.getInt("CommittedMachine")+macp;

PreparedStatement pst3=con.prepareStatement("update AdminData set CommittedMan='"+j+"',CommittedMachine='"+jq+"' where Id='"+rs1.getInt("Id")+"'");

pst3.execute();

pst3.close();

**break**;

}

}

}

rs1.close();

pst1.close();

JOptionPane.*showMessageDialog*(**null**, "Scheduling Successful");

}

**catch** (SQLException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

}//check if all priorities are filled

}//check if admin data is empty

rsisadmin.close();

pstisadmin.close();

} **catch** (SQLException e2) {

// **TODO** Auto-generated catch block

JOptionPane.*showMessageDialog*(**null**, "Error Scheduling!!");

}

CFG for Scheduling:

**WBT for Updating Calendar:**

**try**

{

String query ="select \* from AdminData";

PreparedStatement pst=*conn*.prepareStatement(query);

ResultSet rs=pst.executeQuery();

**while**(rs.next())

{

PreparedStatement pstdel=*conn*.prepareStatement("delete from AdminData where Id='"+rs.getInt("Id")+"'");

pstdel.execute();

pstdel.close();

}

rs.close();

pst.close();

**boolean** isit=**false**;

Object[] q=**new** Object[8];

**int** row = table.getRowCount();

**int** column = table.getColumnCount();

**for** (**int** j = 0; j < row; j++) {

**for** (**int** i = 0; i < column; i++) {

q[i]= table.getValueAt(j,i);

}

String query1 ="insert into AdminData (Id,Date,ManPower,MachinePower,CommittedMan,CommittedMachine,RemainingMan,RemainingMachine) values(?,?,?,?,?,?,?,?)"; PreparedStatement pst1=*conn*.prepareStatement(query1);

**if**(j<chckup.size()){

**if**(chckup.get(j).getid()!=q[0]||chckup.get(j).getholdmanp()!=q[2]||chckup.get(j).getholdmacp()!=q[3]){ SSupervisor.*isupdated*=1;}

}

pst1.setObject(1, q[0]);

pst1.setObject(2, q[1]);

pst1.setObject(3,q[2]);

pst1.setObject(4, q[3]);

pst1.setObject(5, q[4]);

pst1.setObject(6, q[5]);

pst1.setInt(7, 0);

pst1.setInt(8, 0);

pst1.execute();

pst1.close();

isit=**true**;

}

**if**(isit){

JOptionPane.*showMessageDialog*(**null**, "DATA SAVED");

dispose();

}

// dispose();

}

**catch**(Exception e)

{

JOptionPane.*showMessageDialog*(**null**, "Error");

}

CFG for Updating Calendar: