****

Elevator SYSTEm

Testing Report

-Samrat Bikram Shah  
-1815674

1. **Introduction:**

This project expected us to create a desktop application of an Elevator System which had two floors and a Log Button. This task tested our problem-solving skills making us utilize all the things we had learned so far. We were expected to not only create the application but also use all the concepts we have been taught. For instance, using the OOPS approach with features like Encapsulation, Abstraction, Inheritance, etc. Another example is using Timer and delegation to animate the lift. Likewise, we were also asked to use a database to record the required logs and make the program as optimized as possible through Background worker and State patterns. We were also asked to keep in mind the program being able to accommodate to the future changes in the application.

1. **Unit Testing**

Unit Testing a software testing where components of a software are individually or independently tested. It is done to make sure all the components work as they are intended. A unit is the smallest testable part of any software. Unit Testing has been performed for the following tasks that were required in the elevator system.

**Marking Matrix with Self-Assessment:**

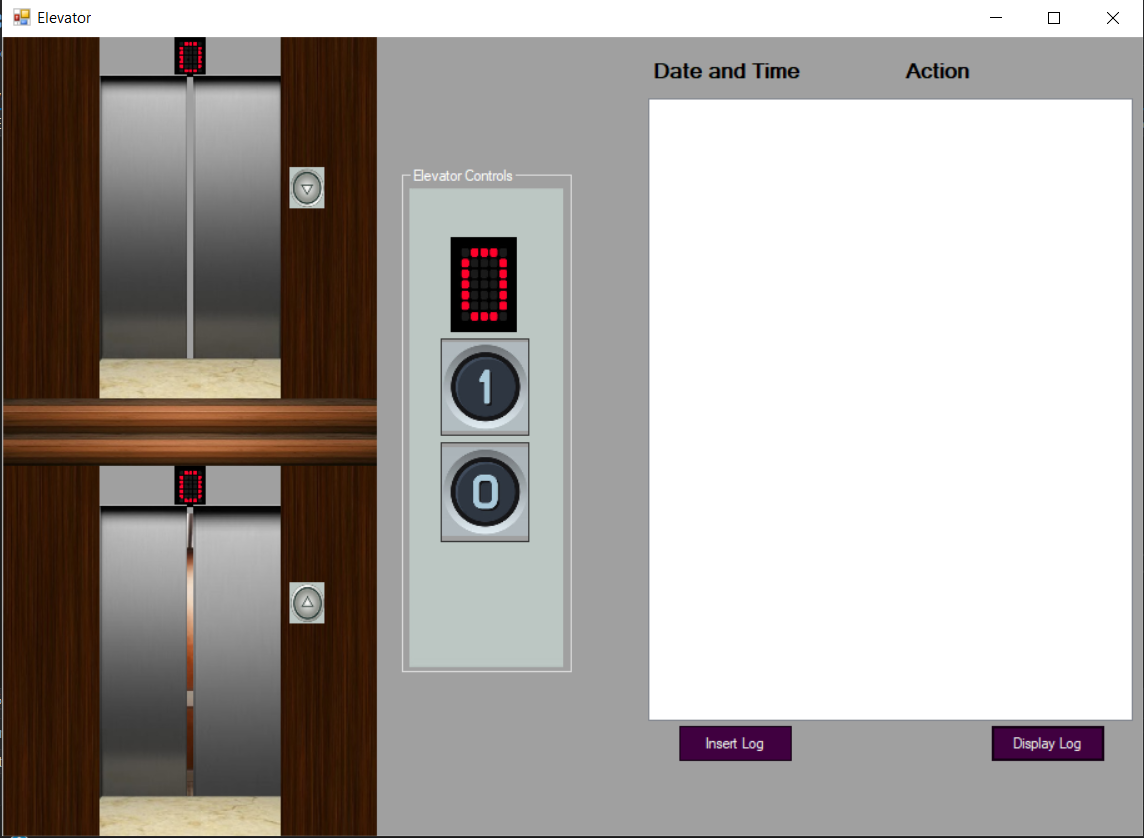
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Task Number** | **Sub-tasks** | **Possible Marks** | **Self-assessment (completed Yes/No)** | **Reference to your testing report** | **Mark Awarded** |
| **Task 1** | Complete GUI for Task 1 | 20 | Yes | [1.](#Test1) | 20 |
| Skeleton of event handlers in place for all buttons | 10 | Yes | [2.](#Test2) | 10 |
| **Task 2** | All event handlers are functional | 10 | Yes | [2.](#Test2) | 10 |
| **Task 3** | Database (DB) is designed and can be connected | 5 | Yes | [3.](#Test3) | 5 |
| Log Information can be retrieved from DB and displayed in the GUI | 5 | Yes | [4.](#Test4) | 5 |
| When the log button is pressed, log information is sent to and stored in the DB | 5 | Yes | [5.](#Test5) | 5 |
| Data source is updated via DataAdapters Update() method instead of ExecuteNonQuery() method | 5 | Yes | [5.](#Test5) | 5 |
| **Task 4** | Events described in Task 2 animated using delegation and timer | 10 | Yes | [2.](#Test2), [6.](#Test6) | 10 |
| **Task 5** | Using relative path instead of absolute path | 5 | Yes | [7.](#Test7) | 5 |
| Avoiding any duplication among the event handlers over the database related functions | 5 | Yes | [8.](#Test8) | 5 |
| Eliminating logical errors and handling exceptions with try and catch | 5 | Yes | [9.](#Test9) | 5 |
| Optimise the efficiency of GUI by implementing multiple tasks concurrently via BackgroundWorker | 5 | Yes | [10.](#Test10) | 5 |
| Use state patterns instead of if-else statements to accommodate future changes of the requirement | 10 | No |  |  |
| **Total** |  | 100 |  |  | 90 |

Additional Work: [Sea-Level diagram](#SeaLevel), [Activity diagram](#Activity), [E-R diagram.](#ER)

1. **Graphical User Interface (GUI):**

The GUI has two floors, display panels indicating the floor we are in, two request buttons one on each floor and a control panel to move up or down while inside the lift. It has a Display Log which displays the log information from the database. Likewise, it also has an Insert Log button which inserts the log data into the database which can be viewed using a display log.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 1. | To ensure the GUI works. | Pass |



1. **Event Handlers:**

For any application to be functional, event handlers are required. They listen to any actions done by the user such as mouse clicks or keystrokes and processes them as accordingly.

Event handlers are used in this application to make the buttons and the PictureBoxes functional in order to make sure they perform the task they are required to.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 2. | The event handlers are working. Buttons are functional. | Pass |

The codes are as follows:

private void Downbtn\_Click(object sender, EventArgs e)

{

Downbtn.Image = Properties.Resources.DownbtnClick;

Downbtn.Enabled = false;

Upbtn.Enabled = false;

AutoUpTimer.Stop();

AutoDownTimer.Stop();

Delay.Stop();

DownCloseTimer.Start();

}

private void Upbtn\_Click(object sender, EventArgs e)

{

Upbtn.Image = Properties.Resources.UpbtnClick;

Downbtn.Enabled = false;

Upbtn.Enabled = false;

AutoUpTimer.Stop();

AutoDownTimer.Stop();

Delay.Stop();

UpCloseTimer.Start();

}

private void FloorOnebtn\_Click(object sender, EventArgs e)

{

if (InsideLift.Top == DownLiftY)

{

DisableButton();

AutoUpTimer.Stop();

AutoDownTimer.Stop();

Delay.Stop();

DownCloseTimer.Start();

}

}

private void FloorZerobtn\_Click(object sender, EventArgs e)

{

if (InsideLift.Top == UpLiftY)

{

DisableButton();

AutoUpTimer.Stop();

AutoDownTimer.Stop();

Delay.Stop();

UpCloseTimer.Start();

}

}

private void Logbtn\_Click(object sender, EventArgs e)

{

try

{

LogBox.Items.Clear();

DataSet ds = conn.DisplayData();

foreach (DataRow row in ds.Tables[0].Rows)

{

LogBox.Items.Add(row["Logtime"].ToString() + "\t\t" + row["Logaction"].ToString());

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

private void InsertLogbtn\_Click(object sender, EventArgs e)

{

DbBackgroundWorker.RunWorkerAsync();

}

Here, every button has an event handler associated with it which performs a particular function as required on click.

1. **Database Connectivity:**

Database is used in this application to store the log events of all the actions that takes place when the elevator is operating. For this the Insert Log button is used which uses the DataAdapter to insert the data into the database. In this case, an Access database is used.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 3. | To make sure the Database is connected. | Pass |

The code are as follows:

static string dbconnection = "Provider=Microsoft.ACE.OLEDB.12.0;" + @"data source = ElevatorLogDatabase.accdb";

static OleDbConnection Connection = new OleDbConnection(dbconnection);

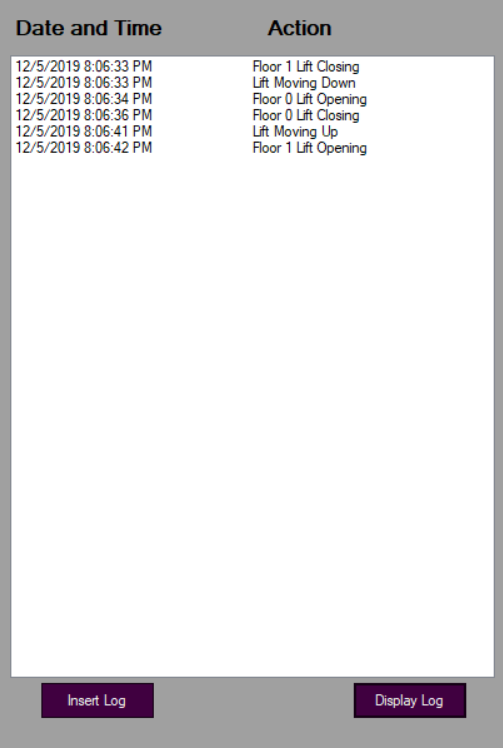
static string dbcommand = "SELECT \* FROM Logtable;";

static OleDbCommand command = new OleDbCommand(dbcommand, Connection);

static OleDbDataAdapter da = new OleDbDataAdapter(command);

The connection string defines the OLEDB driver with the stated location of the database. The connection is established via a Data Adapter.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 4. | To display log information from the Database into the GUI. | Pass |



The codes are as follows:

public DataSet DisplayData()

{

try

{

DataSet ds = new DataSet();

da.Fill(ds);

return ds;

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

return null;

}

}

private void Logbtn\_Click(object sender, EventArgs e)

{

try

{

LogBox.Items.Clear();

DataSet ds = conn.DisplayData();

foreach (DataRow row in ds.Tables[0].Rows)

{

LogBox.Items.Add(row["Logtime"].ToString() + "\t\t" + row["Logaction"].ToString());

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

}

}

The data is displayed into the GUI through a ListBox which uses the DataSet to retrieve the copy of selected fields and records. The data is called in another class via a DisplayData function which returns the DataSet which can be used to add the information from the database into the GUI.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 5. | Log information is stored into the database only after clicking the Insert Log Button via DataAdapter Update() method. | Pass |

The codes are as follows:

public void StoreData(string task)

{

DateTime now = DateTime.Now;

string datenow = now.ToString();

string logevent = task + "," + datenow;

loglist.Add(logevent);

logarray = loglist.ToArray();

}

public void UpdateData()

{

try

{

DataSet ds = new DataSet();

OleDbCommandBuilder ocb = new OleDbCommandBuilder(da);

ds.Clear();

da.Fill(ds);

DataTable dt = ds.Tables[0];

foreach (string log in logarray)

{

DataRow dr = dt.NewRow();

string[] data = log.Split(',');

dr["Logaction"] = data[0];

dr["Logtime"] = data[1];

dt.Rows.Add(dr);

}

DataSet dsc = ds.GetChanges();

da.Update(dsc);

dt.AcceptChanges();

MessageBox.Show("Inserted data into database successfully.");

loglist.Clear();

Array.Clear(logarray, 0, logarray.Length);

}

catch

{

MessageBox.Show("Perform an action first before inserting into database");

}

}

When the Insert Log button is pressed the UpdateData function is called which stores all the action log into the database. In order to make sure data is not stored at every action, the data is temporarily kept in a list via StoreData function which is then used by the UpdateData function as soon as the button is clicked. For updating, the DataAdapter Update() method is used rather than the ExecuteNonQuery() Method. The connection string and the DataAdapter initialization are all done in the class itself.

1. **Delegation and Timer:**

In order to animate the lift to move up and down and to open and close the doors, delegation and timers are used. In this case, the timers were added through drag and drop in the Form selection and the Timer tick events and delegations are auto-generated.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 6. | To animate the lift up and down and open-close the lift doors. | Pass |

The code are as follows:

private void MoveUpTimer\_Tick(object sender, EventArgs e)

{

mov.MovingUp(InsideLift, MoveUpTimer, UpOpenTimer, UpArrowDisplay, FlooroneDisplay, Downbtnimage);

}

private void MoveDownTimer\_Tick(object sender, EventArgs e)

{

mov.MovingDown(InsideLift, MoveDownTimer, DownOpenTimer, DownArrowDisplay, FloorzeroDisplay, Upbtnimage);

}

private void DownCloseTimer\_Tick(object sender, EventArgs e)

{

oc.CloseDown(DownLeftLift, DownRightLift, DownCloseTimer, MoveUpTimer);

}

private void UpCloseTimer\_Tick(object sender, EventArgs e)

{

oc.CloseUp(UpLeftLift, UpRightLift, UpCloseTimer, MoveDownTimer);

}

private void UpOpenTimer\_Tick(object sender, EventArgs e)

{

oc.OpenUp(UpLeftLift, UpRightLift, UpOpenTimer, Delay, EnableButton);

}

private void DownOpenTimer\_Tick(object sender, EventArgs e)

{

oc.OpenDown(DownLeftLift, DownRightLift, DownOpenTimer, Delay, EnableButton);

}

All of the above are generated via the Form and are the Timer Tick events which uses delegates to help us animate the elevator system. All the Timer tick events call a function in another class which does the movement of the PictureBox.

The code are as follows:

public void CloseDown(PictureBox DownLeftLift, PictureBox DownRightLift, Timer downclosetimer, Timer moveuptimer)

{

if (DownLeftLift.Left == 78)

{

downclosetimer.Stop();

conn.StoreData("Floor 0 Lift Closing");

moveuptimer.Start();

}

else if (DownLeftLift.Left >= 9)

{

DownLeftLift.Left += 1;

DownRightLift.Left -= 1;

}

}

public void CloseUp(PictureBox UpLeftLift, PictureBox UpRightLift, Timer upclosetimer, Timer movedowntimer)

{

if (UpLeftLift.Left == 78)

{

upclosetimer.Stop();

conn.StoreData("Floor 1 Lift Closing");

movedowntimer.Start();

}

else if (UpLeftLift.Left >= 9)

{

UpLeftLift.Left += 1;

UpRightLift.Left -= 1;

}

}

public void OpenUp(PictureBox UpLeftLift, PictureBox UpRightLift, Timer upopentimer, Timer delaytimer, Action enablebutton)

{

if (UpLeftLift.Left == 9)

{

upopentimer.Stop();

enablebutton();

conn.StoreData("Floor 1 Lift Opening");

delaytimer.Start();

}

else if (UpLeftLift.Left <= 103)

{

UpLeftLift.Left -= 1;

UpRightLift.Left += 1;

}

}

public void OpenDown(PictureBox DownLeftLift, PictureBox DownRightLift, Timer downopentimer, Timer delaytimer, Action enablebutton)

{

if (DownLeftLift.Left == 9)

{

downopentimer.Stop();

enablebutton();

conn.StoreData("Floor 0 Lift Opening");

delaytimer.Start();

}

else if (DownLeftLift.Left <= 103)

{

DownLeftLift.Left -= 1;

DownRightLift.Left += 1;

}

}

public void MovingUp(PictureBox insidelift, Timer moveuptimer, Timer upopentimer, Action uparrowdisplay, Action flooronedisplay, Action downbtnimage)

{

if (insidelift.Top == 37)

{

moveuptimer.Stop();

conn.StoreData("Lift Moving Up");

flooronedisplay();

downbtnimage();

upopentimer.Start();

}

else if (insidelift.Top <= 381)

{

uparrowdisplay();

insidelift.Top -= 1;

}

}

public void MovingDown(PictureBox insidelift, Timer movedowntimer, Timer downopentimer, Action downarrowdisplay, Action floorzerodisplay, Action upbtnimage)

{

if (insidelift.Top == 381)

{

movedowntimer.Stop();

conn.StoreData("Lift Moving Down");

floorzerodisplay();

upbtnimage();

downopentimer.Start();

}

else if (insidelift.Top >= 37)

{

downarrowdisplay();

insidelift.Top += 1;

}

}

The codes above are called in ticks until the timer is stopped. The timer stops when the picture box that is being animated reaches the position it is needed to go. Hence, by using timers and delegation, the animation of the elevator is handled.

1. **Optimization**

Only creating an application is never sufficient. Developers needs to make sure their app is optimized and consume as less memory as possible. They should also keep in mind that no user is willing to use their application if it lags too much or consumes a lot of memory from the PC.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 7. | To make sure everything works after using a relative path instead of Absolute. | Pass |

Code:

static string dbconnection = "Provider=Microsoft.ACE.OLEDB.12.0;" + @"data source = ElevatorLogDatabase.accdb";

All the files including the database are in the same working directory in the relative path.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 8. | To make sure the data is not being inserted when the previous insert is being done. | Pass |

The code is as follows:

if (!DbBackgroundWorker.IsBusy)

{

DbBackgroundWorker.RunWorkerAsync();

}

else

{

MessageBox.Show("Updating Database");

}

Since BackgroundWorker has been used it was easy to make sure the functions didn’t overlap. When the BackgroundWorker was not busy the data always updated else the Error message would show up.

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 9. | Exception Handling with try-catch is working. | Pass |

Try-catch has been used in a few places especially in the database. It throws an exception whenever an error occurs in the code. One example of an error occurring would be the database. If no action is performed in the elevator and someone tries to add empty value in the database, it will return an error and will be handled by the catch statement.

The code are as follows:

public void UpdateData()

{

try

{

DataSet ds = new DataSet();

OleDbCommandBuilder ocb = new OleDbCommandBuilder(da);

ds.Clear();

da.Fill(ds);

DataTable dt = ds.Tables[0];

foreach (string log in logarray)

{

DataRow dr = dt.NewRow();

string[] data = log.Split(',');

dr["Logaction"] = data[0];

dr["Logtime"] = data[1];

dt.Rows.Add(dr);

}

DataSet dsc = ds.GetChanges();

da.Update(dsc);

dt.AcceptChanges();

MessageBox.Show("Inserted data into database successfully.");

loglist.Clear();

Array.Clear(logarray, 0, logarray.Length);

}

catch

{

MessageBox.Show("Perform an action first before inserting into database");

}

|  |  |  |
| --- | --- | --- |
| Test No. | Objective | Result |
| 10. | BackgroundWorker is working as needed. | Pass |

A basic Background worker has been used in this program for inserting into the database. The background worker helps to execute the line of code in a separate dedicated thread so it won’t cause any delay or issue in the GUI itself.

The code are as follows:

private void InsertLogbtn\_Click(object sender, EventArgs e)

{

if (!DbBackgroundWorker.IsBusy)

{

DbBackgroundWorker.RunWorkerAsync();

}

else

{

MessageBox.Show("Updating Database");

}

}

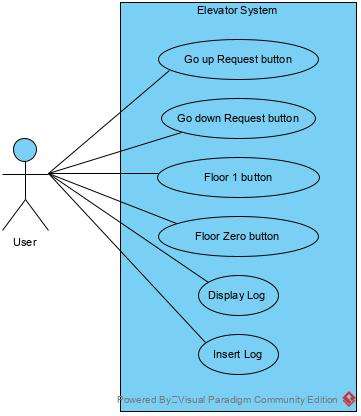
private void DbBackgroundWorker\_DoWork(object sender, DoWorkEventArgs e)

{

conn.UpdateData();

}

Here, the insert button toggles the Background worker which helps to update the database in a separate thread whenever it is not being updated continuously.

1. **Diagrams**

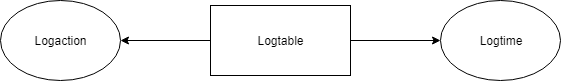


Fig: E-R diagram

Fig: Sea Level Diagram

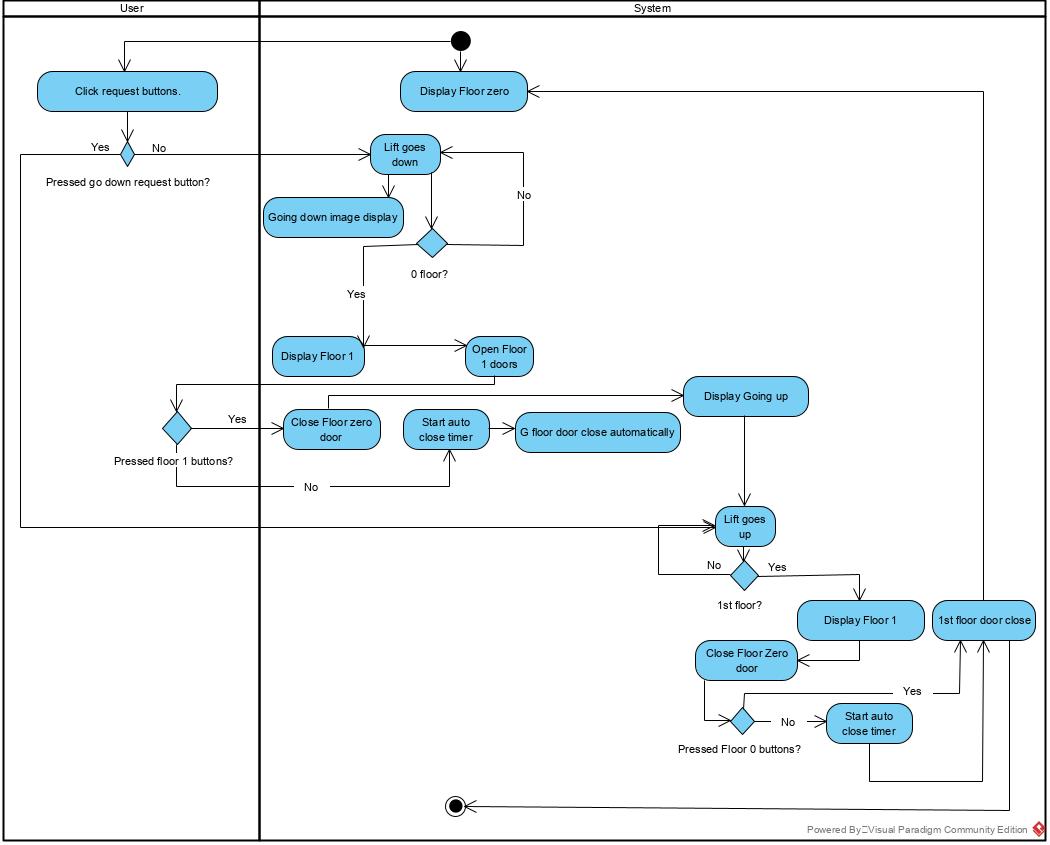


Fig: Activity Diagram

1. **Conclusion**

In conclusion, the application was made according to the requirements. All the features work as it is intended. The only issue is some more optimization and upgradability of the application since the State-patterns weren’t implemented. Besides that, the application works and runs smooth and well.